

**TEHNICI DE PROGRAMARE FUNDAMENTALE
ASSIGNMENT 3**

**ORDER MANAGEMENT
DOCUMENTATIE**

Bakk Cosmin-Robert
Grupa 30227

Cuprins

1. Obiectivul temei
2. Analiza problemei, modelare, scenarii, cazuri de utilizare
3. Proiectare
 - 3.1. Diagrama UML
 - 3.2. Proiectare clase
4. Implementare si testare
 - 4.1. Metode
 - 4.2. Testare
5. Concluzii
6. Bibliografie

1. Obiectivul temei

Obiectivul acestei teme de laborator este de a implementa un sistem de management al unei baze de date. Baza de date pe care am modelat-o este cea a unei firme care se ocupa cu preluarea de comenzi de la clienti pentru produse. Pentru a comunica cu baza de date, un utilizator trebuie sa creeze si sa apeleze printr-o linie de comanda aplicata unei arhive .jar un fisier text de intrare care contine, pe fiecare linie, una dintre comenzile de insert client, insert product, delete client, delete product, order, report client, report product si report order. In urma introducerii acestor date de intrare, se va modifica baza de date sau se vor crea fisiere .pdf de iesire.

Obiectivele secundare sunt: respectarea paradigmei programarii orientate pe obiect, lucrul cu o baza de date, crearea de fisiere .pdf in java, crearea unei documentatii Javadoc si crearea unei arhive .jar pentru portabilitate si testarea mai usoara a programului.

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

Baza de date va contine tabelele client, product, orderitem si orders.

Tabela client contine coloanele name (numele clientului - cheie primara) si address (orasul clientului).

Tabela product contine coloanele name (numele produsului - cheie primara), qty (cantitatea de produse care se afla in stoc) si price (pretul unui astfel de produs).

Tabela orderitem reprezinta o comanda a unui client de un anumit numar de produse de un singur tip. Tabela contine coloanele orderItemID (id-ul unei comenzi – cheie primara), clientName (numele clientului care realizeaza comanda), productName (numele produsului comandat) si qty (cantitatea de produse de acel fel comandate).

Tabela orders reprezinta toate comenzile pentru un client. Tabela contine coloanele orderID (id-ul unei comenzi – cheie primara), client (numele clientului care face comanda) si pret (pretul total al produselor comandate de client).

Fisierul .txt de intrare trebuie sa contina doar comenzi care respecta unul din formatele:

Insert client: <nume client>, <adresa client>

-se insereaza un nou client in baza de date

-daca numele clientului se mai regaseste in baza de date, acesta nu va fi inserat (numele e cheie primara)

Insert product: <nume produs>, <cantitate de produse>, <pretul unui produs>

-se insereaza un nou produs in baza de date

-daca numele produsului se mai regaseste in baza de date, cantitatea de produse aflate in stoc va fi actualizata

Order: <nume client>, <nume produs>, <cantitate de produse>

-se insereaza un nou orderitem in baza de date, in cazul in care sunt destule produse in stoc
-daca se reuseste inserarea, numaru de produse care sunt in stoc este actualizat si se creeaza un raport .pdf pentru acea comanda, altfel fisierul creat va contine informatii despre faptul ca nu s-a putut realiza comanda

-daca se reuseste inserarea, se insereaza un nou order in tabela orders, cu numele clientului si pretul pe care il are acesta de achitat; daca numele clientului exista deja intr-un camp din tabela orders, atunci pretul pe care il are acesta de achitat va fi actualizat

Delete client: <nume client>, <adresa client>

-se sterge un anumit client din baza de date, daca numele specificat exista in baza de date

Delete product: <nume produs>

-se sterge un anumit produs din baza de date, daca numele specificat exista in baza de date

Report client

-se creeaza un fisier .pdf, care contine un tabel cu toti clientii care se afla in acel moment in baza de date si datele despre acestia

Report product

-se creeaza un fisier .pdf, care contine un tabel cu toate produsele care se afla in acel moment in baza de date si datele despre acestea

