



Facultatea de Automatică și Calculatoare

Specializarea Calculatoare

# Tema 3: Aplicație management comenzi

-documentație-

Strujan Florentina

Gr. 302210

An 2, semestrul 2



## Cuprins:

1.Obiectivul temei.....	3
2.Analiza problemei, modelare, scenarii, cazuri de utilizare .....	4
3.Proiectare .....	5
3.1Decizii de proiectare.....	5
3.2Diagrame UML.....	5
3.3Proiectare clase.....	7
3.4Metode.....	7
3.5Pachete.....	16
4. Implementare.....	17
5. Rezultate.....	18
6.Concluzii.....	19
7.Bibliografie.....	20



## 1. Obiectivul temei

Obiectivul temei de laborator a fost proiectarea și implementarea unei aplicații ce se ocupă cu procesarea comenzilor unui depozit. Folosim o bază de date pentru a stoca produsele, clienții, comenzile și produsele din comenzi, după care facem conexiunea cu baza de date și efectuăm diferite operații pe aceasta cu ajutorul programării orientate pe obiecte, rezultatele dându-se în format PDF.

Aplicația trebuie să fie în stare să însereze clienți, produse, comenzi și produsele aferente unei comenzi, să ștergă produse și clienți, la ștergerea clienților fiind nevoie de o actualizare automată a bazelor de date ce conțin numele acestuia, precum și să actualizeze toate tabelele în funcție de inserări ale aceluiași produs, de extrageri ale produselor pentru a fi achiziționate, precum și de apariția de mai multe ori a aceluiași cumpărător a produse diferite. Rezultatele vor fi chitanțe sau înștiințări de esuare a tranzacției, precum și rapoarte cu conținutul bazelor de date dorite în momentul dorit.

Obiectivele secundare sunt:

- folosirea programării orientate pe obiect (definirea de clase, metode, folosirea încapsulării etc.)
- folosirea limbajului de programare Java
- folosirea metodelor de maxim 30 linii și a claselor de maxim 300 linii
- folosirea convențiilor de denumire în Java
- folosirea javadoc pentru a documenta clasele și a genera fișiere JavaDoc corespunzătoare
- folosirea unei baze de date pentru a stoca datele necesare aplicației, cu cel puțin 3 tabele
- crearea unei chitanțe pentru fiecare comandă ca fișier .pdf
- un parser pentru comenzile citite dintr-un fișier text
- decrementarea stocului produsului odată cu finalizarea comenzii
- generarea unui document în cazul în care stocul produsului dorit de client nu este suficient.
- rapoarte în format pdf generate în urma rularii fișierului dat cu comenzi

Optional:

- un fișier .jar creat și configurat pentru a putea fi rulat conform cerințelor
- arhitectura stratificată – cel puțin 4 pachete
- structura de bază de date – mai mult de 3 tabele
- folosirea tehnicii de reflecție



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

Pentru implementarea aplicației cerute, vom analiza cerințele date, iar ulterior vom analiza tipul de programare „Programare Orientată pe Obiecte”, deoarece conceptele acestui tip de programare stau la baza implementării acestei aplicații. Bazându-ne pe aceste concepte, putem începe dezvoltarea aplicației.

O bază de date este o colecție organizată de informații sau de date structurate, stocate electronic într-un computer. O bază de date este controlată, de regulă, de un sistem de management al bazelor de date (DBMS). Cumulat, datele, DBMS și aplicațiile asociate reprezintă un sistem de baze de date, denumit prescurtat bază de date. Bazele de date sunt de obicei utilizate pentru a stoca informații despre obiecte și domenii din viața reală. Obiectivul principal al unei baze de date este de a ține evidențe și de a facilita accesul la informații. Managementul unei comenzi cu ajutorul bazelor de date este interesat în facilitarea realizării comenzii și a actualizării obiectelor afectate de aceasta.

Orice bază de date poate fi văzută ca un depozit, această asociere fiind determinată de următorul aspect: fiecare produs stocat într-un depozit va avea informațiile trecute într-un inventar, iar fiecare client venit la depozit pentru achiziția unui sau a mai multor produse va trebui să prezinte informații despre acesta pentru realizarea comenzii, informații ce vor fi salvate de conducerea depozitului. În momentul cererii unui produs de un client, în inventarul depozitului se vor afla informațiile despre produs și se va ști dacă achiziția este posibilă, asociind produsul clientului. După efectuarea tranzacției, în viața reală, se emite o chitanță cu suma aferentă și produsele cumparate, iar stocul produselor se va actualiza în funcție de vânzarea sau aducerea acestora. Este mai ușoară și mai puțin costisitoare metoda generării unei chitanțe pe baza tuturor produselor cumparate de client, decât generarea unei chitanțe pentru fiecare produs cumparat de același client.



