Documentație de temă

Order Management

Proiect realizat de: Anton Ioana-Carina

Grupa 30225

Contents

[Obiectivul temei 3](#_Toc69886035)

[Obiectivul principal 3](#_Toc69886036)

[Obiectivul secundar 3](#_Toc69886037)

[Analiza problemei 3](#_Toc69886038)

[Diagrama use-case 3](#_Toc69886039)

[Client Operations 4](#_Toc69886040)

[Product operations 5](#_Toc69886041)

[Order Operations 6](#_Toc69886042)

[Cerinte functionale 7](#_Toc69886043)

[Cerinte non-functionale 7](#_Toc69886044)

[Decizii de proiectare 7](#_Toc69886045)

[Layered arhitecture 7](#_Toc69886046)

[Reflection techniques 7](#_Toc69886047)

[Singleton pattern 8](#_Toc69886048)

[Diagrama UML 8](#_Toc69886049)

[Pachete 8](#_Toc69886050)

[Implementare 8](#_Toc69886051)

[Model 8](#_Toc69886052)

[Client 9](#_Toc69886053)

[Comanda 9](#_Toc69886054)

[ComandaProdus 9](#_Toc69886055)

[Product 9](#_Toc69886056)

[Presentation 9](#_Toc69886057)

[ClientGUI + Subpachetul client 9](#_Toc69886058)

[ProductGUI+Subpachetul product 9](#_Toc69886059)

[OrderGUI 9](#_Toc69886060)

[HomepageGUI 9](#_Toc69886061)

[Controller 9](#_Toc69886062)

[Connection 9](#_Toc69886063)

[DAO 9](#_Toc69886064)

[Rezultate 10](#_Toc69886065)

[Concluzii 11](#_Toc69886066)

[Dezvoltari ulterioare 11](#_Toc69886067)

[Ce am invatat 11](#_Toc69886068)

[Bibliografie 11](#_Toc69886069)

# Obiectivul temei

## Obiectivul principal

Implementarea unei aplicatii **OrderManagement** pentru procesarea comenzilor clientilor pentru un depozit. Se vor folosi baze de date relationale pentru stocharea produselor, clientilor si comenzilor. De asemenea, aplicatia va trebui sa fie structurata in pachete folosind arhitectura in straturi prezentata in prezentarea suport si vor trebui folosite (cel putin) urmatoarele clase:

