

**DOCUMENTAȚIE**  
**TEHNICI DE PROGRAMARE**  
**TEMA 3 – Aplicație de gestionare a comenzilor**

NUME STUDENT: ROTARIU LAURA-ALEXANDRA  
GRUPA: 30224-2

# CUPRINS

1. Obiectivul temei .....	3
2. Analiza problemei, modelare, scenarii, cazuri de utilizare.....	3
3. Proiectare.....	6
4. Implementare.....	8
5. Rezultate.....	12
6. Concluzii .....	15
7. Bibliografie.....	16

## 1. Obiectivul temei

Obiectivul principal al temei este de a proiecta și implementa o aplicație de gestionare a comenzilor realizate de clienți pentru un depozit. Trebuie folosite baze de date relaționale pentru a stoca produsele, clienții și comenzile. Aplicația trebuie proiectată conform modelului „layered architecture”.

Obiectivele secundare ale temei (pașii care trebuie urmați pentru îndeplinirea obiectivului principal):

- Analizarea problemei și identificarea funcționalităților de bază care vor fi oferite utilizatorilor prin intermediul interfeței grafice simple: vizualizarea clienților, adăugarea clienților, actualizarea clienților, ștergerea clienților, vizualizarea produselor, adăugarea produselor, actualizarea produselor, ștergerea produselor, realizarea de comenzi și vizualizarea comenzilor efectuate. Acest pas va fi descris în secțiunea Analiza problemei, modelare, scenarii, cazuri de utilizare.

- Proiectarea aplicației pentru simulare: definirea claselor și a interfețelor necesare, implementarea structurilor de date care stochează clienții, produsele și comenzile, stabilirea logicii pentru vizualizare, ștergere, adăugare, actualizare de categorii. Este importantă organizarea codului în clase și pachete în funcție de utilitatea acestuia. Acești pași vor fi descriși în secțiunile Proiectare și Implementare.

- Implementarea aplicației pentru simulare: scrierea codului pentru clasele și interfețele necesare proiectate la punctul anterior, implementarea pentru efectuarea operațiilor asupra clienților, produselor și comenzilor, scrierea codului Java necesar pentru a introduce datele de către utilizator în interfața grafică. Este necesară utilizarea unor componente standard în interfață, precum panouri, butoane, casete de text, etichete, tabele pentru a permite utilizatorilor să introducă datele necesare pentru inserare și actualizare, să șteargă sau să vizualizeze datele. Acest pas va fi descris în secțiunea Implementare.

- Testarea pentru a vedea dacă aplicația funcționează corect. Se va testa pe mai multe seturi de date și se vor urmări rezultatele pentru a identifica erori sau îmbunătățiri posibile. Acest pas va fi descris în secțiunea Rezultate.

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

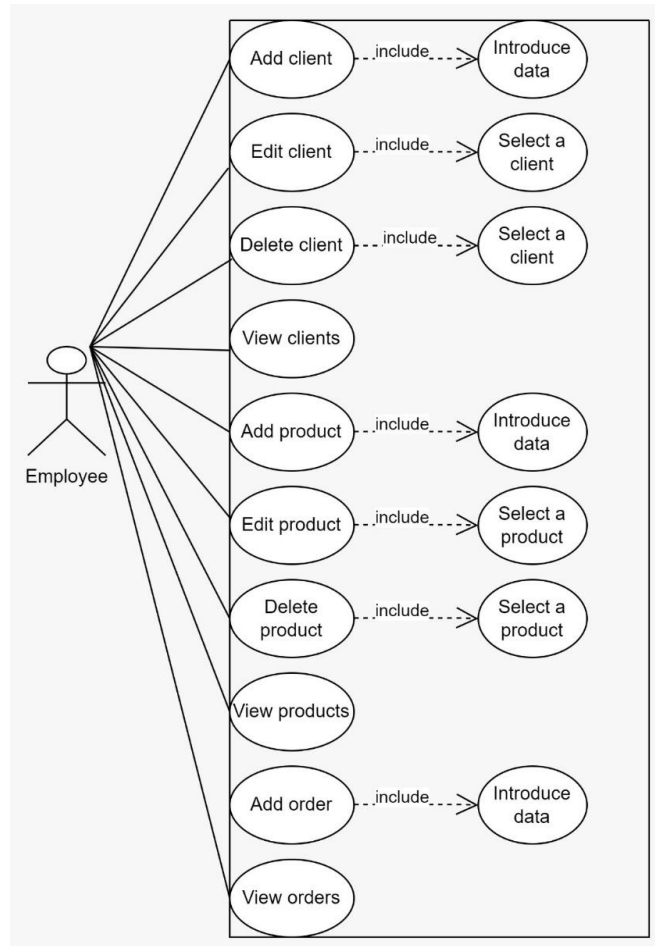
### • Cerințe funcționale:

- Aplicația de gestionare a comenzilor trebuie să permită utilizatorului să adauge un client.
- Aplicația de gestionare a comenzilor trebuie să permită utilizatorului să editeze un client.
- Aplicația de gestionare a comenzilor trebuie să permită utilizatorului să șteargă un client.
- Aplicația de gestionare a comenzilor trebuie să permită utilizatorului să vizualizeze clienții.
- Aplicația de gestionare a comenzilor trebuie să permită utilizatorului să adauge un produs.
- Aplicația de gestionare a comenzilor trebuie să permită utilizatorului să editeze un produs.
- Aplicația de gestionare a comenzilor trebuie să permită utilizatorului să șteargă un produs.
- Aplicația de gestionare a comenzilor trebuie să permită utilizatorului să vizualizeze produsele.
- Aplicația de gestionare a comenzilor trebuie să permită utilizatorului să adauge un produs.
- Aplicația de gestionare a comenzilor trebuie să permită utilizatorului să vizualizeze produsele.
- Aplicația de gestionare a comenzilor trebuie să permită utilizatorului să adauge o comandă.
- Aplicația de gestionare a comenzilor trebuie să permită utilizatorului să vizualizeze comenzile.

