

# DOCUMENTATIE

## TEMA 3

### Orders Management

**NUME STUDENT:** Jişă Diana-Maria  
**GRUPA:** 30224

# CUPRINS

1. Obiectivul temei .....	3
2. Analiza problemei, modelare, scenarii, cazuri de utilizare.....	4
3. Proiectare .....	5
4. Implementare .....	6
5. Rezultate .....	9
6. Concluzii.....	11
7. Bibliografie.....	12

# 1. Obiectivul temei

Obiectivul acestei teme de laborator este de a propune, proiecta și implementa o aplicație de gestionare a comenzilor pentru un depozit, utilizând baze de date relaționale pentru stocarea datelor referitoare la produse, clienți și comenzi. Aplicația trebuie să fie proiectată în conformitate cu modelul arhitectural stratificat și să utilizeze cel puțin următoarele clase: clase de model - reprezentând modelele de date ale aplicației, clase de logică de afaceri - conținând logica aplicației, clase de prezentare - clase legate de interfața grafică și clase de acces la date - clase care conțin accesul la baza de date. Aplicația de gestiune a comenzilor va avea în plus, o interfață grafică prietenoasă pentru utilizator, care să faciliteze accesul și utilizarea într-un mod cât mai intuitiv. Scopul final al acestei teme este de a oferi utilizatorului o experiență cât mai eficientă în procesul de gestionare a comenzilor.

## **Obiective secundare:**

- Analiza problemei și identificarea cerințelor funcționale și non-funcționale ale aplicației de gestionare a comenzilor (Cap. 2)
- Proiectarea soluției utilizând paradigma OOP, diagrame UML de clase și pachete, și definirea structurilor de date și a interfețelor necesare. (Cap. 3)
- Implementarea aplicației de gestionare a comenzilor și a interfeței grafice a utilizatorului (Cap. 4)
- Prezentarea scenariilor de testare (Cap. 5)
- Concluzii și sugestii pentru îmbunătățiri ulterioare (Cap. 6)
- Referințe bibliografice (Cap. 7)

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

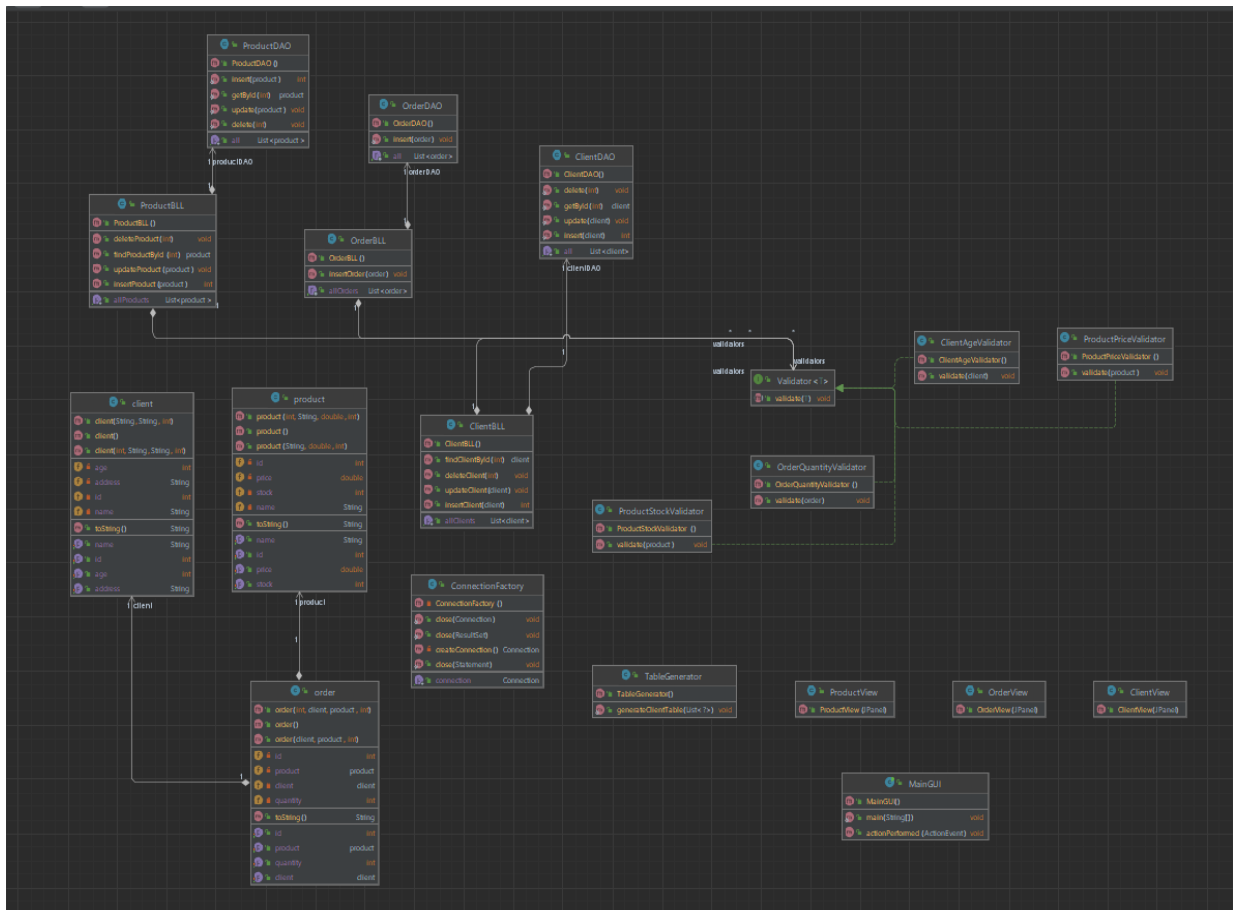
Scopul principal al acestui proiect este de a simplifica și de a eficientiza procesul de gestionare a unui depozit, prin intermediul unei aplicații de gestionare a comenzilor. Prin această soluție software, se va permite crearea și urmărirea comenzilor, precum și gestionarea stocului de produse și a clienților, ceea ce va duce la o îmbunătățire semnificativă a eficienței companiei. În plus, crearea unei baze de date relationale pentru a stoca informații legate de clienți, produse și comenzi va permite o gestionare mai precisă și mai ușoară a datelor.

