# DOCUMENTATIE

## TEMA 3

## *ORDER MANAGEMENT*

*Nume student: SZAKACS EMMA-EVELIN*
*Grupa: 30221*

# CUPRINS

# 1. Obiectivul temei

***Obiectivul principal*** al acestei teme a fost sa proiectez o aplicatie Java care sa gastioneze comenzile unor clienti catre un depozit utilizand o baza de date si metoda reflexiei.

***Obiectivele secundare*** care contribuie la realizarea obiectivului principal sunt:

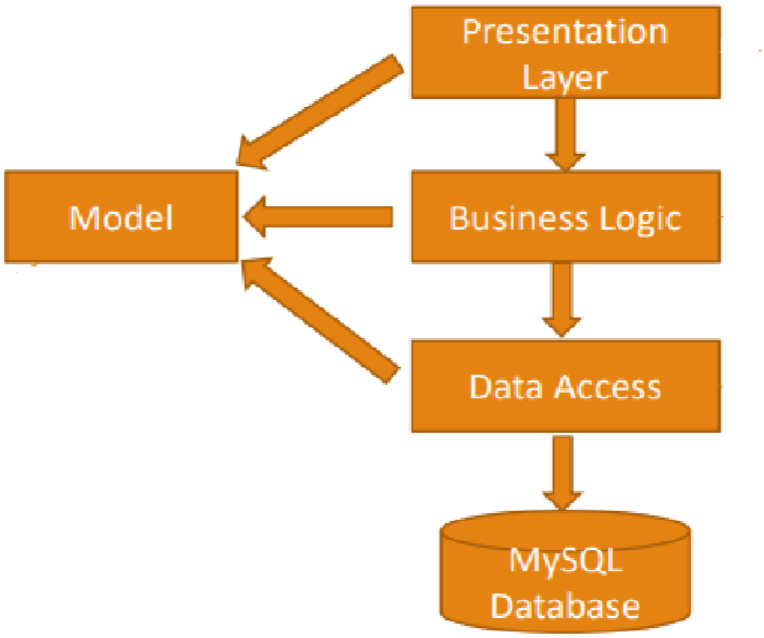
| | |
|---|---|
| 1. Analiza si intelegerea problemei. | Capitolul 1 |
| 2. Folosirea unei baze de date pentru memorarea datelor despre client, comenzi si produse. | Capitolul 3 |
| 3. Modelarea problemei | Capitolul 2 |
| 4. Implementarea propriu zisa a aplicatiei | Capitolul 4 |
| 5. Folosirea unei interfete grafice “User Friendly” – folosind Java Swing | Capitolul 5 - Rezultate |

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

Cel mai comun model de arhitectură este modelul de arhitectură stratificat, altfel cunoscut sub numele de model de arhitectură n-tier. Acest model este standardul de facto pentru majoritatea aplicațiilor Java EE și, prin urmare, este cunoscut pe scară largă de majoritatea arhitecților, designerilor și dezvoltatorilor. Modelul de arhitectură stratificat se potrivește îndeaproape cu structurile tradiționale de comunicare și organizare IT întâlnite în majoritatea companiilor, ceea ce îl face o alegere naturală pentru majoritatea incercarilor de dezvoltare a aplicațiilor de afaceri.