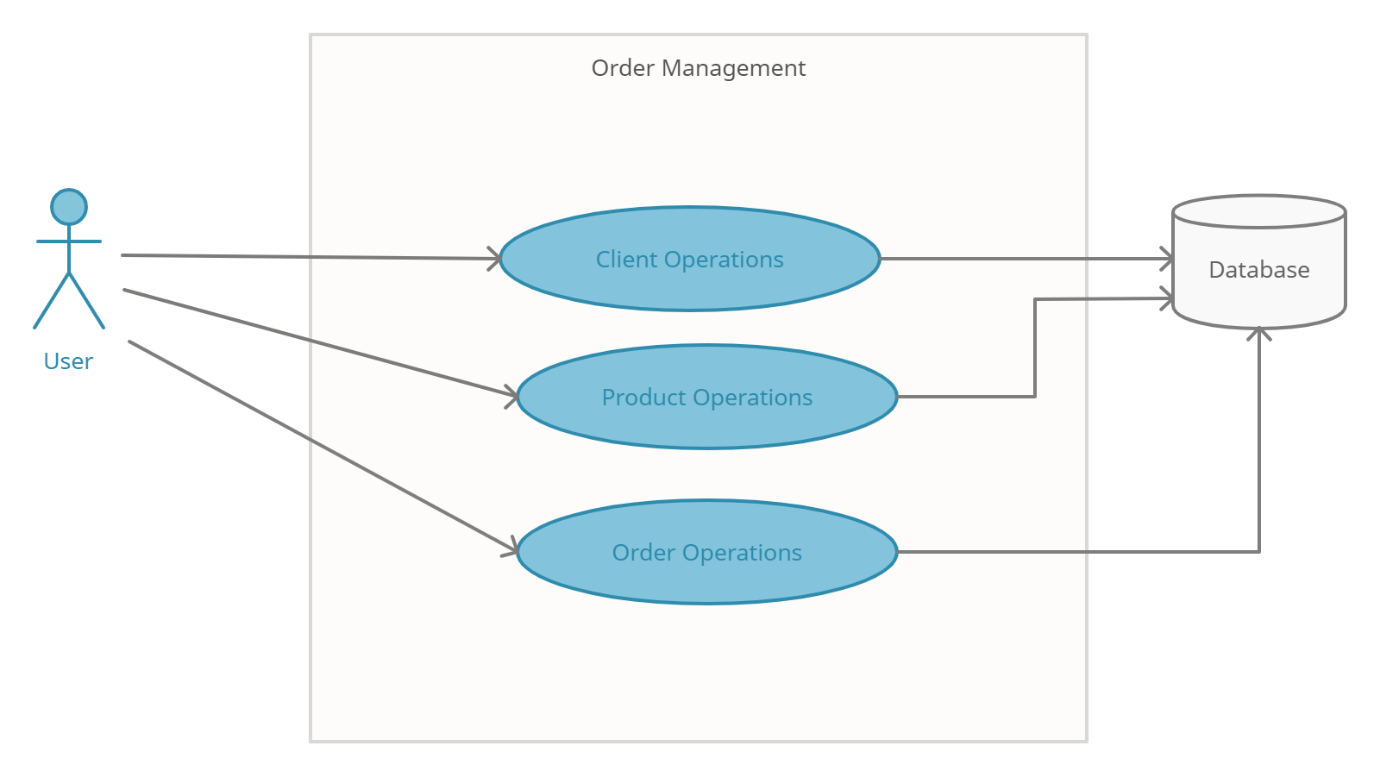
* **Model classes** – modelul de date al aplicatiei
* **Business Logic classes** – logica aplicatiei
* **Presentation classes** – clase de interfata grafica
* **Data access classes** – clase care contin accesul la baza de date

## Obiectivul secundar

|  |  |  |
| --- | --- | --- |
| **Analiza problemei si identificarea cerintelor** | Se stabileste modul de functionare al aplicatiei, se construiesc diagramele use-case si se stabilesc cerintele functionale, respectiv nefunctionale ale problemei | Se detalieaza in capitolul 2 |
| **Proiectare** | Se stabileste structura modelului dupa care se va construi programul, se stabilesc pachetele, clasele, relatiile dintre acestea (Diagrama UML), etc. | Se detaliaza in capitolul 3 |
| **Implementare** | Se transcriu in cod efectiv diagramele de la etapa de proiectare | Se detaliaza in capitolul 4 |
| **Testare** | Se utilizeaza un framework de testare pentru a verifica functionalitatea aplicatiei si cazurile acoperita de acestea | Se detaliaza in capitolul 5 |

