Proiectul 3

Order

Management

Alexuc Tudor  
Grupa 30221/1 - An2 CTI

**1.Obiectivul temei**

**Cerinta :**

Creati o aplicație OrderManagement pentru procesarea comenzilor clienților pentru un depozit. O structura de baze de date relaționale va fi utilizata pentru a stoca produsele, clienții și comenzile. În plus, aplicația ar trebui să fie structurată în pachete folosind o arhitectură stratificată prezentată în suport material (Assignment\_3\_Indications.pdf) și ar trebui să utilizeze (minim) următoarele clase:

• Clasele de modele - modelele de date ale aplicației

• Clase de logică de afaceri - implementează logica aplicației

• Clase de prezentare - implementează intrarea / ieșirea utilizatorului

• Clasele de acces la date - implementează accesul la baza de date

Aplicația ar trebui să permită procesarea comenzilor dintr-un fișier text dat ca argument, efectuați operatiile solicitate, salvați datele din baza de date și generați rapoarte în format pdf. Alte clase și pachete pot fi adăugate pentru a implementa funcționalitatea completă a aplicației.

Asadar, din enuntul dat reiese ca trebuie creata o aplicatie ce ajuta utilizatorul in managementul unui depozit. Prin intermediul aplicatiei utilizatorul va putea introduce, modifica sau sterge produse/clienti. De asemenea, el va putea vizualiza clientii , produsele si cererile clientilor ce vor fi prezentate sub forma unui tabel.

Clientul va avea posibilitatea de a cumpara produse din cadrul magazinului, iar in cazul in care cantitatea pe care si-o doreste sa o achizitioneze este mai mare decat cantitatea disponibila in depozit, se va afisa un mesaj de eroare.

**Obiective secundare :**

1.Crearea bazei de date MySql

2.Crearea modelelor de date in Java

3.Crearea conexiunii intre baza de date si programul nostru

4.Prelucrarea datelor din program in baza de date

5.Afisarea rapoartelor in format pdf

**2.Analiza problemei**

O ordine logica a rezolvarii este :

