

Tehnici de programare

Laborator – Assignment 3

**Order management simulator**

Teacher Assistant: Dr. Marcel Antal

Student: Mihaila Gabriel

Group: 30229

# Assignment Objective

Obiectivul principal al assignment-ului este de a dezvolta o aplicatie care sa permita utilizatorului sa gestioneze comenzile de produse. Practic, aplicatia ar trebui sa permita utilizatorului sa gestioneze clienti, produse si comenzi. Toate acestea ar trebui sa fie construite pe o baza scalabila, care sa permita modificari usoare in viitor folosind Java Reflection.

Toate datele ar trebui stocate intr-o baza de date persistenta si in timp real. live.

Obiectivele secundare ale assignment-ului sunt, dar nu se limiteaza la:

* Avand o interfata cu utilizatorul usor de utilizat, care permite interactiunea utilizatorului cu baza de date.
* Validarea daca este posibila a cantitatii comandate, precum si a restul campurilor din cadrul tabelelor.
* Detinerea unei baze de date relationale care asigura validitatea comenzilor.
* Having a window for all the tables in the database.
* Utilizarea unei arhitecturi stratificate pentru o mai buna intelegere a codului.
* Crearea unei facturi pentru fiecare comanda care poate fi tiparita
* Utilizarea reflectiei pentru toate tipurile de date.
* Utilizati JavaDoc pentru documentarea claselor si metodelor.

# Problem analysis, modeling, scenarios, use case

## Problem analysis and modeling

Problema necesita modelarea datelor intr-un mod care poate fi procesat in Java. Astfel, l-am impartit in 3 parti: Clienti, Produse si Comenzi. Intrarea este data ca datele necesare pentru modelarea fiecarei clase. Rezultatul este dat ca facturi PDF reprezentand comenzile si ca o lista a fiecarei clase care poate fi vazuta in interfata grafica principala.

Modelul pentru fiecare dintre clase este modelat ca: Client

* id - integer
* name - String
* addres-String
* email - String
* age - integer

Product

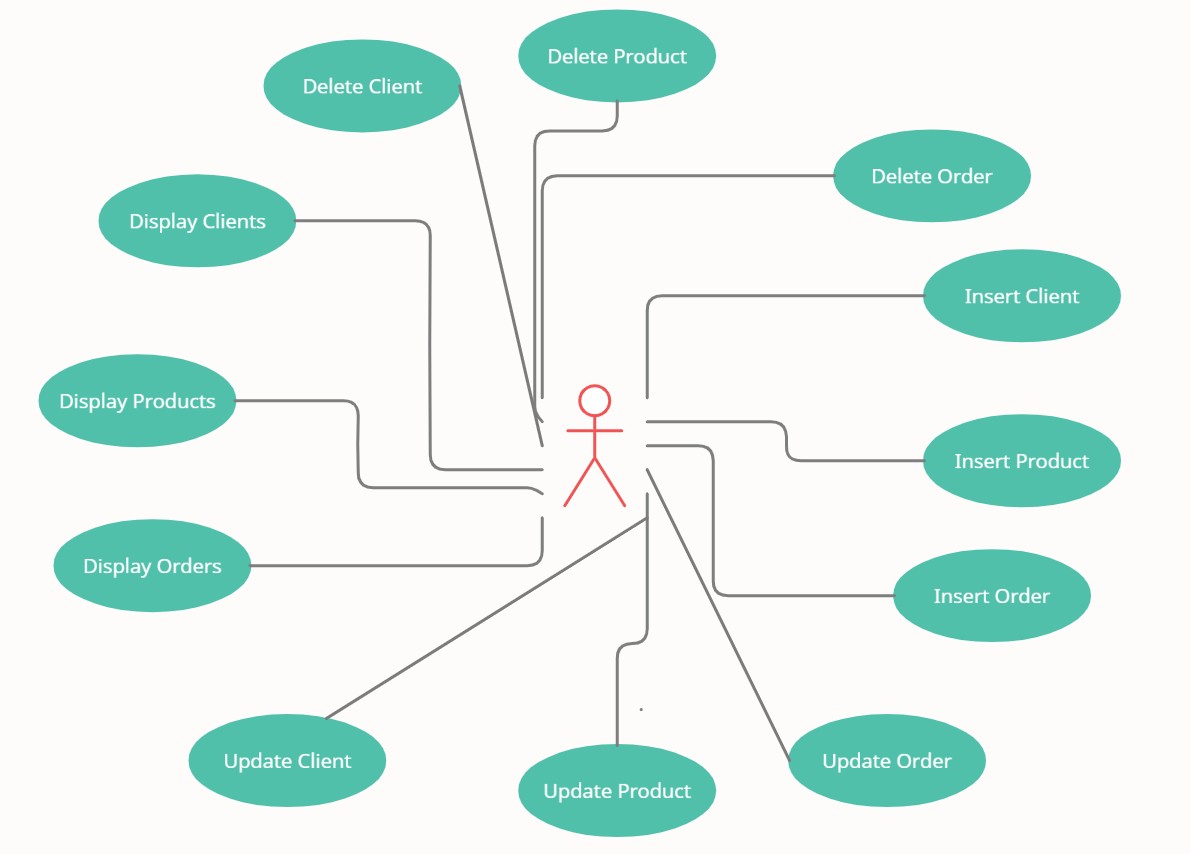
* id – integer
* name – String
* quantity – integer
* price – double