## 3. Proiectare

### 3.1 Decizii de proiectare

Pe plan intern, pentru a implementa aplicația, am făcut conexiunea cu baza de date într-un obiect de tip Singleton, și am apelat-o pentru executarea fiecărui query necesar modificării bazei de date, pentru prelucrarea datelor, folosind validatori pentru fiecare data extrasă din baza de date în vederea folosirii acestora. Am folosit clase diferite pentru citirea din fișier și parsarea comenzilor pentru ușurința apelării acestora, precum și pentru generarea rapoartelor și a chitanțelor.

Pe plan extern, vom afișa în fișiere de tip pdf chitanțele rezultate de efectuarea unor comenzi cu produsele, cantitățile acestora, numele clientului și pretul total, precum și mesajele de înștiințare a eșecului tranzacțiilor însoțite de faptul că comanda nu a putut fi efectuată și rapoartele tabelare folosite în momentul dorit. Comenzile de executat pe baza de date sunt citite dintr-un fișier.

### 3.2 Diagrama UML

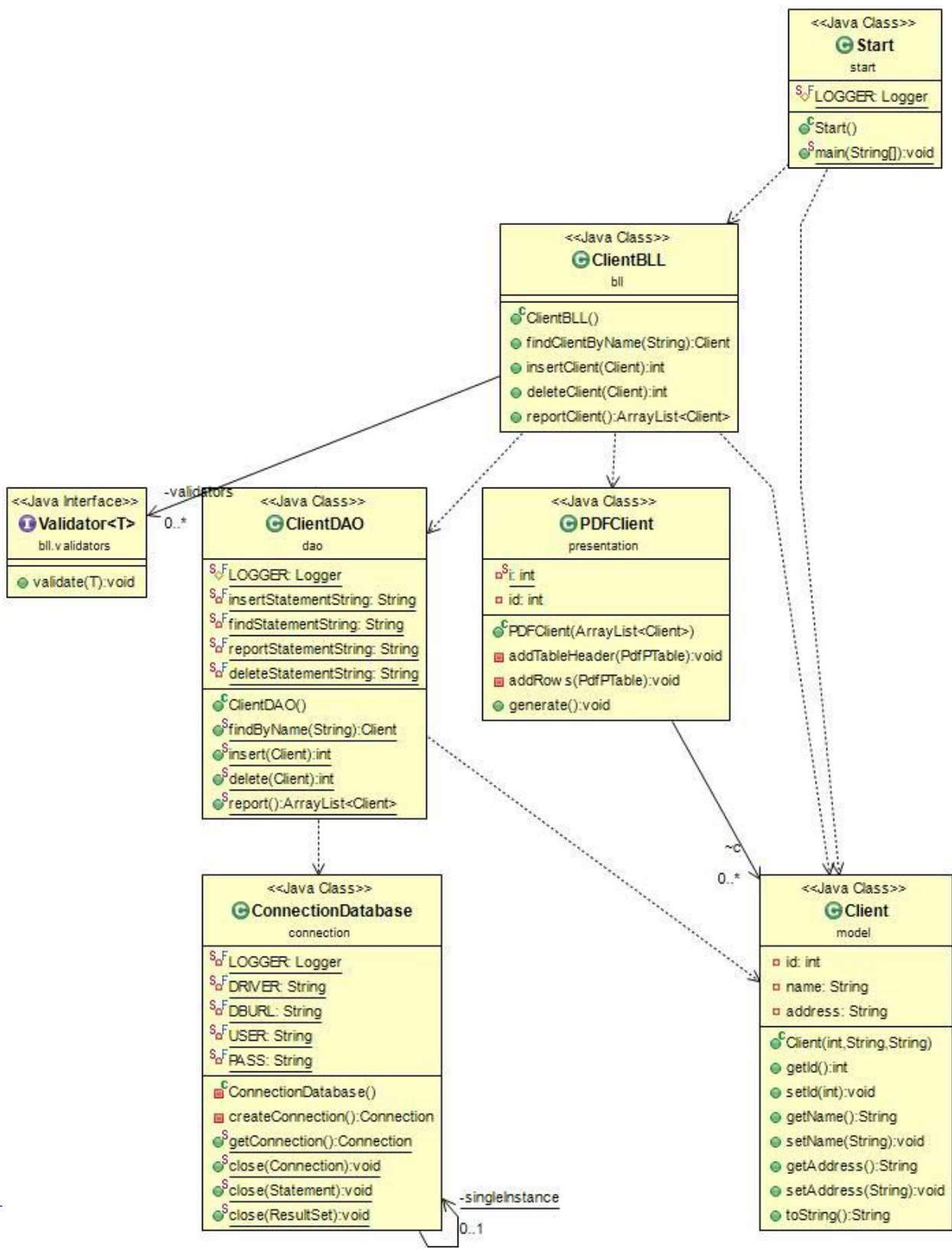
UML este notația internațională standard pentru analiza și proiectarea orientată pe obiecte. Diagramele UML de clase sunt folosite în modelarea orientată pe obiect pentru a descrie structura statică a sistemului, modul în care este el structurat. Oferă o notatie grafică pentru reprezentarea: claselor - entități ce au caracteristici comune relațiilor - relațiile dintre două sau mai multe clase.