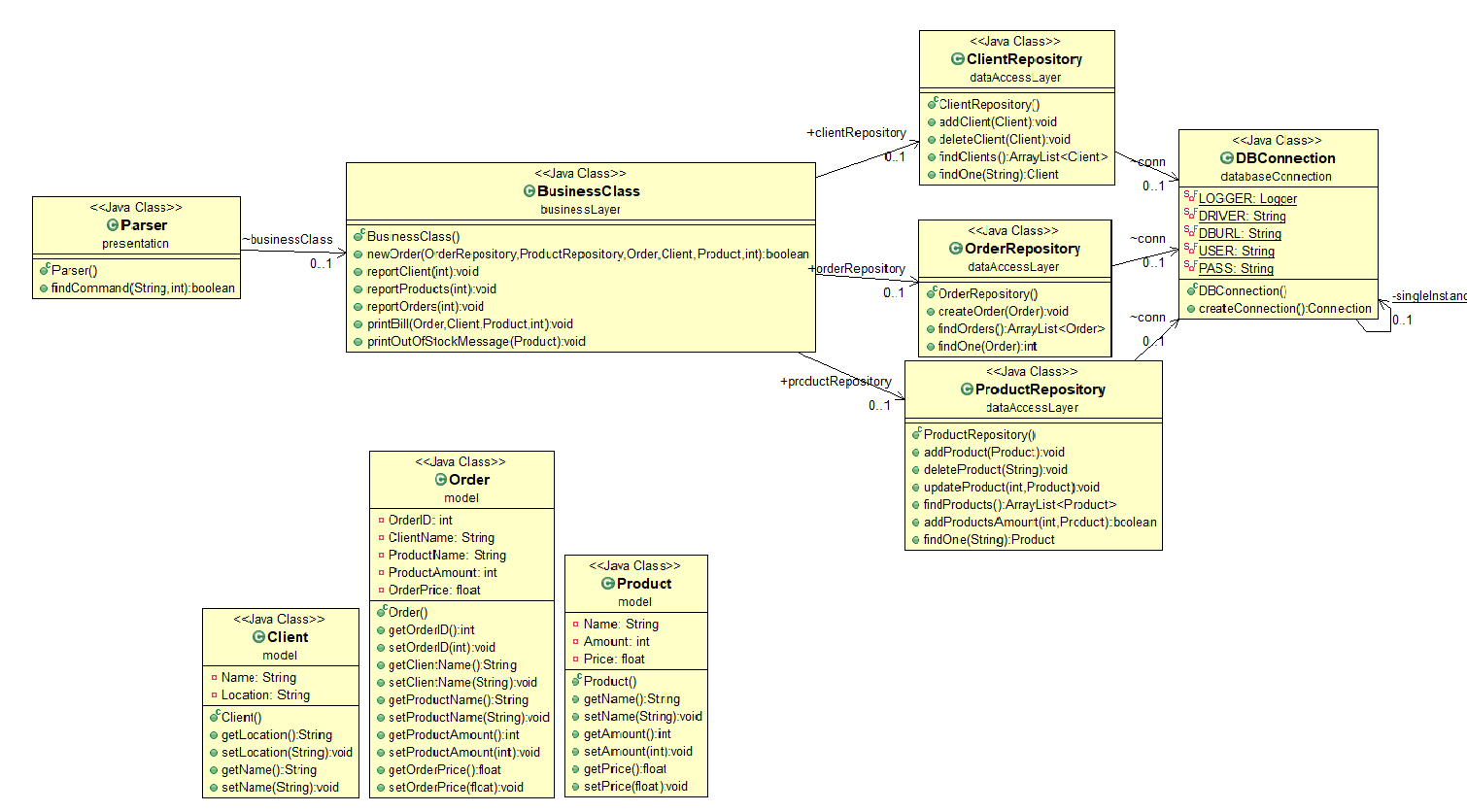
* Crearea bazei de date
* Obtinerea accesului la DB(Database)
* Extragerea informatiilor relevante
* Procesarea si modificarea datelor
* Afisarea rezultatelor cu ajutorul DB-ului si a rapoartelor pdf

Informatiile despre clienti, produse si comenzi vor fi stocate intr-o baza de date de tip relational. Modelarea datelor se va face de la un layer la altul,in primul rand datele vor fi prelucrate de “Data access classes” care sunt reprezentate de “ClientRepository”,”ProductRepository” si “OrderRepository”. Datele de baza vor fi modelate cu ajutorul unor clase de baza, de exemplu clasa “Client” ,”Product”, “Order”. Apoi datele vor fi trimise spre urmatorul layer si anume “Business Logic classes” care reprezinta legatura dintre stratul de data access si “Presentation classes” , care se ocupa de intrarea si iesitrea datelor din program dinspre si catre utilizator .

Majoritatea functiilor implementate sunt simplu de adaugare , stergere , gasire sau modificare a datelor , cu exceptia functiei de creare a comenzilor . Aceasta functie are doua cazuri , cazul in care cantitatea ceruta de client este in stock-ul produsului , caz in care va realiza crearea comenzii si se va crea un pdf pentru chitanta care continue datele tranzactiei : numele clientului , produsul , pretul total, cat si numarul comenzii se si cazul in care cantitatea depaseste stock-ul disponibil si programul va crea un pdf cu un mesaj de “Out of stock”.

**3.Proiectare**

**Diagrama UML:**



**Decizii de proiectare :**

Pentru aceasta aplicatie am folosit un model “layered” care se bazeaza pe transmiterea datelor de la un layer la altul(descrise mai sus). Acest model usureaza intretinerea programului si totodata asigura integritatea datelor si mentinerea securitatii.

