***Aplicație de gestiune a comenzilor***

-Assigment 3-

Student : Brehuescu Andrei

Grupa: 30224

Facultatea CTI RO,an II

Contents

[1.Obiectivul temei 3](#_Toc37784038)

[1.1 Obiectivul principal 3](#_Toc37784039)

[1.2 Obiective secundare 3](#_Toc37784040)

[2. Analiza problemei, modelare, scenarii, cazuri de utilizare 4](#_Toc37784041)

[2.1 Cerințe funcționale 4](#_Toc37784042)

[2.2 Use-case 5](#_Toc37784043)

[2.3 Use-case Diagram 6](#_Toc37784044)

[3. Proiectare (decizii de proiectare, diagrame UML, structuri de date, proiectare clase, interfețe, relații packaged, algoritmi, interfață utilizator). 7](#_Toc37784045)

[3.1 Decizii de proiectare 7](#_Toc37784046)

[3.2 Diagram UML 7](#_Toc37784047)

[3.3 Proiectarea claselor, pachete, interfețe 8](#_Toc37784048)

[4. Implementare 8](#_Toc37784049)

[4.1 Pachetul Start 8](#_Toc37784050)

[4.2 Pachetul Connection 8](#_Toc37784051)

[4.3 Pachetul Model 9](#_Toc37784052)

[4.4 Pachetul Presentation 9](#_Toc37784053)

[4.5 Pachetul Bll 10](#_Toc37784054)

[4.6 Pachetul Validators 11](#_Toc37784055)

[4.7 Pachetul DAO 11](#_Toc37784056)

[5. Rezultate 12](#_Toc37784057)

[6. Concluzii 12](#_Toc37784058)

[7. Bibliografie 12](#_Toc37784059)

# 1.Obiectivul temei

## 1.1 Obiectivul principal

Principalul scop al acestei teme este proiectarea și implementarea unei aplicații de gestiune a comenzilor, având o legătură la o bază de date generată în MySql . Aplicația stochează date cu privire la clienții noi, comenzi plasate și produse în baza de date. Pentru fiecare acțiune cerută, aplicația generează un PDF corespunzător.

Gestionarea comenzilor se face în funcție de produsul și cantitatea comandată, astfel se poate efectua comanda doar în cazul în care există cel puțin cantitatea cerută în depozit .

## 1.2 Obiective secundare

Pentru proiectarea aplicației, problema este împărțită în mai multe obiective secundare, acestea fiind implementate separat . Obiectivele secundare sunt:

* Crearea clasei responsabile de conexiune cu baza de date ( deschiderea conexiunii, a instrucțiunilor, a setului de rezultate cât și închiderea acestora );
* Generearea unei bazei de date potrivite pentru gestionarea aplicației în MySql, stabilirea tabelelor și a relațiilor între ele;
* Definirea claselor ca obiecte transpuse din lumea reala ( clienți, comenzi, produs) și a metodelor;
* Stabilirea atributelor claselor reflectând astfel transpunerea din lumea reală;
* Citirea datelor de intrare de la fișierul de intrare (câte o instrucțiune pentru aplicație, inserare de client, ștergere de client, inserare produs, executare comandă, etc. ), al cărui cale este transmisă ca parametru din linia de comandă;
* Procesarea datelor citite, fiecare linie din fișierul de intrare reprezintă o instrucțiune pentru aplicație, această instrucțiune odată preluată este transmisă controller-ului pentru execuție;
* Definirea regulilor de validare a obiectelor, astfel un obiect va fi acceptat în cazul în care conține date corespunzatoare listei de validatori atribuite acelui obiect;
* Crearea claselor de acces la date prin intermediul conexiunii la baza de date (Data access Layer);
* Definirea unei clase abstracte de acces la date pentru a surprinde metoda Reflection;
* Definirea interfeței de validare a claselor.