### • Cerințe non-funcționale:

- Aplicația de gestionare a comenzilor trebuie să fie intuitivă și ușor de folosit de către utilizator.
- Aplicația de gestionare a comenzilor trebuie să realizeze operațiile precizate mai sus și să afișeze datele într-un timp scurt.
- Aplicația de gestionare a comenzilor trebuie să funcționeze indiferent de sistemul de operare folosit.

- **Cazuri de utilizare:**



- **Descrieri use-case-uri:**

**Use case:** Adăugare client

Actor principal: angajat

Scenariul principal cu succes:

1. Utilizatorul selectează operația de adăugare a unui client.
2. Aplicația va deschide o nouă fereastră în care detaliile clientului trebuie inserate.
3. Angajatul introduce numele clientului, adresa și adresa de email.
4. Utilizatorul apasă butonul de salvare a detaliilor.
5. Aplicația stochează datele introduse în baza de date și afișează un mesaj de confirmare.

Scenariul alternativ: Datele necesare nu sunt introduse sau sunt introduse incorect

- Utilizatorul nu introduce datele necesare pentru a adăuga un client sau introduce date invalide.
- Se afișează un mesaj și se cere utilizatorului să introducă datele necesare și scenariul se va întoarce la pasul 3.

**Use case:** Adăugare produs

Actor principal: angajat

Scenariul principal cu succes:

1. Utilizatorul selectează operația de adăugare a unui produs.
2. Aplicația va deschide o nouă fereastră în care detaliile produsului trebuie inserate.
3. Angajatul introduce numele produsului, prețul și stocul.

4. Utilizatorul apasă butonul de salvare a detaliilor.
5. Aplicația stochează datele introduse în baza de date și afișează un mesaj de confirmare.

Scenariul alternativ: Datele necesare nu sunt introduse sau sunt introduse incorect

- Utilizatorul nu introduce datele necesare pentru a adăuga un produs sau introduce date invalide (stoc sau preț mai mic decât 1).
- Se afișează un mesaj și se cere utilizatorului să introducă datele necesare și scenariul se va întoarce la pasul 3.

**Use case:** Adăugare comandă

Actor principal: angajat

Scenariul principal cu succes:

1. Utilizatorul selectează un client existent, un produs existent și introduce cantitatea dorită.
2. Utilizatorul selectează operația de adăugare a unui comenzi.
3. Aplicația stochează datele introduse în baza de date și afișează un mesaj de confirmare.

Scenariul alternativ: Datele necesare nu sunt introduse sau sunt introduse incorect

- Utilizatorul nu introduce datele necesare pentru a adăuga o comandă sau introduce date invalide (cantitatea mai mică decât 1).
- Se afișează un mesaj și se cere utilizatorului să introducă datele necesare și scenariul se va întoarce la pasul 1.

**Use case:** Editare client

Actor principal: angajat

Scenariul principal cu succes:

1. Angajatul selectează un client existent.
2. Utilizatorul selectează operația de editare a unui client.
3. Aplicația va deschide o nouă fereastră în care sunt detaliile actuale ale clientului și în care utilizatorul va introduce noile detalii ale clientului și apasă butonul de salvare.
4. Aplicația stochează datele introduse în baza de date și afișează un mesaj de confirmare.

Scenariul alternativ: Nu este selectat un client pentru editare sau noile date introduse sunt invalide

- Utilizatorul nu selectează un client existent din tabel sau introduce noi date invalide.
- Se afișează un mesaj și se cere utilizatorului să selecteze un client sau să introducă datele necesare și scenariul se întoarce la pasul 1.

**Use case:** Editare produs

Actor principal: angajat

Scenariul principal cu succes:

1. Angajatul selectează un produs existent.
2. Utilizatorul selectează operația de editare a unui produs.
3. Aplicația va deschide o nouă fereastră în care sunt detaliile actuale ale produsului și în care utilizatorul va introduce noile detalii ale produsului și apasă butonul de salvare.
4. Aplicația stochează datele introduse în baza de date și afișează un mesaj de confirmare.

Scenariul alternativ: Nu este selectat un produs pentru editare sau noile date introduse sunt invalide

- Utilizatorul nu selectează un produs existent din tabel sau introduce noi date invalide.
- Se afișează un mesaj și se cere utilizatorului să selecteze un produs sau să introducă datele scenariul se întoarce la pasul 1.

**Use case:** Ștergere client

Actor principal: angajat

Scenariul principal cu succes:

1. Angajatul selectează un client existent.
2. Utilizatorul selectează operația de ștergere a unui client.
3. Aplicația șterge clientul din baza de date, afișează tabelul actualizat și afișează un mesaj de confirmare.

Scenariul alternativ: Nu este selectat un client pentru ștergere sau este selectat un client care a realizat o comandă.

- Utilizatorul nu selectează un client existent din tabel.
- Se afișează un mesaj și se cere utilizatorului să selecteze un client și scenariul se întoarce la pasul 1.

**Use case:** Ștergere produs

Actor principal: angajat

Scenariul principal cu succes:

1. Angajatul selectează un produs existent.
2. Utilizatorul selectează operația de ștergere a unui produs.
3. Aplicația șterge produsul din baza de date și afișează un mesaj de confirmare.

Scenariul alternativ: Nu este selectat un produs pentru ștergere sau este selectat un produs care a fost comandat.

- Utilizatorul nu selectează un produs existent din tabel.
- Se afișează un mesaj și se cere utilizatorului să selecteze un produs și scenariul se întoarce la pasul 1.

**Use case:** Vizualizare clienți

Actor principal: angajat

Scenariul principal cu succes:

1. Utilizatorul selectează operația de vizualizare a clienților.
2. Aplicația afișează un tabel cu toți clienții existenți în baza de date.

**Use case:** Vizualizare produse

Actor principal: angajat

Scenariul principal cu succes:

1. Utilizatorul selectează operația de vizualizare a produselor.
2. Aplicația afișează un tabel cu toate produsele existente în baza de date.