# Analiza problemei

## Diagrama use-case



### Client Operations

**Use Case**: Add new client

**Primary Actor**: utilizatorul

**Main Success Scenario**:

1. Utilizatorul introduce datele pentru inserarea unui nou client in baza de date

2. Utilizatorul apasa butonul de confirmare

3. Programul va executa interogarea si va insera in tabela client un nou client

**Alternative Sequence**: Incorrect input data

- Utilizatorul introduce datele in mod gresit

- Scenariul se intoarce la pasul 1

**Use Case**: Edit client

**Primary Actor**: utilizatorul

**Main Success Scenario**:

1. Utilizatorul introduce datele pentru modificarea datelor unui client

2. Utilizatorul apasa butonul de confirmare

3. Programul va executa interogarea si va actualiza in tabela client datele clientului precizat

**Alternative Sequence**: Incorrect input data

- Utilizatorul introduce datele in mod gresit

- Scenariul se intoarce la pasul 1

**Alternative Sequence**: Client doesn’t exist

- Clientul nu exista in tabela client

- Aplicatia va atentiona utilizatorul

- Scenariul se intoarce la pasul 1

**Use Case**: Delete client

**Primary Actor**: utilizatorul

**Main Success Scenario**:

1. Utilizatorul introduce cheia primara a clientului pe care doreste sa il stearga

2. Utilizatorul apasa butonul de confirmare

3. Programul va executa interogarea si va sterge din tabela client clientul precizat

**Alternative Sequence**: Incorrect input data

- Utilizatorul introduce datele in mod gresit

- Scenariul se intoarce la pasul 1

**Alternative Sequence**: Client doesn’t exist

- Clientul nu exista in tabela client

- Aplicatia va atentiona utilizatorul

- Scenariul se intoarce la pasul 1

**Use Case**: View all clients in a table

**Primary Actor**: utilizatorul

**Main Success Scenario**:

1. Utilizatorul apasa butonul pentru afisarea tabelei client

2. Programul va afisa in interfata intreaga tabela client din baza de date

### Product operations

**Use Case**: Add new product

**Primary Actor**: utilizatorul

**Main Success Scenario**:

1. Utilizatorul introduce datele pentru inserarea unui nou produs in baza de date

2. Utilizatorul apasa butonul de confirmare

3. Programul va executa interogarea si va insera in tabela product un nou produs

**Alternative Sequence**: Incorrect input data

- Utilizatorul introduce datele in mod gresit

- Scenariul se intoarce la pasul 1

**Use Case**: Edit product

**Primary Actor**: utilizatorul

**Main Success Scenario**:

1. Utilizatorul introduce datele pentru modificarea datelor unui produs

2. Utilizatorul apasa butonul de confirmare

3. Programul va executa interogarea si va actualiza in tabela produs datele produsului precizat

**Alternative Sequence**: Incorrect input data

- Utilizatorul introduce datele in mod gresit

- Scenariul se intoarce la pasul 1

**Alternative Sequence**: Product doesn’t exist

- Produsul nu exista in tabela product

- Aplicatia va atentiona utilizatorul

- Scenariul se intoarce la pasul 1

**Use Case**: Delete product

**Primary Actor**: utilizatorul

**Main Success Scenario**:

1. Utilizatorul introduce cheia primara a produsului pe care doreste sa il stearga

2. Utilizatorul apasa butonul de confirmare

3. Programul va executa interogarea si va sterge din tabela product produsul precizat

**Alternative Sequence**: Incorrect input data

- Utilizatorul introduce datele in mod gresit

- Scenariul se intoarce la pasul 1

**Alternative Sequence**: Product doesn’t exist

- Produsul nu exista in tabela product

- Aplicatia va atentiona utilizatorul

- Scenariul se intoarce la pasul 1

**Use Case**: View all products in a table

**Primary Actor**: utilizatorul

**Main Success Scenario**:

1. Utilizatorul apasa butonul pentru afisarea tabelei product

2. Programul va afisa in interfata intreaga tabela product din baza de date

### Order Operations

**Use Case**: Add new client

**Primary Actor**: utilizatorul

**Main Success Scenario**:

1. Utilizatorul va selecta un produs existent

2. Utilizatorul va selecta un client existent

3. Utilizatorul va preciza cantitatea dorita

4. Programul va executa interogarea si va insera in tabela order noua comanda

5. Programul va actualiza noua cantitate a produsului in tabela product

**Alternative Sequence**: Product quantity is under stock

- Utilizatorul introduce o cantitate prea mare de produse

- Aplicatia va atentiona utilizatorul de acest fapt

- Scenariul se intoarce la pasul 1

## Cerinte functionale

* Aplicatia ar trebui sa permita utilizatorului sa adauge noi clienti
* Aplicatia ar trebui sa permita utilizatorului sa modifice datele unui client
* Aplicatia ar trebui sa permita utilizatorului sa stearga un client
* Aplicatia ar trebui sa permita utilizatorului sa vizualizeze toti clientii intr-o tabela
* Aplicatia ar trebui sa permita utilizatorului sa adauge noi produse
* Aplicatia ar trebui sa permita utilizatorului sa modifice datele unui produs
* Aplicatia ar trebui sa permita utilizatorului sa stearga un produs
* Aplicatia ar trebui sa permita utilizatorului sa vizualizeze toate produsele intr-o tabela
* Aplicatia ar trebui sa permita utilizatorului sa creeze noi comenzi
* Aplicatia ar trebui sa verifice inainte de fiecare operatie daca datele sunt valide, utilizand clase care implementeaza interfata validate

## Cerinte non-functionale

* Aplicatia ar trebui sa aiba o interfata grafica atractiva, intuitiva si usor de utilizat atat pentru fiecare categorie de operatii cat si pentru pagina de pornire.