După accesarea interfeței programului, utilizatorul trebuie să selecteze operația pe care dorește să o realizeze. Pentru a evita erorile, este important ca toate câmpurile să fie completate corespunzător. În cazul în care un câmp a rămas necompletat, va fi generată o excepție. De asemenea, în cazul în care toate câmpurile au fost completate, dar una dintre informații nu este validă, o fereastră de avertizare va fi afișată pentru a indica faptul că operația nu a putut fi realizată.

În general, acest program de gestionare a comenzilor poate fi utilizat cu succes în orice scenariu, iar utilizatorii sunt încurajați să își ajusteze parametrii în funcție de nevoile lor specifice. Prin folosirea acestei aplicații, companiile vor putea să-și optimizeze fluxul de lucru, ceea ce va duce la o creștere semnificativă a eficienței și la o îmbunătățire a experienței clienților și angajaților.

### 3. Proiectare

#### Diagrama UML



Acest program utilizează o arhitectură stratificată, unde straturile sunt împărțite în Presentation Layer, Business Layer și Data Access Layer, iar pachetele sunt organizate astfel încât să aibă următoarele relații: Presentation -> Business -> Acces la Date -> Bază de date MySQL. În plus, fiecare dintre aceste straturi are propriile modele corespunzătoare.

Presentation Layer - conține clasele care definesc interfața utilizatorului

Business Layer - conține clasele care încapsulează logica aplicației

Data Access Layer - conține clasele care conțin interogările și conexiunea la baza de date

Model - conține clasele mapate la tabelul bazei de date

În ceea ce privește interacțiunea cu structurile de date, am folosit în principal obiecte de tipul **List<object>** și **ArrayList** pentru a putea accesa cu ușurință, dar și pentru a prelucra informațiile stocate în baza de date.

## 4. Implementare

Pentru a implementa această aplicație de gestionare a comenzilor, am împărțit codul în cinci pachete, fiecare conținând mai multe clase: **BusinessLogic**, **Connection**, **DataAccess**, **Model** și **Presentation**.