**Use case:** Vizualizare comenzi

Actor principal: angajat

Scenariul principal cu succes:

1. Utilizatorul selectează operația de vizualizare a comenzilor.
2. Aplicația afișează un tabel cu toate produsele existente în baza de date.

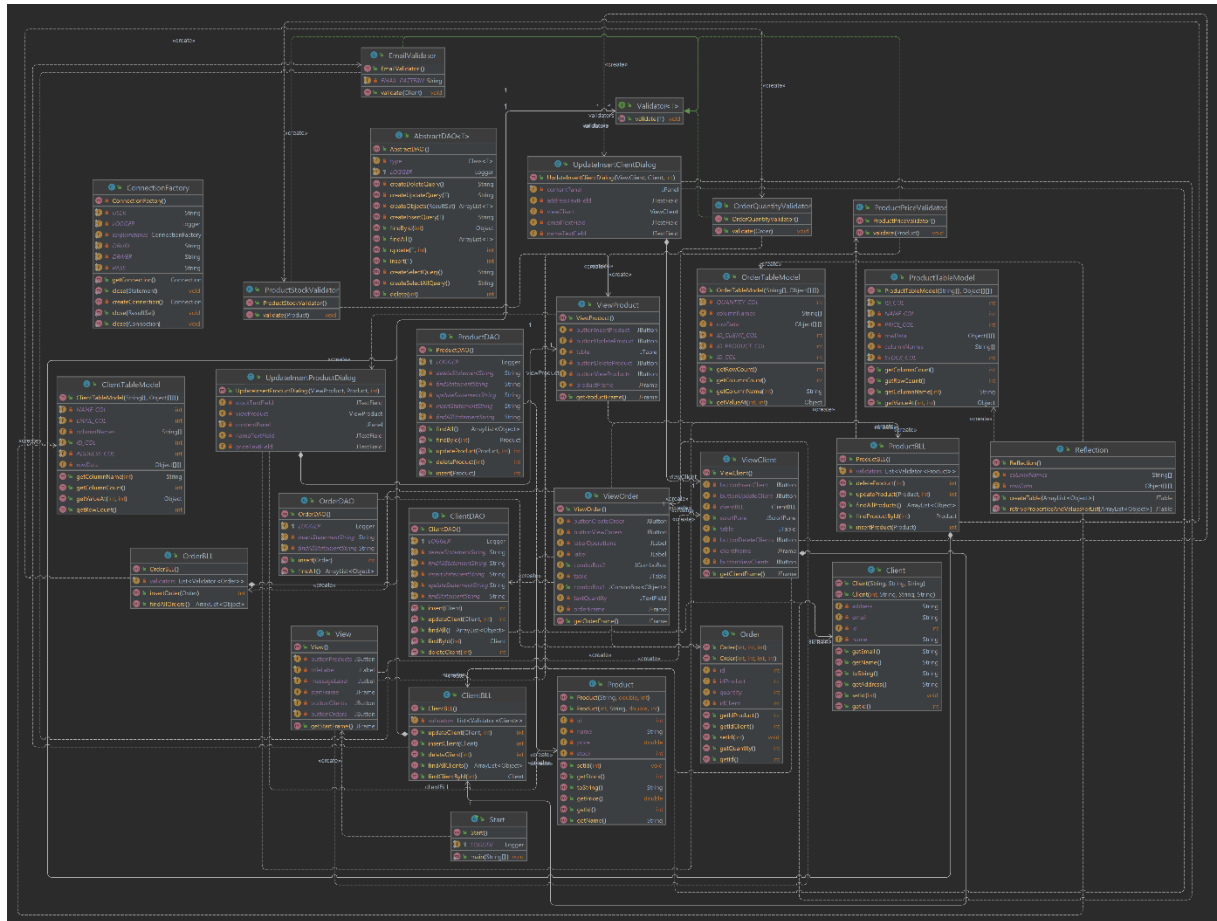
### 3. Proiectare

#### • Proiectarea OOP a aplicației:

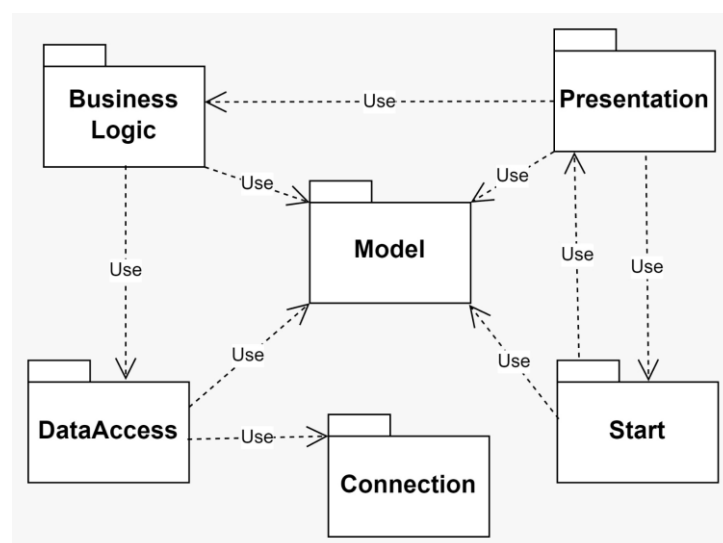
Pentru a rezolva specificațiile problemei, am ales să implementez clasele: Client, Product, Order, ClientDAO, ProductDAO, OrderDAO, ClientBLL, ProductBLL, OrderBLL, ConnectionFactory, Reflection, Start, ClientTableModel, OrderTableModel, ProductTableModel, UpdateInsertClientDialog, UpdateInsertProductDialog, View, ViewClient, ViewOrder, ViewProduct, interfața Validator implementată de clasele EmailValidator, OrderQuantityValidator, ProductPriceValidator și ProductStockValidator, incluse în diferite pachete în funcție de utilitatea lor, urmând modelul pentru layered arhitectură.