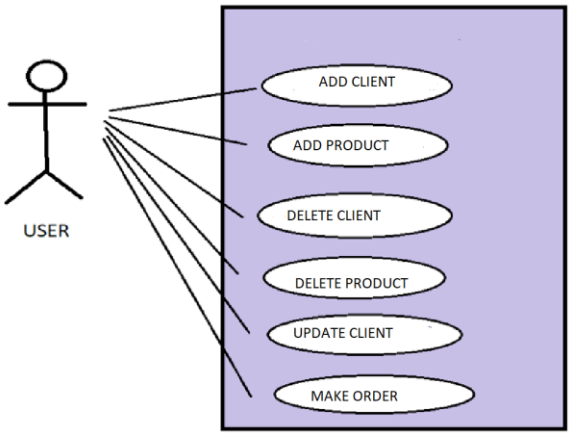
Componentele din modelul de arhitectură stratificată sunt organizate în straturi orizontale, fiecare strat îndeplinind un rol specific în cadrul aplicației (de exemplu, logica de prezentare - BLL sau logica de afaceri - DAO). Deși modelul de arhitectură stratificată nu specifică numărul și tipurile de straturi care trebuie să existe în model, majoritatea arhitecturilor stratificate constau din patru straturi standard: prezentare, afaceri, persistență și bază de date. În unele cazuri, stratul de afaceri și stratul de persistență sunt combinate într-un singur strat de

afaceri, în special atunci când logica de persistență (de exemplu, SQL sau HSQL) este încorporată în componentele stratului de afaceri. Astfel, aplicațiile mai mici pot avea doar trei straturi, în timp ce aplicațiile de afaceri mai mari și mai complexe pot conține cinci sau mai multe straturi.




## *Cazuri de utilizare*

Actorul principal in acest scenario este angajatul care se va ocupa de gestionarea unor comezi. Angajatorul poate selecta optiunea de client, produs sau order. Daca selecteaza produs, are optiunile de inserare, stergere si editare a unui produs. Va avea la vedere un tabel cu toate produsele existente. Daca vrea sa proceseze un produs, va apasa pe randul din tabel specific produsului. La fel va proceda si in cazul in care vrea sa introduca, sa stearga sau sa editeze un client. In cazul in care angajatul vrea sa realizeze o comanda, va alege un client si un produs, va seta cantitatea si va apasa pe un buton care v-a realiza comanda.




## *Modelarea proiectului*

Modelarea problemei a fost facuta dupa exemplul prezentat. Fiecare tabel are asignata o clasa care are ca atribute coloanele tabelului. Modelarea a fost facuta pe straturi, cum a fost cerut, astfel incat sunt 4 nivele de executie, fiecare fiind reprezentat printr-un pachet:

- **Modelul** cu clasele Client, Product si Order. In baza de date se gasesc tabele cu acelasi nume astel incat sa se poata realiza legatura.
- **DAO** care contine clasele ClientDAO, ProductDAO, OrderDAO si AbstractDAO. In AbstractDAO se gasesc queryurile pentru fiecare tip de actiune catre baza de date.
- **BLL** contine clasele ClientsBLL, ProductBLL, OrderBLL. Acestea sunt responsabile cu definirea concreta a actiunilor pentru fiecare clasa. Din aceste clase se apleaza metodele din DAO.
- **Presentation**, care contine Clasele Controller, View, ClientGUI, OrderGUI, ProductGUI si Properties.

