

**ORDER MANAGEMENT**

**DOCUMENTAȚIE**

**Facultatea:** Automatică și Calculatoare

**Specializarea:** Calculatoare și Tehnologia Informației

**Materia:** Tehnici de programare fundamentale

**Nume student:** Petricele Mihaela **Profesor coordonator**

**Grupa:** 30228 Dr. Pop Cristina Bianca

**Cuprins**

[**I.** **Obiectivul temei** 3](#_Toc71762708)

[**II.** **Analiza problemei, modelare, scenarii, cazuri de utilizare** 3](#_Toc71762709)

[***A.*** ***Diagrama Use – case a aplicației*** 4](#_Toc71762710)

[***B.*** ***Descrierea cazurilor de utilizare*** 4](#_Toc71762711)

[**III.** **Proiectare** 5](#_Toc71762712)

[*A)* *Proiectarea OOP a aplicației* 5](#_Toc71762713)

[*B)* *Diagramele UML ale aplicației* 6](#_Toc71762714)

[*a.* *Diagrama de pachete* 6](#_Toc71762715)

[*b.* *Diagrama de clase* 7](#_Toc71762716)

[*C)* *Structurile de date folosite* 7](#_Toc71762717)

[*D)* *Interfețele definite* 7](#_Toc71762718)

[**IV.** **Implementare** 8](#_Toc71762719)

[**A)** **Pachetul BLL** 8](#_Toc71762720)

[**B)** **Pachetul Connection** 9](#_Toc71762721)

[**C)** **Pachetul DAO** 9](#_Toc71762722)

[**D)** **Pachetul Model** 10](#_Toc71762723)

[**E)** **Pachetul Presentation** 11](#_Toc71762724)

[**V.** **Rezultate** 14](#_Toc71762725)

[**VI.** **Concluzii** 15](#_Toc71762726)

[**VII.** **Bibliografie** 15](#_Toc71762727)

# **Obiectivul temei**

*Obiectivul principal* al temei este proiectarea și implementarea unei aplicații OrderManagement pentru procesarea comenzilor clienților pentru un depozit

*Obiectivele secundare* ale temei sunt:

* **Analiza problemei** (în capitolul 2) => se prezintă cerințele funcționale, cazurile de utilizare sub formă de

diagrame use-case

* **Proiectare** (în capitolul 3) => proiectarea OOP aplicației, diagramele UML de clase și de pachete,

structurile de date folosite, interfețele definite și algoritmii folosiți.

* **Implementare** (în capitolul 4) => descrierea claselor, metodelor folosite și descrierea implementării

interfeței cu utilizatorul

* **Rezultate** ( în capitolul 5) => prezentarea facturilor generate

# **Analiza problemei, modelare, scenarii, cazuri de utilizare**

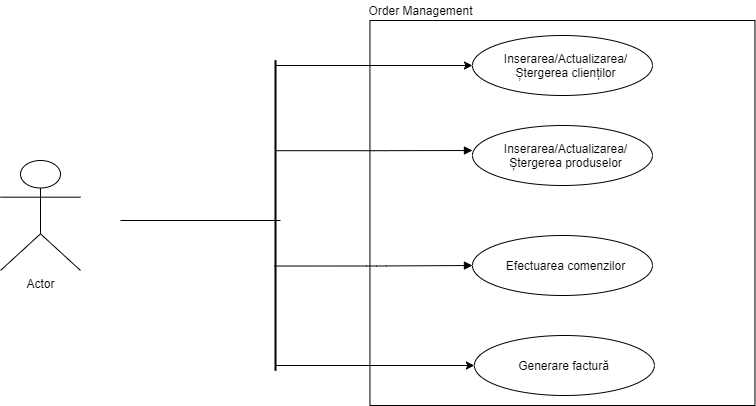
Aplicația ar trebui să permită realizarea operațiilor de insert, update și delete realizate pe o bază de date.

Aceasta totodată ar trebui să țină evidența despre clienți, produse și comenzile realizate de către aceștia și să genereze o factură corespunzătoare care să conțină detaliile despre produsele achiziționate.