\

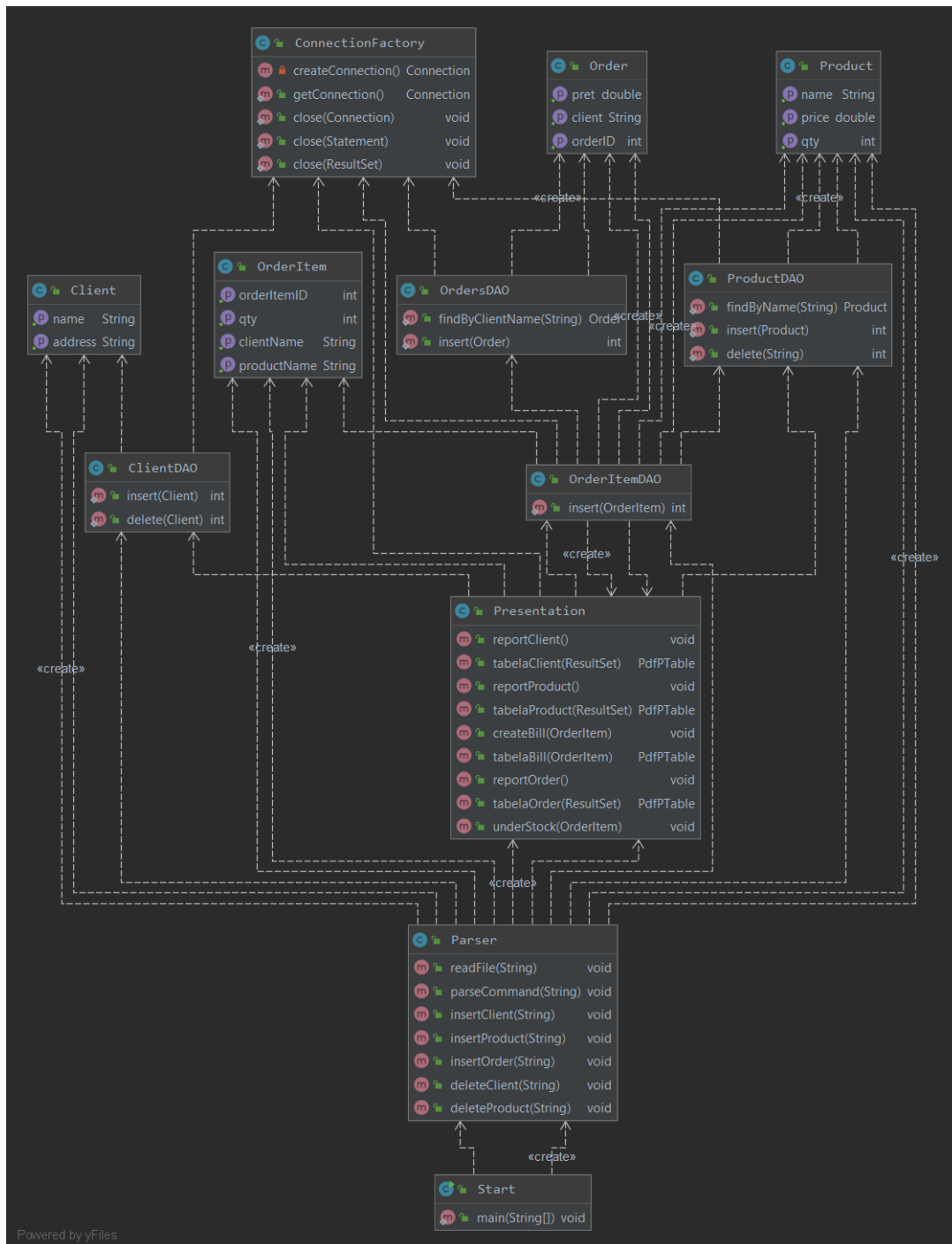
Report order

-se creeaza un fisier .pdf, care contine un tabel cu toate comenzile care se afla in acel moment in baza de date si datele despre acestea

Pentru a structura proiectul, clasele au fost aranjate in urmatoarele pachete: start, connection, presentation, model si dao.

