UNIVERSITATEA DE AUTOMATICA SI CALCULATOARE

DOCUMENTATIE TEMA 3
ORDER MANAGEMENT

Nume si prenume: Timis Iulia Georgeana

Grupa: 30226

Profesor laborator: Dorin Vasile Moldovan

CUPRINS

1.Probleme si solutia problemei

2. Obiecivul temei

3. Studiul problemei, modelare, scenarii, cazuri de utilizare

4. Proiectare(decizii de proiectare, diagrame UML, structuri de date, proiectare clase, interfata utilizator)

5. Implementare

6. Rezultat

7. Concluzii

8. Bibliografie

1. Probleme si solutia problemei:

Gestionarea comenzilor este un lucru destul de greu de facut pentru un angajat, mai ales daca este facuta pe foaie, pot aparea foarte multe greseli in distribuirea comenzilor, astfel creandu-se neplaceri in randul clientilor. O solutie pentru aceasta problema ar putea fi crearea unei aplicatii java de organizare a clientilor, comenzilor si a produselor cu ajutorul unei legaturi cu o baza de date, astfel se poate face mult mai usor o planificare a comenzilor.

2. Obiectivul temei

Obiectivul acestei teme de laborator a fost sa proiectam si sa implementam o aplicatie in Java pentru gestionarea si planificarea comenzilor facute de clienti catre un depozit cu produse. La baza aplicatiei sunt 3 tabele facute cu ajutorul MySQL : Client, Product si Order. Cu ajutorul acestei aplicatii, trebuie sa putem adauga, sterge sau edita un client, un produs, precum si comanda plasata de catre client, depinzand de o anumita cantitate disponibila la momentul respectiv.

Cu ajutorul interfetei grafice vom introduce datele necesare pentru adaugarea unui client, adaugarea unui produs si adaugarea unei comenzi, iar dupa plasarea comenzii va urma generarea unui „bon”, de format PDF, cu datele comenzii.

Aceasta tema a fost realizata cu ajutorul impartirii pe clase, cu nume sugestive, respectand modelui arhitectural Layers, cerut. Pe langa lucrul cu clasele, am avut de realizat conexiunea cu baza de date, prin pacetul dat ConnectionFactory. Implementarea solutiei a fost realizata prin diverse clase si metode folosind paradigma OOP. Pentru generarea bonului a fost necesara folosirea fisierelor, pentru generarea datelor.

Tehnica reflection presupune gestionarea bazei de date prin efectuarea diverselor operatii cu baze de date.

3. Studiul problemei, modelare, scenarii, cazuri de utilizare

Ce reprezinta de fapt o baza de date?

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 (DBSM). Cumulat, datele, DBMS și aplicațiile asociate reprezintă un sistem de baze de date, denumit prescurtat bază de date.
Datele din cele mai obișnuite tipuri de baze de date sunt distribuite de regulă pe linii și coloane, în diferite tabele, pentru eficientizarea procesării și interogării datelor. Datele pot fi accesate, gestionate, modificate, actualizate, controlate și organizate cu ușurință. Majoritatea bazelor de date utilizează un limbaj structurat de interogare (SQL) pentru scrierea și interogarea datelor.

In cazul meu, numele ales de mine pentru baza de date e tp3_database. Aceasta contine 3 tabele: Client, Order si Product.

In cazul tabelului Client, sunt datele despre acesta. Un client este caracterizat de un id unic, de un nume, de o adresa unde va trbui livrata comanda si de o adresa de e-mail.

Tabelul Products prezinta detalii despre produsul comandat. Acesta are de asemenea un id unic, un nume de produs, o cantitate si un pret.

