



**UNIVERSITATEA TEHNICĂ**

DIN CLUJ-NAPOCA

**FACULTATEA DE AUTOMATICĂ ȘI CALCULATOARE  
CATEDRA CALCULATOARE**

# **APLICAȚIE DE GESTIONARE A COMENZILOR**

## **-Documentație-**

**Student:**

Pop Ruxandra Maria

**Univeristatea Tehnică din Cluj-Napoca**

**Facultatea de Automatică și Calculatoare**

**Secția Calculatoare și Tehnologia Informației**

**Anul 2, Grupa 30226**

# Cuprins

1.Obiectivul temei .....	3
2.Analiza problemei .....	4
3.Proiectare .....	8
4.Implementare .....	16
5.Rezultate .....	20
6.Concluzii .....	21
7.Bibliografie .....	22

# 1.Obiectivul temei

## 1.1.Obiectiv principal

Obiectivul principal al acestei teme de laborator a fost să propunem, să proiectăm și să implementăm o aplicație Java pentru gestionarea comenzilor făcute de clienți către un depozit. Aplicația lucrează cu o bază de date,și trebuia să conțină minim 3 tabele (Client/Product/Order).Programul trebuia să permită adăugarea/stergerea/editarea unui client/produs,precum și plasarea unei comenzi ,în limita stocului disponibil.- Datele sunt introduse prin intermediul unei interfațe grafice.După ce se plasează comanda trebuie să se genereze un fisier pdf "order bill" ,care să conțină datele comenzi efectuate.

## 1.2.Objective secundare

**Reprezintă pașii care trebuie urmați pentru atingerea obiectivului principal.**

Obiectiv Secundar	Descriere	Capitol
<b>Impărțirea pe clase</b>	Am împărțit programul în diferite clase,ușor de înțeles ,care respectă modelul arhitectural Layers impus .	3
<b>Realizarea conexiunii cu baza de date</b>	Am realizat conexiunea cu baza de date prin intermediul pachetului connection.	3
<b>Implementarea soluției</b>	Am folosit paradigma OOP,cu multiple clase și metode,care vor fi prezentate mai jos.	4
<b>Lucrul cu fișiere</b>	Pentru realizarea acestei teme, a fost necesar să lucrez cu fișiere PDF ,pentru a genera datele despre comanda efectuată.Astfel în clasa Controller am implementat o metodă care va scrie într-un fișier de tip pdf.	2,4
<b>Tehnica reflection</b>	Gestionarea bazei de date,efectuarea diverselor operații pe această, am realizat-o pe baza tehnicii de reflection care va fi prezentata mai jos.	4

## 2. Analiza problemei

Pentru a înțelege problema și scopul ei, este necesar să definim câteva aspecte teoretice importante legate de bazele de date

**O bază de date este o colecție organizată de informații sau de date structurate, stocate electronic într-un computer.**

Programul conține o baza de date numită "schoolbd", care conține 3 tabele:

- **Tabelul Client** - conține date despre clienți. Un client prezintă un id (care este unic), un nume, o adresă, și vârsta.
- **Tabelul Product** - conține date despre produsele aflate în stoc la depozit. Un produs prezintă un id (care este unic), un nume, cantitatea prezentă în stoc la momentul actual, și un preț.
- **Tabelul Order** - conține date despre comenzile efectuate de către clienți. O comandă prezintă un id (care este unic), id-ul clientului care a efectuat comanda, id-ul produsului comandat, precum și cantitatea comandată și mai conține și totalul comenzi.

**OBS!** Fiecare comandă va conține un singur produs, iar factura se va genera, similar, pentru un singur produs.

**Pentru a crea fișierul PDF am realizat următoarea metodă în clasa controller**

```
public void printBill(OrderW order) {  
    Document document = new Document();  
    try {  
        PdfWriter.getInstance(document, new FileOutputStream("name: " + order.getIdOrder() + ".pdf"));  
    } catch (DocumentException | FileNotFoundException e) {  
        e.printStackTrace();  
    }  
}
```

## Cerințe de funcționare

**Identificarea cerințelor de funcționare constituie un element critic în dezvoltarea unui sistem software.**

*Înainte să ne apucăm de proiectarea și implementarea propriu zisă a sistemului , este necesar să cunoaștem ce cerințe trebuie acesta să îndeplinească ,pentru a descoperii ce trebuie să facă sistemul ,cum trebuie să funcționeze ,pentru a stii ce obiective dorim să atingem la sfârșitul implementari.*

*În continuare voi prezenta ,sub forma unui tabel ,cerințele sistemului și constrângerile,pe care le-am identificat în urma analizei problemei.*

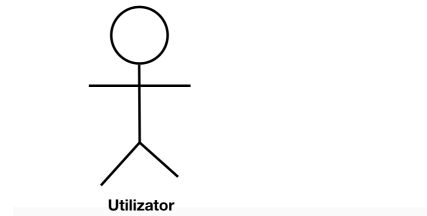
Descriere	
<b>Cerințe funcționale</b>	<ul style="list-style-type: none"><li>-Aplicația trebuie să permită utilizatorului să introducă un client/un produs/o comandă</li><li>-Aplicația trebuie să permită utilizatorului să editeze un client/un produs</li><li>-Aplicația trebuie să permită utilizatorului să steargă un client/un produs</li><li>-Aplicația trebuie să permită introducerea unei comenzi doar dacă există cantitatea de produs dorită în stoc</li><li>-Aplicația trebuie să anunțe utilizatorul în cazul unei erori ,care poate fi adăugarea unui client/produs/comanda a cărei id deja există.</li><li>-Aplicația trebuie să genereze un fisier pdf pentru fiecare comandă efectuată</li></ul>
<b>Cerințe non-funcționale</b>	<ul style="list-style-type: none"><li>-Aplicația trebuie să fie intuitivă și ușor de folosit</li><li>-Aplicația trebuie să afișeze rezultatul într-un mod cât mai interactiv și ușor de citit și înțeles</li><li>-Aplicația trebuie să permită alegerea tabelului pe care vrem să-l vizualizăm</li></ul>
<b>Constrângeri</b>	<ul style="list-style-type: none"><li>-Id-ul produsului/clientului/comenzi trebuie să fie un număr pozitiv mai mare decât 0</li><li>-Cantitatea produsului introdus trebuie să fie mai mare decât 0</li></ul>