Pentru accesul la date am folosit 3 clase Repository(ClientRepository,ProductRepository si OrderRepository), care contin cele 3 tipuri de date din DB si functii specifice pentru fiecare tip.

Structurile de date sunt grupate in layerul “model” al proiectului ele fiind implementarea elementelor bazei de date. Aceste clase au in componenta lor exact aceleasi tipuri primitive de date care sunt accesate in business layer cu ajutorul metodelor de “get” si “set”.

**Pachete:**

Pentru arhitectura layered am facut 5 pachete : dataAccessLayer ,businessLayer,model si presentation fiind principalele si databaseConnection pentru realizarea conexiunii dintre program si DB.

**databaseConnection** se ocupa de realizarea conexiunii cu baza de date implementata in MySQL Workbench . Pentru a stabili aceasta conexiune am declarat 5 constante statice de tip String care reprezinta adresa bazei de date, driver-ul, username-ul si parola. In aceasta clasa sunt implementate metode pentru creeare conexiunii si inchiderea conexiunii.

**dataAccessLayer** se ocupa de prelucrarea datelor prin cele 3 clase Repository

**model** contine tipurile de date extrase din DB si are campuri echivalente acestora

**businessLayer** realizeaza functii complexe ce implica mai multe tipuri de date , acest pachet foloseste repository-urile din dataAccessLayer pentru a obtine accesul la datele din DB si a le modifica

**presentation** se ocupa cu citirea din fisier si parsarea comenzilor date si apeleaza functiile necesare din businessLayer

****

**4.Implementare**

**ClientRepository**

Aceasta clasa se ocupa de functiile de adaugare ,stergere si listare a obiectelor din tabelul Client din baza de date.

De asemenea clasa contine si o functie de gasire a unui anumit client avand datele necesare , functie folosita in crearea unui obiect de tip Client in cadrul metodei newOrder.

public void addClient(Client client)  
{  
 try(Connection c = conn.createConnection())  
 {  
 String query = "INSERT INTO Client (Name,Location)" + "Values (?,?)";  
 PreparedStatement statement = c.prepareStatement(query);  
 statement.setString(1,client.getName());  
 statement.setString(2,client.getLocation());  
 statement.execute();  
 } catch (SQLException throwables) {  
 throwables.printStackTrace();  
 }  
}

public ArrayList<Client> findClients()  
{  
 String prest= "SELECT \* FROM Client";  
 ArrayList<Client> clients = new ArrayList<>();  
 try(Connection c = conn.createConnection()) {  
 PreparedStatement statement = c.prepareStatement(prest);  
 ResultSet rs = statement.executeQuery();  
 while(rs.next())  
 {  
 Client client = new Client();  
 client.setName(rs.getString("Name"));  
 client.setLocation(rs.getString("Location"));  
 clients.add(client);  
 }  
 } catch (SQLException throwables) {  
 throwables.printStackTrace();  
 }  
 return clients;  
}

public void deleteClient(Client client)  
{  
 try(Connection c = conn.createConnection())  
 {  
 String query = "delete from Client where Name = ? and Location = ?";  
 PreparedStatement statement = c.prepareStatement(query);  
 statement.setString(1, client.getName());  
 statement.setString(2, client.getLocation());  
 statement.execute();  
  
 } catch (SQLException throwables) {  
 throwables.printStackTrace();  
 }  
}

public Client findOne(String name)  
{  
 String prest= "select \* from Client where Name = ?";  
 Client client = new Client();  
 try(Connection c = conn.createConnection()) {  
 PreparedStatement statement = c.prepareStatement(prest);  
 statement.setString(1,name);  
  
 ResultSet rs = statement.executeQuery();  
 if(rs.next())  
 {  
 client.setName(rs.getString("Name"));  
 client.setLocation(rs.getString("Location"));  
 }  
 } catch (SQLException throwables) {  
 throwables.printStackTrace();  
 }  
 return client;  
}

**ProductRepository**

Aceasta clasa se ocupa de functiile de adaugare ,stergere si listare a obiectelor din tabelul Product din baza de date.