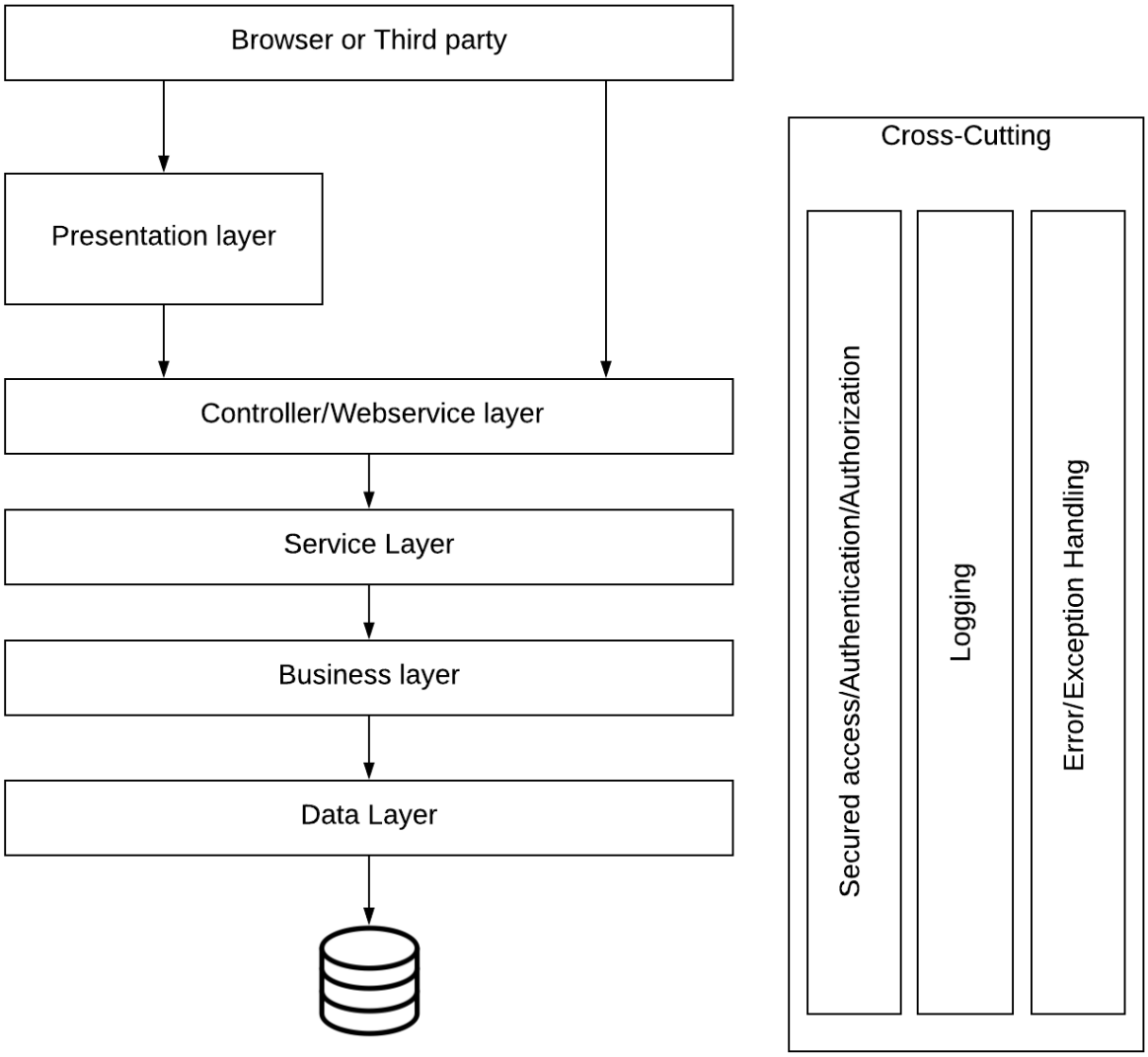
Tabelul orders este tabelul cacarteristic plasarii comenzilor. Acesta are ca cele doua tabele de mai sus, un id unic, un id pentru comanda, un id pentru client si o cantitate pentru a putea vedea disponibilitatea produsului.

4. Proiectare(decizii de proiectare, diagrame UML, structuri de date, proiectare clase, interfata utilizator)

Pentrul designul claselor, proiectarea claselor a fost facuta prin respectarea modelului pe nivele explicat in prezentarea de suport a temei, datorita acestui lucru, fiecare nivel respecta un anumit rol.