## Diagrama de use case (în cazul operației de scadere)

**Diagrama de use case prezintă o colecție de cazuri de utilizare și actori.**

**În cazul acestei teme :**

- **actorul este utilizatorul, cel care folosește calculatorul.**



- **cazurile de utilizare în cazurile operației de adugare a clientului/produsului sunt:**

- **În caz de succes:**

- utilizatorul introduce datele necesare în interfața grafică
- utilizatorul face click pe butonul **"add"**.
- aplicația introduce clientul/produsul în baza de date ,după care se reactualizează tabelul din interfața grafică

- **În caz de nesucces:**

- utilizatorul nu completează toate câmpurile necesare
- utilizatorul introduce un client/produs cu un id care deja există în baza de date

- **cazurile de utilizare în cazurile operației de editare a clientului/produsului sunt:**

- **În caz de succes:**

- utilizatorul introduce id-ul clientului/produsului pe care dorește să-l actualizeze
- utilizatorul introduce datele pe care dorește să le actualizeze în interfața grafică
- utilizatorul face click pe butonul **"edit"**.
- aplicația actualizează datele clientului/produsul în baza de date ,după care se reactualizează tabelul din interfața grafică

- **În caz de nesucces:**

- utilizatorul nu introduce id-ul clientului/produsului
- utilizatorul introduce un client/produs cu un id care nu există în baza de date

- **cazurile de utilizare în cazul operației de ștergere a clientului/produsului sunt:**

- **În caz de succes:**

-utilizatorul introduce id-ul produsului/clientului pe care dorește să-l șteargă din baza de date

-utilizatorul face click pe butonul **“remove”**.

-aplicația șterge clientul/produsul din baza de date ,după care se reactualizează tabelul din interfața grafică

- **În caz de nesucces:**

-utilizatorul nu introduce id-ul clientului/produsului

- utilizatorul introduce un client/produs cu un id care nu există în baza de date

- **cazurile de utilizare în cazul operației de efectuare a unei comenzi sunt:**

- **În caz de succes:**

-utilizatorul introduce în interfața grafică datele necesare .

-utilizatorul face click pe butonul **“place order”**.

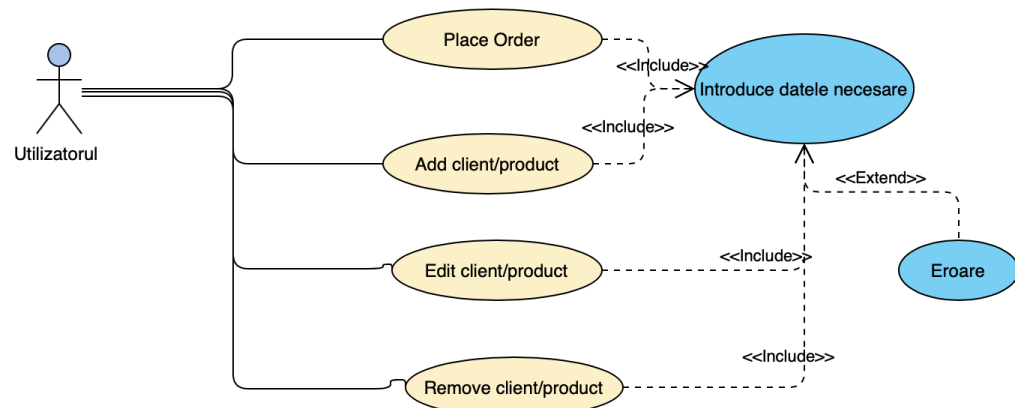
-aplicația introduce comanda în baza de date ,după care se reactualizează tabelul din interfața grafică

- **În caz de nesucces:**

-utilizatorul nu introduce toate datele necesare în interfața grafică

-utilizatorul introduce un id-ul a unui client/produs care nu există în baza de date

-utilizatorul introduce un id-ul a unei comenzi care există deja în baza de date



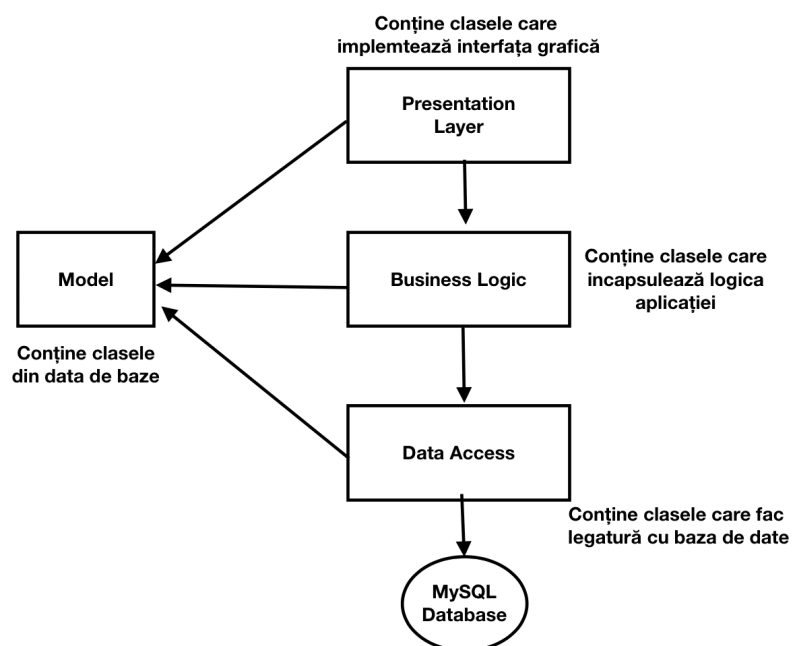
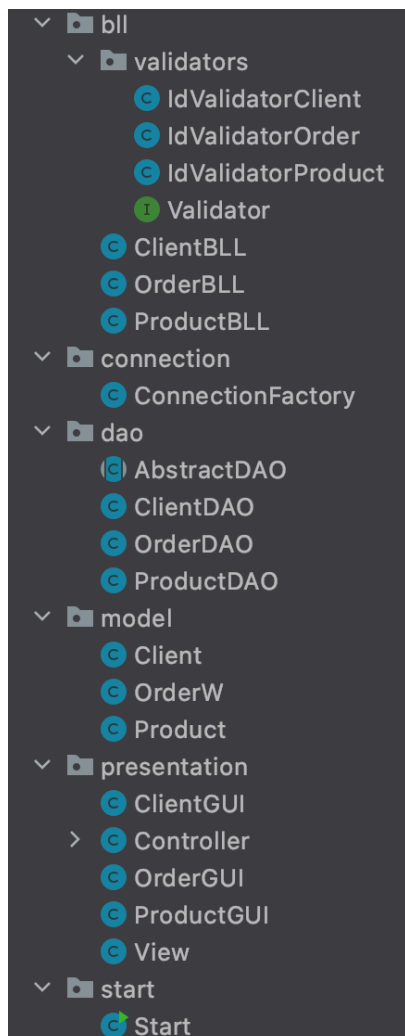
# 3.Proiectare