Order

* id - integer
* clientID - integer
* productID - integer
* quantity - integer

## Scenarios and use case diagram

Cazurile de utilizare sunt definite de imaginea de mai jos, actorul fiind utilizatorul care doreste sa isi proceseze datele folosind aplicatia.

O explicatie detaliata a functionarii poate fi data ca:

* Insert Client -> Insereaza un client care si-a dat numele, adresa, e-mailul si varsta.
* Insert Product -> Insereaza un produs pentru care i-a dat numele, cantitatea si pretul.
* Insert Order -> Insereaza o comanda avand in vedere cantitatea, clientul si produsul din care este compusa.
* Update Client -> Actualizeaza un client dupa ID cu noul sau nume, adresa, email si varsta.
* Update Product -> Actualizeaza un produs caruia i-a fost dat ID-ul, introducand noul nume, cantitatea noua si pretul nou.
* Update Order -> Actualizeaza o comanda dupa ce a dat ID-ul si cantitatea.
* Display Clients -> Afiseaza toti clientii din baza de date intr-un JTable
* Display Products -> Afiseaza toate produsele din baza de date intr-un JTable.
* Display Orders -> Afiseaza toate comenzile din baza de date intr-un JTable.
* Delete Client -> Sterge un client care si-a dat ID-ul.
* Delete Product -> Sterge un produs care a dat ID-ul sau.
* Delete Order -> Șterge o comandă care a dat ID-ul său.

# Design (design decisions, UML diagrams, data structures, class design, interfaces, relationship packages, algorithms, user interface)

## Design decisions

Dupa cum este impus de cerintele atribuirii, codul foloseste o arhitectura stratificata care imparte totul in 4 pachete de baza: dataAccessLayer, businessLayer, model si presentation, alaturi de un pachet de start suplimentar care contine doar clasa principala.

## Relationship Packages

Fiecare dintre pachete imparte clasele dupa cum urmeaza:

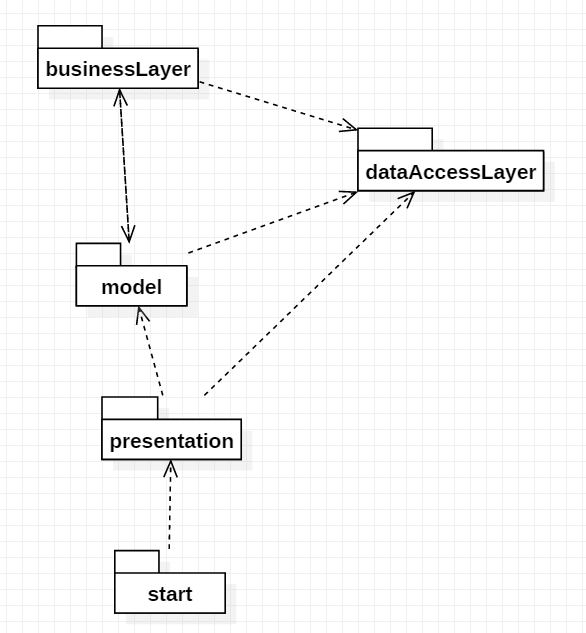
* dataAccessLayer -> Este implicat in clasele care permit citirea si scrierea datelor in baza de date intr-o maniera generica.
* businessLayer -> Este implicat cu unele superclase peste modele care permit accesul si validarea mai usoara a datelor.
* model -> Modeleaza datele intr-un mod care poate fi utilizat de limbajul de programare.
* presentation -> Este implicat in clasele care genereaza interfata grafica si iau intrarea care va fi procesata de dataAccessLayer.
* start -> Conține doar clasa principală care începe acțiunea pentru proiect.
* Vizualizarea care are toate clasele necesare pentru a afisa informatiile pe ecran si, de asemenea, citeste intrarea din primul GUI.