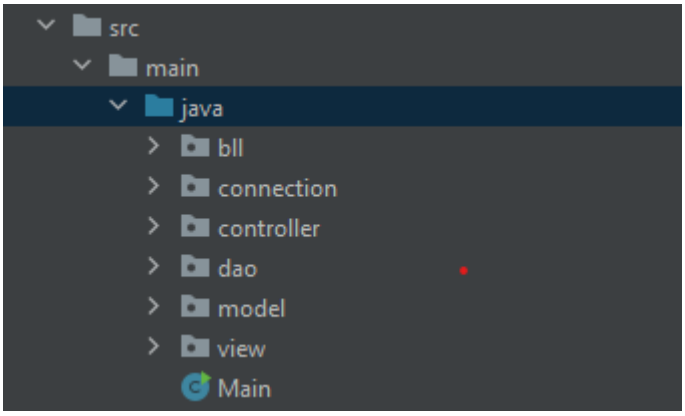
Aplicatia a fost structurata respectand modelul arhitectural Layers care are drept rol impartirea responsabilitatilor claselor pe mai multe nivele, iar acest lucru face ca intelegerea si dezvoltarea aplicatiei sa fie prioritizara.

Exemplu pentru arhitectura Layered



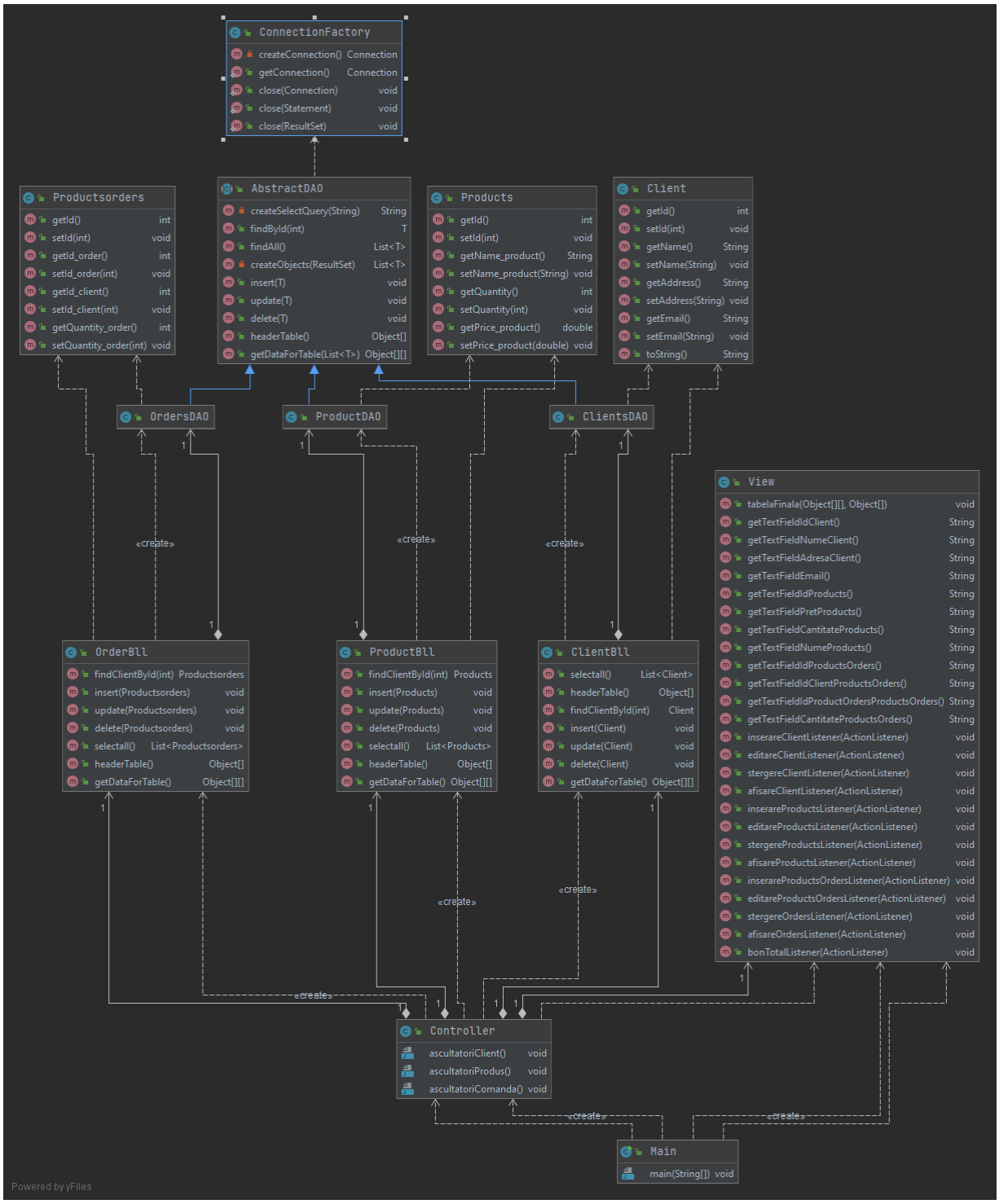
Pentru proiectul meu, am folosit 6 pachete cu nume sugestive: bll, connection, controller, dao, model si view, pe langa acestea mai am clasa main in care se realizeaza „pornirea” aplicatiei.