## 3.1.Etapa de design

Proiectarea claselor a fost făcută ca respectând modelul pe nivele prezentat în prezentarea suport a temei.Astfel fiecare nivel ,are un rol prestabilit,și am implementat toate nivele ,fără să evit vre-unul.

Am structurat aplicația conform modelului arhitectural Layers, ce presupune împărțirea responsabilităților pe mai multe nivele, lucru care facilitează înțelegerea și dezvoltarea ulterioară a aplicației.

Am împărțit aplicația în 6 pachete după cum se poate observa :

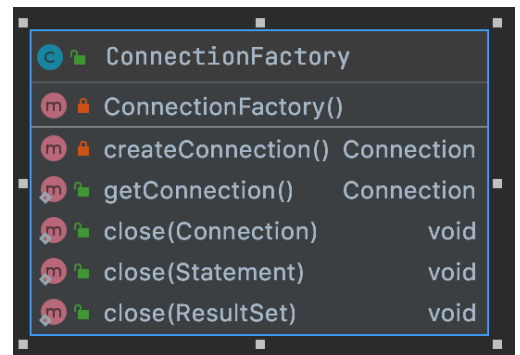


Layered Architectures



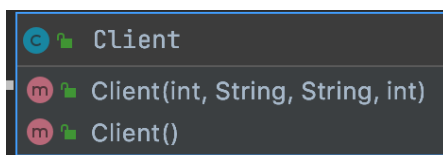
În continuare voi prezenta fiecare pachet pe rând:

1. **Pachetul connection** - conține doar clasa `ConnectionFactory`, care realizează conexiunea la baza de date. Se găsesc metode de `get()` pentru conexiune, `close()` pentru conexiune, `statement` și `resultSet`.

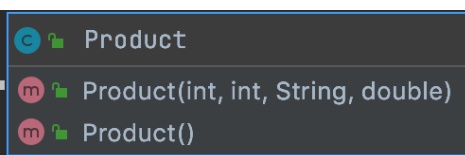


ConnectionFactory
ConnectionFactory()
createConnection() Connection
getConnection() Connection
close(Connection) void
close(Statement) void
close(ResultSet) void

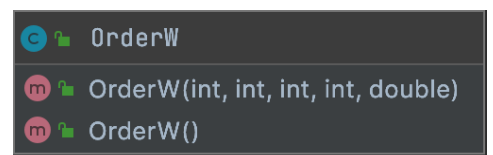
2. **Pachetul model** - conține clasele ce modelează baza de date, corespund tabelor din baza de date: au aceleași câmpuri, aceeași ordine, și aceleași tipuri de date. Conțin doar metode de `get` și `set`. Clasele sunt: `Client`, `Product`, `OrderW`.



Client
Client(int, String, String, int)
Client()

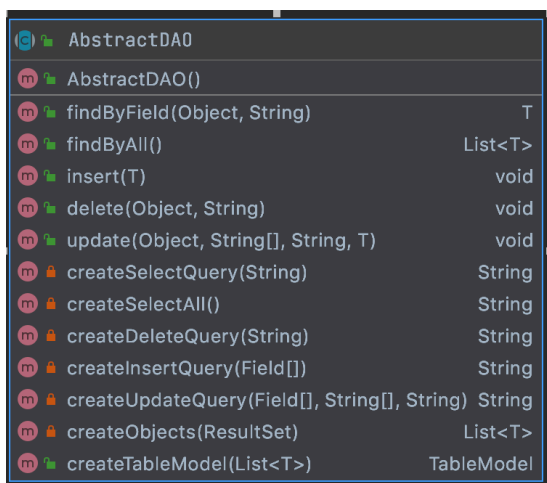


Product
Product(int, int, String, double)
Product()

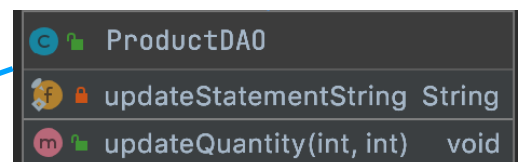


OrderW
OrderW(int, int, int, int, double)
OrderW()

3. **Pachetul dao** - conține o clasă generică, abstractă `AbstractDAO`, și încă 3 clase care extind această clasă, și anume: `ClientDAO`, `ProductDAO`, `OrderDAO`. Acest pachet este folosit pentru interogarea bazei de date și modificarea/stergerea/adaugarea datelor din aceasta. Aici am folosit tehnica reflecției, astfel încât operațiile pe tabele se fac prin aceleași metode indiferent de tipul obiectului din clasa `Model`.



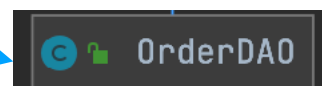
AbstractDAO
AbstractDAO()
findByField(Object, String) T
findByAll() List<T>
insert(T) void
delete(Object, String) void
update(Object, String[], String, T) void
createSelectQuery(String) String
createSelectAll() String
createDeleteQuery(String) String
createInsertQuery(Field[]) String
createUpdateQuery(Field[], String[], String) String
createObjects(ResultSet) List<T>
createTableModel(List<T>) TableModel



ProductDAO
updateStatementString String
updateQuantity(int, int) void



ClientDAO
-----------



OrderDAO
----------

**4. Pachetul bll-** conține 3 clase și anume: ClientBLL, ProductBLL, OrderBLL. Acestea implementează logica de business ,pentru clasele din Model.

