

Documentație laborator tema 3

Gestionarea comenzilor

Tehnici fundamentale de programare

Student: Dunca Denisa Mihaela

Grupa: 30227

1. **Obiectivul temei**

Cerința temei este de a crea și implementa o aplicație pentru gestionarea de comenzi într-un depozit. Aplicația lucrează pe o bază de date și utilizatorul poate sa introducă, să șteargă sau să modifice clienți, produse si comenzi, în limita unui stoc disponibil. Pentru implementare am folosit modelul arhitectural Layers.

Tema de laborator “Gestionarea comenzilor” are ca și obiectiv principal crearea unui mod mai ușor prin care utilizatorul poate sa lucreze cu tabelele dintr-o baza de date, iar aplicația respectă paradigmele de programare orientată pe obiect folosind limbajul de programare Java.

Obiectivele secundare ale proiectului:

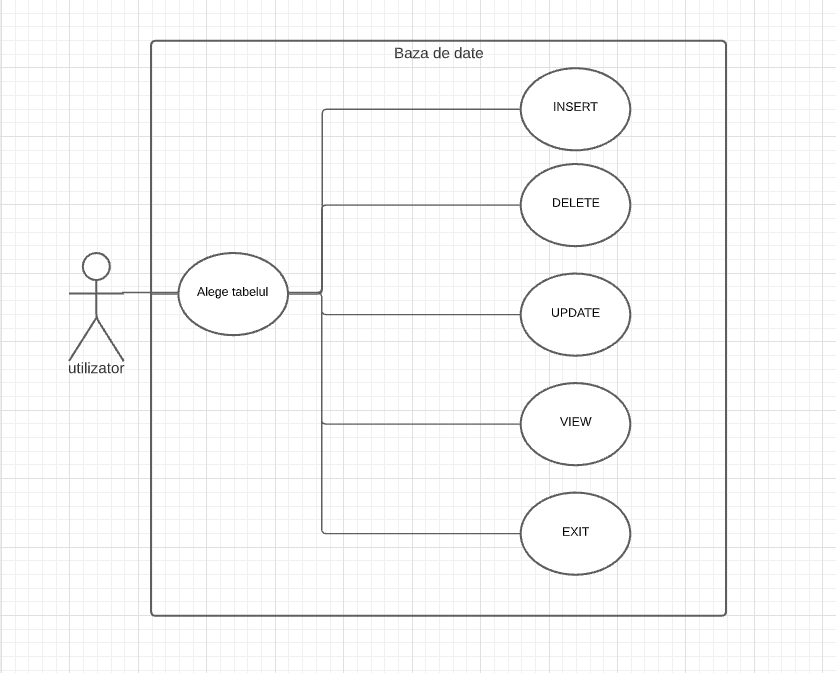
* Analizarea problemei prin înțelegerea corectă a cerinței și modelarea de scenarii și cazuri de utilizare.
* Proiectarea unei soluții care să se plieze pe toate cazurile pe care le introduce utilizatorul.
* Implementarea soluției alese prin scrierea codului implicit și a interfeței utilizator.
* Observarea rezultatelor finale și concluziile.

1. **Analiza problemei**

Analiza problemei a constat prin înțelegerea pe deplin a cerinței și a informațiilor pe care dorim să le primim la generarea unui rezultat final. S-a creat o schemă inițială care trebuie să prezinte în mare ceea ce se urmărește prin acest proiect. Pe de o parte, este necesară crearea unei baze de date și a unei conexiuni cu aceasta prin clasa ConnectionFactory, pe de altă parte proiectul trebuie implementat folosind Layers și validatori, precum și o clasă ambstractă DAO (Data Access Object) care să conțina principalele metode.