Pachetele aplicatiei mele:



Primul pachet este bll, acesta contine 3 clase: ClientBll, ProductBll si OrderBll. Rolul acestuia este de a implementa logica de buissines pentru clasele aflate in pachetul model.
Al doilea pachet este pachetul connection care contine doar o clasa, anume ConnectionFactory. Acesta realizeaza conexiunea aplicatiei cu baza de date. Aceasta clasa contine diferite metode pentru ajutarea realizarii unei conexiuni fara erori sau probleme.
Al 3 lea pachet este pachetul Controller. In acesta se afla doar clasa Controller care are rolul de a transmite datele catre model primite prin intermediul interfetei grafice.
Al 4 lea pachet este pachetul dao. Acesta contine 4 clase, si anume: AbstractDAO, ClientsDAO, OrdersDAO, ProductDAO. Clasele ClientsDAO, OrdersDAO, ProductDAO sunt extinse de la clasa AbstractDAO. Rolul principal al acestui proiect este folosirea interogarilor bazei de date si permiterea utilizarii functiilor de stergere sau adaugare a datelor din baza de date.
Al 5 lea pachet este pachetul model care contine 3 clase: Client, Products si Productsorders. Aceste 3 clase sunt clasele de baza ale aplicatei unde sunt instantate datele despre clienti, comada si produs. De asemenea aceste clase corespund tabelelor din MySql, avand acelasi nume si acelasi nume si numar de variablie.
Al 6 lea pachet este reprezentat de pachetul view. Acesta contine o singura clasa, anume View, cu ajutorul careia se creeaza intefata grafica, cat mai usoara si placuta intelegerii utilizatorilor.