### Connection package

Acest pachet conține o singură clasă numită “ConnectionFactory”, care este o clasă utilitară pentru gestionarea conexiunii la o bază de date MySQL. Aceasta definește metode pentru crearea și închiderea conexiunilor, precum și pentru închiderea declarațiilor și a seturilor de rezultate. Clasa utilizează driverul JDBC MySQL și are parametrii de configurare setați în variabile constante.

### Model package

Acest pachet conține trei clase separate: “client”, “product” și “order”. Cele trei clase din acest pachet au fost proiectate cu scopul de a oferi o abordare simplă și intuitivă pentru gestionarea datelor. Pentru a realiza acest lucru, fiecare clasă are atributele necesare declarate, iar prin intermediul metodelor get și set permite utilizatorilor să acceseze și să modifice datele obținute. În plus, fiecare clasă are și metode de afișare care facilitează vizualizarea informațiilor despre obiectele reprezentate. Prin aceste caracteristici, aceste clase sunt esențiale pentru dezvoltarea aplicațiilor de nivel superior. De exemplu, clasa “client” conține atribute precum numele, adresa și vârsta clientului, iar clasa “product” conține atribute precum numele, prețul și stocul disponibil.

### DataAccess package

Acest pachet conține patru clase “ClientDAO”, “ProductDAO”, “OrderDAO”, “TableGenerator”. Primele trei clase sunt responsabile pentru interacțiunea cu baza de date pentru obiectele client, product respectiv order.

Clasa “ClientDAO” include metode pentru a obține toți clienții, pentru a obține un client specific după ID, pentru a insera un client nou în baza de date, pentru a actualiza un client existent în baza de date și pentru a șterge un client din baza de date. Aceste metode folosesc conexiunea la baza de date și interogările SQL pentru a executa operațiile dorite. Clasa utilizează și o clasă de conexiune (ConnectionFactory) pentru a obține o conexiune la baza de date. Aceleași metode sunt implementate și în clasa “ProductDAO”, dar bineînțeles pentru obiecte de tip product.

Clasa “OrderDAO” are implementate doar două metode: getAll() și insert(). Metoda getAll() returnează o listă de obiecte order, care reprezintă toate comenzile existente în baza de date. Metoda insert() inserează o nouă comandă în baza de date și actualizează stocul produsului comandat.

Clasa TableGenerator utilizează Reflection pentru a genera un tabel dinamic care poate afișa date pentru orice clasă. Acest lucru se realizează prin accesarea câmpurilor unei instanțe a clasei folosind metoda getDeclaredFields(), care returnează un tablou de obiecte Field. Apoi, numele câmpurilor sunt extrase și stocate într-un tablou de șiruri de caractere pentru a fi utilizate ca denumiri de coloană în tabel. Clasa utilizează apoi aceste câmpuri pentru a accesa și a extrage valorile lor din obiectele din lista de date și pentru a le stoca într-un tablou bidimensional de obiecte..

Prin utilizarea Reflection, clasa TableGenerator poate fi utilizată pentru a genera automat tabele dinamice pentru orice clasă, fără a fi necesară scrierea codului pentru fiecare clasă în parte.

```
public static void generateClientTable(List<?> dataList) {
    if (dataList.isEmpty()) {
        JOptionPane.showMessageDialog( parentComponent: null, message: "No data to display.");
        return;
    }

    Class<?> clazz = dataList.get(0).getClass();
    Field[] fields = clazz.getDeclaredFields();
    String[] columnNames = new String[fields.length];
    Object[][] rowData = new Object[dataList.size()][fields.length];

    for (int i = 0; i < fields.length; i++) {
        columnNames[i] = fields[i].getName();
    }

    for (int i = 0; i < dataList.size(); i++) {
        Object dataObj = dataList.get(i);
        for (int j = 0; j < fields.length; j++) {
            fields[j].setAccessible(true);
            try {
                rowData[i][j] = fields[j].get(dataObj);
            } catch (IllegalAccessException e) {
                e.printStackTrace();
            }
        }
    }

    JTable table = new JTable(rowData, columnNames);
    JScrollPane scrollPane = new JScrollPane(table);
    scrollPane.setPreferredSize(new Dimension( width: 600, height: 400));
    JOptionPane.showMessageDialog( parentComponent: null, scrollPane, title: "JTable", JOptionPane.PLAIN_MESSAGE);
}
```