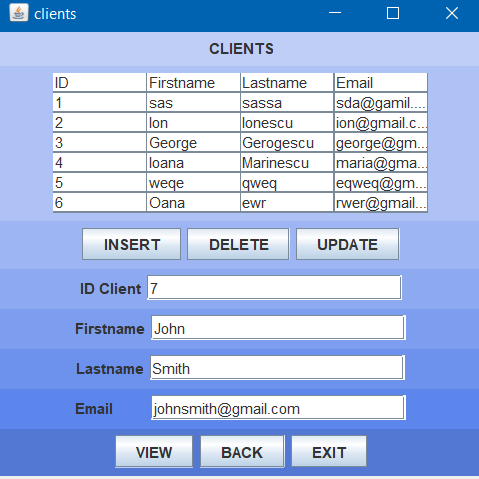
Aplicația are trei tabele: Client, Product și Order, fiecare are diferite atribute și câte o cheie primară ID. Tabelul Client conține date specific clienților, precum: id, nume, prenume și email, la care am pus un validator. Tabelul Product are date despre produsele din deposit și anume id, nume, preț si cantitate, iar tabeleul Order face legatura intre cele doua tabele de mai devreme conținand idOrder, idClient, idProduct si cantitatea comenzii.

Pe fiecare table, utilizatorul poate sa introduca de la tastatura date noi in tabel, poate sa le stearga si poate sa le verifice. Am create si un buton de view care permite utilizatorului sa dea un refresh pentru a vedea ultimele modificari facute. Un mod prin care se poate scoate în evidență această structură este generarea unei Diagrame USE CASE.



În diagrama de mai sus este prezentată structura logică a proiectului. Sistemul este aplicația în sine prin care utilizatorul poate sa introduca date in cele 3 tabele. Introducerea datelor este dependentă de actor, precum și apăsarea pe butoanele de pe interfata și ieșirea din program. După ce au fost introduse datele și aplicația este pornită, utilizatorul poate sa aleaga intre cele 3 operatii de modificare a tabelelor si anume insert, delete si update.

Un scenariu de utilizare ar fi:

* User-ul alege sa modifice tabelul Client

Ex: ID Client: 7

Firstname: John

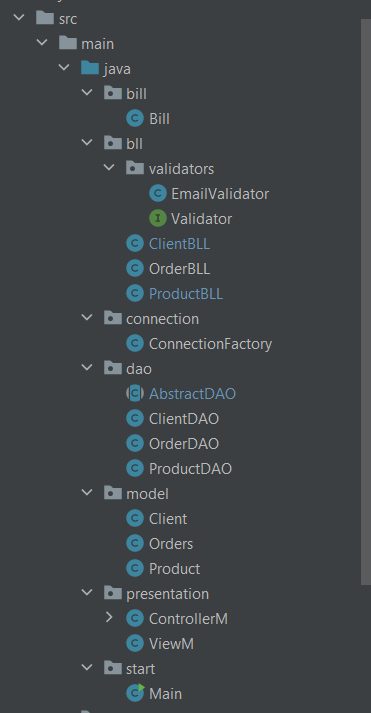
Lastname: Smith

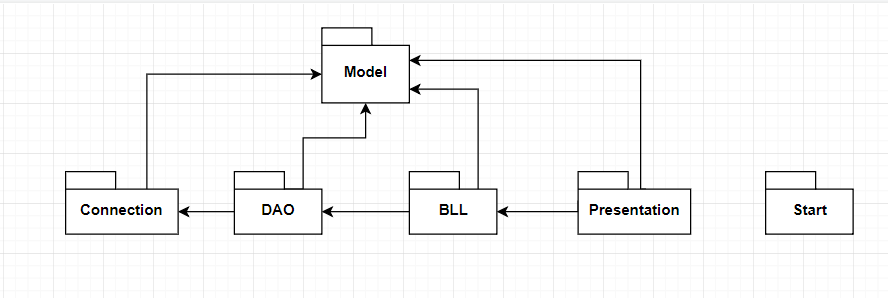
Email: johnsmith@gmail.com

* User-ul alege sa introduca datele pus in tabel deci apasa pe butonul INSERT
* Pentru a vedea tabelul modificat apasa pe butonul View care da un refresh la window
* Poate sa se intoarca la pagina principala folosind butonul BACK sau poate sa iasa din aplicatie folosind butonul EXIT.