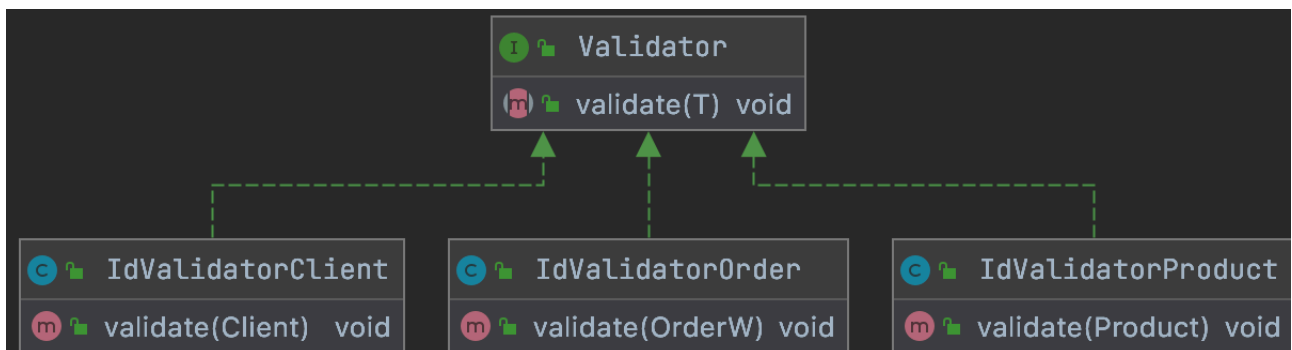
Acest pachet conține un subpachet validators, care conține mai multe clase care implementează interfața Validator ,pentru a nu permite introducerea unor date cu id negativ în tabele.

ProductBLL	
ProductBLL()	
getProducts()	List<Product>
productTable(List<Product>)	TableModel
findProduct(int)	Product
insertProduct(Product)	void
updateQuantity(int, int)	void
updateProduct(int, int, String, double)	void
deleteProduct(Product)	void

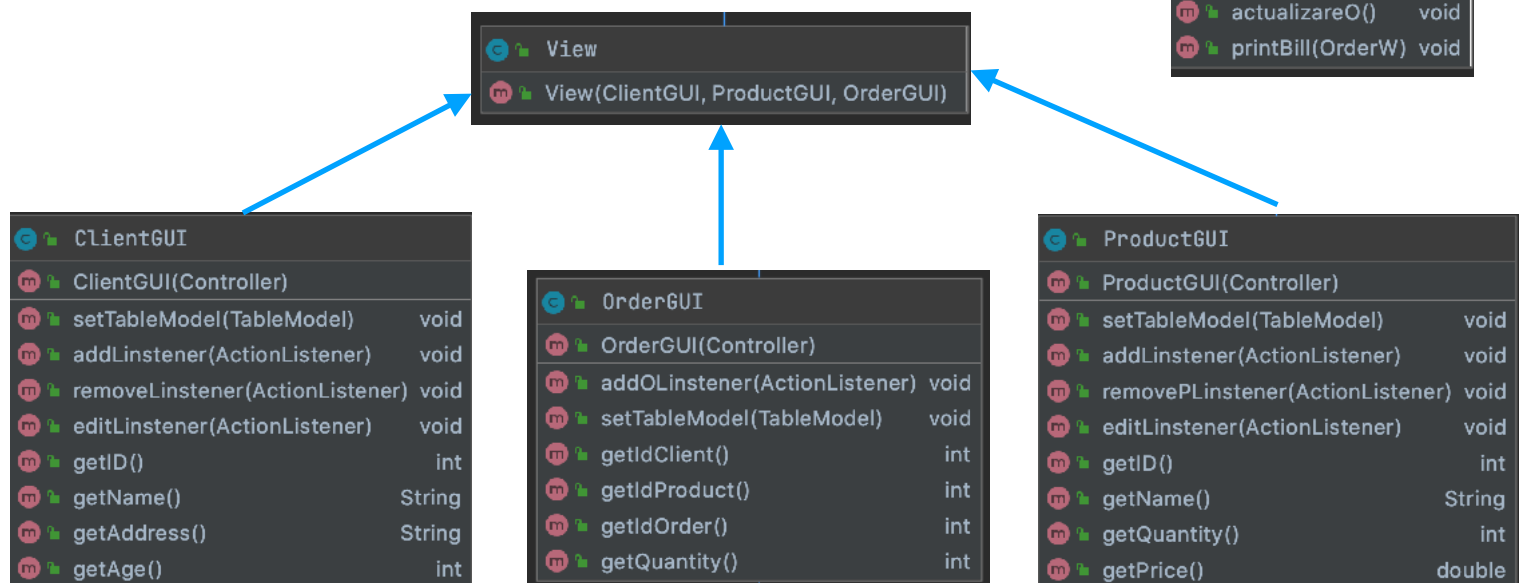
ClientBLL	
ClientBLL()	
getClients()	List<Client>
clientTable(List<Client>)	TableModel
updateClient(int, String, String, int)	void
insertClient(Client)	void
findClient(int)	Client
deleteClient(Client)	void

OrderBLL	
OrderBLL()	
getOrders()	List<OrderW>
orderTable(List<OrderW>)	TableModel
insertOrder(OrderW)	void

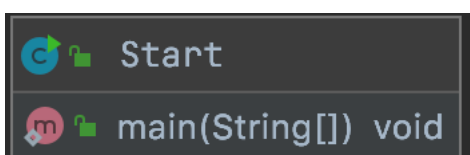
### Subpachetul validators:



**5. Pachetul presentation-** aici se întâlnesc clasele destinate interacțiunii cu utilizatorul. Conține 5 clase, dintre care una este Controller, care are rolul de transmite datele, primite prin interfața grafică, modelului. Celalalte 4 clase sunt destinate modelării interfeței grafice, care va fi prezentată mai detaliat în următorul subcapitol. Am creat câte un nou frame pentru fiecare tabel. Și un frame principal de unde se pot deschide celalalte 3 frame-uri secundare, în funcție de alegerea făcută de utilizator.