Unified Modeling Language sau UML pe scurt este un limbaj standard pentru descrierea de modele și specificații pentru software. UML a fost dezvoltat pentru reprezentarea complexității programelor orientate pe obiect, al cărui fundament este structurarea programelor pe clase, și instanțele acestora (numite și obiecte). Cu toate acestea, datorită eficienței și clarității în reprezentarea unor elemente abstracte, UML este utilizat dincolo de domeniul IT. Așa se face că există aplicații ale UML-ului pentru management de proiecte, pentru business Process Design etc. Reprezentarea UML a claselor pentru Client.



# UNIVERSITATEA TEHNICĂ

DIN CLUJ-NAPOCA





### 3.3 Proiectare clase

În cadrul acestei aplicații avem 31 de clase dintre care 4, cele din pachetul bll, sunt clasele folosite pentru implementarea logică a operațiilor pe fiecare din cele 4 tabele din baza de date warehouse și anume Product, Client, Order și OrderItem.

Următoarele 10 clase implementează interfața Validator și moștenesc metoda validate(T), fiecare din acestea validând câte o coloană din fiecare tabelă. Clasa ConnectionDatabase realizează conexiunea cu baza de date warehouse.

Următoarele 4 clase, din pachetul dao, realizează interacțiunea directă cu cele 4 tabele din baza de date warehouse folosind Connection, PreparedStatement și ResultSet.

Cele 4 clase din pachetul model definesc notiunile abstracte pentru obiectul reprezentat de fiecare tabelă și are metodele de get și set pentru fiecare.

Clasele din pachetul presentation sunt folosite pentru citirea din fișier, parsarea datelor citite și generarea fișierelor .pdf corespunzătoare rapoartelor și chitanțelor.

La final, clasa Start din pachetul start combină toate operațiile implementate în restul claselor.

### 3.4 Metode

În continuare, voi analiza doar metodele corespunzătoare claselor de Client, deoarece sunt similare cu cele ale Client, Produc și OrderItem.

#### Clasa ClientBLL

**Metoda este folosită pentru a găsi un client după numele acestuia, apelându-se metoda din ClientDAO ce face conexiunea cu baza de date warehouse**

```
public Client findClientByName(String name) {
    Client st = ClientDAO.findByName(name);
    if (st == null) {
        throw new NoSuchElementException("The Client with name " + name +
" was not found!");
    }
    return st;
}
```



Metoda este folosită pentru a insera un client în tabela respectivă, apelându-se metoda din ClientDAO ce face conexiunea cu baza de date warehouse.

```
public int insertClient(Client c) {
    for (Validator<Client> v : validators) {
        v.validate(c);
    }
    return ClientDAO.insert(c);
}
```

Metoda este folosită pentru a șterge un client din tabela respectivă, apelându-se metoda din ClientDAO ce face conexiunea cu baza de date warehouse.

```
public int deleteClient(Client c) {
    if (OrderItemDAO.findByClient(c.getName()) != null)
        OrderItemDAO.delete(c.getName());
    if (OrderDAO.findByName(c.getName()) != null)
        OrderDAO.delete(c.getName());
    int st = ClientDAO.delete(c);
    if (st == 0) {
        throw new NoSuchElementException("The Client " + c.toString() + "
was not deleted!");
    }
    return st;
}
```