Aplicatia a fost realizata in pachete si anume presentation contine clasele care definesc interfata grafica( Contorller si View) , bll (business logic layer), contine clasele care incapsuleaza logica aplicatiei si anume validatori si clasele ClientBLL, ProductBLL si OrderBLL, connection face conexiunea bazelor de date cu limbajul java, dao

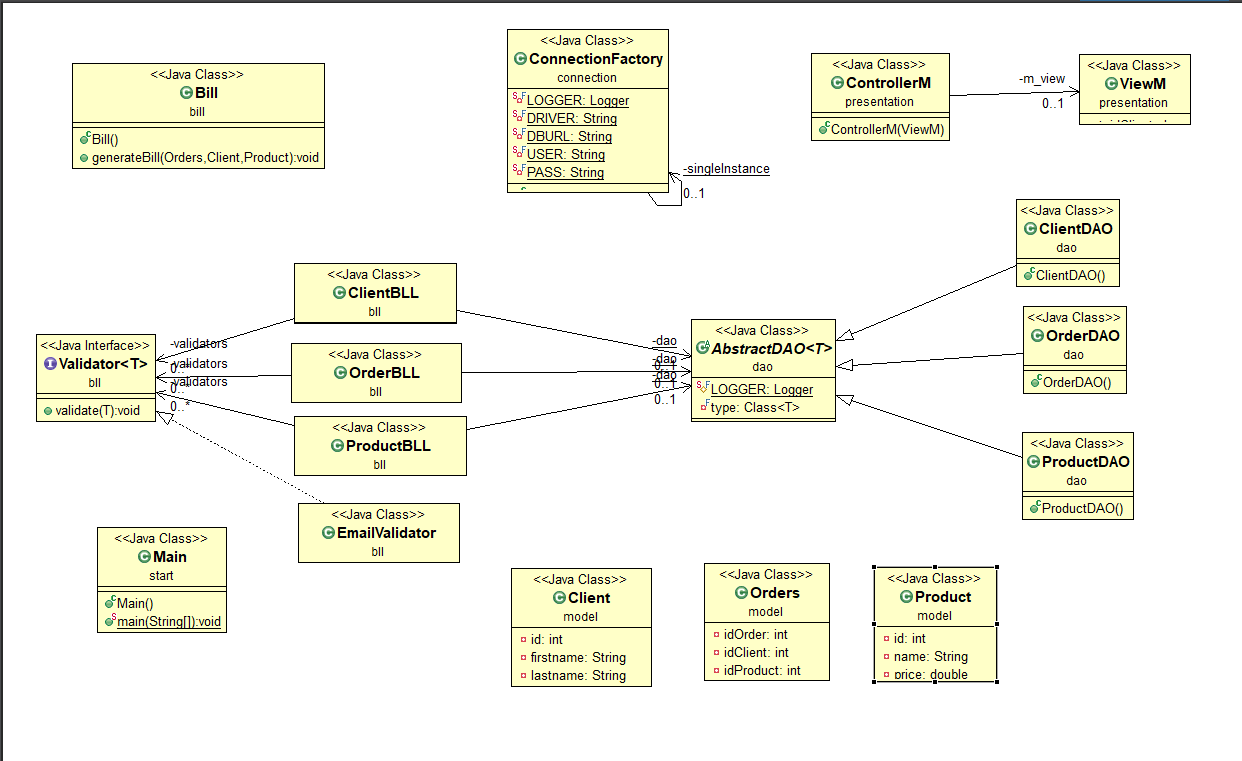
(Data Access Object ) contine clasele care genereaza interogarile in bazele de date, model contine clasele care au corespondent cate o tabela (Client, Order sau Product), iar pachetul start este un pachet cu o singura clasa, si anume clasa main care ruleaza programul.





1. **Proiectare**

Trecând la partea practică de rezolvare a problemei, și anume proiectarea în limbaj OOP, au fost urmăriți mai mulți pași, primul fiind crearea unei Diagrame de pachete urmărind modelul MVC (Model-View-Controller), dar si modelul Layers folosind Business Layer, Presentation Layer, Model Layer. Controllerul face legătura dintre clasele din Model și View prin intermediul butoanelor apăsate.

Intrând într-o analiză mai amănunțită a proiectului, am generat și diagrama UML. Aici se pot vedea în detaliu legăturile dintre clase, metodele folosite, precum și structura logică a întregului proiect.