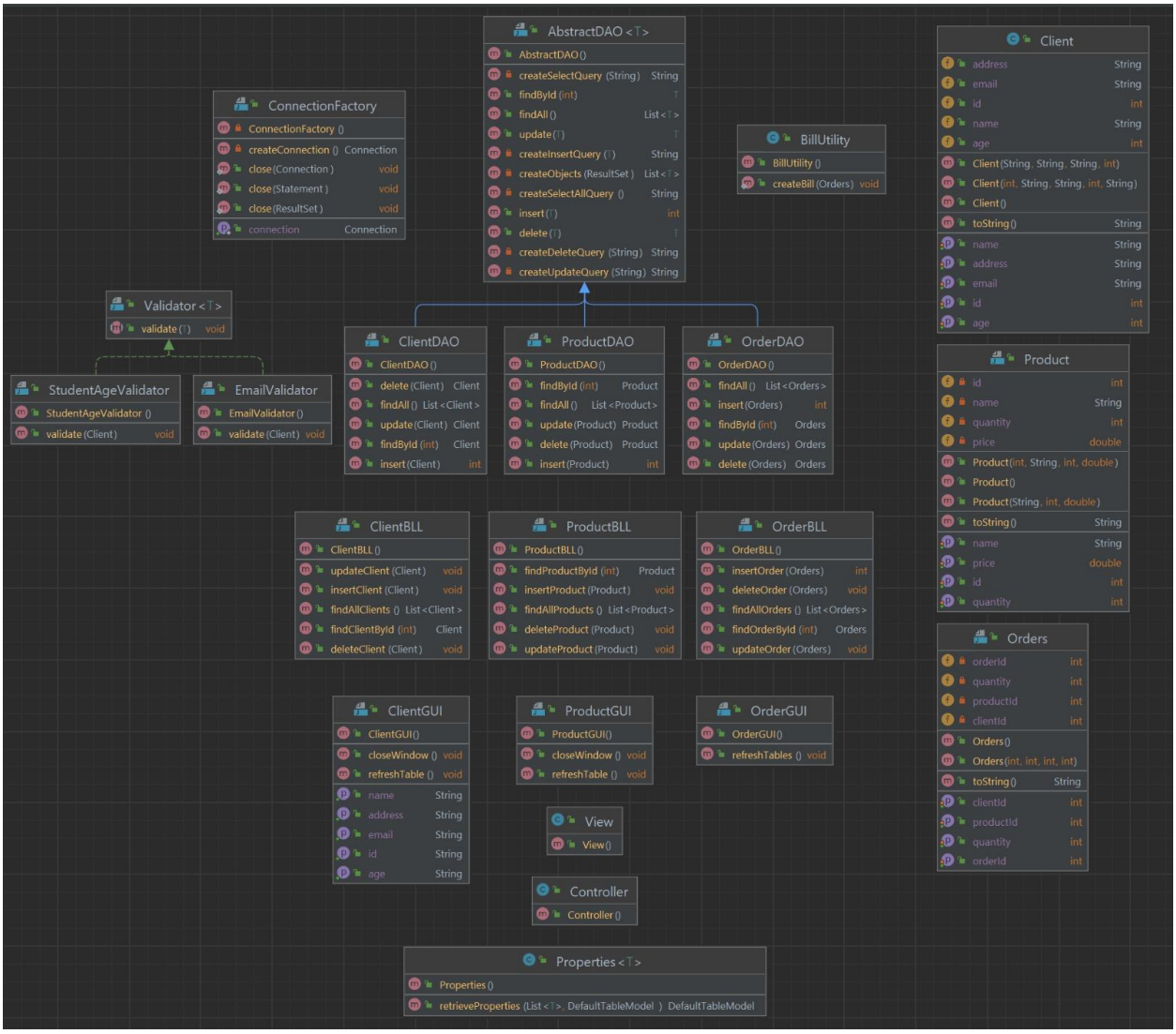
# 3. Proiectare

## 3.1. Diagrama UML

Unified Modeling Language sau UML pe scurt este un limbaj standard pentru descrierea de modele si specificatii pentru software. UML a fost la bază dezvoltat pentru reprezentarea complexității programelor orientate pe obiect, al căror fundament este structurarea programelor pe

clase, și instanțele acestora ( numite și obiecte ). Cu toate acestea, datorită eficienței și clarității în reprezentarea unor elemente abstracte, UML este utilizat dincolo de domeniul IT.

Diagrama UML generata pentru proiectul acesta este urmatoarea:




## *3.2. Structuri de date folosite*

Listele in Java ofera mentinerea colectiei ordonate. Listele contin metoda de inserare, de actualizare,de stergere si de cautare a elementelor, fiind mult mai eficiente decat array-urile. De asemenea listele permit elemente duplicate si pot retine elemente nule.

In acest proiect am folosit liste pentru salvarea obiectelor de acelasi tip intr-o colectie ca mai apo isa le pot gestiona mai usor.