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

## 2.1 Cerințe funcționale

Cerințe:

* Inserarea clienților și a produselor în baza de date și stocarea datelor despre aceștia ( nume și adresa respectiv cantitate, preț și numele produsului );
* Inserarea unui produs deja existent va avea ca efect actualizarea stocului acelui produs;
* Validarea tuturor datelor înaintea introducerii acestora în baza de date;
* Generarea de rapoarte PDF conform cerinței cerute în fișierul de intrare sau în caz de eșec, generarea unui raport cu date despre motivul eșuării ( Under-stock );
* Pentru fiecare produs comandat, stocul acestuia va trebui actualizat și în baza de date;
* Fiecărei comenzii îi va fi calculat un total reprezentând produsul între prețul obiectului comandat și prețul acestuia pe unitate;
* Se vor putea comanda doar produse al căror stock este cel puțin egal cu cantitatea comandată, în caz contrar comanda nu este executată iar cantitatea actuală a produsului ramâne neschimbată;
* Ștergerea unui produs/client va avea ca efect eliminarea acestuia din baza de date.

Aceste cerințe vor avea nevoie de câmpuri de identificare pentru a putea fi manipulate de aplicația proiectată în Java. Astfel, fiecare tabela din cadrul bazei de date va conține un ID drept cheie primară în tabel și identificator pentru aplicație. Acest ID va fi unic pentru fiecare client, produs și comandaProdus.

Pentru a realiza proiectarea unei aplicații de gestiune a comenzilor astfel încât să reflecte cât mai mult realitatea, am ales adăugarea acestei tabele intermediare Comenzi, care leagă tabelul pentru clienți cu tabelul pentru produse comandate pentru a surprinte o relație Many-to-Many ( un client poate comanda mai multe produse și invers, un produs poate fi comandat de mai mulți clienți).

Pentru ștergerea clientului sau a produsului s-a considerat o manieră de eliminare totală a datelor și nu doar de setarea unui flag. Această decizie am ales-o pe baza cerinței, ulterior aflând ca era posibilă și cealaltă.

Pentru executarea corectă a operațiilor se va ține cont de câteva aspecte :

* În baza de date vor fi introduse doar date valide, în caz contrar se va arunca o exceptie și aplicația va fi oprită;
* Nu se vor cere rapoarte asupra unor date inexistente, o excepție va fi aruncată în caz de nerespectare a acestui aspect.

## 2.2 Use-case

Date de intrare:

Un fișier care va conține pe câte o line, o acțiune ce trebuie executată de aplicație. Aceste instrucțiuni pot fi:

* Inserare client: <Nume>, <Adresa>
  + Are loc inserarea unui client în baza de date
* Delete client: <Nume>, <Adresa>
  + Are loc ștergerea clientului din baza de date
* Report [client]/[order]/[product]
  + Generează un raport cu privire la toate datele existente despre client, order sau product.
* Order: <Nume client>, <Nume produs>, <Cantitate>
  + Realizarea unei comenzi pentru clientul specificat