Clasa Client este inclusă în pachetul org.example.Model și implementează modelul pentru un client din aplicație. Clasa Product este inclusă tot în pachetul org.example.Model și aici am implementat un model de produs pentru aplicație. Clasa Order din pachetul org.example.Model implementează modelul pentru o comandă. Clasele ClientDAO, OrderDAO, ProductDAO sunt incluse în pachetul org.example.DataAccess și aici se face accesul la baza de date. Clasele ClientBLL, ProductBLL, OrderBLL sunt incluse în pachetul org.example.BusinessLogic și aici se află logica aplicației. În pachetul org.example.BusinessLogic se află și subpachetul validators cu clasele ce validează obiectele folosite de metodele claselor ClientBLL, ProductBLL, OrderBLL. Clasa ConnectionFactory se află în pachetul org.example.Connection și aici se fac conexiunile la baza de date. În pachetul org.example.Presentation se află clasele pentru realizarea interfeței grafice unde se vor face operațiile pe clienți, produse și comenzi și vizualizarea lor. În pachetul org.example.Start se află clasa Reflection unde se folosește reflecția pentru a genera un tabel în GUI și clasa Start unde se pornește aplicația cu fereastra principală.

- **Diagrama UML de clase:**



- **Diagrama de pachete:**



- **Structurile de date folosite:**

Structurile de date pe care le-am utilizat în implementarea aplicației sunt:

- *List<Validator<T>>*: pentru a stoca validatorii pentru obiectele folosite în aplicație;
- *ArrayList<Object>*: pentru a stoca obiectele folosite în aplicație;
- *StringBuilder*: pentru a compune interogarea;
- *Logger*: pentru a înregistra mesaje de log;
- *PreparedStatement, Statement, ResultSet, Connection*: pentru conexiunile la baza de date;
- *Client*: clasa creată pentru a stoca clienții;
- *Product*: clasa creată pentru a stoca produsele;
- *Order*: clasa creată pentru a stoca comenzile;
- *String[]*: array de String-uri pentru a stoca numele coloanelor din tabel;
- *String[][]*: matrice pentru a stoca datele pentru fiecare rând din tabel;
- *restul claselor necesare la implementarea aplicației de gestionare a comenzilor.*

- **Interfețele definite**

- Am implementat interfața *Validator <T>* care este generică pentru a valida un obiect de tipul T. Aceasta are o metodă *validate(T t)* care primește un obiectul de tipul T pentru a-i valida proprietățile. Interfața este utilă deoarece se pot defini mai mulți validatori pentru diferite obiecte din aplicație. Clasele *EmailValidator*, *OrderQuantityValidator*, *ProductPriceValidator*, *ProductStockValidator* implementează interfața definită și suprascriu metoda acesteia.

## 4. Implementare

- *Clasa Client* - definește modelul pentru un client. Atributele clasei sunt private și sunt id-ul de tip întreg care reprezintă un identificator unic pentru fiecare instanță de tipul Client, name, address și email-ul. Clasa are doi constructori, primul cu toate câmpurile și al doilea fără câmpul id.

Metodele clasei Task sunt:

- *void setID(int id)*: setează valoarea atributului id de tip privat;
- *int getID()*: accesează atributul id de tip privat;
- *int getName()*: accesează atributul name de tip privat;
- *int getAddress()*: accesează atributul address de tip privat;
- *int getEmail()*: accesează atributul email de tip privat;
- *String toString()*: suprascrie metoda *toString* din clasa *Object* astfel încât să se scrie clientul sub formă de șir de caractere;

- *Clasa Product* - definește modelul pentru un produs. Atributele clasei sunt private și sunt id-ul de tip întreg care reprezintă un identificator unic pentru fiecare instanță de tipul Product, name, price și stock. Clasa are doi constructori, primul cu toate câmpurile și al doilea fără câmpul id.

Metodele clasei Task sunt:

- *void setID(int id)*: setează valoarea atributului id de tip privat;
- *int getID()*: accesează atributul id de tip privat;
- *int getName()*: accesează atributul name de tip privat;
- *int getPrice()*: accesează atributul price de tip privat;
- *int getStock()*: accesează atributul stock de tip privat;
- *String toString()*: suprascrie metoda *toString* din clasa *Object* astfel încât să se scrie produsul sub formă de șir de caractere;

- *Clasa Order* - definește modelul pentru o comandă. Atributele clasei sunt private și sunt id-ul de tip întreg care reprezintă un identificator unic pentru fiecare instanță de tipul Product, id-ul clientului care face comanda, id-ul produsul care este comandat și cantitatea, de tip private. Clasa are doi constructori, primul cu toate câmpurile și al doilea fără câmpul id.

Metodele clasei Task sunt:

- *void setID(int id)*: setează valoarea atributului id de tip privat;
- *int getID()*: accesează atributul id de tip privat;



- `int getIdClient()`: accesează atributul `idClient` de tip privat;
- `int getIdProduct()`: accesează atributul `idProduct` de tip privat;
- `int getQuantity()`: accesează atributul `quantity` de tip privat;
- `String toString()`: suprascrie metoda `toString` din clasa `Object` astfel încât să se scrie comanda sub formă de șir de caractere;

- *Clasele `EmailValidator`, `OrderQuantityValidator`, `ProductPriceValidator`, `ProductStockValidator`*–implementează interfața `Validator` care validează un anumit câmp pentru un anumit obiect din aplicație. Metoda validează un obiect `t` de tipul `T` și validează un anumit atribut. Clasele nu au atribute, doar metoda validate suprascrisă în funcție de necesitate.

- *Clasa `ConnectionFactory`* este utilizată pentru a crea conexiunile la baza de date și pentru a închide corect aceste conexiuni, instrucțiuni și seturi de rezultate. Această clasă conține metode pentru a crea și închide conexiuni la baza de date, instrucțiuni și seturi de rezultate și pentru a obține aceste conexiuni. Clasa are un `Logger` pentru a transmite avertismente și constantele `DRIVER`, `DBURL`, `USER`, `PASS` care conțin numele clasei driverului `JDBC`, `URL`-ul bazei de date la care se va conecta aplicația, numele de utilizator și parola pentru conectare. Această clasă are un constructor privat care inițializează driverul `JDBC` și o variabilă `singleInstance`, instanță a clasei `ConnectionFactory` utilizată pentru a returna conexiunea la baza de date.