Diagrama arata o reprezentare mai usor de inteles a claselor si a relatiilor acestora.

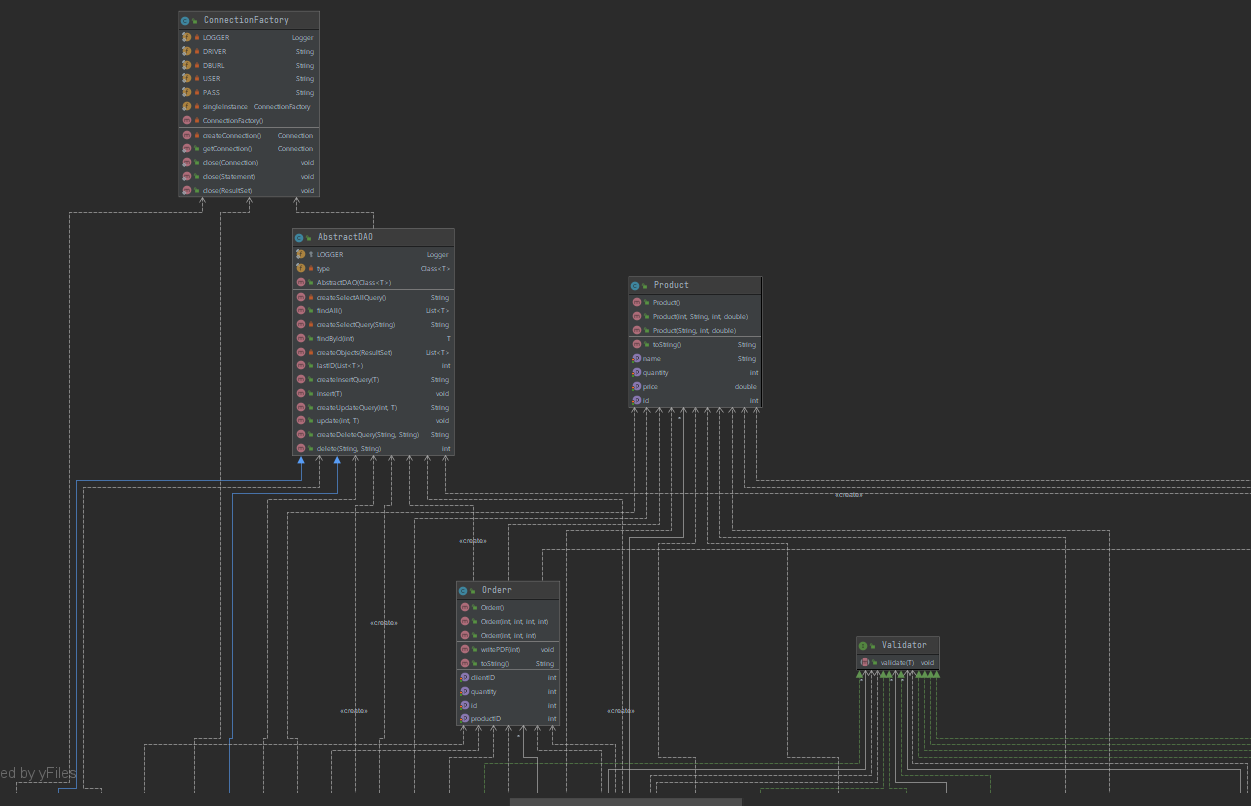
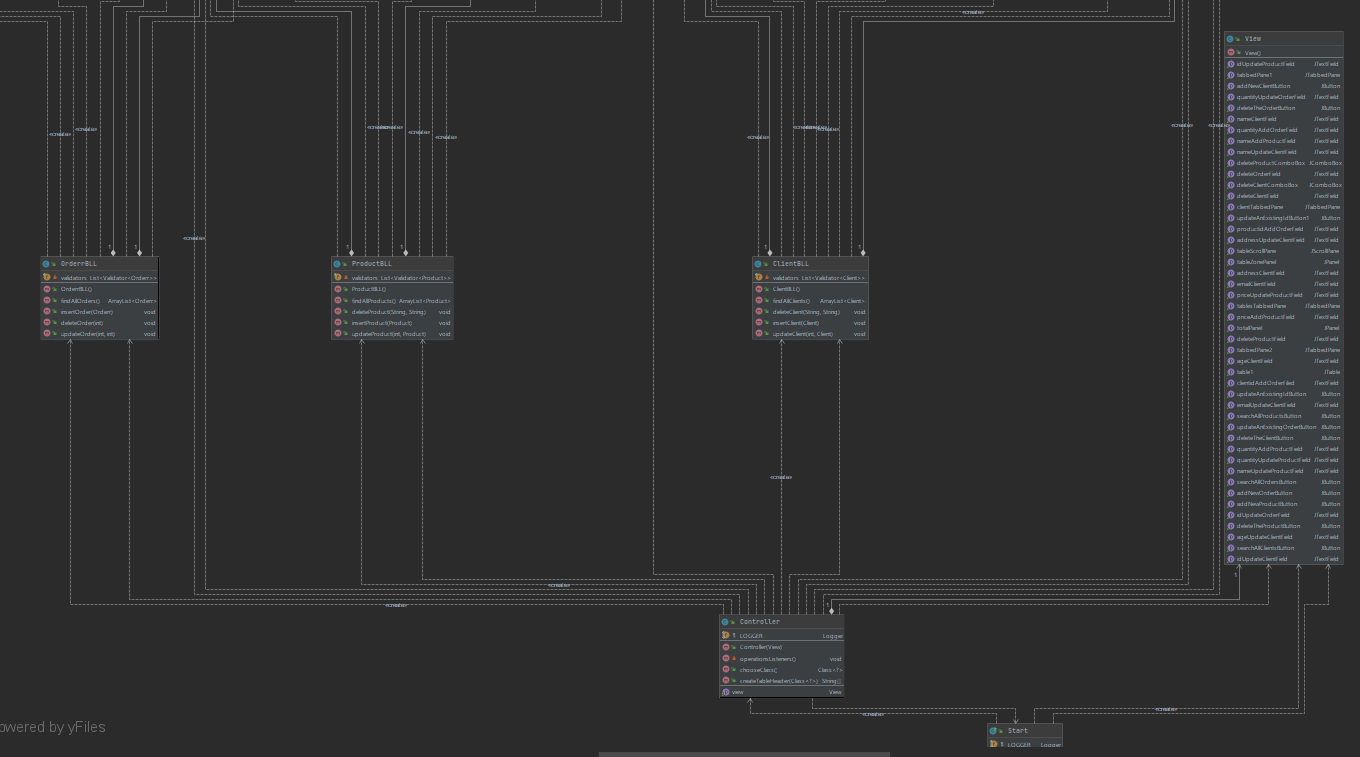
## Data structures