# Decizii de proiectare

## Layered arhitecture

Arhitectura pe straturi implica, dupa cum sugereaza si numele, impartirea in mai multe straturi a aplicatiei. Fiecare strat va avea propriul scop si va apela metodele straturilor de sub el. In acest proiect se vor implementa straturile:

* Model – cate o clasa pentru fiecare tabela din baza de date, care sa reflecte atat denumirea cat si atributele ei
* Presentation Layer – clasele pentru interfatele grafice, cate o interfata pentru fiecare tabela cu operatiile specifice
* Business Logic – incapsuleaza logica aplicatiei
* Data Access – clase care comunica direct prin intermediul unei conexiuni cu baza de date si executa interogari pe aceasta

## Reflection techniques

Reflectia in Java se refera la abilitatea unui program in executie de a se examina pe sine, mediul sau software si sa isi modifice actiunile in functie de ce gaseste. Reflectia ofera acces dinamic la informatiile interne ale claselor incarcate in JVM. De asemenea, reflectia permite programatorului sa contruiasca cod flexibil care poate fi asamblat la rulare, fara sa fie nevoie de legaturi de cod sursa intre componente.

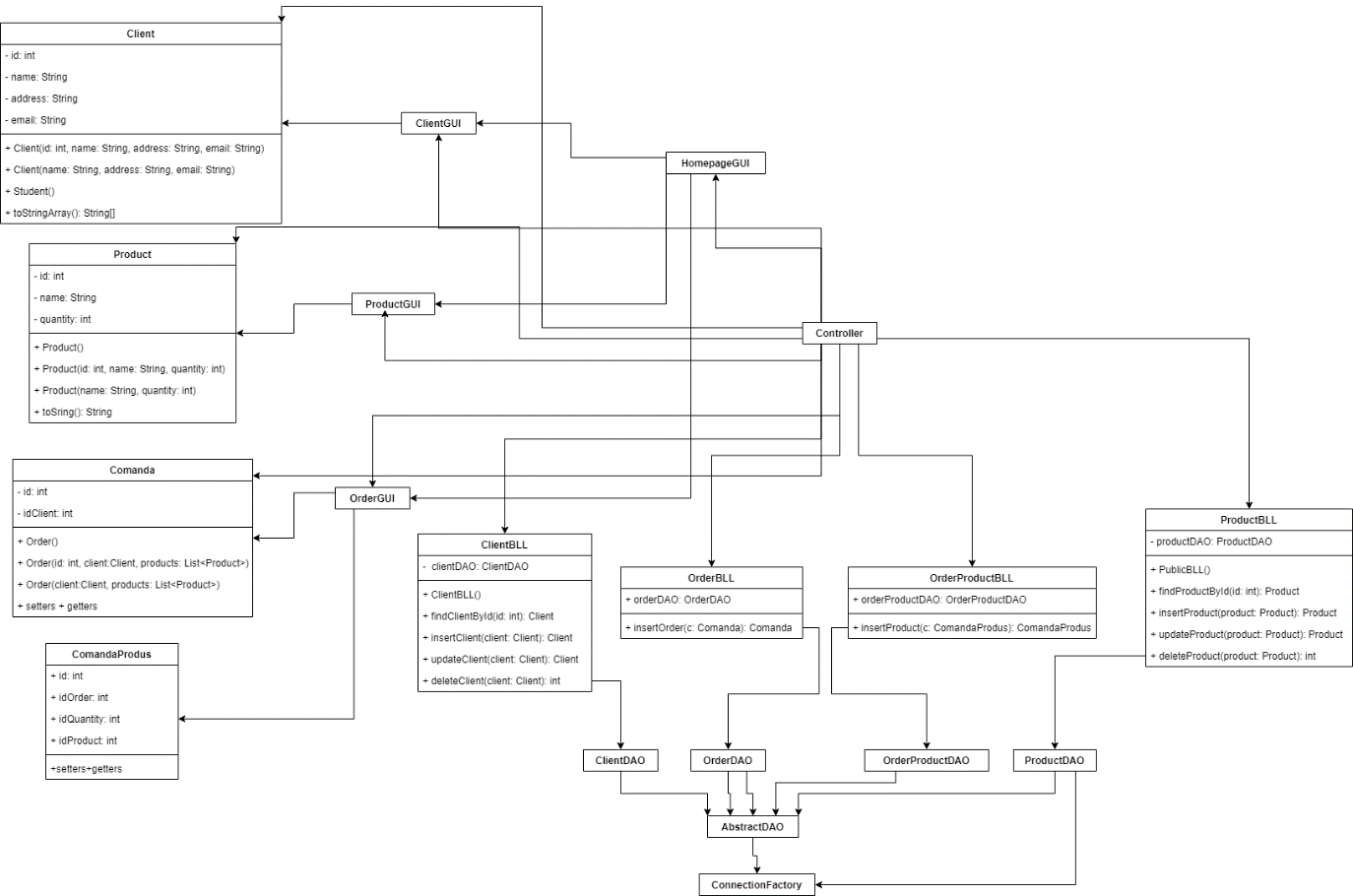
In acest proiect reflectia a fost folosita pentru:

* In clasa **AbstractDAO** atat pentru extragerea campurilor tabelelor cat si pentru prelucrarea datelor din ele pentru a face diferite operatii CRUD
* A fost utilizata pentru generarea header ului tabelei pentru a putea fi afisata intr-un JTable intr-un mod mai usor

## Singleton pattern

Modelul singleton este un model de proiectare software care restrictioneaza instantierea unei clase la o unica instanta. Acest model este folositor cand exact un obiect este necesar pentru a coordona actiunile programului. Acest model va fi utilizat pentru gestionarea conexiunii cu baza de date.

## Diagrama UML



## Pachete

Deoarece am utilizat arhitectura pe straturi, am creat cate un pachet pentru a evidentia fiecare strat in parte:

* **Bll** = business logic classes
* **Connection** = database connection
* **Dao =** data access classes
* **Model =** model classes
* **Presentation =** graphical interface classes

# Implementare

## Model

* Clasele din pachetul model reprezinta tabelele din baza de date.
* Fiecare clasa are ca variabila instanta coloanele din tabela exact asa cum sunt in baza de date.
* Clasele **Client** si **Product** au pe langa metodele de setter si getter si cate o metoda (**toStringArray()**)care transforma o instanta a clasei intr-un tablou unidimensional de String care reprezinta un rand din tabelul din baza de date; acestea au fost folosite pentru crearea modelului pentru JTable.
* **ComandaProdus** = produse care au fost puse intr-o comanda
* **Product** = produse din depozit/stoc

### Client

Variabilele instanta/atributele din tabela **client:**

* int id
* String name
* String address
* String email

### Comanda

Variabilele instanta/atributele din tabela **comanda:**

* Int id
* Int idClient

### ComandaProdus

Variabilele instanta/atributele din tabela **comandaprodus:**

* Int id
* Int idOrder
* Int quantity
* Int idProduct

### Product

Variabilele instanta/atributele din tabela **product:**

* int id
* String name
* Int quantity

## Presentation

* In acest pachet sunt toate clasele de interfata grafica utilizate pentru operatiile pe baza de date.
* **HomepageGUI** = interfata principala;
* **ClientGUI** = interfata principala pentru operatiile pe clienti;
* **ProductGUI** = interfata principala pentru operatiile pe produse;
* **OrderGUI**= fereastra pentru crearea unei comenzi
* **Controller** = clasa care leaga interfetele grafice cu operatiile pe baza de date

### ClientGUI + Subpachetul client

Subpachetul contine cate o interfata pentru fiecare operatie pe tabela Client.