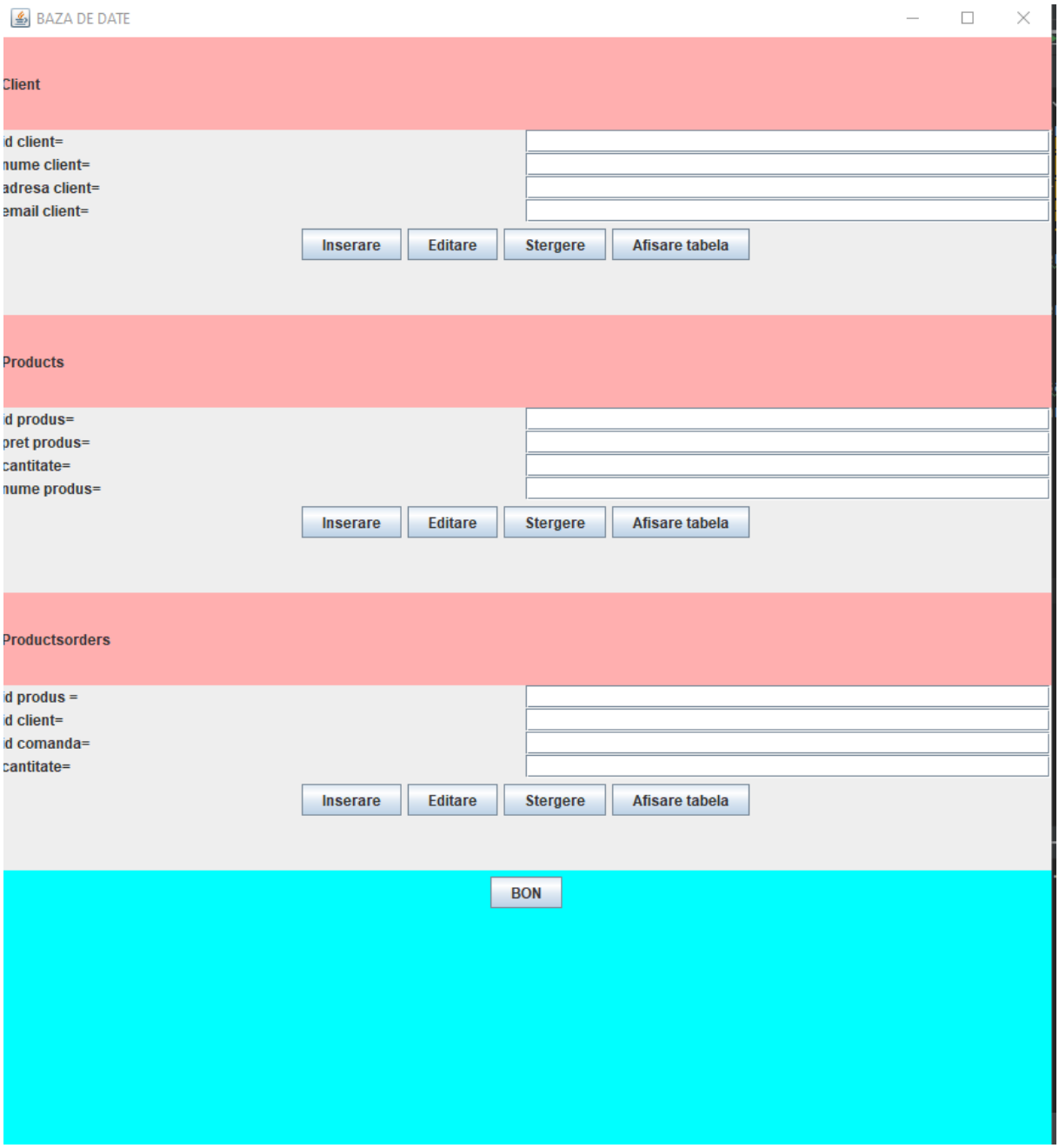
Diagrama UML



Unified Modeling Language (adica prescurtat UML) este un limbaj standard pentru descrierea de modele și specificații pentru software. UML are la baza dezvoltarea pentru reprezentarea complexității programelor orientate pe obiect, al căror fundament este structurarea programelor pe clase, și instanțele acestora (numite și obiecte).In cazul cestui proiect am folosit diagrama te tip clase, diagramele de clasă sunt folosite pentru reprezentarea concretă a unor instanțe de clasă, așadar obiecte, și a legăturilor concrete dintre acestea.

Am generat o diagrama pentru fiecare pachet in functie legaturile pe care le are.

Interfata grafica este de tipul user friendly, si este foarte usor de utilizat, chiar daca este pentru prima data. Interfata contine 2 frame-uri: unul pentu intoducerea datelor, si unul pentu afisarea datelor din tabel.



Avem mai multe butoane cu nume sugestive, iar la apasare lor vor face functiile dorite.

Butonul BON genereaza bonul cu toate datele necesare.

5. Implementare

Implementarea claselor a fost luat dupa exemplu dat pe GitLab.