Datorita naturii sarcinii, nu exista multe structuri de date notabile, unele dintre cele mai interesante ar putea fi:

* JTabbedPane -> Care este utilizat în interfața grafică pentru a comuta între datele care sunt editate sau vizualizate.
* ConnectionFactory -> Care este utilizat pentru stabilirea conexiunii intre codul Java si baza de date.

## Class design and UML diagrams

Voi incepe cu o diagrama care prezinta toate clasele si relatiile dintre ele ca un UML.



Principalele clase necesare pentru proiect pot fi impartite in:

* Start -> Contine doar functia principala care porneste executarea programului.
* View -> Contine elementele de baza ale GUI, procesarea butoanelor apasate de utilizator si cateva metode necesare pentru actualizarea GUI cu informatiile furnizate de dataAccessLayer.
* Client -> Modelează datele necesare pentru reprezentarea unui client.
* Orderr -> Modelează datele necesare pentru reprezentarea unei comenzi.
* Product -> Modeleaza datele necesare pentru reprezentarea unui produs.
* ConnectionFactory -> Gestioneaza conexiunea dintre cod si baza de date MySQL data.
* AbstractDAO -> Modeleaza o interfata de baza de date generica care permite extragerea si inserarea datelor din si in baza de date folosind o clasa specificata in constructor. (Se bazeaza pe reflectie)
* ClientBLL -> Adauga capacitatea de a adauga validatori deasupra clasei Client.
* OrderrBLL -> Adauga capacitatea de a adauga validatori peste clasa Orderr.
* ProductBLL -> Adauga capacitatea de a adauga validatori peste clasa de produse.
* Validator<T> -> Defineste o interfata pentru validatorii dati.
* ClientAddressValidator -> Un validator care valizeaza adresa clientului.
* ClientAgeValidator -> Un validator care valideaza varsta clientului.
* ClientEmailValidator -> Un validator care valideaza email-ul clientului.
* ClientNameValidator -> Un validator care valideaza numele clientului.
* OrderQuantityValidator -> Un validator care valideaza daca cantitatea din orderr este mai mica decat cantitatea disponibila pentru produsul cu productID-ul corespunzator din clasa Product.
* ProductNameValidator -> Un validator care valideaza numele produsului.
* ProductPriceValidator -> Un validator care valideaza pretul produsului.
* ProductNameValidator -> Un validator care valideaza numele validatorului.

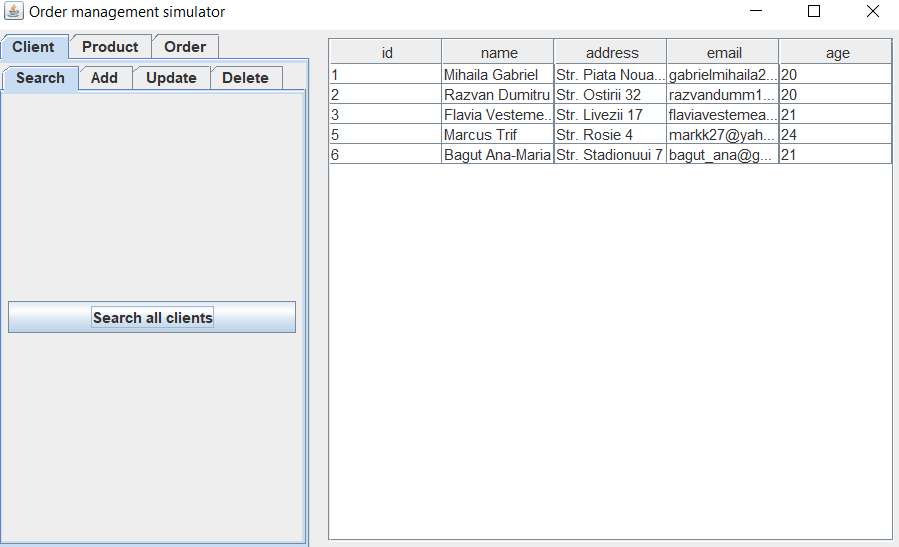
## Interfaces

Singura interfata prezenta in proiect este interfetele Validator care permit crearea mai usoara a validatorilor care asigura validitatea intrarii.

## Algorithms

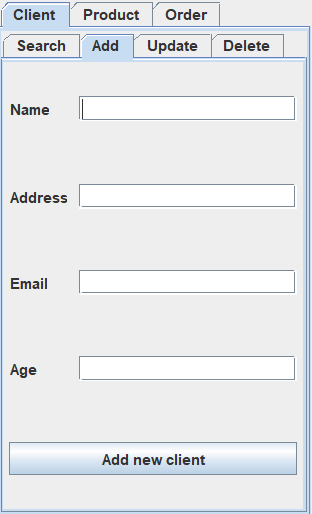
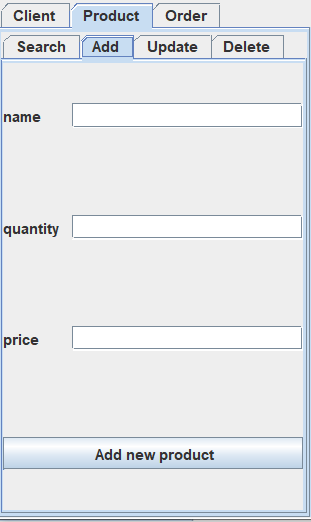
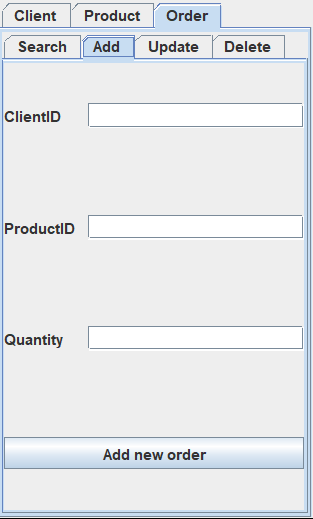
Natura assignment-ului nu a necesitat utilizarea unor algoritmi specifici pentru o buna implementare.