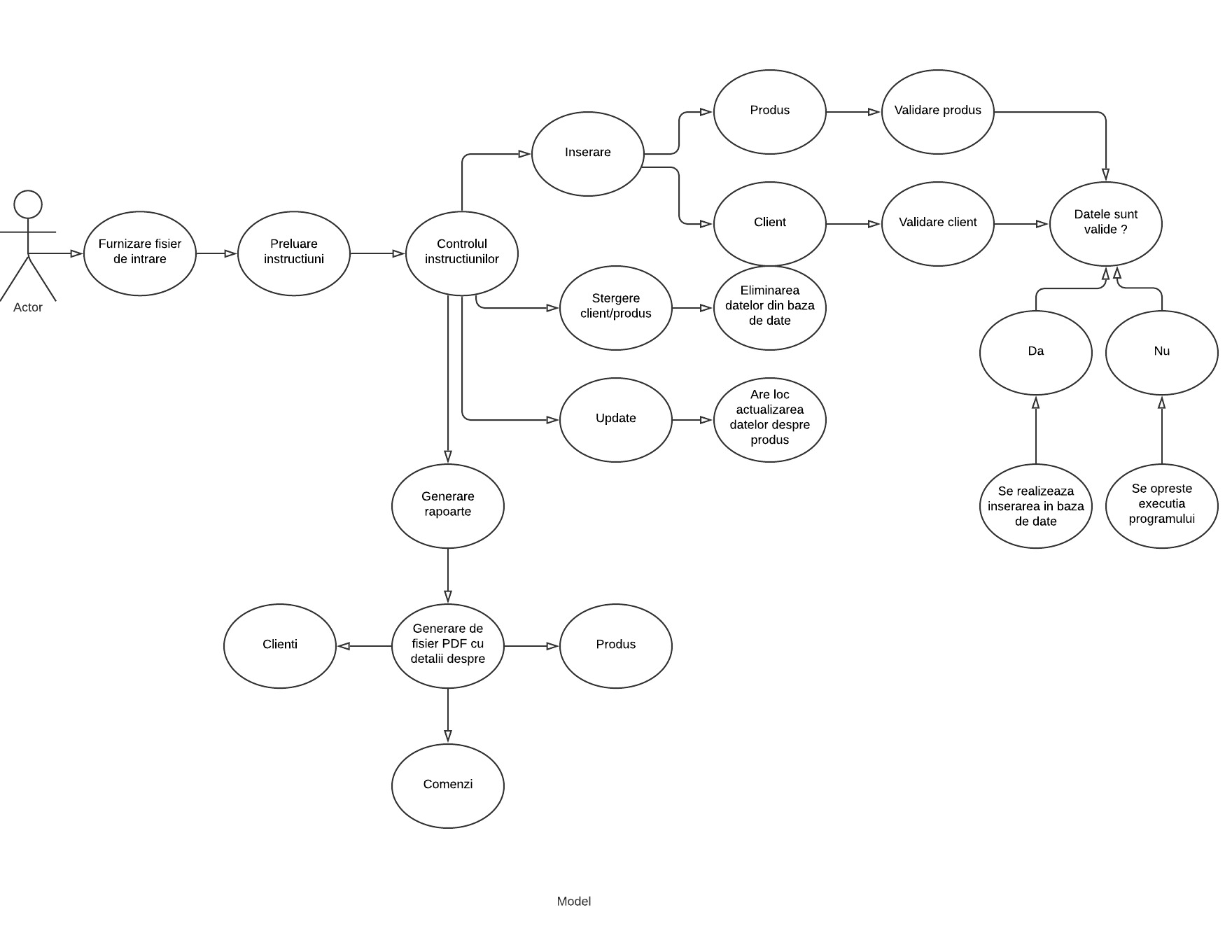
După furnizarea datelor de intrare sub această formă (câte o instrucțiune pe fiecare linie), aplicația va începe gestionarea datelor și în caz de succes vor fi generate rapoarte PDF în directorul aplicației .

Date de ieșire:

Aplicația generează 5 tipuri de rapoarte, fișiere în format PDF. Aceste rapoarte pot fi :

* UnderStock\_<Report Number>
  + Raport ce conține detalii despre eșuarea generării comenzii în caz că a fost cerută o cantitate ce depașește limita stocului existent;
* Report\_Customer\_<ReportNumber>
  + Raport despre toți clienții existenți în baza de date;
* Report\_Product\_<ReportNumber>
  + Raport despre toate produsele existente în baza de date;
* Report\_Order\_<ReportNumber>
  + Raport despre comenzile efectuate, conține detalii despre clientul ce a efectuat comanda, produsul comandat și totalul sumei de plată;
* Bill\_Order\_<ReportNumber>
  + Raport despre fiecare comandă în parte.