# 4. Implementare

## 4.1. Clase si metode

Proiectarea aplicatiei se face prin intermediul pachetelor, pe nivele. Astfel fiecare pachet face apel doar la niveluri de sub.

### 1. PACHETUL CONNECTION

- contine o singura clasa si anume ConnectionFactory. Astesta clasa realizeaza conexiunea cu baza de date. Are ca variabile instanta LOGGER, DRIVER, DBURL, USER si PASS. Ca metode se gasesc constructor, createConnection(), metode de get pentru conexiune, metoda de inchidere a conexiunii, close().

### 2. PACHETUL MODEL

Contine urmatoarele clase:

- ***Clasa Client*** simuleaza un client care are ca variabile instanta id, nume, adresa, email, varsta. Clasa contine doar gettere si settere si contructori.
- ***Clasa Product*** simuleaza un produs care are ca variabile instanta id, nume, pret, cantitate. Clasa contine doar gettere si settere si constructori.
- ***Clasa Order*** simuleaza o comanda care are ca variabile instanta id, idClient, idProdus, cantitate. Clasa contine doar gettere si settere si constructori.

Ideea din spatele modelui este sa aiba clase care sa coincide cu tabelele din baza de date. Astfel se poate folosi reflexia daca denumim field-urile la fel ca atributele din baza de date.

### 3. PACHETUL DAO

Acest pachet contine clasele care contin query-urile si face conexiunea intre aplicatia java si baza de date.

- ***Clasa ClientDAO*** extinde clasa AbstarctDAO si face legatura cu baza de date prin apelarea metodei din ConnecionFactory si se implementeaza query-urile pentru clasa/tabelul Clients. Are o singura variabila instanta, si anume LOGGER. Metodele din aceasta clasa sunt insert, delete, update, findAll si findById. In clasa nu se afla implementarea propriu zisa a metodelor, deoarece se apeleaza cu super metoda respectiva din clasa AbstractDAO.
- ***Clasa ProductDAO*** extinde clasa AbstarctDAO si face legatura cu baza de date prin apelarea metodei din ConnecionFactory si se implementeaza query-urile pentru clasa/tabelul Product. Are o singura variabila instanta, si anume LOGGER. Metodele din aceasta clasa sunt insert, delete, update, findAll si findById. In clasa nu se afla implementarea propriu zisa a metodelor, deoarece se apeleaza cu super metoda respectiva din clasa AbstractDAO.
- ***Clasa OrdersDAO*** extinde clasa AbstarctDAO si face legatura cu baza de date prin apelarea metodei din ConnecionFactory si se implementeaza query-urile pentru clasa/tabelul Orders. Are o singura variabila instanta, si anume LOGGER. Metodele din aceasta clasa sunt insert, delete,

update, findAll si findById. In clasa nu se afla implementarea propriu zisa a metodelor, deoarece se apeleaza cu super metoda respectiva din clasa AbstractDAO.

- ***Clasa AbstractDAO***
  Metode:
  - public List<T> findAll() – returneaza lista de obiecte de tipul T, T reprezentand tipul elementelor dintr-o tabela. Se face conexiunea cu baza de date si se apeleaza query-ul createSelectAllQuery().
  - public <T> dindById(int id) – returneaza un obiect care are id-ul respectiv. Mai intai se face conexiunea cu baza de date si se trimite la query-ul createSelectQuery id-ul respectiv. In cazul in care obiectul a fost gasit, acesta se returneaza.
  - public List<T> createObjects(ResultSet resultSet) – fiind dat ResultSet, returneaza o lista de obiecte de tipul T. Pentru fiecare resultat din ResultSet se creeaza o noua instanta de tipul T. Pentru fiecare field al clasei T se extrage din rezultatul curent valoarea field-ului curent, se obtine setter-ul pentru a seta valoarea la field-ul respectiv si folosind metoda de obtain se seteaza valoarea field-ului.
  - public int insert(T t) – insereaza un obiect de tipul T in tabela specifica lui T. De asemenea, metoda returneaza id-ul specific obiectului inserat in baza de date, ceea ce voi volosi ulterior in metodele din BLL.
  - public T update(T t) – modifica field-urile coresunzatoare obiectului care are id-ul obiectului t.
  - public T delete(T t) – sterge obiectul t din tabelul specific lui T. In mod asemanator cu metodele anterioare, se face conexiunea cu baza de date si se cauta obiectul t de tipul T pentru a putea fi sters ulterior.