De asemenea clasa contine si o functie de gasire a unui anumit produs in DB avand numele acestuia , o functie pentru modificarea cantitatii si o functie pentru adaugarea unei cantitati suplimentare.

public ArrayList<Product> findProducts()  
{  
 String prest= "SELECT \* FROM Product";  
 ArrayList<Product> products = new ArrayList<>();  
 try(Connection c = conn.createConnection()) {  
 PreparedStatement statement = c.prepareStatement(prest);  
 ResultSet rs = statement.executeQuery();  
 while(rs.next())  
 {  
 Product product = new Product();  
 product.setName(rs.getString("Name"));  
 product.setAmount(rs.getInt("Amount"));  
 product.setPrice(rs.getFloat("Price"));  
 products.add(product);  
 }  
 } catch (SQLException throwables) {  
 throwables.printStackTrace();  
 }  
 return products;  
}

public void addProduct(Product product)  
{  
 try(Connection c = conn.createConnection())  
 {  
 String query = "INSERT INTO Product (Name,Amount,Price)" + "Values (?,?,?)";  
 PreparedStatement statement = c.prepareStatement(query);  
 statement.setString(1,product.getName());  
 statement.setInt(2,product.getAmount());  
 statement.setFloat(3,product.getPrice());  
 statement.execute();  
 } catch (SQLException throwables) {  
 throwables.printStackTrace();  
 }  
}

public void deleteProduct(String Name)  
{  
 try(Connection c = conn.createConnection())  
 {  
 String query = "delete from Product where Name = ?";  
 PreparedStatement statement = c.prepareStatement(query);  
 statement.setString(1, Name);  
 statement.execute();  
  
 } catch (SQLException throwables) {  
 throwables.printStackTrace();  
 }  
}

public boolean addProductsAmount(int amountAdded,Product product)  
{  
 String prest= "select \* from Product where Name = ?";  
 try(Connection c = conn.createConnection()) {  
 PreparedStatement statement = c.prepareStatement(prest);  
 statement.setString(1,product.getName());  
  
 ResultSet rs = statement.executeQuery();  
 if(rs.next())  
 {  
 updateProduct(product.getAmount()+amountAdded,product);  
 return true;  
 }  
 } catch (SQLException throwables) {  
 throwables.printStackTrace();  
 }  
 return false;  
}

public void updateProduct (int amount,Product product)  
{  
 try(Connection c = conn.createConnection())  
 {  
 String query = "update Product set Amount = ? where Name = ?";  
 PreparedStatement statement = c.prepareStatement(query);  
 statement.setInt(1, amount);  
 statement.setString(2, product.getName());  
 statement.execute();  
  
 }catch (SQLException throwables) {  
 throwables.printStackTrace();  
 }  
}

public Product findOne(String name)  
{  
 String prest= "select \* from Product where Name = ?";  
 Product product = new Product();  
 try(Connection c = conn.createConnection()) {  
 PreparedStatement statement = c.prepareStatement(prest);  
 statement.setString(1,name);  
  
 ResultSet rs = statement.executeQuery();  
 if(rs.next())  
 {  
 product.setName(rs.getString("Name"));  
 product.setAmount(rs.getInt("Amount"));  
 product.setPrice(rs.getFloat("Price"));  
 }  
 } catch (SQLException throwables) {  
 throwables.printStackTrace();  
 }  
 return product;  
}

**OrderRepository**

Aceasta clasa se ocupa de functiile de adaugare, listare si gasire a obiectelor din tabelul Order din baza de date.