**Implementare**

**Pachetul bll**

Pachetul bll (Business Logic Layer) contine, atat validatori cat si trei clase in care sunt aplicate metodele din DAO (Data Access Object) cu validatori

**Validators**

Este o interfata numita validator ce este implementata de clasa EmailValidator care valideaza daca informatia introdusa de utilizator in textfieldul dedicat emailului indeplineste formatul unui email.

**ClientBLL/ OrderBLL/ ProductBLL**

Cele trei clase cheama metodele de insert, delete respectiv update din DAO (Data Access Object) si le adauga o verificare suplimentara pentru validatori. Pe langa operatiile principale am mai creat doua metode, getAll care, folosind reflexia ia datele din data base si le pune intr-un tabel pentru ca acesta sa poata fi afisat in interfata si metodele getClient, getPorduct si getOrder, care fac o verificare suplimentara pentru a sti atunci cand se creaza chitanta de unde trebuie sa ia informatia verificand egalitatea dintre id-uri.

**ClientBLL**

Implementeaza business logic pentru clienti si apeleaza metode specifice pe un obiect din clasa abstracta DAO (Data Access Object) ce opereaza pe obiecte din clasa Client. Aici sunt toate metodele ce sunt implementate in dao si anume, inserare, stergere si update, dar pe langa aceastea, se afla si o metoda getAll care imi creaza tabela clienti luand informatia direct din baza de date si o pun in tabelul specific clientilor cu tot cu headere implementate.

**ProductBLL**

Implementeaza business logic pentru produse si apeleaza metode specifice pe un obiect din clasa abstracta DAO (Data Access Object) ce opereaza pe obiecte din clasa Product. Aici sunt toate metodele ce sunt implementate in dao si anume, inserare, stergere si update, dar pe langa aceastea, se afla si o metoda getAll care imi creaza tabela product luand informatia direct din baza de date si o pun in tabelul specific ptoduselor cu tot cu headere implementate.

**OrderBLL**

Implementeaza business logic pentru comenzi si apeleaza metode specifice pe un obiect din clasa abstracta DAO (Data Access Object) ce opereaza pe obiecte din clasa Orders. Aici sunt toate metodele ce sunt implementate in dao si anume, inserare, stergere si update, dar pe langa aceastea, se afla si o metoda getAll care imi creaza tabela order luand informatia direct din baza de date si o pun in tabelul specific comenzilor cu tot cu headere implementate.

**Pachetul dao**

Contine o clasa abstracta DAO care este alcatuita din toate metodele necesare pentru a face operatii pe tabele. Aceste metode sunt metode generale care pot fi apelate pentru orice tip de data indiferent de tabel. ClientDAO, ProductDAO si OrderDAO implementeaza aceasta clasa si folosesc aceste metode.

**AbstractDAO**

Clasa AbstractDAO implementeaza metode generale pentru clasele din model si corespondentele acestora prin intermediul reflexiei. Aici se folosește tehnica de reflexie în Java pentru a nu fi nevoie să scriem mai multe metode similare particularizate pentru fiecare clasă ce modelează obiectele aplicației. În acest sens, a fost definit un tip generic asociat clasei AbstractDAO, care este o clasă abstractă.

Fiecare operatie pe tabel este alcatuita din doua metode, una care selecteaza folosind SQL datele din tabel si una care gaseste in ce tabel trebuie introduse, sterse sau actualizate. În acest pachet se folosește tehnica reflexiei, așa încât introducerea, ștergerea și interogarea în baza de date se fac folosind aceleași metode, indiferent de tipul obiectului instanță al unei clase din pachetul Model. Operațiile care sunt specifice fiecărei clase Model sunt implementate prin metode în afara clasei abstracte AbstractDAO (aici se folosește reflexia).

**Pachetul model**

**Client/ Product/ Order**

In pachetul model sunt cele 3 clase Client, Product si Orders cu atributele lor care corespund cu tabelele facute in baza de date.

**Pachetul presentation**

Conține clase destinate interacțiunii cu utilizatorul, mai exact Controller-ul si View-ul.