### ProductGUI+Subpachetul product

Subpachetul contine cate o interfata pentru fiecare operatie pe tabela Product.

### OrderGUI

### HomepageGUI

### Controller

* Are ca variabile instanta:
  + Toate interfetele grafice
  + Toate tabelele/clasele din model
  + Toate clasele pentru tabele din BLL
* Contine mai multe subclase pentru a adauga ActionListeners la toate butoanele din interfete

## Connection

* **ConnectionFactory** = clasa **Singleton**, ea face conexiunea dintre baza de date si aplicatia facuta; are o singura instanta => o singura conexiune care va fi pornita si oprita dupa fiecare operatie pe baza de date

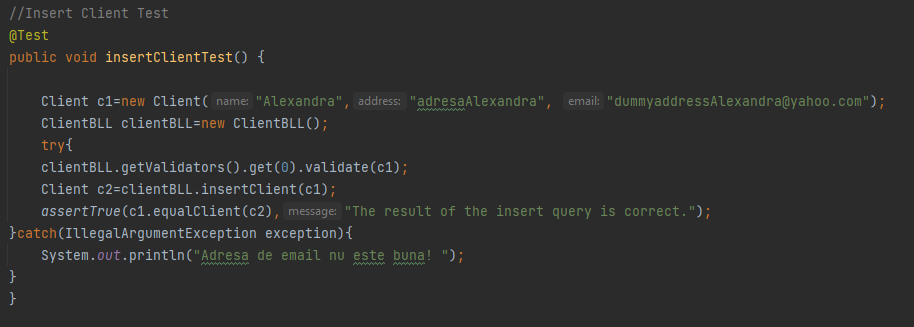
## DAO

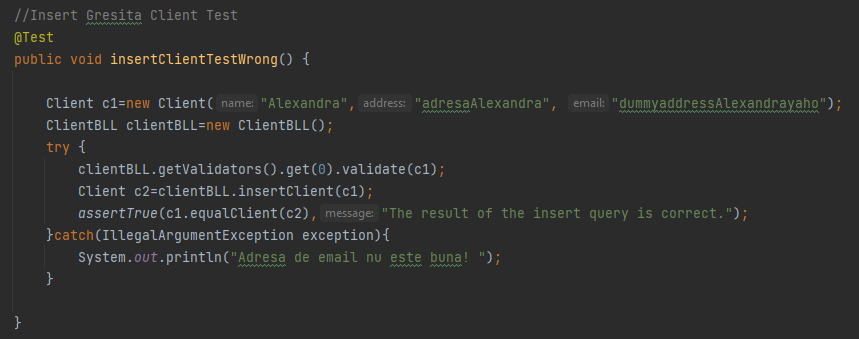
* In acest pachet se afla clasele care au acces la datele din baza de date, cea mai importanta clasa fiind **AbstractDAO**
* **AbstractDAO** = o clasa generica care implementeaza toate operatiile CRUD pentru baza noastra de date; metodele implementate:
  + **findAll** = metoda pentru o interogare de select care returbeaza intreaga tabela, metoda returneaza o lista cu toate elementele din tabel
  + **findById** = metoda pentru o interogare de select care gaseste o entitate din tabel dupa id/cheia primara, metoda returneaza obiectul respesctiv entitatii cu id ul specificat.
  + **Insert=** metoda utilizata pentru inserarea unui nou obiect in tabel, functia va returna obiectul inserat sau null daca acesta nu a putut fi inserat;
  + **Update=** metoda utilizata pentru actualizarea unei entitati intr-un tabel, se apeleaza folosind un obiect si se va interoga baza de date pentru a aduce modificari entitatii din tabel conform noilor valori din obiect; functia returneaza obiectul pentru care s-a apelat sau null in caz de eroare
  + **Delete=**metode utilizata pentru a sterge o intrare dintr-un tabel, se apeleaza folosind **id-ul** respesctiv entitatii dorite a fi stearsa
  + **getColumnNames()**= metoda folosita pentru a crea **header ul** tabelului pentru a putea fi afisat intr-un JTable
* Toate clasele **ClientDAO, OrderDAO, OrderProductDAO si ProductDAO** extind clasa AbstractDAO<T> ; Tipul generic T fiind modelul de date respectiv fiecarui tabel
* Clasa **ProductDAO** contine o metoda specifica numite **updateQuantity** folosita la plasarea comenzilor pentru a actualiza noua cantitate