## User interface

Interfata este construita ca mai multe JTextField-uri etichetate care permit introducerea datelor de catre utilizator, alaturi de 3 butoane J pentru toate procesarile de date necesare. JTextFields sunt distribuite intr-un JTabbedPane care le face mai usor de utilizat. Exista, de asemenea, un tabel JT care permite ca toate datele dintr-o anumita clasa sa fie vizualizate in partea dreapta a interfetei grafice.

Interfata arata dupa cum urmeaza:

JTabbedPane poate fi in urmatoarea stare, care arata ca:



Fiecare dintre ghiduri care dintre cele 3 tipuri de date vor fi procesate de cele 3 butoane, care functioneaza dupa cum urmeaza:

* Search -> listeaza tabelul respectiv cu informatiile aferente (se creeaza si header-ul tabelului folosind metoda reflectiei)
* Add -> Adauga o noua entitate in tabel
* Delete -> Sterge o entitate sau mai multe entitatie, specifacandu-se si fieldul dupa care sa se faca stergerea.
* Update -> Se actualizeaza o entitate existenta ( care are un id existent), modificand toate celelalte field-uri.

# Implementation

Vom lua toate clasele pe rand si vom analiza toate partile importante.

## Start

Singurul rol al clasei este de a initializa clasele necesare programului si de a-l porni.

## View

Genereaza interfata grafica, avand in acelasi timp grija sa o actualizeze si sa utilizeze butoanele ascultatoare pentru a procesa datele.

Functia serveste ca un exemplu de procesare a actiunilor prin prezentarea a ceea ce se intampla la apasarea butonului de stergere. Mai intai verifica daca campul de text ID contine date, altfel nu putem sti ce rand al tabelului ar trebui sters. Apoi apeleaza AbstractDAO cu tipul corect de date selectat din JTabbedPane si sterge ID-ul dat.

Functia calculeaza ce date ar trebui sa fie afisate in JTable avand tipul de date selectat din JTabbedPane ca parametru 

Mai intai converteste indexul selectat in clasa corecta si acceseaza dataAccessLayer pentru a prelua datele necesare care sunt reprezentate ca campuri private care trebuie setate ca accesibile.

Iesirea finala este o lista de liste de siruri, dintre care prima lista de siruri reprezinta antetul tabelului si urmatoarele reprezinta datele care ar trebui afisate.

## Client

Creeaza o noua instanta de client avand urmatoarele field-uri: id, nume, adresa, email si varsta.