## 4. PACHETUL BLL

**Clasa ClientBLL** are ca variabile instanta Lista de validatori si clientDAO care se vor instatia in constructor. Ca metode are urmatoarele:

-findClientById(int id) care primeste un id si apeleaza metoda findById din clasa ClientDAO pentru a gasi clientul respective, apoi il returneaza.

-insertClient(Client client) – apeleaza metoda de insert din clasa ClientDAO si insereaza clientul respectiv in baza de date.

-deleteClient(Client client) – primeste un client ca parametru si apeleaza metoda de delete din ClientDAO pentru a sterge clientul respectiv.

-updateClient(Client client) - primeste un client ca parametru si apeleaza metoda de update din ClientDAO pentru a sterge clientul respectiv.

-findAllClients() – salzeaza intr-o lista de client lista de client returnata de metoda findAll din clientDAO.

**Clasa ProductBLL** are ca variabila instanta productDAO care se v-a instatia in constructor. Ca metode are urmatoarele:

-findProductById(int id) care primeste un id si apeleaza metoda findById din clasa ProductDAO pentru a gasi produsul respective, apoi il returneaza.

-insertProduct(Product product) – apeleaza metoda de insert din clasa ProductDAO si insereaza produsul respectiv in baza de date.

-deleteProduct(Product product) – primeste un produs ca parametru si apeleaza metoda de delete din ProductDAO pentru a sterge produsul respectiv.

-updateProduct(Product product) - primeste un client ca parametru si apeleaza metoda de update din ProductDAO pentru a sterge produsul respectiv.

-findAllProducts() – salzeaza intr-o lista de produse lista de produse returnata de metoda findAll din productDAO.

**Clasa OrderBLL** - are ca variabile instanta orderDAO si productDAO care se vor instatia in constructor. Ca metode are urmatoarele:

-findOrderById(int id) care primeste un id si apeleaza metoda findById din clasa OrderDAO pentru a gasi comanda respectiva, apoi o returneaza.

-insertOrder(Order order) – cauta produsul specific comezii pentru a putea scadea din cantitatea totala de produse numarul de produse specific comenzii, iar in cazul in care nu sunt suficiente produse in stoc se afiseaza un mesaj de eroare. Dupa ce se da update la produs se apeleaza inserarea din orderDAO.

-deleteOrder(Order order) – primeste o comanda ca parametru si apeleaza metoda de delete din OrderDAO pentru a sterge comada respective.

## 5. PACHETUL PRESENTATION

Contine urmatoarele clase:

**Clasele View, ClientGUI, ProductGUI si OrderGUI** care vor fi prezentate in urmatorul capitol.

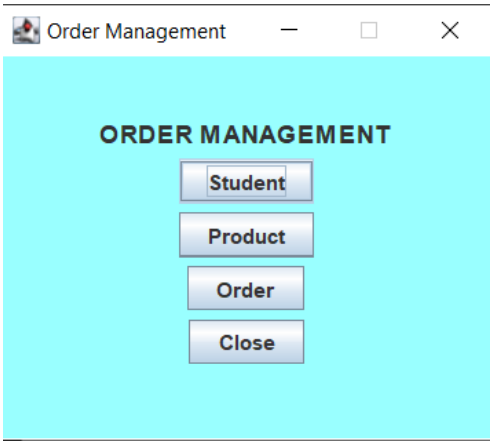
**Clasa controller** care instantiaza View-ul.