Aplicația trebuie să permită utilizatorului:

* Să vizualizeze trei interfețe grafice:
* Fereastra pentru Client
  + - Permite introducerea clienților in baza de date
    - Permite realizarea operației de update a anumitor detalii ale clienților
    - Permite ștergerea clienților prin introducerea Id-ului specific
    - Permite vizualizarea clienților prin intermediul unui tabel
* Fereastra pentru Produs
  + - Permite introducerea produselor in baza de date
    - Permite realizarea operației de update a anumitor detalii ale produselor
    - Permite ștergerea produselor prin introducerea Id-ului specific
    - Permite vizualizarea produselor prin intermediul unui tabel
* Fereastra pentru Comanda
  + - Permite realizarea de comenzi prin introducerea: IdClient, IdProdus și cantitatea care dorește a fi comandată
    - Permite vizualizarea comenzilor prin intermediul unui tabel

## ***Diagrama Use – case a aplicației***



## ***Descrierea cazurilor de utilizare***

***Use case:*** Inserarea/Actualizarea/Ștergerea clienților

***Scenariul de success:*** Utilizatorul poate introduce valori pentru:

1. Inserare clienți

* Utlizatorul introduce corect valorile în fiecare TextField și la apăsarea butonului de *Insert* se introduce un client în baza de date.

1. Actualizare clienți

* Utlizatorul introduce corect valorile în fiecare TextField și la apăsarea butonului de *Update* se actualizează datele clientului în baza de date.

1. Ștergere clienți

* Utlizatorul introduce corect valoarea id-ului clientului pe care dorește să îl șteargă în TextField și la apăsarea butonului de *Delete* clientul este șters din baza de date.

***Scenariu alternativ:***

1. Utilizatorul introduce în TextField-ul destinat pentru Email, un email incorect, caz în care se va afișa un mesaj de eroare.
2. Utilizatorul introduce în TextField-ul destinat pentru Age, un număr în afara intervalului 7-70 pentru ani, caz în care se va afișa un mesaj de eroare.

***Use case:*** Inserarea/Actualizarea/Ștergerea produselor

***Scenariul de success:*** Utilizatorul poate introduce valori pentru:

1. Inserare produse

* Utlizatorul introduce corect valorile în fiecare TextField și la apăsarea butonului de *Insert* se introduce un produs în baza de date.

1. Actualizare produse

* Utlizatorul introduce corect valorile în fiecare TextField și la apăsarea butonului de *Update* se actualizează datele produsului în baza de date.

1. Ștergere produse

* Utlizatorul introduce corect valoarea id-ului produsului pe care dorește să îl șteargă în TextField și la apăsarea butonului de *Delete* produsul este șters din baza de date.

***Scenariu alternativ:*** Utilizatorul introduce într-un TextField o valoare care nu este compatibilă cu tipul câmpului, caz în care se va afișa un mesaj de eroare.

***Use case:*** Efectuarea comenzilor

***Scenariul de success:*** Utilizatorul introduce toate valorile corect în zonele de text și la apăsarea butonului de *Order* se efectuează comanda.

***Scenariu alternativ:***

1. Utilizatorul introduce un client sau un produs care nu se află în baza de date, caz în care se va afișa un mesaj de eroare.
2. Utilizatorul introduce o cantitate prea mare care nu se află disponibilă pentru un produs, caz în care se va afișa un mesaj de eroare.

***Use case:*** Generare factură

***Scenariul de success:*** Utilizatorul introduce toate valorile corect pentru realizarea comenzii, iar comanda se efectuează, totodată generându-se și factura corespunzătoare.