## 2.3 Use-case Diagram



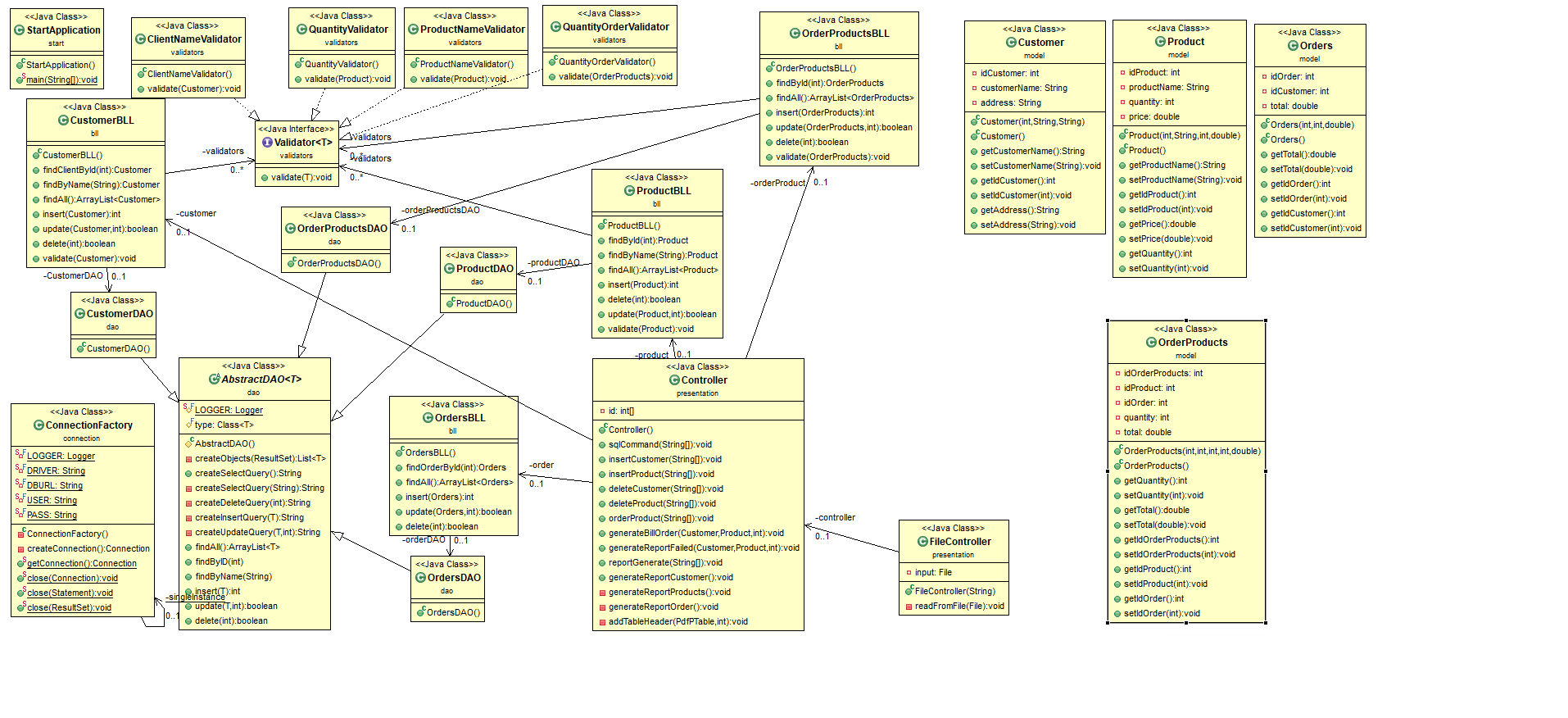
# 3. Proiectare (decizii de proiectare, diagrame UML, structuri de date, proiectare clase, interfețe, relații packaged, algoritmi, interfață utilizator).

## 3.1 Decizii de proiectare

Programarea orientată pe obiecte reprezintă o paradigmă de transpunere a obiectelor din lumea reală în clase și pachete. Tema nu specifică existența unei interfețe grafice, interacțiunea cu utilizatorul se realizează prin intermediul de fișiere de intrare-ieșire. Tema este structurată într-o arhitectură definită pe diferite nivele ( Layered Architecture). Astfel clasele vor fi împărțite în pachetele: dao (Data Access Layer), bll( Business Layer ), model și presentation.

Pentru baza de date, vor fi create 4 tabele pentru stocarea clienților, produselor, comenzilor și o tabelă intermediară pentru realizarea relației Many-to-Many între produse și clienți. Fiecare tabel are o cheie primară ( un id unic în cadrul tabelei ), iar tabelele sunt legate între ele cu chei străine, folosite pentru a se executa o ștergere în cascadă ( ștergerea unei înregistrari are ca efect și ștergerea înregistrărilor legate de cea principală prin cheie straină).