**Clasa Properties** care are o singura metoda, si anume retriveProperties(List<T> objects, DefautTableModel tableModel) care returneaza un defaultTableModel care va avea ca header-uri la coloane numele field-urilor specific clasei T si va contine pe linii obiectele de tipul T trimise ca argumente.

## 4.2. Graphical User Interface

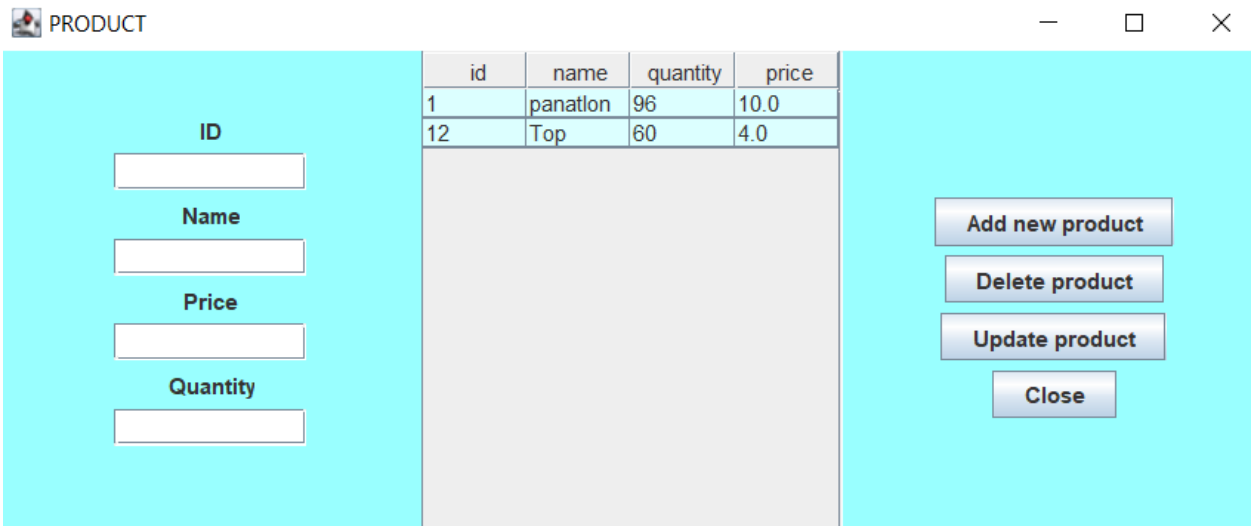
Pentru realizarea interfetei grafice am implementat 4 clase diferite.

**Clasa View** contine pagina principala de unde utilizatorul poate apasa unul din butoanele “Order”,”Client” si “Product”. In functie de ce buton apasa, se va deschide o noua fereasta specifica butonului respectiv. Ca sa realizez asta, am pus actionListener pe fiecare buton, iar in momentul in care unul este apasat se face o instanta noua pentru clasa specificata.

