A picture containing drawing

Description automatically generated

# Tema 3

ORDER MANAGEMENT

Indrumator de laborator: Moldovan Dorin

Student: Manici Valentin-Dan

Grupa: 30226

# Contents

Tema 3 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .. . . . . . . 1

ORDER MANAGEMENT. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .1

1.Obiectivul temei. . . .. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 3

Obiective secundare. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .3

2.Analiza problemei, modele, scenarii, cazuri de utilizare . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 3

[3. Proiectare. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .](#_Toc36769940)3

[4.Implementare. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .](#_Toc36769945)7

[5.Concluzii. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .](#_Toc36769946)

[6.Bibliografie. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .](#_Toc36769947)

# 1.Obiectivul temei

Proiectarea și implementarea unui proiect de tipul order management.

## Obiective secundare:

1. Parsarea fisierului pentru a obtine comenzile
2. Implementarea corecta pentru queries si functiile care le executa

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

Problema cere sa se foloseasca parsarea pentru obtinerea comenzilor din fisierul dat ca input si pe baza acestor comenzi trebuie sa se execute diferite comenzi asupra bazei de date. De asemenea proble cere sa se genereze ca output fisiere de tip pdf pentru fiecare comanda de tip report, si pentru pentru comenzilor care se pot realiza cu success, in cazul in care produsul necesar pentru o comanda nu se afla pe stoc se cere sa se genereze un pdf in care sa apara acesta eroare de stoc insufficient.

Programul se poate rula de mai multe ori dar nu va reseta implicit baza de date, daca nu se sterg datele direct din baza pentru fiecare fisier executat se vor executa comenzile din fisier si se vor adauga datele suplimentar.

Datele din fisier apar in urmatorul format:

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

# 3. Proiectare

## Decizii de implementare

Am decis sa stochez datele din tabele din baza de date intr-un Array List sis a le returnez in functia care genereaza tabelul corespunzator tipului de date in fisierul pdf. Obtinerea comenzilor am realizat-o folosind in principal functia split.

## Pachete

Programul este organizat in cinci pachete:

* “model” : Acest pachet contine clasele Customer, Earnings, Product, si Order care sunt necesare pentru realizarea proiectului, atat pentru queries cat si pentru raporturile in format pdf;
* “dao”: Acest pachetul contine clasele ConnectionFactory, CustomerDAO, ProductDAO, Order DAO care contin codul pentru conectarea la baza de date, cat si queries pentru comenzile de insert, delete, find.
* “Main”(default package): Contine clasa Main.
* “presentation”: Acest pachet contine clasele GeneratorPdf care este responsabila de generarea rapoartelor pdf si clasa ReadFile in care se citesc informatiile din fisier si se parseaza comenzile.
* “bll”: Acest pachet contine clasele CustomerBLL, OrderBLL, ProductBLL, Reflection, clase responsabile de logica proiectului.

## Proiectare clase

1. Clasa Customer contine campurile: id, name, location;

Constructorii: public Customer(int id, String name, String location) { }

public Customer(String name, String location) { }

public Customer() { }

Metodele:

public int getId() { }

public void setId(int id) { }

public String getName() { }

public void setName(String name) { }

public String getLocation() { }

public void setLocation(String location) { }

1. Clasa Product contine campurile: id, name,quantity, price

Constructorii: public Product(int id, String name,int quantity, float price){}

public Product(String name, int quantity, float price) { }

public Product() { }

Metodele:

public Integer getId() { }

public void setId(int id) { }

public String getName() { }

public void setName(String name) { }

public Integer getQuantity() { }

public void setQuantity(int quantity) { }

public Float getPrice() { }

public void setPrice(int price) { }

1. Clasa Order contine campurile: id, customerName, productName, quantity

Constructor: public Order(int id, String customerName, String productName, int quantity) { }

public Order(String customerName, String productName, int quantity) { }

Metode:

public int getId() { }

public void setId(int id) { }

public String getCustomerName() { }

public void setCustomerName(String customerName) { }

public String getProductName() { }

public void setProductName(String productName) { }

public int getQuantity() { }

public void setQuantity(int quantity) { }

1. Clasa Earnigs contine campurile: productName, price

Constructori: public Earnings(int id, String productName, float price) { }

public Earnings(String productName, float price) { }

Metode:

public String getProductName() { }

public void setProductName(String productName) { }

public float getPrice() { }

public void setPrice(float price) { }

1. Clasa GeneratorPdf : metode :

public static void generateClientReport(ArrayList<Customer> clients, int number) { }

public static void generateProductReport(ArrayList<Product> products, int number) { }

public static void generateOrderReport(ArrayList<Order> orders, int number) { }

private static void addRows(PdfPTable table, String content) { }

public static void generateStockError(String product) { }

public static void createBill(Order order, float cost) { }

1. Clasa ReadFile: contine campurile fisier, lista, reportClient, reportProduct, reportOrder

Constructori: public ReadFile(String fisier) { }

public ReadFile() { }

Metode:

public String getFisier() { }

public static void readFile() { }

public void parseCommand(ArrayList<String> lista) { }

1. Clasa Main: aici se intializeaza ReadFile
2. Clasa CustomerBLL

Constructor: public CustomerBLL() { }

Metode:

public static void deleteCustomer(Customer customer) { }

public static void insertCustomer(Customer customer) { }

public static ArrayList<Customer> select(){ }

1. Clasa OrderBLL

Constructor: public OrderBLL() { }

Metode:

public static void insert(Order order) { }

public static ArrayList<Order> select(){ }

public static void insertInEarnings(Earnings earnings) { }

1. Clasa Reflection: metode public static void retrieveProperties(Object object) { }
2. Clasa ProductBLL:

Constructor: public ProductBLL() { }

Metode:

public static void insert(Product product) { }

public static void deleteByName(Product product){ }

public void findByName(Product product) { }

public static ArrayList<Product> select(){ }

1. Clasa ConnectionFactory

Metode:

private ConnectionFactory() { }

private Connection createConnection() { }

public static Connection getConnection() { }

public static void close(Connection connection) {}

public static void close(Statement statement) { }

public static void close(ResultSet resultSet) { }

1. Clasa CustomerDAO:

Metode:

public static void deleteByName(Customer customer){ }

public static void insert(Customer customer) { }

public static ArrayList<Customer> select(){ }

1. Clasa ProductDAO:

Metode:

public static void deleteByName(Product product){ }

public static Product findByName(Product product) { }

public static void insert(Product product) { }

public static ArrayList<Product> select(){ }

1. Clasa OrderDAO:

Metode:

public static void insert(Order order) { }

public static ArrayList<Order> select() { }

public static void insertInEarnings(Earnings earnings) { }

Diagrama UML:

A screenshot of a cell phone

Description automatically generated

1. Implementare

Fiecare client este descries pentru baza de date ca avand un id, nume si o locatie (adresa). Constructorii sunt folositi la instantierea de client atat cu id dat cat si cu id automat. Metodelet de get si set preiau respective seteaza aceste attribute ale clientului.

Fiecare produs este descries ca avand id, nume, cantitate sip ret. Constructorii sunt folositi pentru instantierea produselor. Metodele de get si set sunt folosite pentru preluarea respective setarea atributelor produsului respective.

Fiecare earning este descries de numele produsului si pretul total obtinut prin vanzarea produsului cu numele nameProduct. De asemenea este instantiate de constructori.

Fiecare order este descries de numele consumatorului, numele produsului, si cantitatea comandata, de asemenea cantitatea comandata este scazuta din stocul produsului respecti.

Functia care realizeaza aceasta cerinta se afla in clasa OrderDAO si anume functia public void insert(Order order) { } . Aceasta insereaza orderul in baza de date, verifica disponibilitatea produsului, decrementeaza stocul produsului si insereaza in tabelul earnings castigul total obtinut din vanzarea produsului.