3. Proiectare

3.1. Diagrama UML



3.2. Proiectare clase

1. Pachetul model

Clasa Client: aceasta clasa defineste conceptul de client, acesta fiind caracterizat printr-un nume, specific fiecarui client si o adresa, address, aceasta reprezentand numele orasului unde locuieste clientul.

Clasa Product: aceasta clasa defineste conceptul de produs, acesta fiind caracterizat printr-un nume, specific fiecarui produs, o cantitate de tipul intreg, qty si un pret de tipul double, price.

Clasa OrderItem: aceasta clasa defineste conceptul de comanda a unui anumit produs, acesta fiind caracterizat printr-un ID, specific fiecarei comenzi, un nume de client, un nume de produs si o cantitate de produse comandate. De asemenea, clasa are o variabila de clasa care este incrementata la crearea unei comenzi noi.

Clasa Order: aceasta clasa defineste conceptul de comanda a unui client, acesta fiind caracterizat printr-un ID, specific fiecarei comenzi, un nume de client, si un pret pe care il are de achitat clientul. De asemenea, clasa are o variabila de clasa care este incrementata la crearea unei comenzi noi.

2. Pachetul connection

Clasa ConnectionFactory: aceasta clasa se ocupa cu pornirea/oprirea conexiunii la baza de date si cu inchiderea statement-urilor si a rezultatelor.

3. Pachetul start

Clasa Start: aceasta clasa este clasa main a proiectului. Se ocupa cu crearea unui nou obiect de tip Parser si trimiterea numelui fisierului de intrare ca parametru in metoda de citire a clasei Parser.

4. Pachetul presentation

Clasa Parser: aceasta clasa se ocupa cu citirea fiecarei linii din fisierul text de intrare. Fiecare linie este tradusa apoi in comanda corespunzatoare, iar in functie de comanda se va realiza una din operatiile aflate intr-o clasa a pachetului dao.

Clasa Presentation: aceasta clasa se ocupa cu crearea rapoartelor sau a facturilor .pdf, folosind ca query pentru a comunica cu baza de date "SELECT * FROM <nume tabela>".

5. Pachetul dao

Clasa ClientDAO: aceasta clasa comunica cu tabela client din baza de date. Are metode pentru a insera sau pentru a sterge campuri din tabela.

Clasa ProductDAO: aceasta clasa comunica cu tabela product din baza de date. Are metode pentru a insera/actualiza sau pentru a sterge campuri din tabela.

Clasa OrderItemDAO: aceasta clasa comunica cu tabela orderitem din baza de date. Are metoda care insereaza/actualizeaza campuri din tabela.

Clasa OrdersDAO: aceasta clasa comunica cu tabela orders din baza de date. Are metode care insereaza/actualizeaza campuri din tabela sau care cauta un camp dupa numele unui client.

4. Implementare si testare

4.1. Metode

Pachetul model

Clasa Client

public Client(String name, String address)

-constructorul clasei, primeste doua variabile de instanta, care sunt primite din clasa Parser – numele si adresa clientului

public String getName()

-returneaza numele clientului

public String getAddress()

-returneaza adresa clientului

Clasa Product

public Product(String name, int qty, double price)

-constructorul clasei, primeste trei variabile de instanta, care sunt primite din clasa Parser – numele produsului, cantitatea de produse din acel tip, pretul unui astfel de produs

public String getName()

-returneaza numele clientului

public String getQty()

-returneaza cantitatea de produse

public String getPrice()

-returneaza pretul produsului

Clasa OrderItem

public OrderItem(String clientName, String productName, int qty)

-constructorul clasei, primeste trei variabile de instanta, care sunt primite din clasa Parser – numele clientului care realizeaza comanda, numele produsului pentru care se face comanda si cantitatea de produse

-la crearea unui nou obiect de tipul OrderItem, variabila de clasa noOrders este incrementata si este asignata variabilei de instanta orderItemID

public String getOrderItemID()