**6. Pachetul start-** conține clasa Start, cu metoda main, de unde începe execuția programului. Aici se apelează o nouă instanță de controller, care va duce la pornirea întregii aplicații.



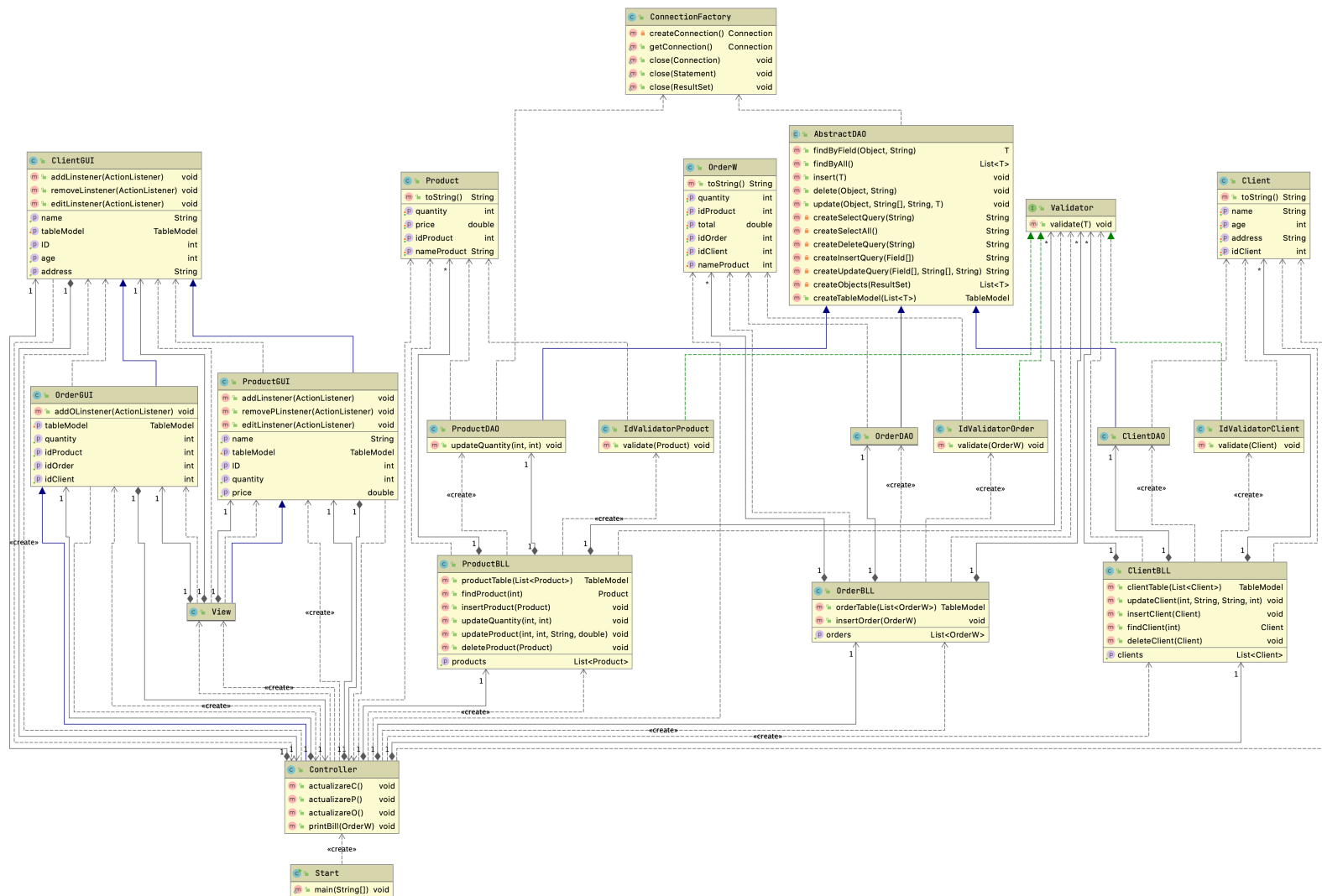
```

public class Start {

    public static void main(String[] args) {
        new Controller();
    }
}
  
```

# UML-Unified Modeling Language.

Reprezintă un set de clase,interfețe,colaborări și alte relații.



## 3.2.Structuri de date

Am decis ca în acest proiect să folosesc ca și structuri de date colecțiile.

**Colecția : orice clasă care păstrează obiecte și implementează interfața Collection.**

Ca structuri de date am folosit List ,pentru a reține clienți ,produsele,comenzi-ile.Iar pentru validatori am folosit și ArrayList.Folosirea acestor colecții m-au ajutat la implementarea soluției.

```
public class ClientBLL {  
    private ClientDAO clientDao;  
    private List<Client> clients;  
    private List<Validator<Client>> validators;  
  
    public ClientBLL() {  
        validators = new ArrayList<Validator<Client>>();  
        validators.add(new IdValidatorClient());  
    }  
}
```

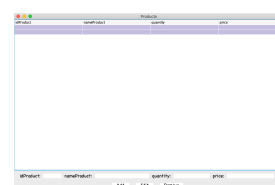
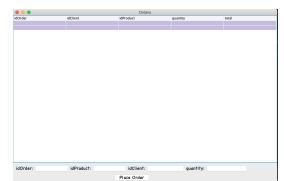
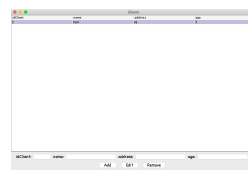
## 3.3.Interfața grafică (Graphical User Interface)

**Reprezintă un mecanism prietenos pentru interacțiunea utilizatorului cu programul.**

Folosim o interfață User-Friendly pentru a permite utilizatorilor să se simtă mai familiarizați cu programul chiar înainte de a-l fi utilizat. Printr-o interfață utilizatorul poate înțelege mai bine cum funcționează programul ,poate învăța mai repede modul de utilizare a acestui calculator.Aceasta interfață cuprinde 4 frame-uri(windows), unul principal ,și 3 secundare.