## Product

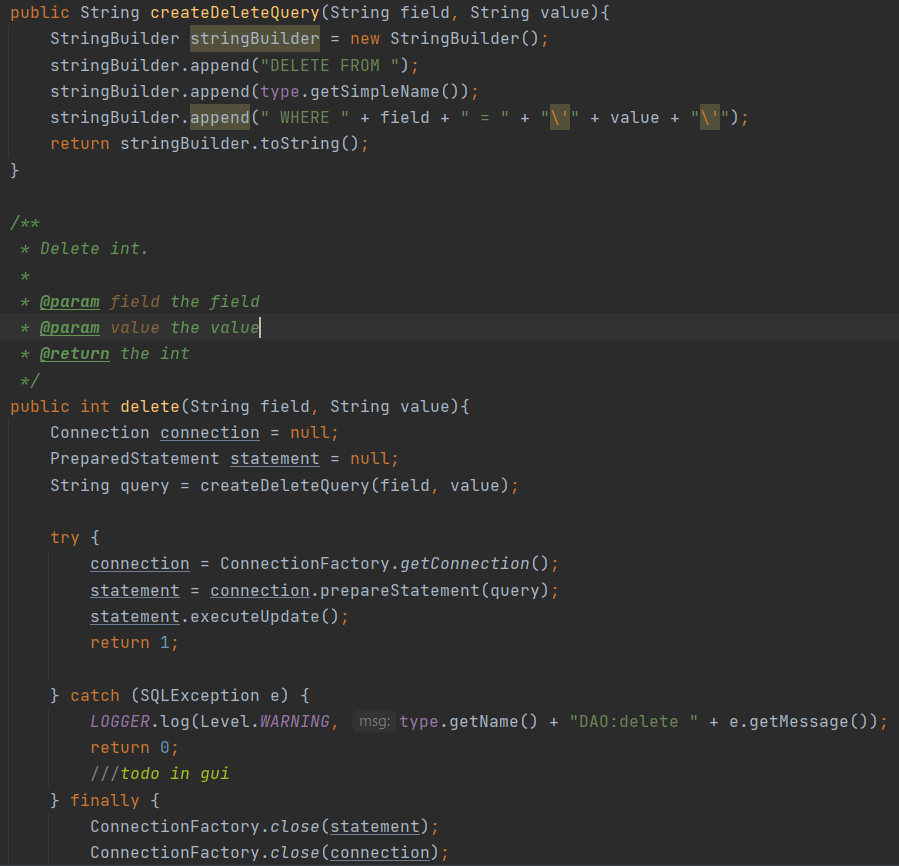
Creeaza o noua instanta de produs cu urmatoarele field-uri: id, nume, cantitate si pret.

## Orderr

Modeleaza comanda ca avand 4 campuri importante, ID-ul intreg, ID-ul intreg al clientului, ID-ul intreg al produsului si numarul intreg care reprezinta cantitatea.

## AbstractDAO

Provides a generic way to access the data from the database that can be used by any of the 3 possible classes: Product, Client and Ordert that are specified in the constructor.



Functia genereaza sirul de interogare SQL care introduce obiectul generic T dat in tabelul corect. Utilizeaza numele clasei pentru a gasi care este tabelul corect si apoi introduce fiecare dintre campurile clasei ca Siruri in sirul de interogare rezultat.

## ConnectionFactory

O clasa utilizata ca conector principal intre cod si baza de date MySQL.A class used as the main connector between the code and the MySQL database.

Creeaza conexiunea la baza de date cu DBURL, USER si PASS valorile finale specificate in cod.

## ClientBLL

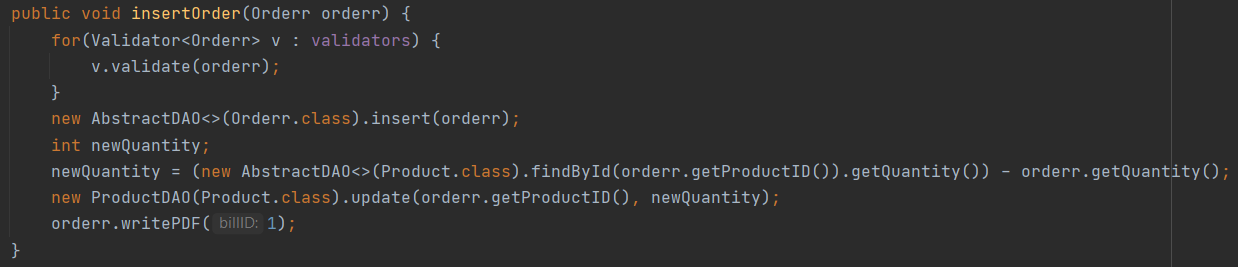
Folosit pentru validarea insertiei clientilor cu validatorii dati.

## ProductBLL

Folosit pentru validarea inserției produselor cu validatorii dați.

## OrderrBLL

Folosit pentru a valida inserarea comenzilor cu validatorii dati.