Metoda este folosită pentru a popula un vector de clienți cu clienții din tabela respectivă, apelându-se metoda din ClientDAO ce face conexiunea cu baza de date warehouse.

```
public ArrayList<Client> reportClient() throws FileNotFoundException,
DocumentException {
    ArrayList<Client> st = ClientDAO.report();
    if (st == null) {
        throw new NoSuchElementException("The Client was not found!");
    } else {
        PDFClient p = new PDFClient(st);
        p.generate();
    }
    return st;
}
```





## Clasa OrderItemBLL

Voi adauga aceasta metoda din OrderItemBLL deoarece este cea mai ampla din aceasta aplicatie. Metoda este folosita pentru a insera un order item in tabela respectiva,apelandu-se metode din OrderItemDAO, OrderDAO, ClientDAO si ProductDAO ce fac conexiunea cu baza de date warehouse. Aceasta metoda este cea mai ampla din proiect din cauza faptului ca in momentul inserarii unui order item exista mai multe cazuri de inserari si updatari la nivelul celorlalte tabele din baza de date astfel: in momentul inserarii unui order item e necesar ca produsul sa existe in tabela Product, iar catitatea acestuia sa fie mai mare sau egala cu cantitatea produsului de inserat in order item si este necesar ca si numele clientului de inserat sa existe in tabela Client. In caz de succes, se updateaza cantitatea produsului corespunzator din tabela Product. Daca numele clientului exista in tabela Orderc, se updateaza totalul corespunzator.Altfel, se va adauga clientul cu pretul aferent produsului cumparat.Mai avem si situatia cand numele produsului si numele clientului de inserat exista in tabela OrderItem. In acest caz, se va updata cantitatea aferenta.

```
public int insertOrderItem(OrderItem p) throws FileNotFoundException,
DocumentException {
    int result;
    Product prod = ProductDAO.findByName(p.getProductName());
    Client cli = ClientDAO.findByName(p.getClient());
    Orderc ord = OrderDAO.findByName(p.getClient());
    float newPrice = prod.getPrice() * p.getProductQuantity();
    for (Validator<OrderItem> v : validators) {
        v.validate(p);
    }
    if (OrderItemDAO.findByName(p.getClient(), p.getProductName()) == null
    && prod != null
        && prod.getQuantity() >= p.getProductQuantity() && cli !=
    null) {
        if (ord != null)
            OrderDAO.update(newPrice, p.getClient());
        else
            OrderDAO.insert(new Orderc(0, p.getClient(), newPrice));
        OrderDAO.update(p.getProductQuantity(), p.getProductName());
        result = OrderItemDAO.insert(p);
        ProductDAO.updateMinus(p.getProductQuantity(),
        p.getProductName());
        ArrayList<OrderItem> arr =
        OrderItemDAO.findByClient(p.getClient());
        Orderc orde = OrderDAO.findByName(p.getClient());
        Bill bill = new Bill(orde, arr);
        bill.generate();
        return result;
    } else if (prod != null && prod.getQuantity() >= p.getProductQuantity()
    && cli != null) {
        if (ord != null)
            OrderDAO.update(newPrice, p.getClient());
        else
            OrderDAO.insert(new Orderc(0, p.getClient(), newPrice));
    }
}
```



```

OrderDAO.update(p.getProductQuantity(), p.getProductName());
result = OrderItemDAO.updateAdauga(p);
ProductDAO.updateMinus(p.getProductQuantity(),
p.getProductName());
ArrayList<OrderItem> arr =
OrderItemDAO.findByClient(p.getClient());
Orderc orde = OrderDAO.findByName(p.getClient());
Bill bill = new Bill(orde, arr);
bill.generate();
return result;
} else {
BillError bi = new BillError(p);
bi.generate();
return -1;
}
}
}

```

### Clasa ClientAddressValidator

Metoda primește ca argument un Client și îi validează adresa în funcție de niste limite impuse.

```

@Override
public void validate(Client t) {
    if (t.getAddress().length() < MIN_SIZE || t.getName().length() >
MAX_SIZE) {
        throw new IllegalArgumentException("The Client Name limit is not
respected!");
    }
}
}