Principal



Secundare

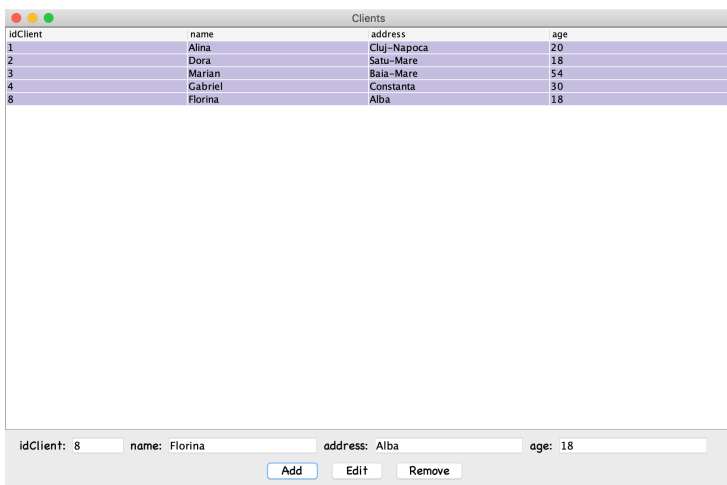
**Frame-ul principal-** *Este implementat de clasa View.*



Prezintă :- 3 butoane —"Clients"  
—"Products"  
—"Place Orders"  
-un JLabel — care este titlul frame-ului

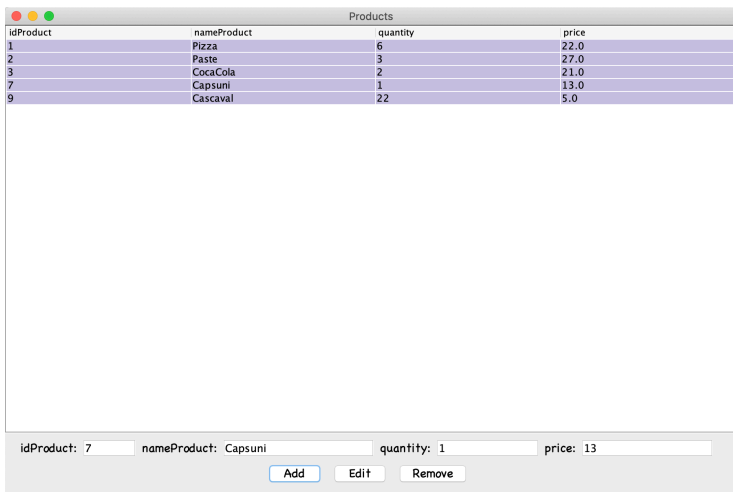
În funcție de butonul apăsăat de catre utilizator ,se va deschide unul dintre frame-urile secundare. Cele trei frame-uri secundare sunt asemanatoare ca design.

**Frame-ul secundar (Clients)-** *Este implementat de clasa ClientGUI.*



Prezintă :- 3 butoane —"Add"  
—"Edit"  
—"Remove"  
-un JTable—care conține datele din baza de date(coloanele sale fiind corespunzătoare câmpurilor din tabelul Client)  
-4 JLabel-uri—care indică utilizatorului, atributele clientului  
-4 JText-uri — unde va introduce utilizatorul datele

## Frame-ul secundar (Products)- Este implementat de clasa ProductGUI.



idProduct	nameProduct	quantity	price
1	Pizza	6	22.0
2	Paste	3	27.0
3	CocaCola	2	21.0
7	Capsuni	1	13.0
9	Cascaval	22	5.0

idProduct: 7   nameProduct: Capsuni   quantity: 1   price: 13

Add   Edit   Remove

Prezintă :- 3 butoane —"Add"

—"Edit"

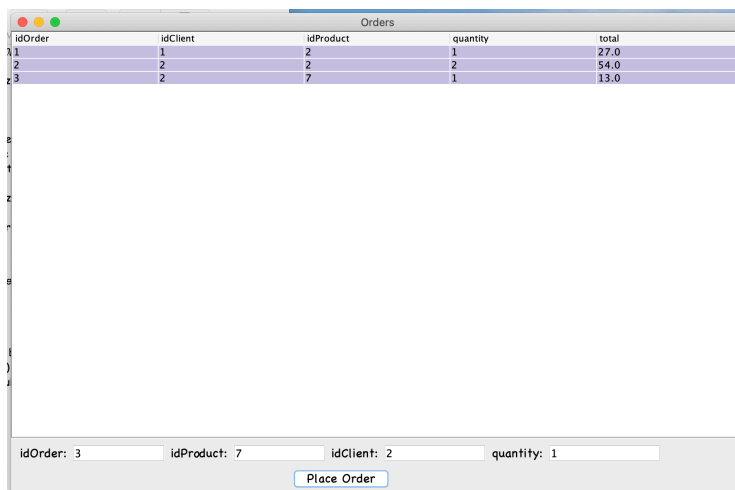
—"Remove"

-un JTable—care conține datele din baza de date(coloanele sale fiind corespunzătoare câmpurilor din tabelul Product)

-4 JLabel-uri—care indică utilizatorului, attributele produsului

-4 JText-uri — unde va introduce utilizatorul datele

## Frame-ul secundar (PlaceOrders)- Este implementat de clasa OrderGUI.



idOrder	idClient	idProduct	quantity	total
1	1	2	1	27.0
2	2	2	2	54.0
3	2	7	1	13.0

idOrder: 3   idProduct: 7   idClient: 2   quantity: 1

Place Order

Prezintă :- 1 buton —"Place Order"

-un JTable—care conține datele din baza de date(coloanele sale fiind corespunzătoare câmpurilor din tabelul Order)

-4 JLabel-uri—care indică utilizatorului, attributele comenzii

-4 JText-uri — unde va introduce utilizatorul datele

!

În main se instanțiază interfața grafică prin intermediul controller-ului, deci de fiecare dată când este rulat main ,se formează o nouă interfață.

# 4.Implementare

Clasele folosite sunt destul de ușor de înțeles și interpretat.M-am inspirat din idei pe care ați pus-o în prezentarea suport și pe GitLab.

Clasele din pachetul validators au fost prezentate în capitolul 3.