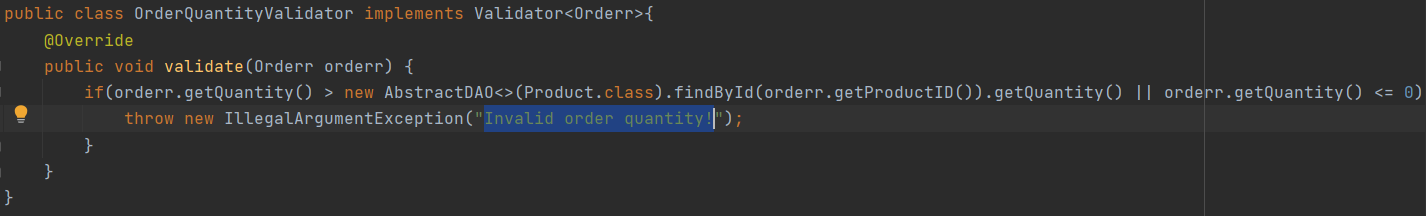
Functia de inserare a comenzii este putin mai speciala decat restul, deoarece dupa validarea comenzii ca fiind valabila, scade cantitatea produsului dat, atat in ​​cod, cat si in baza de date si apoi genereaza un PDF care permite utilizatorului pentru a vedea toate comenzile care au fost plasate in viitor.

## Validator

Reprezinta o interfata care este utilizata pentru a genera cat mai multi validatori pentru date, dupa cum este necesar.

## OrderQuantityValidator

Folosit pentru a valida faptul ca comanda de produs data are suficient stoc pentru ca comanda sa fie indeplinita.



Primeste o comanda ca parametru si verifica daca cantitatea comandata este mai mare decat cantitatea din stoc a produsului, arunca o exceptie spunand ca cantitatea este invalida.

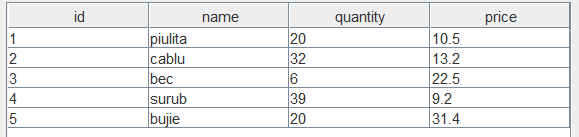
# Results

Rezultatele sunt reprezentate ca datele care ajung in baza de date si pot fi vazute in partea dreapta a

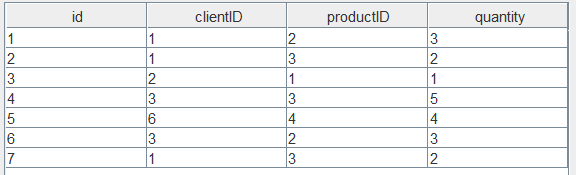
GUI-ului.

De exemplu, produs introdus curent

in baza de date cu cantitatea lor data sunt:



Si unele dintre comenzile date de clienti pot fi, de asemenea, vazute in GUI:



# Conclusion

In concluzie, assignment-ul a servit ca o modalitate excelenta de a afla despre modul in care un program Java este capabil sa interactioneze cu o baza de date, in timp ce invata multe despre reflectie. Cred ca reflectarea este o tehnica cu adevarat importanta, care permite scrierea mult mai putin cod.

In acelasi timp, a servit si ca un exemplu real din lumea in care ar putea fi folosite acele idei, ceea ce a ajutat la invatarea lor mult mai rapida.

De asemenea, a oferit un bun exemplu in cazul in care arhitectura stratificata ar fi utilizata in viata reala.

In ceea ce priveste evolutiile viitoare, cred ca aplicatia ar trebui sa accepte comenzi mai dinamice pentru introducerea utilizatorului, ar trebui sa ofere o interfata grafica cu utilizatorul si poate oferi mai multe operatiuni pentru gestionarea unui depozit.

Extinderea datelor care pot fi modelate prin adaugarea de clase suplimentare, care pot fi usor integrate de baza de reflectie deja existenta.

* + Furnizarea unei interfete mai usor de utilizat.
  + Furnizarea notificarilor pentru momentul procesarii datelor.
  + Permiterea utilizatorului sa editeze datele direct in JTable.
  + Utilizarea aplicatiei pentru o companie din lumea reala care are nevoie de ea.

# Bibliography

* <https://www.baeldung.com/java-pdf-creation>
* <https://online.visual-paradigm.com/>
* <https://app.creately.com/>
* <https://dzone.com/articles/layers-standard-enterprise>
* <https://www.oracle.com/technical-resources/articles/java/javareflection.html>
* [https://www.geeksforgeeks.org/reflection-in-java/](https://www.oracle.com/technical-resources/articles/java/javareflection.html)
* [https://www.javatpoint.com/steps-to-connect-to-the-database-in-java](https://www.oracle.com/technical-resources/articles/java/javareflection.html)
* [https://dev.mysql.com/doc/refman/8.0/en/insert-on-duplicate.html](https://www.oracle.com/technical-resources/articles/java/javareflection.html)