```

### Clasa ConnectionDatabase

Metoda este folosită pentru a se crea conexiunea cu baza de date, conexiune care se va plasa într-un obiect de tip Singleton

```

private Connection createConnection() {
    Connection connection = null;
    try {
        connection = DriverManager.getConnection(DBURL, USER, PASS);
    } catch (SQLException e) {
        LOGGER.log(Level.WARNING, "An error occurred while trying to
connect to the database");
        e.printStackTrace();
    }
    return connection;
}

```



Metoda este folosita pentru a obtine o conexiune active.

```
public static void close(Connection connection) {  
    if (connection != null) {  
        try {  
            connection.close();  
        } catch (SQLException e) {  
            LOGGER.log(Level.WARNING, "An error occured while trying  
to close the connection");  
        }  
    }  
}
```

Metoda este folosita pentru a inchide un Statement.

```
public static void close(Statement statement) {  
    if (statement != null) {  
        try {  
            statement.close();  
        } catch (SQLException e) {  
            LOGGER.log(Level.WARNING, "An error occured while trying  
to close the statement");  
        }  
    }  
}
```

Metoda este folosita pentru a inchide un ResultSet

```
public static void close(ResultSet resultSet) {  
    if (resultSet != null) {  
        try {  
            resultSet.close();  
        } catch (SQLException e) {  
            LOGGER.log(Level.WARNING, "An error occured while trying  
to close the ResultSet");  
        }  
    }  
}
```



## Clasa ClientDAO

Metoda este folosită pentru a găsi un client după numele acestuia, creând o conexiune cu baza de date, adăugând parametrii necesari query-ului definit și inițializat în loc de '?', apoi executând query-ul. Rezultatul execuției query-ului este stocat într-un `ResultSet`, fiecare element al acestuia corespunzându-i unui rând al tabelului. Valorile din coloane sunt extrase, știind denumirile coloanelor.

```
public static Client findByName(String ClientName) {
    Client toReturn = null;

    Connection dbConnection = ConnectionDatabase.getConnection();
    PreparedStatement findStatement = null;
    ResultSet rs = null;
    try {
        findStatement =
dbConnection.prepareStatement(findStatementString);
        findStatement.setString(1, ClientName);
        rs = findStatement.executeQuery();
        rs.next();

        int id = rs.getInt("id");
        String address = rs.getString("address");
        toReturn = new Client(id, ClientName, address);
    } catch (SQLException e) {
        LOGGER.log(Level.WARNING, "ClientDAO:findByName " +
e.getMessage());
    } finally {
        ConnectionDatabase.close(rs);
        ConnectionDatabase.close(findStatement);
        ConnectionDatabase.close(dbConnection);
    }
    return toReturn;
}
```

Metoda este folosită pentru a insera un client în tabela respectivă, creând o conexiune cu baza de date, adăugând parametrii necesari query-ului definit și inițializat în loc de '?', apoi executând query-ul.

```
public static int insert(Client Client) {
    Connection dbConnection = ConnectionDatabase.getConnection();

    PreparedStatement insertStatement = null;
    int insertedId = -1;
    try {
        insertStatement =
dbConnection.prepareStatement(insertStatementString,
Statement.RETURN_GENERATED_KEYS);
```



```

insertStatement.setString(1, Client.getName());
insertStatement.setString(2, Client.getAddress());
insertStatement.executeUpdate();

ResultSet rs = insertStatement.getGeneratedKeys();
if (rs.next()) {
    insertedId = rs.getInt(1);
}
} catch (SQLException e) {
    LOGGER.log(Level.WARNING, "ClientDAO:insert " + e.getMessage());
} finally {
    ConnectionDatabase.close(insertStatement);
    ConnectionDatabase.close(dbConnection);
}
return insertedId;
}

```

Metoda este folosită pentru a șterge un client din tabela respectivă, creând o conexiune cu baza de date, adăugând parametrii necesari query-ului definit și inițializat în loc de '?', apoi executând query-ul.

```

public static int delete(Client Client) {
    int ok = 0;
    Connection dbConnection = ConnectionDatabase.getConnection();
    PreparedStatement deleteStatement = null;
    try {
        deleteStatement =
dbConnection.prepareStatement(deleteStatementString);
        deleteStatement.setString(1, Client.getName());
        deleteStatement.setString(2, Client.getAddress());
        deleteStatement.executeUpdate();
    } catch (SQLException e) {
        LOGGER.log(Level.WARNING, "ClientDAO:delete " + e.getMessage());
    } finally {
        ConnectionDatabase.close(deleteStatement);
        ConnectionDatabase.close(dbConnection);
        ok = 1;
    }
    return ok;
}