Functia public static ArrayList<Order> select() { } returneaza o lista cu toate datele din tabelul orders. De asemenea exista o astfel de functie pentru fiecare tabela.

**public** **static** ArrayList<Order> select() {

ArrayList<Order> lista = **new** ArrayList<Order>();

Connection dbConnection = ConnectionFactory.*getConnection*();

PreparedStatement selectStatement = **null**;

ResultSet rs = **null**;

Order o = **null**;

**try** {

selectStatement = dbConnection.prepareStatement(***selectStatementString***);

rs = selectStatement.executeQuery();

**while** (rs.next()) {

**int** id = rs.getInt("id");

String customerName = rs.getString("customerName");

String productName = rs.getString("productName");

**int** quantity = rs.getInt("quantity");

o = **new** Order(id, customerName, productName, quantity);

lista.add(o);

}

} **catch** (SQLException e) {

e.printStackTrace();

} **finally** {

ConnectionFactory.*close*(selectStatement);

ConnectionFactory.*close*(dbConnection);

}

**return** lista;

}

In pachetul bll se afla clasele corespunzatoare claselor DAO care contin logica programului. In pachetul presentation se afla partea de input si de output al programului.

Prin input, clasa ReadFile se citeste fisierul care contine comenzile, se parseaza comenzile si se executa aceste comenzi.

Exemplu de parsare:

**if** (l.equalsIgnoreCase("Report product"))

GeneratorPdf.*generateProductReport*(ProductBLL.*select*(), ++reportOrder);

**else** {

String[] parsedInfo = l.split(": ");

String[] parsedData = parsedInfo[1].split(", ");

**if** (parsedInfo[0].equalsIgnoreCase("Insert client")) {

Customer c = **new** Customer(parsedData[0], parsedData[1]);

CustomerBLL.*insertCustomer*(c);

} **else** **if** (parsedInfo[0].equalsIgnoreCase("Delete client")) {

Customer c = **new** Customer(parsedData[0], parsedData[1]);

CustomerBLL.*deleteCustomer*(c);

In clasa GeneratorPdf se genereaza pdf-urile care trebuie sa contina tabelele cu elementele din baza de date, products, orders, customers, sau mesaje cum este cel din cazul in care nu sunt suficiente produse in stoc.

Exemplu generare de raport pentru tabel:

**public** **static** **void** generateProductReport(ArrayList<Product> products, **int** number) {

String fileName = "products\_report\_" + number + ".pdf";

Document document = **new** Document();

**try** {

PdfWriter.*getInstance*(document, **new** FileOutputStream(fileName));

document.open();

PdfPTable productsTable = **new** PdfPTable(4);

**for** (Product p : products) {

*addRows*(productsTable, p.getId() + "\_" + p.getName() + "\_" + p.getQuantity() + "\_" + p.getPrice());

}

document.add(productsTable);

document.close();

} **catch** (DocumentException | FileNotFoundException e) {

e.printStackTrace();

}

}

Exemplu de generare raport pentru mesaj scris:

**public** **static** **void** generateStockError(String product) {

String fileName = product + "\_stock\_err.pdf";

Document document = **new** Document();

**try** {

PdfWriter.*getInstance*(document, **new** FileOutputStream(fileName));

document.open();

Font font = FontFactory.*getFont*(FontFactory.***COURIER***, 16, BaseColor.***BLACK***);

Chunk chunk = **new** Chunk("The product " + product + " is out of stock.", font);

document.add(chunk);

document.close();

} **catch** (DocumentException | FileNotFoundException e) {

e.printStackTrace();

}

}

# 5.Concluzii

Am invatat cum sa conectez o baza de date in java si cum sa lucrez cu aceasta, de asemenea sa generez rapoarte pdf.

# 6.Bibliografie

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

<https://dev.mysql.com/doc/workbench/en/wb-admin-export-import-management.html>

<https://www.sqlitetutorial.net/sqlite-java/insert/>

<https://www.w3schools.com/sql/sql_update.asp>

<https://www.geeksforgeeks.org/split-string-java-examples/>