-returneaza ID-ul comenzii

public String getClientName()

-returneaza numele clientului

public String getProductName()

-returneaza numele produsului

public String getQty()

-returneaza cantitatea de produse

Clasa Order

public Order (String client, double pret)

-constructorul clasei, primeste doua variabile de instanta, care sunt primite din clasa OrderItemDAO – numele clientului care realizeaza comanda si pretul pe care acesta il are de achitat

-la crearea unui nou obiect de tipul OrderItem, variabila de clasa noOrders este incrementata si este asignata variabilei de instanta orderID

```
public String getOrderID()
    -returneaza ID-ul comenzii

public String getClient ()
    -returneaza numele clientului

public String getPret()
    -returneaza suma pe care clientul o are de achitat
```

Pachetul connection

Clasa ConnectionFactory

```
private ConnectionFactory()
    -constructorul clasei, incarca clasa Driver din fisierul .jar al bazei de date MySQL

private Connection createConnection()
    -metoda care creeaza o conexiune la baza de date si o returneaza

public static Connection getConnection()
    -metoda care apeleaza metoda createConnection() si ii da return conexiunii

public static void close(Connection connection)
    -metoda care inchide conexiunea la baza de date

public static void close(Statement statement)
    -metoda care inchide un statement

public static void close(ResultSet resultSet)
    -metoda care inchide un rezultat
```

3. Pachetul start

Clasa Start

```
public static void main(String[] args)
    -este functia main a clasei. Se initializeaza un obiect de tipul Parser, iar apoi se apeleaza metoda readFile,
    avand ca parametru args[0], acesta fiind numele fisierului .txt de input
```

4. Pachetul presentation

Clasa Parser

```
public void readFile(String in)
    -se parcurge fisierul de input folosind o instanta a clasei Scanner. Scanner-ul parcurge fiecare linie si o trimite
    apoi metodei parseCommand

public void parseCommand(String data)
    -se initializeaza un obiect de tipul Presentation
    -se verifica daca string-ul primit ca parametru este "Report client", "Report product" sau "Report order". In
    aceste cazuri, se apeleaza metodele reportClient, reportProduct, respectiv reportOrder ale clasei Presentation
    -in caz contrar, string-ul este impartit prin metoda split dupa ":", primul sir fiind verificat cu string-urile
    "Insert client", "Insert product", "Order", "Delete client", "Delete product", iar al doilea sir este trimis ca parametru
    uneia din metodele insertClient, insertProduct, insertOrder, deleteClient, respectiv deleteProduct

public void insertClient(String client)
    -se creeaza un nou obiect de tipul Client, care este trimis apoi ca parametru in metoda insert a clasei
    ClientDAO
```

public void insertProduct(String product)

-se creeaza un nou obiect de tipul Product, care este trimis apoi ca parametru in metoda insert a clasei ProductDAO

public void insertOrder(String order)

-se creeaza un nou obiect de tipul OrderItem, care este trimis apoi ca parametru in metoda insert a clasei OrderItemDAO

public void deleteClient(String client)

-se creeaza un nou obiect de tipul Client, care este trimis apoi ca parametru in metoda delete a clasei ClientDAO

public void deleteProduct(String product)

-se trimite string-ul primit ca parametru metodei delete a clasei ProductDAO

Clasa Presentation

public void reportClient()

-metoda face conexiunea la baza de date

-se creeaza un statement folosind preparedStatement obiectului de tip Connection, cu parametrul "SELECT * FROM client"

-se executa query-ul

-se creeaza un document .pdf, avand numele "ReportClient<numar>.pdf", numarul fiind incrementat la fiecare report nou

-se creeaza un tabel nou, ca fiind tabelul returnat de metoda tabelaClient, care primeste ca parametru rezultatul query-ului executat

-se adauga tabelul la document

-se inchid rezultatul, statement-ul si conexiunea

public PdfPTable tabelaClient(ResultSet rs)