## 3.2 Diagram UML



## 3.3 Proiectarea claselor, pachete, interfețe

Pachetele aplicației sunt: Start, Presentation, Model, Dao, Connection, Bll si Validators. Main-ul aplicației se află în pachetul Start și are ca scop lansarea spre execuție a programului. Pachetul Presentation controlează execuția aplicației prin doua clase: FileReader si Controller, clase ce se vor ocupa de citirea fișierulul și procesarea datelor în funcție de fiecare instrucțiune preluată. Pachetul Model conține datele transpuse în diferite obiecte: Customer ( clasa ce reține datele despre clienți ), Product ( transpunerea produselor ), OrderProducts ( definirea unui produs comandat cu detalii despre comandă ), Orders ( comanda efectivă, legătura între client și produsele pe care le-a comandat). Pachetul Dao conține definirea claselor responsabile de acces asupra datelor din baza de date, aceasta poate efectua operațiile de inserare, ștergere, actualizare sau extragerea datelor pentru afișare. Pachetul Connection furnizează pentru nivelul de acces la date, conexiunea cu baza de date pentru executarea operațiilor. Pachetul Bll conține la rândul lui un pachet, Validators, în acesta se află definirea diferitelor clase de validare , nume, cantitate, etc. Clasele din Bll validează informațiile introduse urmând a fi introduse în baza de date de obiecte de tip Dao.

Clasele din model au aceeași formă ( numele atributelor, ordinea atributelor și numele clasei ) cu tabelul corespunzător în MySql pentru stocarea datelor.

# 4. Implementare

## 4.1 Pachetul Start

**Clasa MainClass**

Clasa MainClass conține doar metoda main() în care este definită o instanță la clasa FileReader în care este trimis ca parametru calea către fișierul din intrare și se începe execuția programului.

## 4.2 Pachetul Connection

**Clasa ConnectionFactory**

Clasa ConnectionFactory conține ca atribut static datele necesare conexiunii cu baza de date ( driver-ul, user-ul, parola și url-ul bazei de date). Constructorul clasei are ca scop crearea unui obiect de pe baza driver-ului inițializat.

Metoda createConnection realizează conexiunea la baza de date sau aruncă o excepție în cazul în care aceasta nu poate fi creată. Clasa mai conține 3 metode ( close() ), acestea sunt supraîncărcate, diferă doar parametrul transmis. În aceste metode are loc ” închiderea parametrului transmis” , astfel dacă este transmisă conexiunea cu baza de date aceasta este închisă, asemănător și pentru setul de rezultate și instrucțiunea.

## 4.3 Pachetul Model

**Clasa Customer**

Clasa Customer conține ca atribute: idCustomer, customerName și address. Aceste atribute sunt datele ce vor fi stocate în baza de date. Atributul idCustomer este folosit pentru identificarea unică a clienților, customerName reține numele clientului, iar address - adresa acestuia.

Clasa conține atât gettere și settere pentru atribute cât și un constructor pentru setarea atributelor la instanțierea obiectului.

**Clasa OrderProducts**

Clasa OrderProducts reprezintă produsele comandate și conține ca atribute: idOrderProducts ( id de identificare unic în tabele din baza de date ), idProduct ( id-ul produsului comandat ), idOrder ( id-ul comenzii ), quantity (cantitatea comandată ) și total ( totalul de plata a acelui produs din cadrul comenzii ).

Clasa conține atât gettere și settere cât și un constructor ce primește ca parametrii valorile pentru atribute.

**Clasa Orders**

Clasa Orders reprezintă legătura între clienți și produsele pe care aceștia le-au comandat. Atributele clasei sunt : idOrder ( id-ul comenzii ), idCustomer ( id-ul clientului care a realizat comanda) și total ( totalul de plata al întregii comenzi ).

Clasa conține atât gettere și settere cât și un constructor ce primește ca parametrii valorile pentru atribute.

**Clasa Product**

Clasa Product reprezintă transpunerea produsului și are ca atribute : idProduct ( id-ul produsului pentru identificare unică în tabel ), productName ( numele produsului ), quanitity ( cantitatea disponibilă pentru comenzi ) și price ( prețul unei unități ).

Clasa conține atât gettere și settere cât și un constructor ce primește ca parametrii valorile pentru atribute.

## 4.4 Pachetul Presentation

**Clasa FileController**

Clasa este responsabilă de gestionarea citirii fișierului de intrare. Atributele sunt : input ( de tipul File, reține instanța la fișierul de intrare, și o instanță la obiectul Controller ( obiectul care lansează instrucțiunile spre execuție ).

Constructorul clasei primește ca argument un șir de caractere reprezentând calea catre fișierul de intrare și instanțiază atributul input.