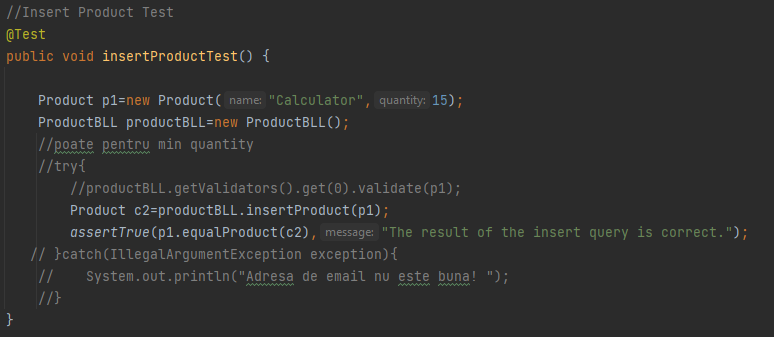
## BLL

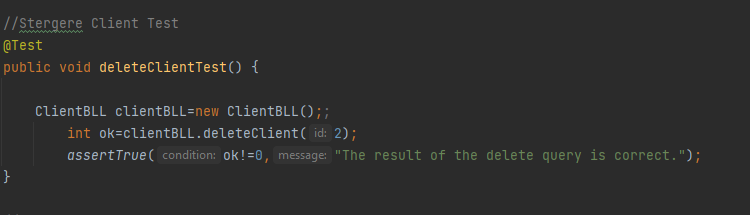
Pe langa clasele care implementeaza logica din clasele **DAO** avem si niste clase care implementeaza interfata **Validator** care contine o metoda de validare pentru verificarea datelor introduse de utilizator inainte ca acestea sa fie introduse in baza de date si posibil sa o eroneze.

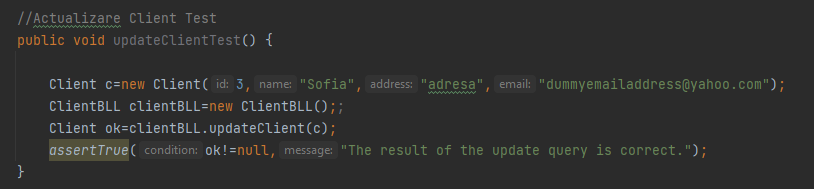
# Rezultate











# Concluzii

## Dezvoltari ulterioare

* Interfete grafice mai atractive
* Adaugare mai mult validators
* Eficientizare creare noua comanda si crestere numar maxim de produse per comanda
* Adaugare butoane pentru o navigare mai usoara si mai fluida

## Ce am invatat

* Arhitectura Layered – o structura mai potrivita decat MVC pentru proiecte de gestiune a unei baze de date
* Exersare de Java Reflection Techniques – destul de usor de folosit dupa putina exersare
* Clase cu tip generic – ajuta la reutilizarea codului
* Clasa Singleton
* Generarea de JavaDoc in IntelliJ

# Bibliografie

[1] https://gitlab.com/utcn\_dsrl/pt-layered-architecture

[2] https://gitlab.com/utcn\_dsrl/pt-reflection-example

[3] https://en.wikipedia.org/wiki/Singleton\_pattern

[4] https://docs.oracle.com/

[5] <https://stackoverflow.com/>

[6] <https://www.baeldung.com/javadoc>

[7] <https://dev.mysql.com/doc/workbench/en/wb-admin-export-import-management.html>

[8] <https://www.baeldung.com/java-pdf-creation>

[9] https://www.jetbrains.com/help/idea/working-with-code-documentation.html