Voi incepe prin prezentarea claselor din pachetul model:

Clasa Client

```
private int id;
private String name;
private String address;
private String email;


public Client(int id, String name, String address, String email) {
    this.id = id;
    this.name = name;
    this.address = address;
    this.email = email;
}
```

Clasa Client are ca atribute id-ul clientului, numele acestuia, adresa unde locuieste si adresa de email, date prin variabile cu nume sugestiv: id, name, address si email. Coloanele din tabelul cu acelasi nume din baza de date, corespund de asemenea cu numele variabileor din aplicatie. Id-ul clientului este unic Pentru realizare acestor functii avem nevoie de un constructor pentru initializarea functiilor.Avem diverse functii cu nume sugestiv folosite pentru returnarea diverselor valori dee care avem nevoie Tot in aceasta clasa se afla suprascrierea metodei pentru o afisare usoara si placuta utilizatorului.

Clasa Products:

```
public class Products {
    private int id;
    private String name_product;
    private int quantity;
    private double price_product;
    public Products(int id_product, String name_product, int quantity, double price_product )
    {
        this.id=id_product;
        this.name_product=name_product;
        this.quantity=quantity;
       this.price_product=price_product;
    }
    public Products(){};
```

Clasa Products are ca atribute id-ul produsului, numele acestuia, cantitatea acestuia, si pretul , date prin variabile cu nume sugestiv: id, name_product, quantity si price_product. Coloanele din tabelul cu acelasi nume din baza de date, corespund de asemenea cu numele variabileor din aplicatie. Id-ul produsului este unic Pentru realizare acestor functii avem nevoie de un constructor pentru initializarea functiilor.Avem diverse functii cu nume sugestiv folosite pentru returnarea diverselor valori dee care avem nevoie.

Clasa ProductsOrders

```java
public class Productsorders {

    private int id;
    private int id_order;
    private int id_client;
    private int quantity_order;
    public Productsorders(int id_order, int id_client, int product_order, int quantity_order)
    {
        this.id=product_order;
        this.id_order=id_order;
        this.id_client=id_client;
        this.quantity_order=quantity_order;
    }
    public Productsorders(){};
```

Clasa Productsorders are ca atribute id-ul, id-ul comenzii, id_ul clientului si cantitatea comenzii, date prin variabile cu nume sugestiv: id, id_order,id_client si quantity_order . Coloanele din tabelul cu acelasi nume din baza de date, corespund de asemenea cu numele variabileor din aplicatie. Id-ul produsului este unic Pentru realizare acestor functii avem nevoie de un constructor pentru initializarea functiilor.Avem diverse functii cu nume sugestiv folosite pentru returnarea diverselor valori dee care avem nevoie.

Urmeaza prezentarea claselor din pachetul Bll:

Clasa ClientBll:

```java
package bll;
import java.util.List;
import java.util.NoSuchElementException;
import dao.*;
import model.*;
public class ClientBll
{
    private ClientsDAO client;
    public ClientBll() { this.client = new ClientsDAO(); }
    public List<Client>selectall()
    {
        return client.findAll();
    }
    public Object[] headerTable()
    {
       return client.headerTable();
    }
```

Aceasta clasa implementeaza layer-ul business logic pentru tabelul Client. Clientii care se afla la momentul respectiv in baza de date vor fi returnati prin apelarea diverselor metode pe un obiect de tip client. Metoda findclientbyid returneaza clientii gasiti in baza de date, daca nu va fi niciunul gasit, se va arunca o exceptie unde se va da id-ul dat de utilizator care nu se afla in baza de date. Functia insert

insereaza clientii in tabel iar functia update, actualizeaza lista de clienti. Functia delete, sterge clientii, iar functia getdatafor table da datele gaside despre client.