***Scenariu alternativ:*** Comanda nu s-a efectuat, caz în care factura nu a putut să se genereze.

# **Proiectare**

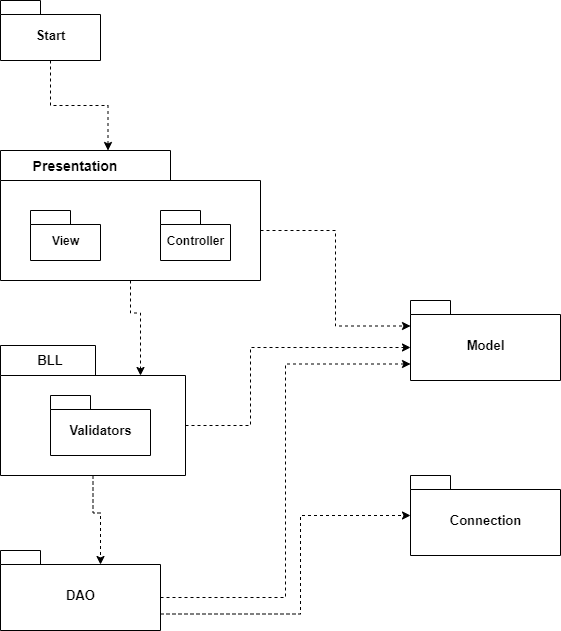
## *Proiectarea OOP a aplicației*

Aplicația este organizată pe Layere. Aceasta conține pachetele BLL (BusinessLayer) cu subpachetul

Validators, DAO (dataLayer), Model, Presentation cu subpachetele View și Controller și Connection. În pachetul Bll se află Clasele care implementează logica de business a aplicației, iar în subpachetul Validators se regăsesc Clasele pentru descrierea validatorilor anumitor field-uri. În pachetul de Dao se află clasa generică AbstractDao care implementează toate operațiile comune realizate asupra tabelelor din DB. În Model se află clasele cu aceași denumire ca tabelele din baza de date, iar în Connection se află Clasa care implementează metoda pentru conectarea cu baza de date și metode pentru închiderea conexiunii. Iar în pachetul Presentation se implementează interfața cu utilizatorul a aplicației.

## *Diagramele UML ale aplicației*

### *Diagrama de pachete*



### *Diagrama de clase*

O imagine care conține text, ecran, captură de ecran

Descriere generată automat

## *Structurile de date folosite*

Pentru implementarea acestei aplicații am folosit tipuri generice pentru declararea tuturor operațiilor

comune pentru accesarea tabelei: insert, update, delete, findById, findAll. În același timp am folosit fișiere pentru generarea facturii unei comenzi.

## *Interfețele definite*

În cadrul acestei teme am lucrat cu următoarele interfețe pentru a mă ajuta sa implementez mai eficient

aplicația:

* Interfața ActionListener
* Interfața definită Validator care definește o metodă pentru validarea anumitor câmpuri din aplicație

# **Implementare**

În acest capitol se prezintă clasele și metode corespunzătoare care au ajutat la realizarea aplicației

OrderManagement.

## **Pachetul BLL**

1. **Clasa ClientBLL**

Această clasă conține anumite metode care realizează operații asupra tabelei de clienți apelând

metodele de insert, update și delete din clasa ClientDAO. În același timp se verifică cu ajutorul validatorilor integritatea datelor. În metoda *insertClient()* înainte să se apeleze metoda de insert din ClientDAO se verifică cu ajutorul validatorilor dacă field-ul de ,,Age’’ și field-ul de ,,Email’’ respectă formatul specific.

1. **Clasa ProductBLL**

Această clasă conține anumite metode care realizează operații asupra tabelei de produse apelând

metodele de insert, update și delete din clasa ProductDAO.

1. **OrderTableBLL**

Această clasă conține anumite metode care realizează operații asupra tabelei de produse apelând

metodele de insert și findAll din clasa OrderTableDAO. În metoda *insertOrder()* înainte să se apeleze metoda de inserare a unei comenzi în tabelă se verifică cu ajutorul metodei *validate()* din clasa OrderValidator dacă comanda care dorește a fi introdusă în tabelă are datele despre IdProdus, IdClient și cantitate valide. În același timp în această clasă am implementat metoda pentru generarea unei facturi sub formă de fișier text în momentul realizării comenzii. Metoda *bill()* primește ca argumente comanda de inserat și id-ul acesteia care se generează Random. Descrierea metodei este următoarea:

*public void bill(OrderTable orderTable, int idOrder) {*

*try {*

*File myFile = new File("bill" + idOrder + orderTable.getIdClient() + orderTable.getIdProduct() + ".txt");*

*FileWriter myWriter = new FileWriter("bill" + idOrder + orderTable.getIdClient() + orderTable.getIdProduct() + ".txt");*

*ClientBLL clientBLL = new ClientBLL();*

*ProductBLL productBLL = new ProductBLL();*

*Client client = clientBLL.findClientById(orderTable.getIdClient());*

*Product product = productBLL.findByIdProdus(orderTable.getIdProduct());*

*myWriter.write(" The Bill \n");*

*myWriter.write(" Client details\n" + "Id Client: " + orderTable.getIdClient() + "\nName: " + client.getName() + "\nAdress: " + client.getAddress() + "\nEmail: " + client.getEmail() + "\nAge: " + client.getAge() + "\n");*