## 1.Clasa Client

Această clasă simulează un client din baza de date.Are ca și attribute ,coloanele din baza de date corespunzătoare tabelului Client: idClient care este unic în tabela,name- numele clientului,address -adresa unde locuiește ,age-vârsta acestuia.Se găsesc cei doi constructorii ,precum și numeroase metode de get și set.

```
public class Client {
    private int idClient;
    private String name;
    private String address;
    private int age;

    public Client(int idClient, String name, String address, int age) {
        this.idClient = idClient;
        this.name = name;
        this.address = address;
        this.age = age;
    }

    public Client() {
    }
}
```

## 2.Clasa Product

Această clasă simulează un produs vândut de către magazin ,produs din baza de date.Are ca și attribute ,coloanele din baza de date corespunzătoare tabelului Product: idProduct care este unic în tabela,nameProduct- numele produsului,quantity-numărul de bucăți care se găsesc în magazin ,price-prețul unei bucați.Se găsesc cei doi constructorii ,precum și numeroase metode de get și set.

```
public class Product {
    private int idProduct;
    private String nameProduct;
    private int quantity;

    private double price;

    public Product(int idProduct, int quantity, String nameProduct, double price) {
        this.idProduct = idProduct;
        this.quantity = quantity;
        this.nameProduct = nameProduct;
        this.price = price;
    }

    public Product() {
    }
}
```

## 3.Clasa OrderW

Această clasă simulează o comandă a unui client pentru un singur produs,comandă din baza de date.Are ca și attribute ,coloanele din baza de date corespunzătoare tabelului Order: idOrder care este unic în tabela,idClient- id-ul clientului care a efectuat comandă,idProduct-id-ul produsului comandat de către client,quantity-nr de bucăți comandate,și total-care reprezintă suma pe care trebuie să o plătească clientul. Se găsesc cei doi constructorii ,precum și numeroase metode de get și set.

```
public class OrderW {
    private int idOrder;
    private int idClient;
    private int idProduct;

    private int quantity;
    private double total;

    public OrderW(int idOrder, int idProduct, int idClient, int quantity, double total) {
        this.idOrder = idOrder;
        this.idProduct = idProduct;
        this.idClient = idClient;
        this.quantity = quantity;
        this.total = total;
    }

    public OrderW() {
    }
}
```



## 4. Clasa ClientBLL

Această clasă implementează layer-ul business logic pentru tabelul Client. Apelând diverse metode pe un obiect de tip ClientDAO. Metoda getClients(), returnează clienți care se găsesc la momentul actual în baza de date. Metodele insertClient, updateClient, deleteClient, findClient, au ca scop, inserarea unui client în baza de date, actualizarea unui client din baza de date, stergerea unui client din baza de date, precum căutarea unui client cu un anumit id. Totodată aici se găsește și metoda clientTable care are ca scop crearea în interfața grafică a tabelului corespunzător modelului Client. Ca atribut prezintă clientDao care este de tip ClientDAO, o listă de clienți, și o listă de validatori.

```
public class ClientBLL {
    private ClientDAO clientDao;
    private List<Client> clients;
    private List<Validator<Client>> validators;

    public ClientBLL() {
        validators = new ArrayList<Validator<Client>>();
        validators.add(new IdValidatorClient());
        clientDao = new ClientDAO();
        clients = clientDao.findAll();
    }
}
```

## 5. Clasa ProductBLL

Această clasă implementează layer-ul business logic pentru tabelul Product. Apelând diverse metode pe un obiect de tip ProductDAO. Metoda getProducts(), returnează produsele care se găsesc la momentul actual în baza de date. Metodele insertProduct, updateProduct, deleteProduct, findProduct, au ca scop, inserarea unui produs în baza de date, actualizarea unui produs din baza de date, stergerea unui produs din baza de date, precum căutarea unui produs cu un anumit id. Totodată aici se găsește și metoda productTable care are ca scop crearea în interfața grafică a tabelului corespunzător modelului Product și o metodă de updateQuantity care va modifica cantitatea produsului atunci când este comandat. Ca atribut prezintă productDao care este de tip ProductDAO, o listă de produse, și o listă de validatori.

```
public class ProductBLL {
    private ProductDAO productDao;
    private List<Product> products;
    private List<Validator<Product>> validators;

    public ProductBLL() {
        productDao = new ProductDAO();
        products = productDao.findAll();
        validators = new ArrayList<Validator<Product>>();
        validators.add(new IdValidatorProduct());
    }
}
```

**6. Clasa OrderBLL**-Această clasă implementează layer-ul business logic pentru tabelul Order. Apelând diverse metode pe un obiect de tip OrderDAO. Metoda `getOrders()`, returnează comenzile care se găsesc la momentul actual în baza de date. Metoda `insertOrder` inserează o comandă în baza de date. Totodată aici se găsește și metoda `orderTable` care are ca scop crearea în interfața grafică a tabelului corespunzător modelului Order. Ca atribut prezintă `orderDao` care este de tip OrderDAO, o listă de comenzi și o listă de validatori.

```
public class OrderBLL {
    private OrderDAO orderDao;
    private List<OrderW> orders;
    private List<Validator<OrderW>> validators;

    public OrderBLL() {
        validators = new ArrayList<Validator<OrderW>>();
        validators.add(new IdValidatorOrder());
        orderDao = new OrderDAO();
        orders = orderDao.findAll();
    }
}
```

## 7. Clasa AbstractDAO

Este clasa generică care conține metodele pentru modelare bazei de date.

Această clasă prezintă metodele de insert, update, delete, find pentru obiecte instanță ale claselor din pachetul Model. Aceste metode sunt implementate prin tehnica reflection, pentru a nu trebui să scriem mai multe metode similare dar pe alt tip de obiect. Astfel, a fost definit un tip generic asociat, care se obține în mod dinamic. Metoda `findByField` efectuează o interogare creată de către metoda `createSelectQuery`, și returnează un obiect. Metoda `findByAll` efectuează o interogare creată de metoda `createSelectALL`, și returnează toate obiectele de tip T din baza de date. Metoda `insert` efectuează o interogare creată de metoda `createInsertQuery`, și are ca scop inserarea unui element dat ca atribut, în tabelul corespunzător tipului sau. Metoda `delete` efectuează o interogare formată de metoda `createDeleteQuery`, și are ca scop stergerea unui element din tabel în funcție de id-ul dat de către utilizator. Metoda `update` efectuează o interogare creată de metoda `createUpdateQuery`, și are ca scop actualizarea anumitor date ale elementului dat ca argument, datele cu care urmează să fie actualizate coloanele elementului se regăsesc în tabelul de tip string `update`. Metoda `createObjects` are ca scop crearea unui noi obiect de tip T. Metoda `createTableModel`, ajută la formarea unui tabel de tip T care va fi afișat în interfața grafică specifică.