public int findOne(Order order)  
{  
 int orderID = 0;  
 String prest= "select *max*(OrderID) from `Order` where ClientName = ? and ProductName = ? " +  
 "and ProductAmount = ? and OrderPrice = ?";  
 try(Connection c = conn.createConnection()) {  
 PreparedStatement statement = c.prepareStatement(prest);  
 statement.setString(1,order.getClientName());  
 statement.setString(2,order.getProductName());  
 statement.setInt(3,order.getProductAmount());  
 statement.setFloat(4,order.getOrderPrice());  
  
 ResultSet rs = statement.executeQuery();  
 if(rs.next())  
 {  
 orderID = rs.getInt("max(OrderID)");  
 }  
 } catch (SQLException throwables) {  
 throwables.printStackTrace();  
 }  
 return orderID;  
}

}

public void createOrder(Order order)  
{  
 try(Connection c = conn.createConnection())  
 {  
 String query = "INSERT INTO `Order` (ClientName,ProductName,ProductAmount,OrderPrice)"  
 + "Values (?,?,?,?)";  
  
 PreparedStatement statement = c.prepareStatement(query);  
 //statement.setInt(1,order.getOrderID());  
 statement.setString(1,order.getClientName());  
 statement.setString(2,order.getProductName());  
 statement.setInt(3,order.getProductAmount());  
 statement.setFloat(4,order.getOrderPrice());  
 statement.execute();  
 } catch (SQLException throwables) {  
 throwables.printStackTrace();  
 }  
}

public ArrayList<Order> findOrders()  
{  
 String prest= "SELECT \* FROM `Order`";  
 ArrayList<Order> orders = new ArrayList<>();  
 try(Connection c = conn.createConnection()) {  
 PreparedStatement statement = c.prepareStatement(prest);  
 ResultSet rs = statement.executeQuery();  
 while(rs.next())  
 {  
 Order order = new Order();  
 order.setOrderID(rs.getInt("OrderID"));  
 order.setClientName(rs.getString("ClientName"));  
 order.setProductName(rs.getString("ProductName"));  
 order.setProductAmount(rs.getInt("ProductAmount"));  
 order.setOrderPrice(rs.getFloat("OrderPrice"));  
 orders.add(order);  
 }  
 } catch (SQLException throwables) {  
 throwables.printStackTrace();  
 }  
 return orders;  
}

**BusinessRepository**

Aceasta clasa realizeaza crearea de comenzi noi modificand si cantitatea din product . De asemenea , tot in aceasta clasa se realizeaza si creare facturilor , a mesajelor de stoc indisponibil si a rapoartelor cu elementele fiecarui tabel.

public void printBill(Order order,Client client,Product product,int requiredAmount)  
{  
 Document document = new Document();  
 try {  
 PdfWriter.*getInstance*(document, new FileOutputStream("Bill"+order.getOrderID()+".pdf"));  
 } catch (DocumentException e) {  
 e.printStackTrace();  
 } catch (FileNotFoundException e) {  
 e.printStackTrace();  
 }  
  
 document.open();  
 Font font = FontFactory.*getFont*(FontFactory.*COURIER*, 16, BaseColor.*BLACK*);  
 Chunk chunk = new Chunk(client.getName() + " ordered " + requiredAmount +' '+ product.getName()  
 + " for a total of " + requiredAmount\*product.getPrice() +".", font);  
 try {  
 document.add(chunk);  
 } catch (DocumentException e) {  
 e.printStackTrace();  
 }  
 document.close();  
}

public boolean newOrder(OrderRepository orderRepository, ProductRepository productRepository,Order order,Client client,  
 Product product,int requiredAmount)  
{  
 if(requiredAmount < product.getAmount()) {  
 order.setProductAmount(requiredAmount);  
 order.setOrderPrice(requiredAmount\*product.getPrice());  
  
 orderRepository.createOrder(order);  
 productRepository.updateProduct(product.getAmount()-requiredAmount,product);  
  
 order.setOrderID(orderRepository.findOne(order));  
  
 printBill(order,client,product,requiredAmount);  
 return true;  
 }  
 else{  
 printOutOfStockMessage(product);  
 return false;  
 }  
  