Clasa ProductBll:

```
package bll;
import java.util.List;
import java.util.NoSuchElementException;
import dao.ProductDAO;
import model.Products;
public class ProductBll
{
    private ProductDAO product1;

    public ProductBll() { this.product1=new ProductDAO(); }

    public Products findClientById(int id)
    {
        Products p = product1.findById(id);
        if(p == null){
            throw new NoSuchElementException("The student with id =" + id + " was not found!");
        }
        return p;
    }
```

Aceasta clasa implementeaza layer-ul business logic pentru tabelul Product. Produsele care se afla la momentul respectiv in baza de date vor fi returnate prin apelarea diverselor metode pe un obiect de tip produs . Metoda findclientbyid returneaza produsele gasite in baza de date, daca nu va fi niciunul gasit, se va arunca o exceptie unde se va da id-ul dat de utilizator care nu se afla in baza de date. Functia insert insereaza produsele in tabel iar functia update, actualizeaza lista de produse. Functia delete, sterge produsele, iar functia getdatafor table da datele gaside despre produs.

Clasa OrderBll:

```java
package bll;

import java.util.List;
import java.util.NoSuchElementException;
import dao.OrdersDAO;
import model.Productsorders;


public class OrderBll
{
    private OrdersDAO order;

    public OrderBll() { this.order = new OrdersDAO(); }
    public Productsorders findClientById(int id){
        Productsorders order1 = order.findById(id);
        if(order1 == null){
            throw new NoSuchElementException("The client with id =" + id + " was not found!");
        }
        return order1;
    }
```

Aceasta clasa implementeaza layer-ul business logic pentru tabelul Productorders. Comenzile care se afla la momentul respectiv in baza de date vor fi returnate prin apelarea diverselor metode pe un obiect de tip order . Metoda findclientbyid returneaza comenzile gasite in baza de date, daca nu va fi niciunul gasit, se va arunca o exceptie unde se va da id-ul dat de utilizator care nu se afla in baza de date. Functia insert insereaza comrnzile in tabel iar functia update, actualizeaza lista de comenzi. Functia delete, sterge comenzile din tabel, iar functia getdatafor table da datele gasite despre comanda.

Urmeaza prezentarea claselor din pachetul controller:

Clasa Contoller

```java
public class Controller {
    private View v;
    private ClientBll client_bll;
    private ProductBll product_bll;
    private OrderBll order_bll;

    public Controller(View view) {
        this.v = view;
        this.client_bll = new ClientBll();
        this.product_bll = new ProductBll();
        this.order_bll = new OrderBll();


        this.ascultatoriClient();
        this.ascultatoriProdus();
        this.ascultatoriComanda();
        v.bonTotalListener(new bonTotalActionListener());

    }
```

Pentru ascultatorii butoanelor am transformat acolo unde a fost nevoie datele in intregi sau in numere cu virgula flotanta de exemplu la id-uri sau la pret sau la cantitate, daca nu faceam acest lucru nu puteam insera datele in tabela. Pot spune ca controllerul este foarte strans legat de view, el avand ca prim atribut un obiect de tipul view.

Aceasta clasa este alcatuita din multe subclase, una pentru fiecare ascultator al butonului de care apartine. Dintre atributele acestei clase avem un model un view si un contoller, specifice afisarii model view controller.

Tot in aceasta clasa se genereaza bonul, am facut legatura intre tabele si dupa da datele cerute intr-un fisier PDF. Pentru generarea bonului, am un buton la apasarea caruia apare fisierul PDF.

Urmeaza prezentarea claselor din pachetul dao:

Clasa AbstractDAO

```java
public abstract class AbstractDAO<T> {
    protected static final Logger LOGGER = Logger.getLogger(AbstractDAO.class.getName());
    private final Class<T> type;

    /unchecked/
    public AbstractDAO() {
        this.type = (Class<T>) ((ParameterizedType) getClass().getGenericSuperclass()).getActualTypeArguments()[0];
    }
```