-metoda creeaza un tabel nou, cu 2 coloane

-se adauga cele 2 celule din header, campul "Name" si campul "Address"

-pentru fiecare camp din rezultatul query-ului, se adauga cate 2 celule, reprezentand numele clientului si adresa acestuia

public void reportProduct()

-metoda face conexiunea la baza de date

-se creeaza un statement folosind preparedStatement obiectului de tip Connection, cu parametrul "SELECT * FROM product"

-se executa query-ul

-se creeaza un document .pdf, avand numele "ReportProduct<numar>.pdf", numarul fiind incrementat la fiecare report nou

-se creeaza un tabel nou, ca fiind tabelul returnat de metoda tabelaProduct, care primeste ca parametru rezultatul query-ului executat

-se adauga tabelul la document

-se inchid rezultatul, statement-ul si conexiunea

public PdfPTable tabelaProduct(ResultSet rs)

-metoda creeaza un tabel nou, cu 3 coloane

-se adauga cele 3 celule din header, campul "Name", campul "Qty" si campul "Price"

-pentru fiecare camp din rezultatul query-ului, se adauga cate 3 celule, reprezentand numele produsului, cantitatea de produse care se afla in stoc si pretul unui astfel de produs

public void createBill(OrderItem orderItem)

-se creeaza un document .pdf, avand numele "Bill<numar>.pdf", numarul fiind incrementat la fiecare bill nou

- se creeaza un tabel nou, ca fiind tabelul returnat de metoda tabelBill, care primeste ca parametru obiectul de tip OrderItem care a fost primit ca parametru in metoda curenta
- se adauga tabelul la document

```
public PdfPTable tabelaBill(OrderItem orderItem)
    -metoda creeaza un tabel nou, cu 3 coloane
    -se adauga cele 3 celule din header, campul "Client name", campul "Product name" si campul "Qty"
    -se adauga 3 celule noi, cele returnate de metodele getClientName, getProductName si getQty ale obiectului orderItem
```

```
public void reportOrder()
    -metoda face conexiunea la baza de date
    -se creeaza un statement folosind preparedStatement obiectului de tip Connection, cu parametrul "SELECT * FROM orders"
    -se executa query-ul
    -se creeaza un document .pdf, avand numele "Order<numar>.pdf", numarul fiind incrementat la fiecare report nou
    -se creeaza un tabel nou, ca fiind tabelul returnat de metoda tabelaOrder, care primeste ca parametru rezultatul query-ului executat
    -se adauga tabelul la document
    -se inchid rezultatul, statement-ul si conexiunea
```

```
public PdfPTable tabelaOrder(ResultSet rs)
    -metoda creeaza un tabel nou, cu 2 coloane
    -se adauga cele 2 celule din header, campul "Client name" si campul "Price"
    -pentru fiecare camp din rezultatul query-ului, se adauga cate 2 celule, reprezentand numele clientului si pretul pe care il are acesta de achitat
```

```
public void underStock(OrderItem orderItem)
    -se creeaza un document .pdf, avand numele "Bill<numar>.pdf", numarul fiind incrementat la fiecare bill nou
    -se creeaza un string nou, in care se scriu informatii despre faptul ca nu a fost realizata comanda
    -se adauga paragraful care contine string-ul creat la document
```

4. Pachetul dao

Clasa ClientDAO

```
public static int insert(Client client)
    -metoda face conexiunea la baza de date
    -se creeaza un statement folosind preparedStatement obiectului de tip Connection, cu parametrul "INSERT INTO client (name,address)" + " VALUES (?,?)"
    -se seteaza cele doua "?" cu valorile returnate de metodele getName si getAddress aplicate obiectului client
    -se executa query-ul
    -se inchid statement-ul si conexiunea
    -se returneaza pozitia campului care a fost inserat sau -1 in caz de esec
```

```
public static int delete(Client client)
    -metoda face conexiunea la baza de date
    -se creeaza un statement folosind preparedStatement obiectului de tip Connection, cu parametrul "DELETE FROM client WHERE name = ?"
    -se seteaza "?" cu valoarea returnata de metoda getName aplicata obiectului client
    -se executa query-ul
    -se inchid statement-ul si conexiunea
    -se returneaza pozitia campului care a fost sters sau -1 in caz de esec
```

Clasa ProductDAO

public static Product findByName(String name)

- metoda face conexiunea la baza de date

- se creeaza un statement folosind preparedStatement obiectului de tip Connection, cu parametrul "SELECT * FROM product where name = ?"