```



Metoda este folosită pentru a popula un vector de clienți cu clienții din tabela respectivă, creând o conexiune cu baza de date, pe baza query-ului definit și inițializat, apoi executat. Rezultatul execuției query-ului este stocat într-un `ResultSet`, fiecare element al acestuia corespunzându-i unui rând al tabelului. Valorile din coloane sunt extrase, știind denumirile coloanelor.

```
public static ArrayList<Client> report() {
    ArrayList<Client> toReturn = new ArrayList<Client>();

    Connection dbConnection = ConnectionDatabase.getConnection();
    PreparedStatement findStatement = null;
    ResultSet rs = null;
    try {
        findStatement =
dbConnection.prepareStatement(reportStatementString);

        rs = findStatement.executeQuery();
        while (rs.hashCode() != 0 && rs.next()) {
            int id = rs.getInt("id");
            String name = rs.getString("name");
            String address = rs.getString("address");
            toReturn.add(new Client(id, name, address));
        }
    } catch (SQLException e) {
        LOGGER.log(Level.WARNING, "ClientDAO:report " + e.getMessage());
    } finally {
        ConnectionDatabase.close(rs);
        ConnectionDatabase.close(findStatement);
        ConnectionDatabase.close(dbConnection);
    }
    return toReturn;
}
```

## Clasa Client

Singurele metode prezente în această clasă sunt de `get` și `set`

```
public int getId() {
    return id;
}

public void setId(int id) {
    this.id = id;
}
```



## Clasa PDFClient

Metoda este folosita pentru a formata header-ul tabeli Client.

```
private void addTableHeader(PdfPTable table) {
    Stream.of("ID", "Name", "Adress").forEach(columnTitle -> {
        PdfPCell header = new PdfPCell();
        header.setBackgroundColor(BaseColor.LIGHT_GRAY);
        header.setBorderWidth(3);
        header.setPhrase(new Phrase(columnTitle));
        table.addCell(header);
    });
}
```

Metoda este folosita pentru a popula tabela cu detaliile corespunzatoare fiecarui rand

```
private void addRows(PdfPTable table) {
    for (Client client : c) {
        table.addCell(Integer.toString(client.getId()));
        table.addCell(client.getName());
        table.addCell(client.getAddress());
    }
}
```

Metoda genereaza tabela

```
public void generate() throws FileNotFoundException, DocumentException {
    Document document = new Document();
    String denumire = "Client" + Integer.toString(id) + ".pdf";
    PdfWriter.getInstance(document, new FileOutputStream(denumire));
    document.open();
    PdfPTable table = new PdfPTable(3);
    addTableHeader(table);
    addRows(table);
    document.add(table);
    document.close();
}
```

## Clasa Start

In metoda Main combinam toate clasele, medodele si operatiile definite in cadrul acestui proiect



## 3.5 Pachete

În cadrul acestei aplicații avem 6 pachete și un subpachet.

Primul, pachetul `bll` (Business Logic Classes) conține clasele folosite pentru implementarea logică a operațiilor pe fiecare din cele 4 tabele din baza de date warehouse și anume `Product`, `Client`, `Order` și `OrderItem`. Acest pachet conține un subpachet, `bll.validators`, ce conține clasele ce validează fiecare câte o coloană din fiecare tabelă.

Pachetul `connecton` (Database Connection) conține clasa `ConnectionDatabase` care realizează conexiunea cu baza de date warehouse.

Pachetul `dao` (Database Access Classes) conține clase ce realizează interacțiunea directă cu cele 4 tabele din baza de date warehouse pe baza de `query-uri`, folosind `Connection`, `PreparedStatement` și `ResultSet`.

În pachetul `model` (Model Classes) se definesc noțiunile abstracte pentru obiectul reprezentat de fiecare tabelă.

Pachetul `presentation` (UI Classes) face operațiile pe obiectele cele mai apropiate de interfața utilizator.

Pachetul `start` (Main Class) conține legarea tuturor claselor în obținerea rezultatului final.