}

public void reportClient(int c)  
{  
 Document document = new Document();  
 try {  
 PdfWriter.*getInstance*(document, new FileOutputStream("ClientReport"+c+".pdf"));  
 } catch (DocumentException e) {  
 e.printStackTrace();  
 } catch (FileNotFoundException e) {  
 e.printStackTrace();  
 }  
 document.open();  
 PdfPTable table = new PdfPTable(2);  
 Stream.*of*("Name", "Location")  
 .forEach(columnTitle -> {  
 PdfPCell header = new PdfPCell();  
 header.setBackgroundColor(BaseColor.*LIGHT\_GRAY*);  
 header.setBorderWidth(2);  
 header.setPhrase(new Phrase(columnTitle));  
 table.addCell(header);});  
 for (Client client: clientRepository.findClients())  
 {  
 table.addCell(client.getName());  
 table.addCell(client.getLocation());  
 }  
  
 try {  
 document.add(table);  
 } catch (DocumentException e) {  
 e.printStackTrace();  
 }  
 document.close();  
}

**DBConnection**

Realizeaza legatura dintre program si baza de date

private static final Logger *LOGGER* = Logger.*getLogger*(DBConnection.class.getName()); ;  
private static final String *DRIVER* = "com.mysql.jdbc.Driver";  
private static final String *DBURL* = "jdbc:mysql://localhost:3306/ordermanagement?useSSL=false";  
private static final String *USER* = "root";  
private static final String *PASS* = "";

private static DBConnection *singleInstance* = new DBConnection();  
  
public DBConnection()  
{  
 try  
 {  
 Class.*forName*(*DRIVER*);  
 }catch(ClassNotFoundException e)  
 {  
 e.printStackTrace();  
 }  
}

public Connection createConnection()  
{  
 Connection conn = null;   
 try  
 {  
 conn = DriverManager.*getConnection*(*DBURL*,*USER*,*PASS*);  
 } catch (SQLException e)  
 {  
 System.*out*.println("Invalid connection index");  
 e.printStackTrace();  
 }  
 return conn;  
}

**Parser**

Aceasta clasa realizeaza citirea din fisier si interpretarea comenzilor si apoi apelarea lor.

public boolean findCommand(String command,int reportNumber)  
{  
 String[] phrase = command.split(" ");  
 if(phrase[0].equals("Insert") || phrase[0].equals("insert"))  
 {  
 if(phrase[1].equals("Client:") || phrase[1].equals("client:"))  
 {  
 Client client = new Client();  
 client.setName(phrase[2]+" "+phrase[3].substring(0, phrase[3].length() - 1));  
 client.setLocation(phrase[4]);  
 businessClass.clientRepository.addClient(client);  
 }  
 if(phrase[1].equals("Product:") || phrase[1].equals("product:"))  
 {  
 Product product = new Product();  
 product.setName(phrase[2].substring(0, phrase[2].length() - 1));  
 product.setAmount(Integer.*parseInt*(phrase[3].substring(0, phrase[3].length() - 1)));  
 product.setPrice(Float.*parseFloat*(phrase[4]));  
 if(!businessClass.productRepository.addProductsAmount(product.getAmount(),product))  
 businessClass.productRepository.addProduct(product);  
 }  
 }

if(phrase[0].equals("Order:") || phrase[0].equals("order:"))  
{  
 Order order = new Order();  
 order.setClientName(phrase[1]+" "+phrase[2].substring(0, phrase[2].length() - 1));  
 order.setProductName(phrase[3].substring(0, phrase[3].length() - 1));  
 int requiredAmount = Integer.*parseInt*(phrase[4]);  
  