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

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

- Această clasă este extinsă de următoarele clase ,care ii vor putea implementa toate metodele în funcție de tipul ei :ClientDAO,OrderDAO,ProductDAO.În clasa ProductDAO , am ales să implementez metoda updateQuantity, care va modifica cantitatea produsului atunci când este comandat ,pentru a imi fi mai usor în implementarea aplicației.

```
public class ClientDAO extends AbstractDAO<Client> {
```

```
public class OrderDAO extends AbstractDAO<OrderW> {
```

```
public class ProductDAO extends AbstractDAO<Product> {
    private static final String updateStatementString =
```

## 6. Clasa view

Este clasa care implementează Presentation Layer. Aici se face legătura cu celelalte 3 clase de tip GUI. Prezintă doar metodele fundamentale de tip setVisible, pentru cele 3 butoane prin care se vor putea accesa celelalte clase. Constructorul are 3 attribute care sunt instanțe ale celorlalte 3 clase de tip Presentation.

```
public View(ClientGUI clientGUI, ProductGUI productGUI, OrderGUI orderGUI) {
    this.clientGUI = clientGUI;
    this.productGUI = productGUI;
    this.orderGUI = orderGUI;
```

- Clasa ClientGUI extinde JFrame.

```
public ClientGUI(Controller controller) {
    super( title: "Clients");
    this.controller = controller;
```

Această clasa implementează Presentation Layer pentru tabelul Client din baza de date. Prezintă metode pentru cele 3 butoane ,precum și metode de get care au ca scop preluarea datelor din interfața grafică. Si metoda de setTableModel care ajută la crearea tabelui din interfață Are ca attribute numeroase textField-uri, butoane ,label , un controler.

- **Clasa ProductGUI extinde JFrame.**

```
public ProductGUI(Controller controller) {
    super( title: "Products");
    this.controller = controller;
}
```

Această clasă implementează Presentation Layer pentru tabelul Product din baza de date. Prezintă metode pentru cele 3 butoane, precum și metode de get care au ca scop preluarea datelor din interfața grafică. Și metoda de setTableModel care ajută la crearea tabelului din interfață. Are ca atribute numeroase textField-uri, butoane, label, un controller.

- **Clasa OrderGUI extinde JFrame.**

```
public OrderGUI(Controller controller) {
    super( title: "Orders");
}
```

Această clasă implementează Presentation Layer pentru tabelul OrderW din baza de date. Prezintă o metodă pentru butonul de "place order", precum și metode de get care au ca scop preluarea datelor din interfața grafică. Și metoda de setTableModel care ajută la crearea tabelului din interfață. Are ca atribute numeroase textField-uri, butoane, label, un controller.

## 7. Clasa Controller

Această clasă implementează Presentation Layer, aici se face legătura între utilizator și interfața grafică. Aici se instantiază obiectele de tip GUI. Prezintă numeroase metode, pe care le-am explicat mai detaliat în javadoc, urmând ca aici să le prezint pe scurt. Prezintă 3 metode de inserare a unui client în baza de date, a unui produs sau a unei comenzi. Prezintă 2 metode de stergere care au ca scop stergerea unui client sau produs din baza de date. Prezintă 2 metode de update, care au ca scop actualizarea datelor unor client, sau produs. Metodele de stergere și actualizare necesită introducerea id-ului, astfel ca aplicația să știe pe ce produs/client să efectueze operația. Mai întâlnim aici și o metodă printBill care are ca scop crearea unui document PDF, care să conțină datele comenzii efectuate.

```
public Controller() {
    client = new ClientBLL();
    clientGUI = new ClientGUI( controller: this);
    product = new ProductBLL();
    productGUI = new ProductGUI( controller: this);
    order = new OrderBLL();
    orderGUI = new OrderGUI( controller: this);
}
```

## 5.Rezultate

Pentru testarea aplicației am introdus diferite date în interfața grafică,și am testat fiecare operație pe care o poate efectua.Iar înainte să implementez interfața grafică am introdus date de la tastatură, și am efectuat diverse operații. Pe parcurs am întâmpinat diverse erori,pe care le-am rezolvat de-a lungul dezvoltării aplicației. În varianta finală a proiectului nu au mai aparut erori sau greseli .

În ceea ce privește javadoc am reușit să-l generez (se află în folder-ul javaDoc). Pentru fiecare clasă și metodă am scris câte ceva în javadoc.



javaDoc

Un extras din cod,unde am scris o documentație pentru metoda de creare a unui query de update.

```
/**
 * Formează o interogare de tip update pentru un obiect de tip T.
 *
 * @param fields contine toate coloanele din tabelul corespunzator obiectului de tip T.
 * @param update contine valorile campurilor care urmeaza sa fie schimbate.
 * @param fieldId este denumirea coloanei tabelului.
 * @return interogarea de update.
 */
```

### All Packages

#### Package Summary

Package	Description
bll	
bll.validators	
connection	
dao	
model	
presentation	
start	

OVERVIEW PACKAGE CLASS USE TREE DEPRECATED INDEX HELP

## 6.Concluzie

În concluzie, acest proiect ma ajutat să învăț cum să lucrez cu baze de date în Java,fiind prima dată când fac acest lucru.Am văzut cum trebuie făcută conexiunea cu baza de date.Totodată am acumulat cunostințe cu privire la generarea și scrierea într-un fisier pdf.Am învățat să folosesc tehnica de reflection.

Ca și dezvoltare ulterioară, interfața grafică ar putea fi îmbunătățită,astfel încât să prezinte mai multe obțiuni.Claselor model să prezinte mai multe atribute,iar comanda să fie alcătuită din mai multe tipuri de produse ,nu doar din unul.

## 7.Bibliografie

1. <https://ro.wikipedia.org>

2. <https://stackoverflow.com>

3. <https://www.tutorialspoint.com/index.htm>

4. <https://www.youtube.com>

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

6. <https://www.baeldung.com/java-reflection>