*myWriter.write(" Product details\n" + "Id Product: " + orderTable.getIdProduct() + "\nName: " + product.getNume() + "\nQuantity: " + orderTable.getCantitate() + "\nPrice: " + product.getPret() + "\n");*

*myWriter.write(" TOTAL price is: " + orderTable.getCantitate() \* product.getPret());*

*myWriter.close();*

*} catch (IOException e) {*

*e.printStackTrace();*

*}*

*}*

1. **Pachetul validators**

Acest pachet este un subpachet al pachetului BLL și conține clasele ,,ClientAgeValidator”,

,,EmailValidator’’, ,,OrderValidator’’ și interfața Validator care definește metoda *validate()* implementată în aceste clase. Acesti validatori sunt apelați în clasele de Bll specifice fiecărei tabele.

## **Pachetul Connection**

1. **Clasa ConnectionFactory**

Această clasă conține numele driver-ului (inițializat prin reflecție), locația bazei de date, și

utilizatorul și parola pentru accesarea serverului MySQL. Conexiunea la baza de date va fi plasată într-un Obiect ,,Singleton’’. Clasa totodată, conține metode pentru crearea unei conexiuni, obținerea unei conexiuni active și închiderea unei conexiuni, a unei declarații sau a unui rezultat.

## **Pachetul DAO**

1. **Clasa AbstractDAO**

Această clasă definește operațiile comune asupra tabelelor: insert, update, delete, findById și

findAll. Clasa este definită pe tipul generic T, iar operațiile la fel. Metodele sunt implementate folosind Generice si utilizând Reflection. Metoda *insert()* primește ca parametru un obiectul de tip T care trebuie inserat și returnează 1 dacă inserarea s-a realizat cu succes și 0 altfel. Descrierea metodei este următoarea:

*public int insert(T obj) {*

*Connection connection = null;*

*PreparedStatement insertStatement = null;*

*String query = "";*

*query = "INSERT INTO " + type.getSimpleName() + " VALUES(";*

*int insertedId = -1;*

*int l = type.getDeclaredFields().length;*

*for (int k = 0; k < l; k++) {*

*if (k != l - 1) {*

*query += "?,";*

*} else {*

*query += "?)";*

*}*

*}*

*try {*

*int nr = 1;*

*connection = ConnectionFactory.getConnection();*

*insertStatement = connection.prepareStatement(query, Statement.RETURN\_GENERATED\_KEYS);*

*for (Field field : obj.getClass().getDeclaredFields()) {*

*PropertyDescriptor propertyeDescriptor = new PropertyDescriptor(field.getName(), type);*

*Method method = propertyeDescriptor.getReadMethod();*

*insertStatement.setObject(nr++, method.invoke(obj));*

*}*

*assert insertStatement != null; insertedId = insertStatement.executeUpdate();*

*} catch (SQLException | IntrospectionException | IllegalAccessException | InvocationTargetException throwables) {*

*LOGGER.log(Level.WARNING, "DAO:insert " + throwables.getMessage());*

*} finally {*

*ConnectionFactory.close(insertStatement);ConnectionFactory.close(connection);*

*}*

*return insertedId;*

*}*

1. **Clasele ClientDAO, ProductDAO și OrderTableDAO**

Aceste clase extend AbstractDAO deci implementează ca atare metodele din această clasă.

Accesarea acestor metode se realizează prin apelarea constructorului *super()*, acesta fiind constructorul clasei părinte AbstractDAO.

## **Pachetul Model**

1. **Clasele Client, Product și OrderTable**

Aceste clase au aceași denumire ca și tabelele din baza de date, deorece este nevoie de acest lucru

pentru a extrage elementele din baza de date. Aceste clase trebuie să aibă denumirea și tipul câmpurilor exact ca și coloanele din tabelul corespunzător. Clasele trebuie să aibă de asemenea constructori, gettere și settere.