 Product product = businessClass.productRepository.findOne(order.getProductName());  
 Client client = businessClass.clientRepository.findOne(order.getClientName());  
 businessClass.newOrder(businessClass.orderRepository,businessClass.productRepository,order,client,product,requiredAmount);  
}  
  
if(phrase[0].equals("Delete") || phrase[0].equals("delete"))  
{  
 if(phrase[1].equals("Client:") || phrase[1].equals("client:"))  
 {  
 Client client = new Client();  
 client.setName(phrase[2]+" "+phrase[3].substring(0, phrase[3].length() - 1));  
 client.setLocation(phrase[4]);  
 businessClass.clientRepository.deleteClient(client);  
 }  
 if(phrase[1].equals("Product:") || phrase[1].equals("product:"))  
 {  
 businessClass.productRepository.deleteProduct(phrase[2]);  
 }  
}

if(phrase[0].equals("Report") || phrase[0].equals("report"))  
 {  
 if(phrase[1].equals("Client") || phrase[1].equals("client"))  
 {  
 businessClass.reportClient(reportNumber);  
 return true;  
 }  
 if(phrase[1].equals("Product") || phrase[1].equals("product"))  
 {  
 businessClass.reportProducts(reportNumber);  
 return true;  
 }  
 if(phrase[1].equals("Order") || phrase[1].equals("order"))  
 {  
 businessClass.reportOrders(reportNumber);  
 return true;  
 }  
 }  
 return false;  
}

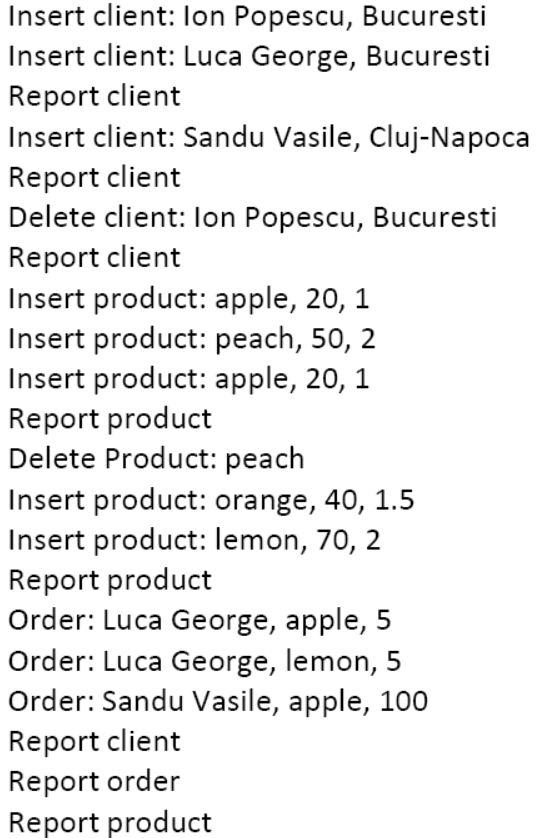
**5.Testare**

**Clasa main + .jar file**

Citirea datelor din fisier se face cu ajutorul java.io.File . Liniile din fisier au despartite in cuvinte, folosind functia split() ,apoi prelucrate de parser.

De asemenea , pentru a putea rula fisierul .jar si a da argumentele functiei din Command Prompt, in loc de path-ul spre fisierul de input am pus args[0].

Programul a fost testat pe exemplul dat si baza de data a fost modificata ca atare.



**Exemplul dat**

**6.Concluzii**

Programul este un exercitiul bun pentru implementarea unei aplicatii cu conexiune la baze de date si prin crearea lui am invatat si mi-am amintit atat concepte si moduri de lucru cu baze de date cat si cu arhitectura layered si conexiunea cu un program aditional a unui cod java.

Aplicatia poate fi extinsa sa contina si alte tabele sau alte functii de afisare a anumitor coloane si chiar unirea anumitor tabele compatibile.

**7.Bibliografie**

<https://www.baeldung.com/java-pdf-creation>

<https://stackoverflow.com/>

<https://www.javahelps.com/>

Warehouse Management Indications