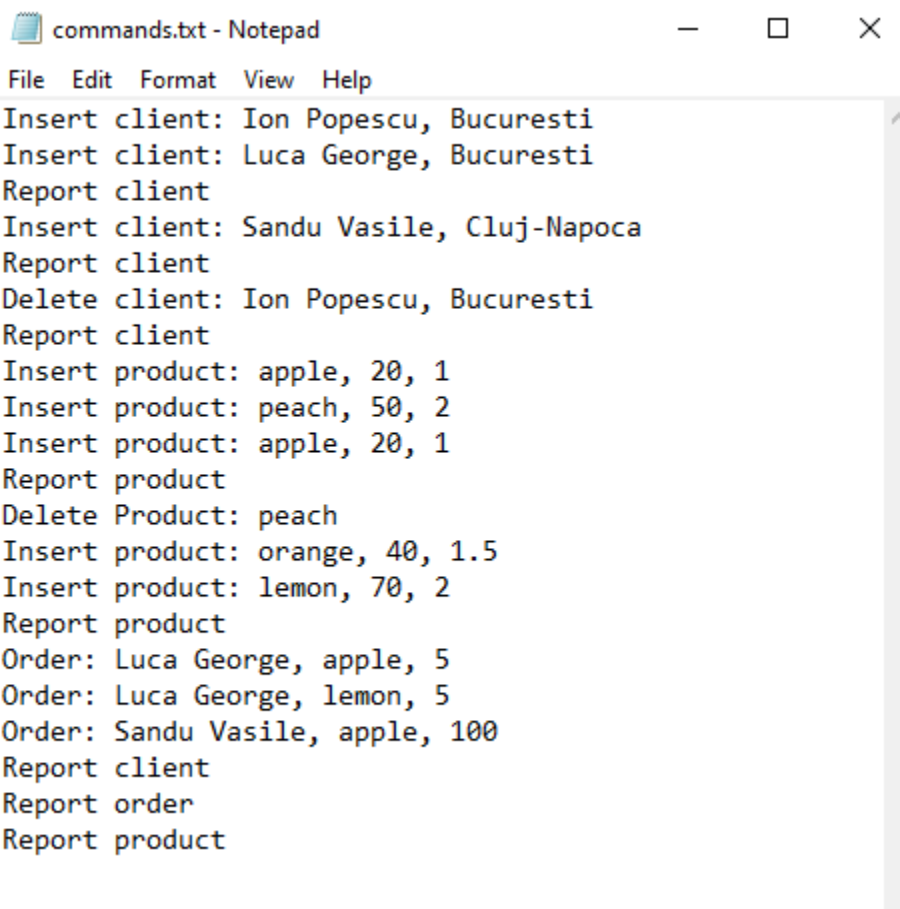




## 4.Implementare

Initial, luam un fisier text continand comenzile de executat asupra bazei de date, pe care l-am denumit in cazul meu commands.txt .

La rularea aplicatiei, in functie de comenzile reprezentate de datele de intrare, aplicatia va putea executa in timp real operatiile pe baza de date in vederea obtinerii rezultatului final.



```
File Edit Format View Help
Insert client: Ion Popescu, Bucuresti
Insert client: Luca George, Bucuresti
Report client
Insert client: Sandu Vasile, Cluj-Napoca
Report client
Delete client: Ion Popescu, Bucuresti
Report client
Insert product: apple, 20, 1
Insert product: peach, 50, 2
Insert product: apple, 20, 1
Report product
Delete Product: peach
Insert product: orange, 40, 1.5
Insert product: lemon, 70, 2
Report product
Order: Luca George, apple, 5
Order: Luca George, lemon, 5
Order: Sandu Vasile, apple, 100
Report client
Report order
Report product
```



## 5.Rezultate

Am obtinut o implementare a cerintei proiectului , punand in practica o aplicatie a managementului unei comenzi, generand in timp real rapoartele corespunzatoare fiecarei tabela in functie de comanda data in fisier.

Daca datele sunt introduse corespunzator in fisierul de intrare, rezultatele generate sunt corecte si utile pentru utilizator.

The screenshot displays a web application interface with two main panels. The left panel shows a bill for client Luca George, and the right panel shows a bill for client Sandu Vasile. Below the bills, there is a table of products and their prices.

**Bill for client Luca George:**

Product Name	Product Quantity
apple	5
lemon	5

The client Luca George bought products worth 15.0 €.

Acquisition Date: Mon 2020.04.13 at 08:52:03 PM EEST

**Bill for client Sandu Vasile:**

The transaction desired by Sandu Vasile could not be processed.