- se seteaza "?" cu String-ul name primit ca parametru

- se executa query-ul

- se creeaza un obiect nou de tipul Product, avand ca parametrii valorile din campurile citite din rezultat

- se inchid rezultatul, statement-ul si conexiunea

- se returneaza obiectul de tip Product care a fost gasit sau null, in caz ca acesta nu a fost gasit

public static int insert(Product product)

- metoda face conexiunea la baza de date

- se apeleaza metoda findByName cu parametrul fiind valoarea returnata de metoda getName aplicata obiectului product

- daca findByName este null:

- se creeaza un statement folosind preparedStatement obiectului de tip Connection, cu parametrul "INSERT INTO product (name,qty,price)" + " VALUES (?,?,?)"

- se seteaza cele trei "?" cu valorile returnate de metodele getName, getQty si getPrice aplicate obiectului product

- daca findByName nu este null:

- se creeaza un intreg nou, qty, acesta fiind suma dintre valorile returnate de metoda getQty aplicata obiectului product si obiectului returnat de metoda findByName

- se creeaza un statement folosind preparedStatement obiectului de tip Connection, cu parametrul "UPDATE product SET qty = ? WHERE name = ?"

- se seteaza cele doua "?" cu cantitatea calculata, respectiv cu valoarea returnata de metoda getName aplicata obiectului product

- se executa query-ul

- se inchid statement-ul si conexiunea

- se returneaza pozitia campului care a fost inserat sau actualizat sau -1 in caz de esec

public static int delete(String product)

- metoda face conexiunea la baza de date

- se creeaza un statement folosind preparedStatement obiectului de tip Connection, cu parametrul "DELETE FROM product WHERE name = ?"

- se seteaza "?" cu string-ul primit ca parametru

- se executa query-ul

- se inchid statement-ul si conexiunea

- se returneaza pozitia campului care a fost sters sau -1 in caz de esec

Clasa OrderItemDAO

public static int insert(OrderItem orderItem)

- metoda face conexiunea la baza de date

- se initializeaza un nou obiect de tipul Presentation

- se creeaza un statement folosind preparedStatement obiectului de tip Connection, cu parametrul "INSERT INTO orderitem (orderItemID,clientName,productName,qty)" + " VALUES (?,?,?,?)"

- se seteaza cele patru "?" cu valorile returnate de metodele getOrderItemID, getClientName, getProductName si getQty aplicate obiectului orderItem

- daca cantitatea de produse a OrderItem-ului din parametru este mai mica decat cantitatea de produse de tipul celui cerut(gasit cu metoda findByName a clasei ProductDAO), atunci:

- se executa query-ul

- se creeaza un nou produs, avand variabila qty egala cu - cantitatea de produse a obiectului orderItem

- se apeleaza metoda insert a clasei ProductDAO, avand ca parametru produsul creat; astfel, cantitatea de produse care se afla in stoc va fi scazut

- se creeaza o factura folosind metoda createBill a obiectului de tip Presentation

- se calculeaza pretul care trebuie platit de catre client

- se creeaza un nou obiect de tipul Order, care va fi parametru pentru metoda insert a clasei OrdersDAO
- altfel: se creeaza o factura folosind metoda underStock a obiectului de tip Presentation
- se inchid statement-ul si conexiunea
- se returneaza pozitia campului care a fost inserat sau -1 in caz de esec

Clasa OrdersDAO

public static Order findByClientName(String clientName)