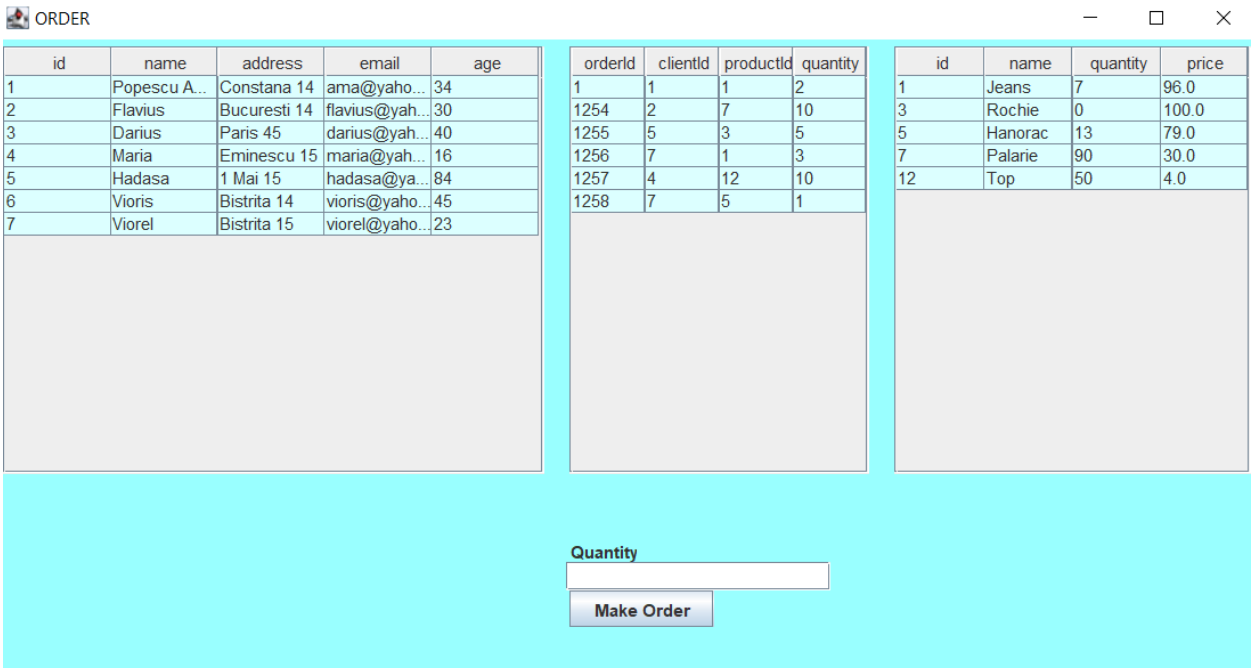



**Clasa ClientGUI** contine un JTable care contine toti clientii prezenti in baza de date, field-uri speficice pentru atributele clientului(id, nume, adresa, email, varsta) si nu in ultimul rand butoane pentru stergerea, adaugarea sau modificarea unui client. Dupa fiecare apasare a unui buton se apeleaza functia refreshTable care sterge toate datele din tabel si le insereaza iar pentru a vedea modificarile in timp real.