## **Pachetul Presentation**

1. **Clasa TableView**

Această clasă utilizează tehnica de Reflection și creează o metodă *retrieveProprieties()* care

primește o listă de obiecte și generează antetul tabelului prin extragerea cu ajutorul reflexiei a proprietăților obiectului și apoi populează tabelul cu valorile elementelor din listă. Descrierea metodei este următoarea:

*public static <T> TableModel retrieveProperties(List<T> object) {*

*int nr = 0;*

*for (Object object1 : object) {*

*nr = 0;*

*for (Field ignored : object1.getClass().getDeclaredFields()) {*

*nr++;*

*}*

*}*

*Object[] columnNames = new Object[nr];*

*Object[][] data = new Object[100][nr];*

*try {*

*for (int i = 0; i < object.size(); i++) {*

*Object object1 = object.get(i);*

*int counter = 0;*

*for (Field field : object1.getClass().getDeclaredFields()) {*

*field.setAccessible(true);*

*Object value;*

*try {*

*columnNames[counter] = field.getName();*

*value = field.get(object1);*

*data[i][counter] = value;*

*counter++;*

*}catch (NullPointerException exception) {*

*exception.printStackTrace();*

*}*

*}*

*}*

*}catch(IllegalArgumentException | IllegalAccessException e){*

*e.printStackTrace();*

*}*

*TableModel tableModel = new DefaultTableModel(data, columnNames);*

*return tableModel;*

*}*

1. Acest pachet implementează totodată și logica interfeței cu utilizatorul a aplicației. Aceasta este formată din trei

ferestre:

* + Fereastra pentru Client
  + Fereastra pentru Produs
  + Fereastra pentru Comandă

Implementarea acestor ferestre se realizează în subpachetele View și Controller.

**Pachetul View**

În acest pachet se implementează interfața cu utilizatorul a aplicației. Aceasta este formată din trei ferestre:

1. Fereastra pentru Client

Această fereastră este formată din 4 zone:

* Zona de *Insert Client* unde se introduc datele aferente pentru inserarea unui client în baza de date.
* Zona de *Update Client* unde se introduc datele pentru actualizarea unui anumit client specificat prin IdClient.
* Zona de *Delete Client* unde se introduce id-ul clientului care se dorește a fi șters din baza de date.
* Zona de afișare clienți în tabel.

O imagine care conține masă

Descriere generată automat

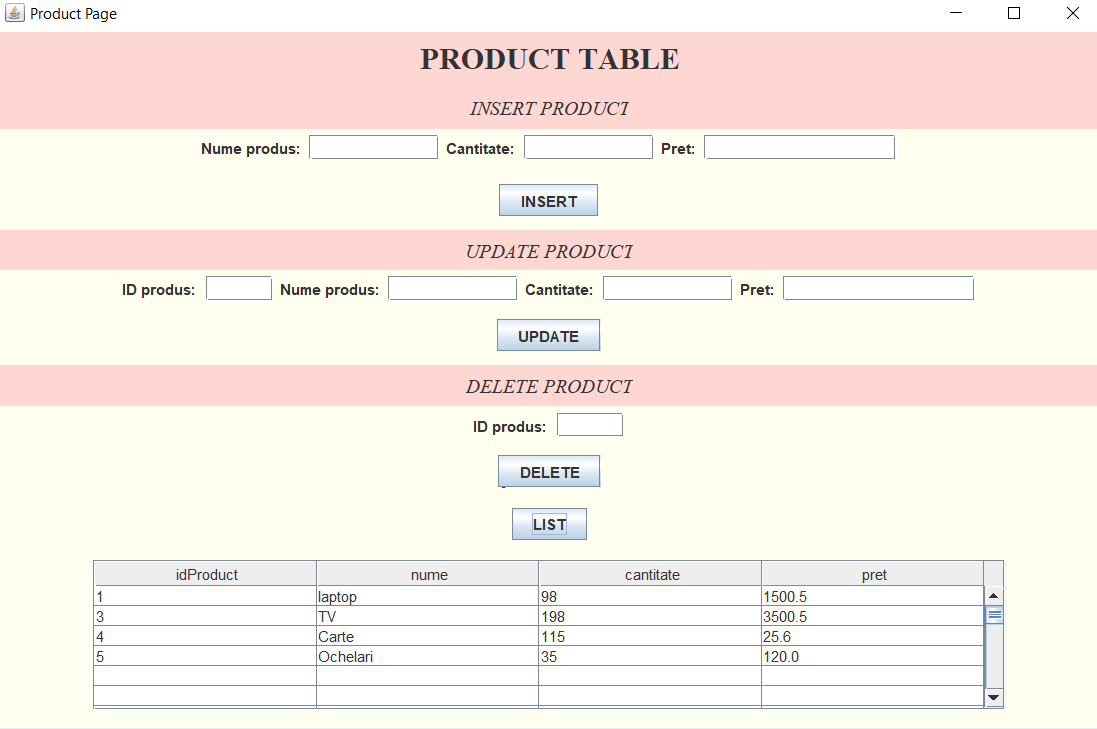
În cazul în care utilizatorul introduce în momentul inserării unui client un Email greșit sau un număr pentru