Clasa AbstractDAO este o clasa abstracta foarte importanta deoarece aici se gasesc toate instructiunile in SQL necesare inserarii, stergeri, si editarii. Fiecare metoda are cate o sub functie care genereaza interogarile propriu zise iar mai apoi acestea sunt apelate din alte metode pentru o compatibilitate cat mai mare. Aici este implementata reflexia, nefiind nevoiti asfel sa scriem de mai multe ori acelasi lucru economisind asfel o multime de timp. Pe langa functiile de insert update si delete, mai exista o functie care gaseste un client, o comanda sau un produs dupa

id. Tot aici se gaseste functia care selecteaza toate datele dintr-o tabela dar si functia care construieste obiectele propriu zise. Pentru a putea afisa in GUI intr-un Jtabel am facut metode care sa puna intr un tablou de tipul Object antetele tabelelor, adicab numele lor; si intr-o matrice tot de tipul Object toate datele din tabela. Pentru aceasat clasa am avut foarte mult de lucru pentru a putea fi implementata deoarece au fost de prins o multime de exceptii deoarec pentru a putea crea un obiect pe cazul general. M-am inspirat din modelul dat de GitLab.

Aceasta clasa are trei clase extensii: ClientsDAO, OrdersDAO, ProductDAO. Acestea sunt clase goale.

Clasa ClientsDAO:

```
package dao;
import model.Client;
import java.util.logging.Logger;
public class ClientsDAO extends AbstractDAO<Client>{};
```

Clasa OrdersDAO:

```
1  package dao;
2  import model.Productsorders;
3  public class OrdersDAO extends AbstractDAO<Productsorders>{};
4
```

Clasa ProductDAO:

```
package dao;
import model.Productsorders;
public class OrdersDAO extends AbstractDAO<Productsorders>{};
```

Urmeaza prezentarea claselor din pachetul connection:

Clasa ConnectionFactory:

```
public class ConnectionFactory {

    private static final Logger LOGGER = Logger.getLogger(ConnectionFactory.class.getName());
    private static final String DRIVER = "com.mysql.cj.jdbc.Driver";
    private static final String DBURL = "jdbc:mysql://localhost:3306/tp3_database";
    private static final String USER = "root";
    private static final String PASS = "Lolipop@99";
```

Cu ajutorul acestei clase se realizeaza conexiunea cu baza de date, adica se realizeaza log-area la serverul mySql. Aceasta clasa are drept utilizare inserarea, editarea, si stergerea datelor din tabel.

Urmeaza prezentarea claselor din pachetul view:

Clasa View

Aceasta clasa reprezinta tot ce inseamna implementarea interfetei grafice facute in acest caz cu java swing.

Clasa incepe prin declararea casutelor de text necesare pentru toate argumentele cerute in problema,. Casutele de text au nume suggestive: pentru id-ul clientului, al comenzii si al produsului, numele coleintului si al produsului, adresa si email ul clientului, cantitatea produsului, pretul produsului, si cantitatea comenzii.

Clasa Main:

```
import controller.Controller;
import view.*;
public class Main {

    public static void main(String[] args)  {
        View view =new View();
        Controller controller= new Controller(view);
    }


    }
```

In aceasta clasa avem parametrul destinate pentru view si controller, de aici executandu-se rularea programului.

6. Rezultat

Finalul aplicatiei se face cu generarea unui bon.

Numar bon: 1
Client cu numarul:1 si numele: Timis Iulia
Cu domiciliul in: Oltului nr 84
Cu adresa de email: t@aaa.com
Produs cu numarul: 1
Nume produs:Cola-Zero 3 bucati.
Are de platit: 12.0 lei

6. Concluzii

Aceasta aplicatie a fost foarte de ajutor pentru aprofundarea legaturilor cu baze de date. A fost o aplicatie interactive, dar a necesitat destul de mult timp. Totodată am acumalat cunostințe cu privire la generarea și scrierea într-un fisier pdf.Am învățat să folosesc tehnica de reflection.

7. Bibliografie

https://ro.wikipedia.org/wiki/Model-view-controller

http://www.mkyong.com/jdbc/how-to-connect-to-mysql-with-jdbc-driver-java

Java Reflection Tutorial (jenkov.com)

Utcn Dsrl / pt-reflection-example · GitLab

Utcn Dsrl / pt-reflection-example · GitLab

.