- metoda face conexiunea la baza de date
- se creeaza un statement folosind preparedStatement obiectului de tip Connection, cu parametrul "SELECT * FROM orders where client = ?"
- se seteaza "?" cu String-ul clientName primit ca parametru
- se executa query-ul
- se creeaza un obiect nou de tipul Order, avand ca parametrii valorile din campurile citite din rezultat
- se inchid rezultatul, statement-ul si conexiunea
- se returneaza obiectul de tip Order care a fost gasit sau null, in caz ca acesta nu a fost gasit

public static int insert(Order order)

- metoda face conexiunea la baza de date
- se apeleaza metoda findByClientName cu parametrul fiind valoarea returnata de metoda getClient aplicata obiectului order
- daca findByClientName este null:
 - se creeaza un statement folosind preparedStatement obiectului de tip Connection, cu parametrul "INSERT INTO orders (orderID,client,pret)" + " VALUES (?,?,?)"
 - se seteaza cele trei "?" cu valorile returnate de metodele getOrderID, getClient si getPret aplicate obiectului order
- daca findByClientName nu este null:
 - se creeaza un double nou, pret, acesta fiind suma dintre valorile returnate de metoda getPret aplicata obiectului order si obiectului returnat de metoda findByClientName
 - se creeaza un statement folosind preparedStatement obiectului de tip Connection, cu parametrul "UPDATE orders SET pret = ? WHERE client = ?"
 - se seteaza cele doua "?" cu pretul calculata, respectiv cu valoarea returnata de metoda getClientName aplicata obiectului order
- se executa query-ul
- se inchid statement-ul si conexiunea
- se returneaza pozitia campului care a fost inserat sau actualizat sau -1 in caz de esec

4.2. Testare












Pentru fisierul commands.txt, datele de intrare sunt:

```
Delete client: Ion Popescu, Bucuresti
Delete client: Luca George, Bucuresti
Delete client: Sandu Vasile, Cluj-Napoca
Delete product: apple
Delete product: peach
Delete product: orange
Delete product: lemon
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
```

In urma executiei comenzii

```
java -jar PT2020_30227_Bakk_Cosmin-Robert_Assignment_3.jar commands.txt
```

se vor crea urmatoarele rapoarte .pdf:

-  Bill1
-  Bill2
-  Bill3
-  Order1
-  ReportClient4
-  ReportProduct3
-  ReportClient1
-  ReportClient2
-  ReportClient3
-  ReportProduct1
-  ReportProduct2

Pentru o factura creata cu succes, de exemplu Bill1.pdf, continutul documentului arata astfel:

Client name	Product name	Qty
Luca George	apple	5

Pentru o factura care nu a putut fi creata datorita numarului insuficient de produse aflate in stoc, de exemplu Bill3.pdf, continutul documentului arata astfel:

Client Sandu Vasile placed an order of 100 products of type apple.

There are only 35 products of type apple in stock.

Continutul documentului Order1 arata astfel:

Client name	Price
Luca George	15.0

Continutul documentului ReportClient4 arata astfel:

Name	Address
Luca George	Bucuresti
Sandu Vasile	Cluj-Napoca

Continutul documentului ReportProduct3 arata astfel:

Name	Qty	Price
apple	35	1.0
lemon	65	2.0
orange	40	1.5

5. Concluzii

Acest proiect poate fi util pentru comunicarea cu o astfel de baza de date. Proiectul m-a ajutat sa invat mai multe despre lucrul cu o baza de date si realizarea conexiunii dintre baza de date si un proiect java si sa invat sa folosesc o structura a proiectului bazata pe impartirea claselor in mai multe pachete. De asemenea, am invatat sa comentez adecvat codul pentru a putea face o documentatie de tip Javadoc.

6. Bibliografie

1. http://coned.utcluj.ro/~salomie/PT_Lic/4_Lab/
2. <http://thinktobits.blogspot.com/2012/12/iText-JDBC-SQL-Table-PDF-Report-Java-Example-Program.html>
3. <https://www.jetbrains.com/help/idea/packaging-a-module-into-a-jar-file.html>