Age înafara intervalului 7-70 se vor afișa anumite mesaje de eroare.

1. Fereastra pentru Produs

Această fereastră este formată din 4 zone:

* Zona de *Insert Product* unde se introduc datele aferente pentru inserarea unui produs în baza de date.
* Zona de *Update Product* unde se introduc datele pentru actualizarea unui anumit client specificat prin IdProdus.
* Zona de *Delete Product* unde se introduce id-ul produsului care se dorește a fi șters din baza de date.
* Zona de afișare produse în tabel.



1. Fereastra pentru Order

Aceasta conține o zonă pentru introducerea datelor specifice unei comenzi: IdClient, IdProduct și

Cantitatea dorită pentru a fi comandată. Totodată există și o zonă pentru afișarea comenzilor deja existente.

O imagine care conține masă

Descriere generată automat

În cazul în care se introduce un IdClient sau IdProdus inexistent se va afișa un mesaj de eroare. În același timp dacă se introduce o cantitate care nu există pentru un anumit produs se va semnala acest lucru printr-un mesaj de eroare.

**Pachetul Controller**

Această pachet descrie Clasele care implementează interfața *ActionListener*, deci trebuie să implementeze metoda *actionPerformed().* În această metodă am descris logica care se întâmplă la apăsarea fiecărui buton din ferestrele interfeței de către utilizator. În momentul apăsării unui anumit buton se preiau datele din interfața grafică și se crează un nou obiect de tipul *Client, Product* sau *OrderTable* care deservește ca și câmp pentru fiecare metodă de insert, update și delete.

# **Rezultate**

Pentru testarea aplicației am folosit următoarele date de intrare pentru realizarea unei comenzi:

|  |  |  |
| --- | --- | --- |
| Comanda1 | Comanda2 | Comanda3 |
| IdClient = 2  IdProduct = 1  Cantitate = 2 | IdClient = 1  IdProduct = 4  Cantitate = 5 | IdClient = 5  IdProduct = 3  Cantitate = 2 |

Generarea facturii pentru Comanda1 este următoarea:

*The Bill*

*Client details*

*Id Client: 2*

*Name: Maria*

*Adress: Cluj Napoca*

*Email: maria@yahoo.com*

*Age: 20*

*Product details*

*Id Product: 1*

*Name: laptop*

*Quantity: 2*

*Price: 1500.5*

*TOTAL price is: 3001.0*

# **Concluzii**

În concluzie acest proiect mi-a aprofundat cunoștințele despre limbajul Java. Am învățat să lucrez cu

tipuri Generice și cu tehnica de Reflection pe care nu le-am mai folosit până acum. Mi-am aprofundat totodată și cunoștințele pe care le-am dobândit până acum.

Ca dezvoltări ulterioare aș propune implementarea verificării cu chei străine în momentul ștergerii unui produs. Adică, să se verifice dacă este în cadrul unei comenzi, dacă este să se afișeze un mesaj de eroare și să nu se realizeze ștergerea produsului.

# **Bibliografie**

1. <https://app.diagrams.net/>
2. Stack Overflow
3. Youtube
4. Assignment\_3\_Suport\_Presentation
5. <https://gitlab.com/utcn_dsrl/pt-reflection-example>
6. <https://gitlab.com/utcn_dsrl/pt-layered-architecture>
7. <https://www.baeldung.com/javadoc>
8. <https://dev.mysql.com/doc/workbench/en/wb-admin-export-import-management.html>