- *Clasele `View`, `ViewClient`, `ViewOrder`, `ViewProduct`*- implementează interfața grafică pentru o aplicație de gestionare a comenzilor. Aceste clase folosesc elemente specifice interfețelor grafice (panouri, etichete, casete, text, butoane, tabele) și ascultători pentru butoane pentru a fi efectuate operații la apăsarea lor și au constructori unde sunt inițializate componentele și metode de returnare a ferestrelor. *Clasele `UpdateInsertClientDialog` și `UpdateInsertProductDialog`* sunt folosite tot în interfața grafică, fiind ferestre pentru inserare de date. *Clasele `ClientTableModel`, `OrderTableModel` și `ProductTableModel`* sunt folosite pentru a realiza tabelele în care vor fi afișate obiectele din baza de date.

- *Clasele `ClientDAO`, `OrderDAO`, `ProductDAO`*- sunt folosite pentru accesarea datelor din baza de date utilizată de aplicație. Clasele `ClientDAO` și `ProductDAO` au metode pentru căutare, inserare, editare și ștergere, iar clasa `OrderDAO` are metode pentru adăugare și vizualizare. De asemenea, clasele au ca și atribute `String`-uri pentru a realiza query-urile necesare fiecărei metode. Clasa `AbstractDAO` este generică și implementează metodele pentru vizualizare, adăugare, ștergere și editare obiect și generează query-urile pentru accesarea bazei de date prin reflecție.

- *Clasele `ClientBLL`, `ProductBLL`, `OrderBLL`* implementează logica pentru obiectele de tip `Client`, `Product` și `Order`. Clasele utilizează obiecte de tipul `Validator` pentru a valida obiectele înainte de le insera/actualiza în baza de date folosind metodele din clasele `ClientDAO`, `ProductDAO` și `OrderDAO`. De asemenea, au metode pentru vizualizare și căutare. Clasele au ca și atribut câte o listă de validatori.

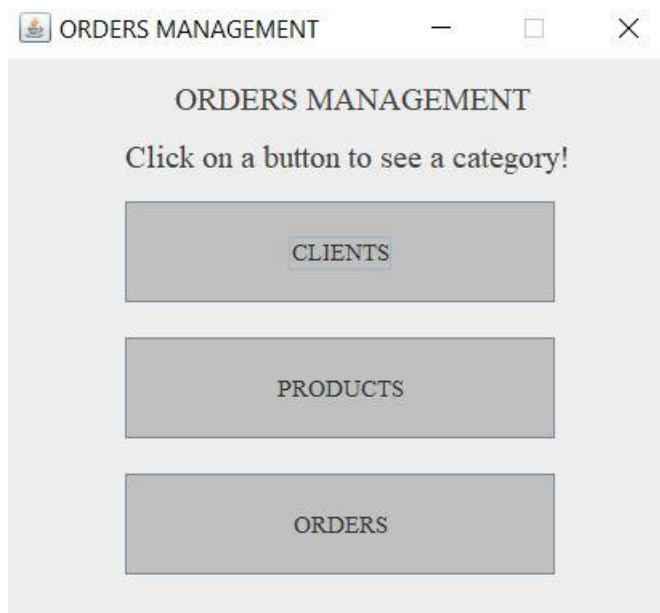
- *Clasa `Reflection`*- utilizează tehnici de reflecție pentru a crea un tabel `JTable` dintr-o listă de obiecte. Clasa are ca și atribute un `String[] columnNames` pentru numele coloanelor și un `Object[][] rowData` pentru datele de pe fiecare rând din tabel. Metodele clasei sunt

- `createTable(ArrayList<Object> objects)`: primește o listă de obiecte și returnează un tabel în funcție de tipul clasei obiectelor. Această metodă este folosită în metoda `retrievePropertiesAndValuesForList(ArrayList<Object> objects)`.
- `retrievePropertiesAndValuesForList(ArrayList<Object> objects)`: creează un tabel `JTable` utilizând metoda `createTable` și completează coloanele și rândurile acestuia cu proprietățile și valorile obiectelor din lista data utilizând reflecția.

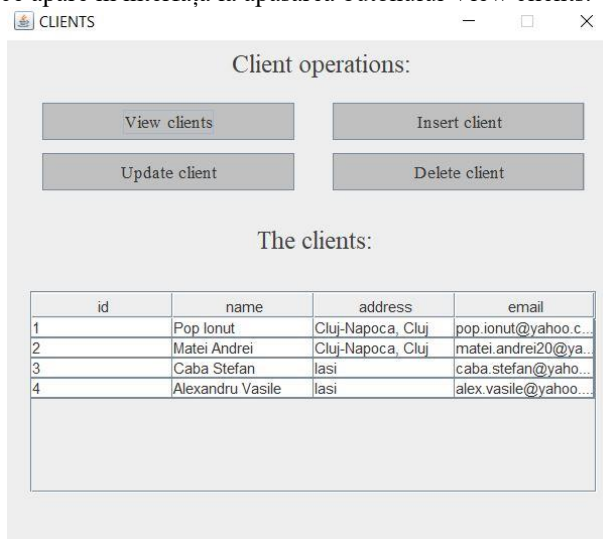
- *Clasa `Start`*-are un obiect `Logger` pentru a transmite mesaje de avertisment și o metodă `main(String[] args)` în care are loc pornirea aplicației prin deschiderea ferestrei principale a aplicației de gestionare a comenzilor.