Transaction Date: Mon 2020.04.13 at 08:52:03 PM EEST

**Product List:**

ID	Name	Quantity	Price
117	apple	40	1.0
118	peach	50	2.0

**Order Summary:**

ID Order Item	Product Name	Product Quantity	Client Name
32	apple	5	Luca George
33	lemon	5	Luca George

ID Order	Client Name	Total
21	Luca George	15.0



Javadoc este un generator de documentație creat de Sun Microsystems pentru limbajul Java pentru generarea documentației API în format HTML din codul sursă Java. Formatul HTML este utilizat pentru a adăuga comoditatea de a putea hiperliga documente asociate. Am folosit Javadoc pentru documentare proiectului meu, adaugand comentarii fiecărei metode a fiecărei clase, exceptand metodele validate din cele 10 clase de validare, din motive exidente.

SUMMARY: NESTED   FIELD   CONSTR   METHOD    DETAIL: FIELD   CONSTR   METHOD    SEARCH: <input type="text"/>		
Modifier and Type	Method	Description
static int	<code>delete</code> ( <code>java.lang.String c1Name</code> )	Metoda este folosita pentru a sterge un order din tabela respectiva, creand o conexiune cu baza de date, adaugand parametrii necesari query-ului definit si initializat in loc de '?', apoi executand query-ul.
static Orderc	<code>findByName</code> ( <code>java.lang.String clientName</code> )	Metoda este folosita pentru a gasi un order dupa numele clientului ce realizeaza comanda, creand o conexiune cu baza de date, adaugand parametrii necesari query-ului definit si initializat in loc de '?', apoi executand query-ul.
static int	<code>insert</code> ( <code>Orderc o</code> )	Metoda este folosita pentru a insera un order in tabela respectiva, creand o conexiune cu baza de date, adaugand parametrii necesari query-ului definit si initializat in loc de '?', apoi executand query-ul.
static <code>java.util.ArrayList&lt;Orderc&gt;</code>	<code>report()</code>	Metoda este folosita pentru a popula un vector cu order-urile din tabela respectiva, creand o conexiune cu baza de date, pe baza query-ului definit si initializat, apoi executat.
static int	<code>update</code> ( <code>float total</code> , <code>java.lang.String name</code> )	Metoda este folosita pentru a modifica pretul total a unui order din tabela respectiva pe baza numelui clientului, creand o conexiune cu baza de date, adaugand parametrii necesari query-ului definit si initializat in loc de '?', apoi executand query-ul.
<b>Methods inherited from class <code>java.lang.Object</code></b>		
<code>equals</code> , <code>getClass</code> , <code>hashCode</code> , <code>notify</code> , <code>notifyAll</code> , <code>toString</code> , <code>wait</code> , <code>wait</code> , <code>wait</code>		
<b>Constructor Details</b>		
<b>OrderDAO</b>		
<code>public OrderDAO()</code>		

## 6.Concluzii

Sunt de părere că în urma realizării acestei teme am reușit să-mi reamintesc și să aprofundez materia de semestrul trecut, să-mi îmbunătățesc tehnicile de programare în acest limbaj, am deprins o mai buna aptitudine in a rezolva algoritmi pe obiecte si in a manipula baze de date cu ajutorul limbajului Java, precum si generarea si rularea unui jar. In acelasi timp, am invatat cum sa fac o documentatie Javadoc si cat este de importanta si folositoare, precum si cum sa generez pdf-uri, si sa le manipulez,cum sa generez un Dump file si cum sa autoincrementez o variabila.

Ca o posibila dezvoltare ulterioara, aplicatiei ii mai pot fi aduse unele imbunatatiri, ca de exemplu:

- generarea unui fisier .fdf mai detaliat in privinta esuarii tranzactiei
- utilizarea tehnicilor de reflexie pentru usurarea muncii



## 7. Bibliografie

[http://coned.utcluj.ro/~salomie/PT\\_Lic/4\\_Lab/Assignment\\_3/Assignment\\_3\\_Indications.pdf](http://coned.utcluj.ro/~salomie/PT_Lic/4_Lab/Assignment_3/Assignment_3_Indications.pdf)  
<https://dzone.com/articles/layers-standard-enterprise>  
<https://dev.mysql.com/doc/workbench/en/wb-admin-export-import-management.html>  
<https://www.baeldung.com/javadoc>  
<https://www.oracle.com/technetwork/java/javase/documentation/index-137868.html>  
<https://www.baeldung.com/java-pdf-creation>  
<http://pages.cs.wisc.edu/~hasti/cs302/examples/Parsing/parseString.html>  
<https://stackoverflow.com/questions/24305830/java-auto-increment-id>