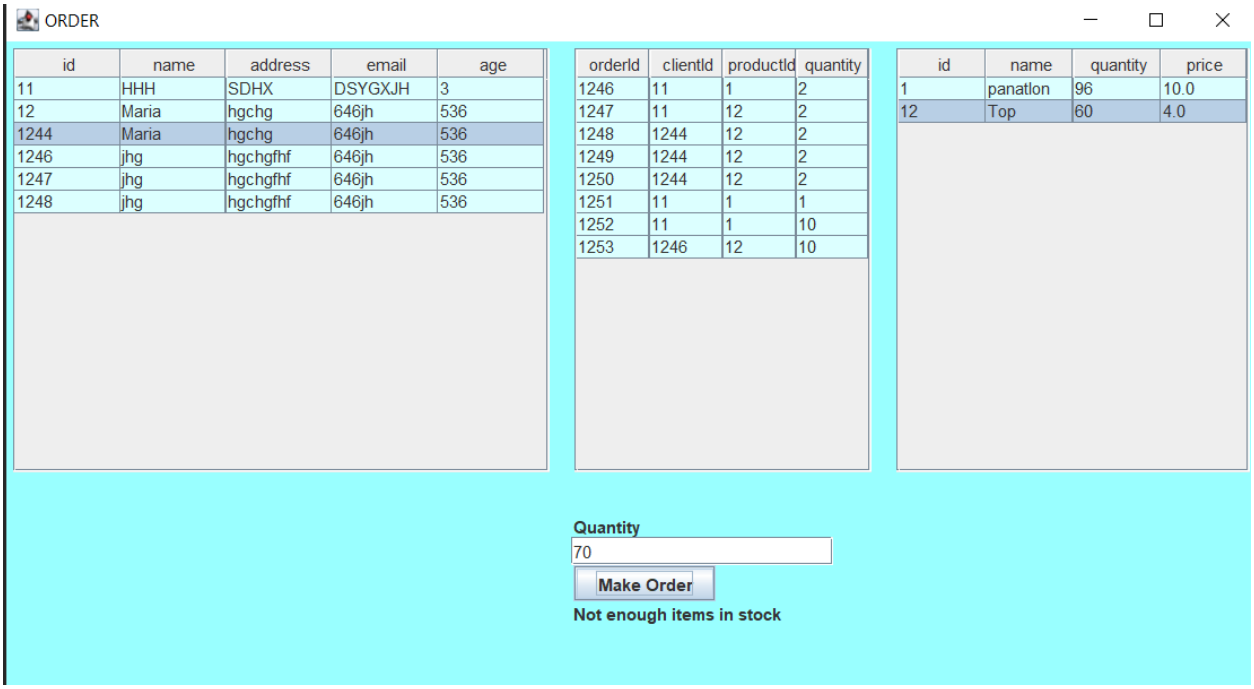

**Clasa ProductGUI** contine un JTable care contine toate pprodusele prezente in baza de date, field-uri speficice pentru atributele produsului(id, nume,pret, cantitate) si nu in ultimul rand butoane pentru stergerea, adaugarea sau modificarea unui produs. Dupa fiecare apasare a unui buton se apeleaza functia refreshTable care sterge toate datele din tabel si le insereaza iar pentru a vedea modificarile in timp real.




**Clasa OrderGUI** contine 3 JTable-uri. Primul reprezinta tabelul cu client, al doilea reprezinta lista cu comenzile, iar al treilea reprezinta lista cu produsele. Cu ajutorul unui mouseListener se va selecta un rand din tabelul clientilor si un rand din tabelul produselor, iar dupa ce se va introduce o valoare in JTextField-uri pentru cantitate, se va apasa butonul make order si de va face automat o comanda noua care va fi inserata in tabel.



In cazul in care nu sunt suficiente produse in stoc se va afisa un mesaj de eroare sub butonul Make Order, iar comanda nu va fi realizata.




## 5. Rezultate

Pentru testare am creeat 7 clienti si 5 produse, apoi am realizat 3 comenzi. Fisierele generate in urma realizarii cu succes a comenzilor sunt:

---

Factura 1254

Client: Flavius

Email: flavius@yahoo.com

Adresa de livrare: Bucuresti 14

Produs: Palarie  Pret: 30.0

Cantitate: 10

Pret total: 300.0

---

Factura 1255

Client: Hadasa

Email: hadasa@yahoo.com

Adresa de livrare: 1 Mai 15

Produs: Rochie  Pret: 100.0

Cantitate: 5

Pret total: 500.0

---

Factura 1257

Client: Viorel

Email: viorel@yahoo.com

Adresa de livrare: Bistrita 15

Produs: Jeans  Pret: 96.0

Cantitate: 3

Pret total: 288.0

## 6. Concluzii

Prin urmare, acest proiect a fost util pentru aprofundarea cunostintelor acumulate la limbajul de programare Java. Am invatat cum sa folosesc reflexia in contextual comunicarii cu baza de date mySQL. De asemenea, am inteles utilitatea structurarii proiectului in presentation layer, business layer, data access layer si model.

Ca dezvoltare ulterioara s-ar mai putea lucra la interfata grafica. Ceea ce s-ar mai putea imbunatati, folosind regex-ul, este atunci cand utilizatorul introduce ceva gresit, sa se genereze un mesaj “Input gresit” si sa se stearga automat respectivul input, cerandu-i sa il introduca din nou corect.

# 7. Bibliografie


- Java Swing: https://docs.oracle.com/javase/tutorial/uiswing/
- Cursul de OOP din semestrul 1
- https://jenkov.com/tutorials/java-reflection/index.html
- https://dzone.com/articles/layers-standard-enterprise