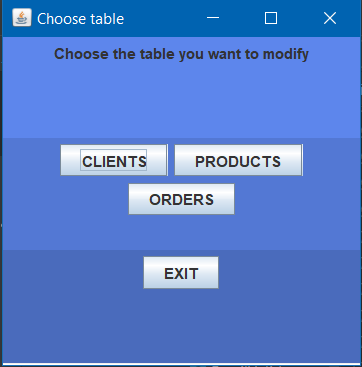
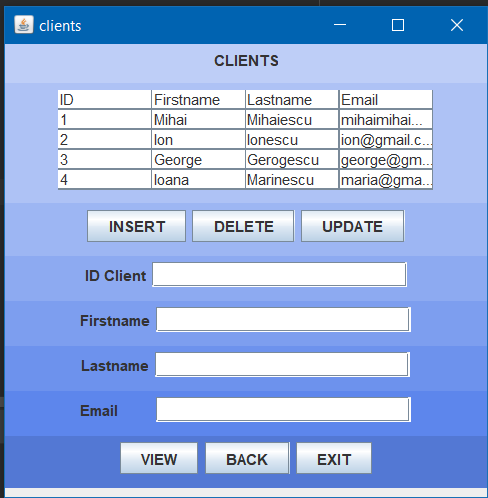
**ControllerM**

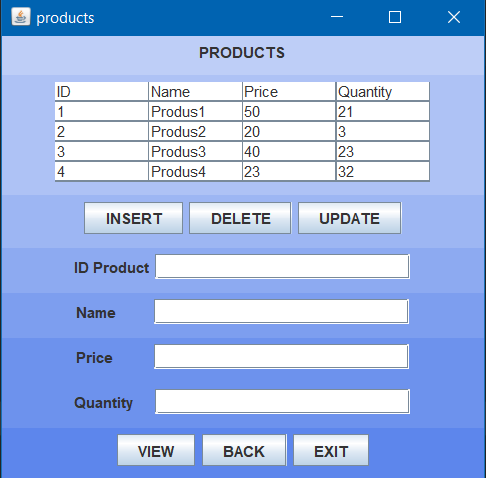
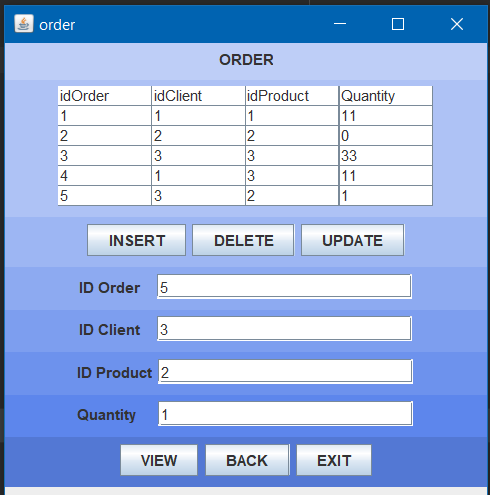
In Controller se intampla toata logica proiectului. Toate butoanele au listenere si aici sunt apelate toate metodele implementatea in celelalte clase. Tot aici am apelat si metoda de generare a chitantei la inserarea unui nou order in tabel.

**ViewM**

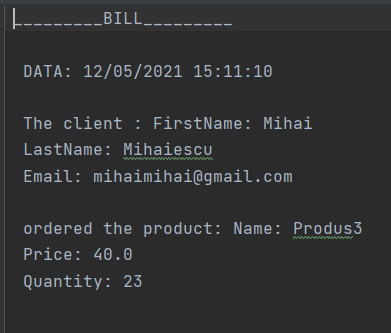
In view am creat toata interfata cu mai multe frame uri si mai multe metode care sa ma ajute sa ajung la revultatul final a proiectului. Am creat si o metoda de refresh care sterge ce am pe framul initial si apoi apeleaza din nou framul care trebuie afisat dar de data aceasta este actualizat. View contine mai multe variabile de tipul textfield, button si 3 variabile de tipul JTabel.

Am facut si 3 metode de refresh pe care le folosesc atunci cand apas pe butonul de view. Metodele sunt diferite in functie de tabelul pe care vrem sa lucram si presupune sa nu mai faca vizibil frame-ul apoi sa apeleze functia dispose pentru a sterge informatia din frame, apoi adauga din nou framul actualizat in functie de tabel si la final in il face vizibil.