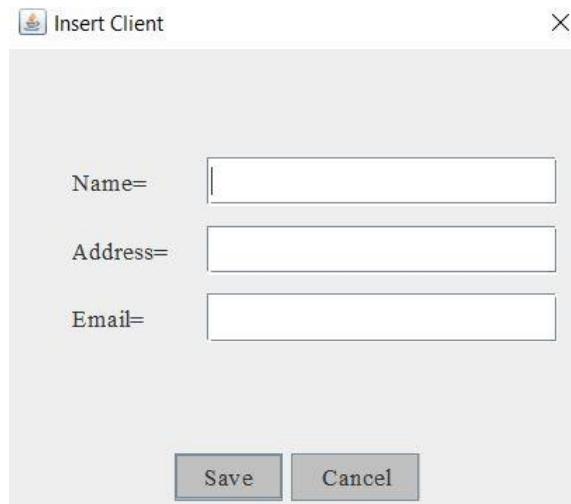
## Implementarea interfeței grafice:

Interfața grafică a aplicației de gestionare a comenzilor este simplă și ușor de utilizat. Fereastra principală a aplicației are ca titlu ORDERS MANAGEMENT. Este scris un mesaj pentru a apăsa pe categoria dorită. Există trei butoane denumite corespunzător: butonul CLIENTS pentru a deschide fereastra pentru operațiile pe clienți, butonul PRODUCTS pentru a deschide fereastra pentru operațiile pe produse și butonul ORDERS pentru a deschide fereastra pentru operațiile pe comenzi.



Utilizatorul trebuie să apese pe butonul pentru categoria dorită. La apăsarea butonului CLIENTS se deschide fereastra pentru clienți. Această fereastră are două etichete și patru butoane, fiecare corespunzător unei operații (vizualizare clienți, adăugare client, actualizare client și ștergere client). La apăsarea butoanelor Update client și Insert client se va deschide o fereastră de dialog pentru introducerea datelor, la apăsarea butonului View clients va apărea un tabel cu clienții din baza de date. Înainte de apăsarea butoanelor Delete client și Update client trebuie selectat un client din tabelul ce apare în interfață la apăsarea butonului View clients.





Insert Client

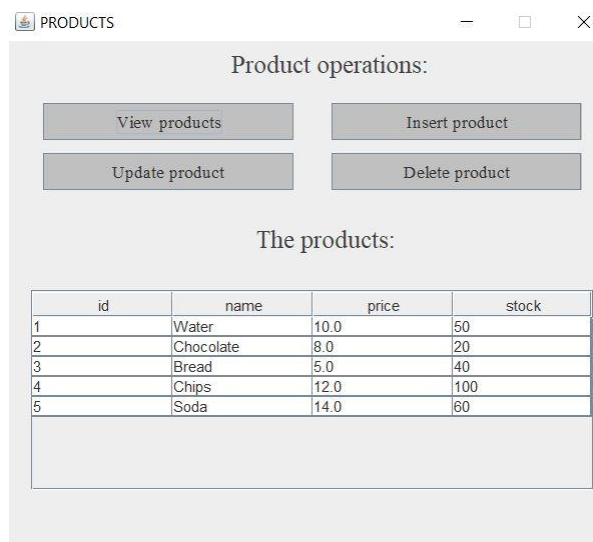
Name=

Address=

Email=

Save Cancel

La apăsarea butonului PRODUCTS se deschide fereastra pentru produse. Această fereastră are două etichete și patru butoane, fiecare corespunzător unei operații (vizualizare produse, adăugare produs, actualizare produs și ștergere produs). La apăsarea butoanelor Update product și Insert product se va deschide o fereastră de dialog pentru introducerea datelor, la apăsarea butonului View products va apărea un tabel cu produsele din baza de date. Înainte de apăsarea butoanelor Delete product și Update product trebuie selectat un produs din tabelul ce apare în interfață la apăsarea butonului View products.



PRODUCTS

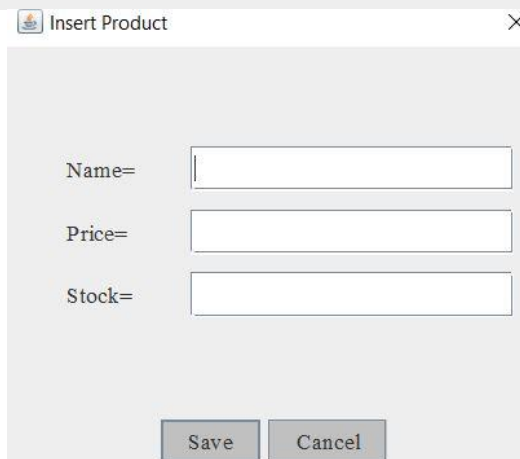
Product operations:

View products Insert product

Update product Delete product

The products:

id	name	price	stock
1	Water	10.0	50
2	Chocolate	8.0	20
3	Bread	5.0	40
4	Chips	12.0	100
5	Soda	14.0	60



Insert Product

Name=

Price=

Stock=

Save Cancel

La apăsarea butonului ORDERS se deschide fereastra pentru comenzi. Această fereastră are două etichete și două butoane, fiecare corespunzător unei operații (vizualizare comenzi, adăugare comandă). La apăsarea butonului View orders va apărea un tabel cu comenzile din baza de date. Înainte de apăsarea butonului Create order trebuie selectat un client din JComboBox-ul din fereastră și un produs și introdusă cantitatea pentru comandă.