## **BusinessLogic package**

Acest pachet contine trei clase separate “ClientBLL”, “ProductBLL” si “OrderBLL”. Acestea implementeaza logica de afaceri a aplicației de gestionare a comenzilor. Aceste clase utilizeaza metodele din DataAccess pentru a accesa și modifica datele din bază de date

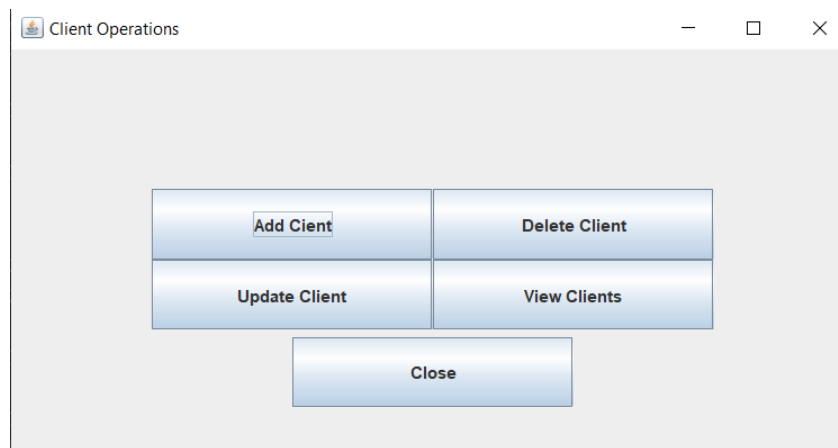
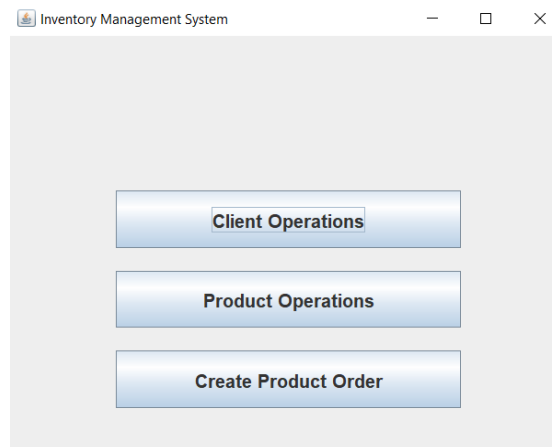
## **Presentation package**

Acest pachet contine patru clase diferite, una pentru fiecare fereastră, ce urmeaza sa apara pe ecran atunci cand aplicatia de gestionare a cozilor este rulata.

Clasa “MainGUI” reprezintă interfața grafică principală a aplicației, prin care se poate accesa meniul de operațiuni cu clienți, produse sau comenzi.

Clasa “ClientView” reprezintă o fereastră a interfeței grafice prin care se pot efectua operații asupra entităților de tip client. Această clasă conține metode pentru afișarea, adăugarea, ștergerea și actualizarea informațiilor despre clienți. Aceleași metode le conține și clasa “ProductView”, clasa care reprezintă o altă fereastră a interfeței prin care se pot realiza operații asupra produselor.

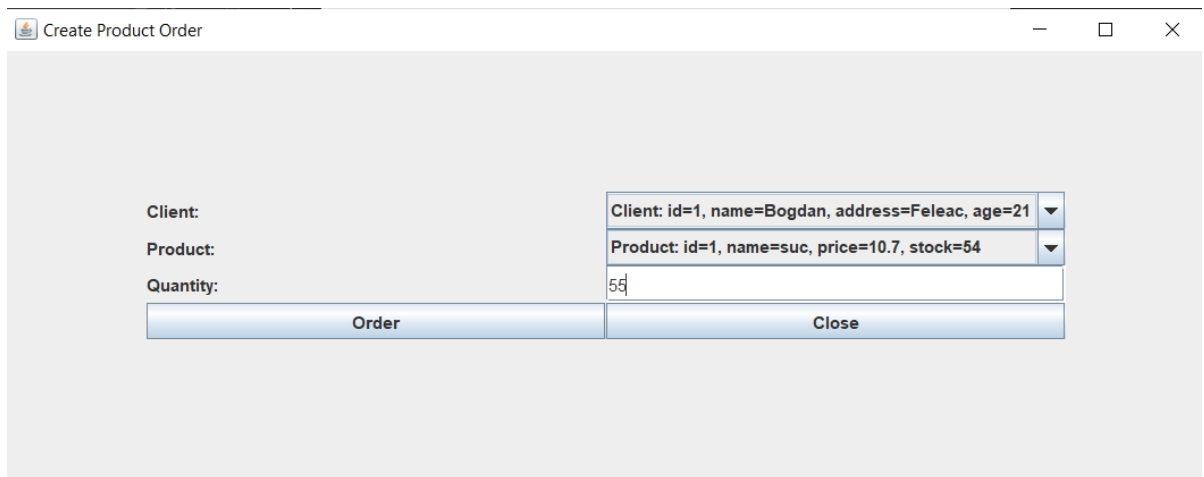
Clasa “OrderView” reprezintă o fereastră a interfeței grafice prin care se pot crea comenzi, prin asocierea unui client cu una sau mai multe produse. Această clasă conține metode pentru afișarea informațiilor despre clienți și produse, selectarea acestora și crearea comenzii.