****

**ConnectionFactory**

**** Conține doar clasa ConnectionFactory, care realizează conexiunea la baza de date folosind o singură instanță de obiect Connection, prin implementarea modelului arhitectural Singleton. De asemnea, sunt puse la dispoziție metode de obținere și închidere a conexiunii,precum și de închidere a unui PreparedStatement și un ResultSet. Prin clasa de connection se realizeaza conectarea cu baza de datele si toate informatiile deja introduse.

**Bill**

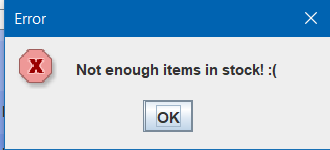
Clasa Bill are rolul de a genera un fisier \*.txt care atunci cand este aplicat un insert in tabelul order se creaza o chitanta automat cu informatii legate de data, ora, client si produs. Am creat un dateformat pe care sa il poata introduce in fisierul final si un string care are toate informatiile din poza de alaturi.

**Pachetul start**

Conține clasa Main, cu metoda main, de unde începe execuția aplicației.

1. **Rezultate**

După rezolvarea problemelor din cod și ajungerea la proiectul final am testat introducand si modificand diferite date de la tastatura. Rezultatele sunt vizibile in fiecare tabela si se pot verifica operatiile de inserare, actualizare si stergere, atat pentru clienti cat si pentru produse si comenzi. Pentru fiecare comanda se poate observa, in primul rand, fisierul \*.txt generat cu o chitanta ce contine informatiile necesare despre data si ora la care s-a facut comanda, clientul si produsul in cauza, si in al doilea rand, se observa cum ca in cazul in care se insereaza sau se face update la o comanda, iar cantitatea de produse introdusa nu e mai mica sau egala cu cantitatea produselor din stoc atunci apare un mesaj de eroare care anunta ca un exista atatea produse in stoc. Un plus pe care l-as aduce proiectului ar fi ca in loc sa trebuiasca sa apas pe butonul de VIEW pentru a face un refresh la tabel, as face acest lucru odata cu inserarea, stergerea sau modificarea informatiei, pentru ca eu consider ca acolo e un buton in plus si se putea si fara el. Un alt lucru pe care l-as modifica ar fi modul prin care arata tabelul si eventual o imbunatatire si mai avansata ar fi ca sa pot sa modific informatiile direct in tabel si apoi sa dau update.



1. **Concluzii**

Per total, întregul proiect este destinat utilizării, și pentru că testărilor și verificărilor făcut aș putea spune că ar trebui să funcționeze pentru orice caz pe care utilizatorul îl introduce.

Prin acest proiect am reusit sa dobandesc cunostinte cu privire la utilizarea modelelor arhitecturale Layers, folosirea tehnicii reflexiei in Java pentru a scrie metode CRUD ce interactioneaza cu baza de date, precum si generearea unui fisier si crearea javadoc-ului. De asemenea am reusit sa inteleg cum sunt accesate datele dintr- o baza de date pentru a putea sa lucram cu ele direct dintr-o interfata si mi-am imbunatatit modul prin care imi creez eu interfata pe mai multe frame-uri. In plus am aprofundat cunstintele cu privire la citirea si scrierea pe o interfata, la crearea diagramelor UML precum si realizarea javadocului si a documentatiei de fata. Pe langa asta, am invatat sa utilizez validatori, in cazul de fata am folosit doar pentru verificarea daca informatia pus in texfieldul destinat emailului indeplineste formatul unui email.

1. **Bibliografie**
2. LucidChart pentru USE CASE Diagram

<https://lucid.app/lucidchart/>

<https://youtu.be/zid-MVo7M-E>

1. Javadoc

<https://www.jetbrains.com/help/idea/working-with-code-documentation.html#troubleshoot>

<https://maven.apache.org/pom.html>

<https://stackoverflow.com/questions/1843072/intellij-generate-javadoc-for-methods-and-classes>

1. MySQL

<https://www.jetbrains.com/help/idea/connecting-to-a-database.html>

1. Dump file

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