id	idClient	idProduct	quantity
1	1	2	10
2	1	2	20

## 5. Rezultate

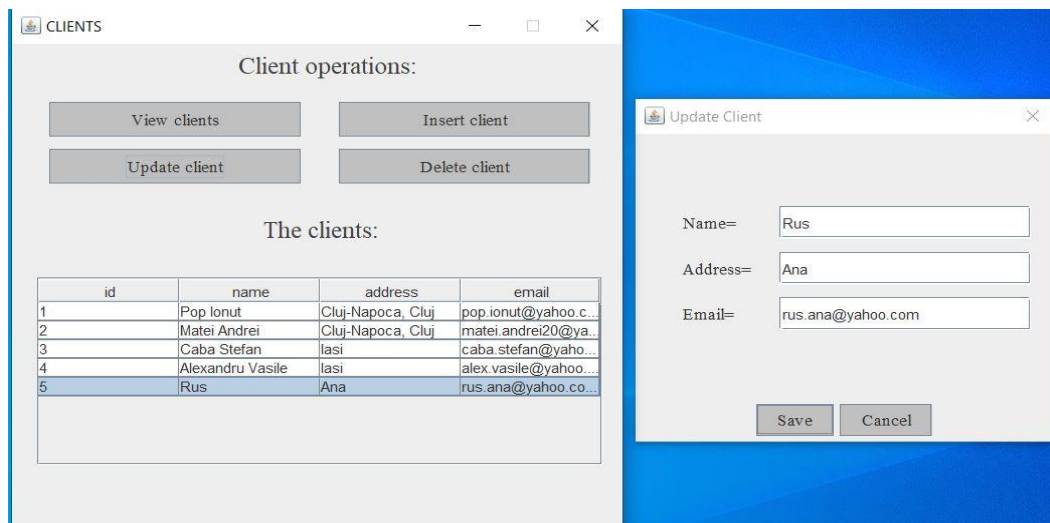
Scenariile pentru testare au fost prezentate în secțiunea a doua a documentației.

Am testat aplicația de gestionare a comenzilor prin verificarea clienților, produselor și comenzilor afișate sub formă de tabel în interfața grafică și prin inserarea, ștergerea, actualizarea clienților și a produselor. Am adăugat comenzi pentru a vedea dacă se realizează în mod corect.

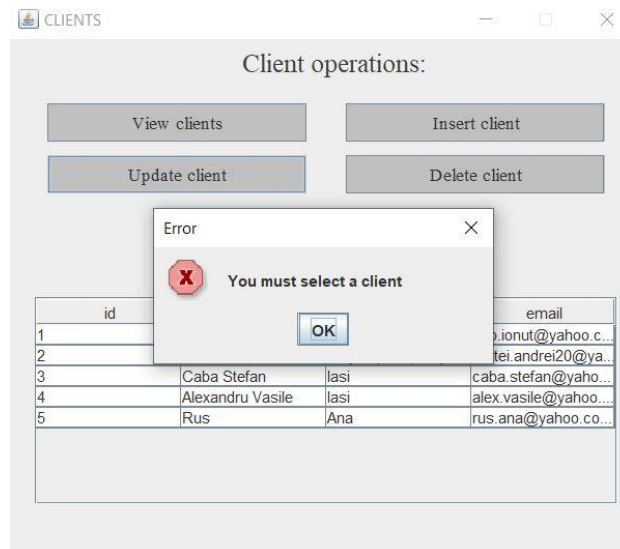
- Un exemplu de inserare a unui client:

id	name	address	email
1	Pop Ionut	Cluj-Napoca, Cluj	pop.ionut@yahoo.c...
2	Matei Andrei	Cluj-Napoca, Cluj	matei.andrei20@ya...
3	Caba Stefan	Iasi	caba.stefan@yaho...
4	Alexandru Vasile	Iasi	alex.vasile@yahoo...
5	Rus	Ana	rus.ana@yahoo.co...

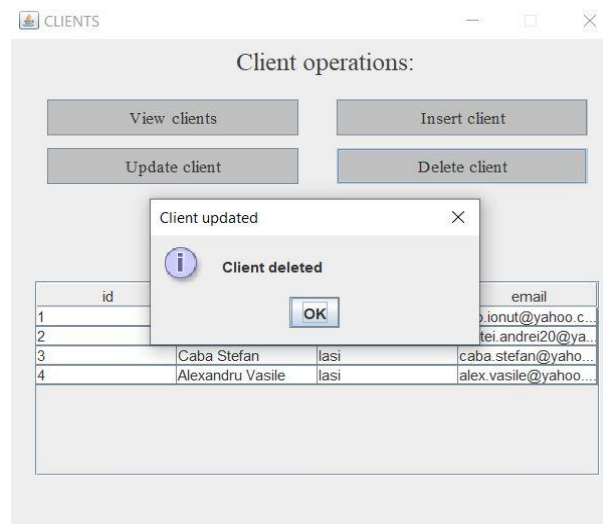
- Editarea unui client:



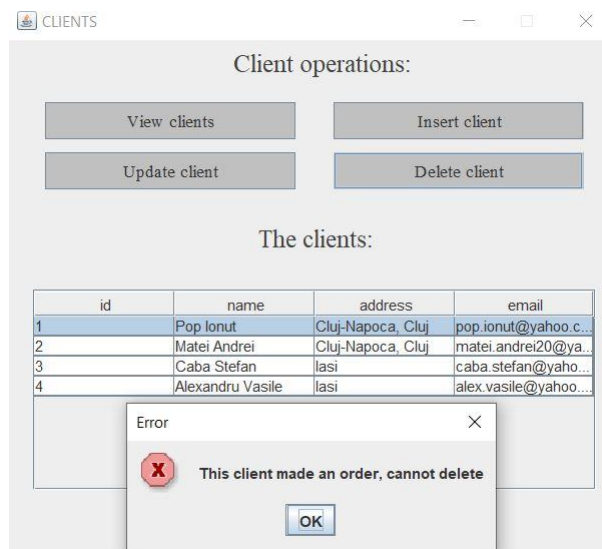
- Dacă nu se selectează un client pentru actualizare:



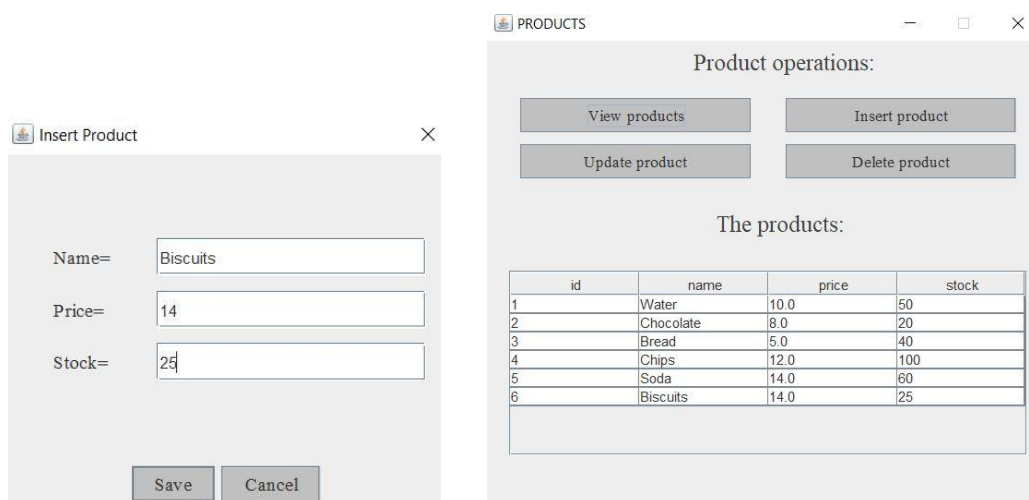
- Ștergerea unui client după selectare:



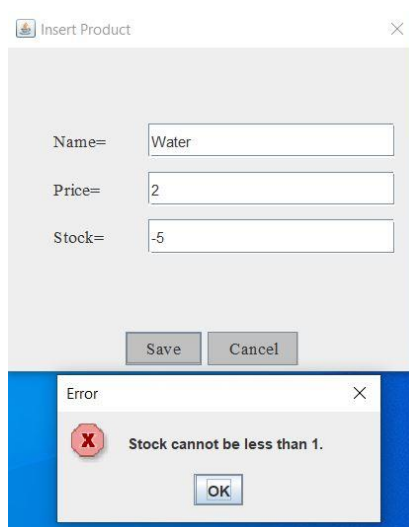
- Dacă se încearcă ștergerea unui client care a efectuat o comandă (id-ul său este folosit la comenzi, în tabela orders din baza de date):



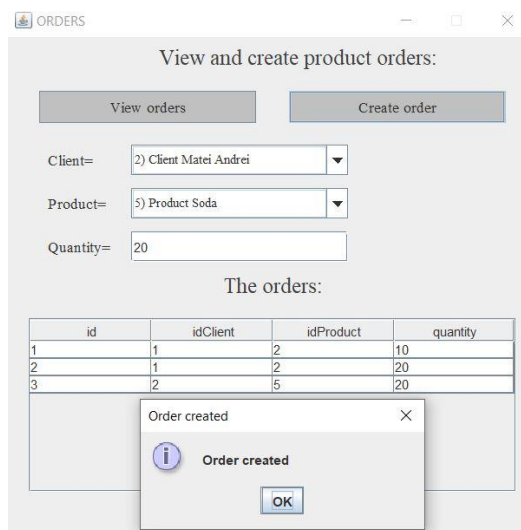
- Operațiile asupra produselor se realizează similar celor pe clienți. De exemplu, inserarea unui produs:



- Dacă datele nu sunt valide la inserarea unui produs:

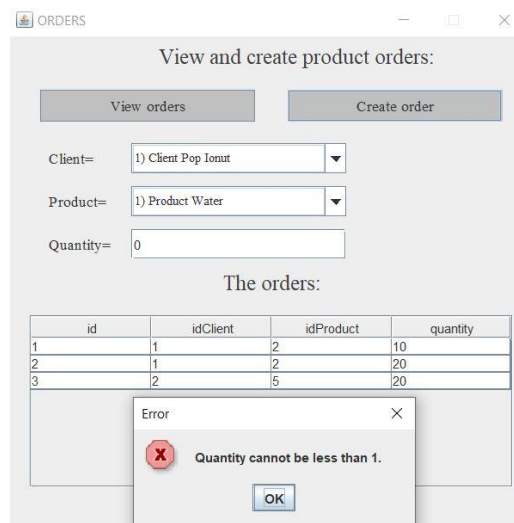


- Adăugarea unei comenzi:



- Dacă datele pentru comandă nu sunt valide:

++



## 6. Concluzii

În concluzie, proiectarea și implementarea unei aplicații menite să gestioneze comenzile pentru un depozit este destul de dificilă atât din punct de vedere al implementării, cât și al interfeței grafice.

Această temă a ajutat la recapitularea conceptelor legate de Programarea orientată pe obiect, a lucrului cu baze de date relaționale și la acumularea unor noi cunoștințe, întrucât am învățat din această temă cum se folosesc tehnicile de reflecție în Java. De asemenea, am învățat că este foarte important să fie urmați pașii următori pentru a ajunge la finalizarea sarcinii: analizarea problemei și identificarea funcționalităților aplicației de gestionare a comenzilor, proiectarea aplicației de gestionare, apoi implementarea acestora și testarea pentru a vedea dacă funcționează fără erori.

Posibile dezvoltări ulterioare ar fi implementarea unei interfețe grafice mai complexe și vizualizarea comenzilor cu mai multe informații despre clienți și produse comandate, nu doar cu id-ul lor și posibilitatea adăugării de comenzi cu mai multe produse.

## 7. Bibliografie

1. *Fundamental Programming Techniques – Suport Presentations and Lectures* - <https://dsrl.eu/courses/pt/>
2. *Reflection example* - [https://gitlab.com/utcn\\_dsrl/pt-reflection-example](https://gitlab.com/utcn_dsrl/pt-reflection-example)
3. *Simple layered project* - [https://gitlab.com/utcn\\_dsrl/pt-layered-architecture](https://gitlab.com/utcn_dsrl/pt-layered-architecture)
4. *Java Reflection Tutorial*- <https://jenkov.com/tutorials/java-reflection/index.html>
5. *Java Swing JTable* - <https://www.codejava.net/java-se/swing/a-simple-jtable-example-for-display>
6. *JavaDoc* - <https://www.baeldung.com/javadoc>