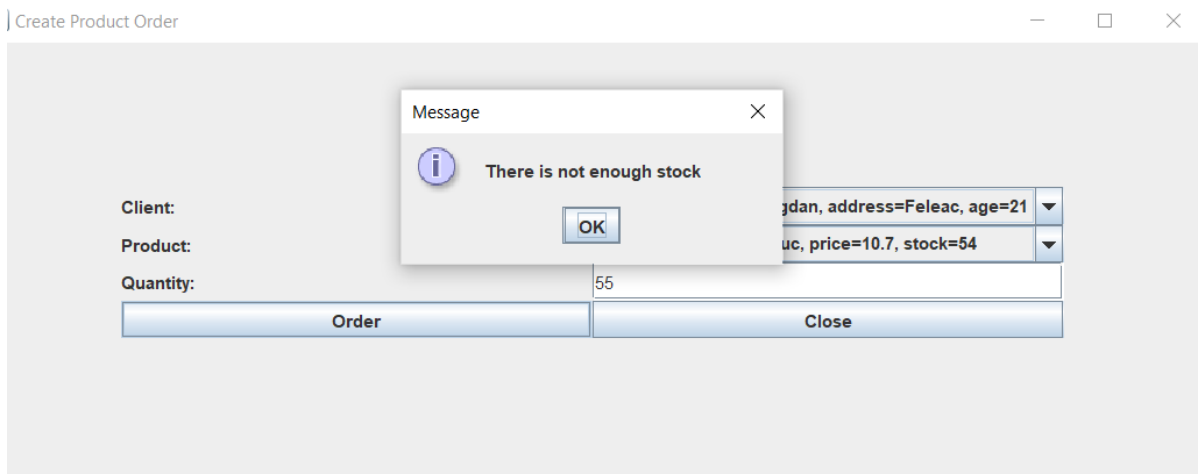
## 5. Rezultate

Am efectuat teste multiple ale acestui program și am constatat că a furnizat rezultate corecte pentru fiecare dintre ele. Mai jos este prezentat un exemplu de plasare a unei comenzi, în care utilizatorul selectează un client și un produs, iar apoi introduce cantitatea dorită. După plasarea comenzii, utilizatorul poate observa că numărul de produse disponibile în stoc a fost actualizat și decrementat conform cantității comandate. De asemenea, după plasarea comenzii, aceasta este introdusă în tabelul corespunzător comenzilor, astfel încât utilizatorul poate accesa informații relevante despre aceasta în orice moment.

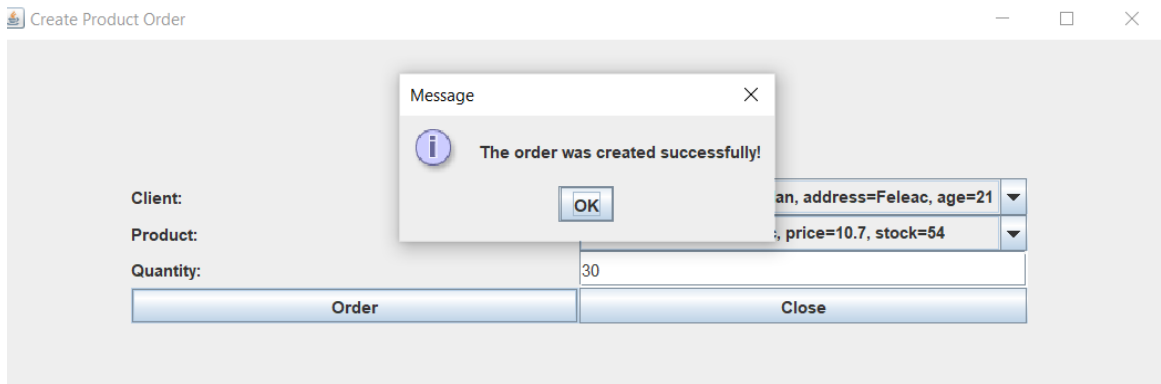


The screenshot shows a window titled "Create Product Order". It contains a form with the following elements:

- Client:** A dropdown menu showing "Client: id=1, name=Bogdan, address=Feleac, age=21".
- Product:** A dropdown menu showing "Product: id=1, name=suc, price=10.7, stock=54".
- Quantity:** A text input field containing the value "54".
- Buttons:** Two buttons at the bottom, "Order" and "Close".



This screenshot shows the same "Create Product Order" window, but with a "Message" dialog box displayed in the center. The dialog box contains an information icon and the text "There is not enough stock", with an "OK" button at the bottom. The background form is partially visible, showing the same dropdowns and input field as the previous screenshot, but the "Quantity" field now contains the value "55".



### Tabelul de comenzi actualizat

JTable

id	client	product	quantity
1	Client: id=1, name=null, add.	Product: id=2, name=null, p.	2
2	Client: id=1, name=null, add.	Product: id=1, name=null, p.	55
3	Client: id=1, name=null, add.	Product: id=1, name=null, p.	30

### Stocul decrementat

JTable

id	name	price	stock
1	suc	10.7	24
2	paine	4.5	8

După finalizarea testelor, am constatat că aplicația de gestionare a comenzilor funcționează corespunzător. Acest lucru demonstrează importanța testării în dezvoltarea unei aplicații fiabile și asigură faptul că aceasta poate fi utilizată eficient și intuitiv de către utilizatori

## 6. Concluzii

În concluzie, dezvoltarea aplicației de management a comenzilor pentru un depozit a fost o experiență captivantă din punct de vedere tehnic și mi-a oferit oportunitatea de a învăța despre arhitectura stratificată și utilizarea bazelor de date relaționale. Aplicația a fost concepută în conformitate cu modelul de arhitectură stratificată și a utilizat clasele de model, clasele de logică de afaceri, clasele de prezentare și clasele de acces la date. Utilizarea bazelor de date relaționale a permis stocarea eficientă a informațiilor despre produse, clienți și comenzi.

În viitor, există posibilitatea de a îmbunătăți această aplicație prin adăugarea unor funcționalități suplimentare, cum ar fi implementarea unui sistem de plată online sau integrarea cu un sistem de livrare. În plus, ar putea fi dezvoltate și funcții de raportare pentru a permite managerilor să urmărească mai ușor starea comenzilor și să ia decizii informate.

În general, dezvoltarea acestei aplicații a fost o experiență benefică în dezvoltarea abilităților mele tehnice și a cunoștințelor despre arhitectura stratificată și utilizarea bazelor de date relaționale. Acest proiect mi-a oferit, de asemenea, oportunitatea de a obține o experiență practică în proiectarea și dezvoltarea de aplicații de management.

## 7. Bibliografie

1. Reflection in Java( <https://www.geeksforgeeks.org/reflection-in-java/> )
2. Using Java Reflection ( <https://www.oracle.com/technical-resources/articles/java/javareflection.html#:~:text=Reflection%20is%20a%20feature%20in,its%20members%20and%20display%20them.> )
3. Layered Architecture ( <https://www.baeldung.com/cs/layered-architecture> )
4. MySQL Java ( <https://zetcode.com/db/mysqljava/> )
5. How to insert data into a MySQL  
( <https://www.tutorialspoint.com/how-to-insert-data-into-a-mysql-database-with-java> )  
( <https://stackoverflow.com/questions/59147960/how-to-insert-into-mysql-database-with-java> )