Metoda readFromFile() citește fiecare linie din fișierul de intrare și împarte linie după separatori ( ”,:” ). Linia ce conține instrucțiunea odată împărțită într-un șir de cuvinte este transmisă controllerului pentru a decide ce tip de instrucțiune este și executarea acesteia.

**Clasa Controller**

Clasa controller este responsabilă pentru gestionarea instrucțiunilor, apelarea metodelor de inserare, ștergere, actualizare și de generarea fișierelor de ieșire. Clasa are 4 instanțe din nivelul Bll. Aceste instanțe sunt utilizate în apelul metodelor de inserare, ștergere sau actualizare a datelor. Clasa mai dispune de un atribut, un șir de numere întregi ( id ), utilizat în compunerea numelor pentru fișierele de ieșire generate.

Metoda sqlCommand( String[] command ) primește ca argument un șir de cuvinte reprezentând instrucțiunea segmentată după separatorii impuși. Pe baza șirului de cuvinte transmis se realizează identificarea acțiunii ce trebuie executată de o metodă potrivită.

Metoda insertCustomer ( String[] customerInfo ) primește ca argument instrucțiunea de inserare ce conține și date despre clientul ce trebuie adăugat în baza de date. În cazul în care datele clientului sunt validate acesta va fi inserat în baza de date.

Metoda insertProduct ( String[] productInfo ) primește ca argument instrucțiunea de inserare ce conține și date despre produsul ce trebuie adăugat în baza de date. În cazul în care datele produsului sunt validate acesta va fi inserat în baza de date.

Metoda deleteCustomer ( String[] customerInfo ) și deleteProduct ( String[] productInfo ) primesc ca argument instrucțiunea de ștergere ce conține date despre obiectul ce trebuie șters. După acest nume se va executa căutarea id-ului obiectului în baza de date pentru a se obține id-ul. Odata obținut id-ul se va executa ștergerea clientului sau a produsului din baza de date.

Metoda orderProduct ( String[] orderInfo ) realizează crearea unei comenzi pe baza detaliilor din orderInfo. Metoda generează un PDF ce conține detalii despre comandă ( cantitatea comandată, clientul, prețul total ) în cazul în care comanda se poate efectua sau un PDF cu detalii despre motivul pentru care nu se poate crea comanda ( UnderStock ). Generarea acestor fișiere este realizată prin apelarea metodelor generateBillOrder, în caz de succes și generateReportFailed în caz de eșec. Aceste metode scriu în fișiere informațiile corespunzătoare.

Metoda reportGenerate( String[] command ) realizează selecție instrucțiunii de generare a rapoartelor complete asupra clientilor, produselor sau comenzilor. Această funcție apelează la rândul ei metoda potrivită pentru generarea fișierelor PDF ce conțin rapoartele complete. Metodele pot fi : generateReportProducts, generateReportOrder, generateReportCustomer.

Metodele de generare a rapoartelor complete creează fișierului în care se va scrie. În acest fișier vor fi scrise datele sub forma unor tabele. Pentru scrierea în aceste tabele va fi apelată o metodă, addTableHeader, care în funcție de ce raport trebuie generat, va adăuga un header diferit pentru fiecare tabel.

## 4.5 Pachetul Bll

Clasele din acest pachet sunt: CustomerBLL, OrderProductBLL, OrdersBLL si ProductBLL. În aceste clase sunt adăugați validatorii respectivi pentru fiecare model de date. Aceste clase au ca atribute o listă de validatori și o instanță la clasa corespunzătoare din nivelul de acces la date. Clasele din pachetul BLL sunt ”clase intermediare” având ca rol validarea datelor înainte ca acestea să fie inserate, respectiv de interceptarea erorilor prin aruncarea de excepții.

## 4.6 Pachetul Validators

**Interfața Validator**

Interfața conține doar metoda validate(T t) care primește ca argument un obiect generic. Acestă interfață este implementată de celelalte clase în pachet, oferind câte o implementare diferită a metodei în funcție de fiecare validator.

Pentru această temă au fost creați validatori pentru numele clientului, numele produsului, cantitatea produsului inserat și cantitatea comandată.

## 4.7 Pachetul DAO

Pachetul DAO conține 4 clase , câte una pentru fiecare tip de date ( client, produs, comandă, produs comandat). Aceste clase nu conțin nici metode nici atribute dar extind clasa AbstractDAO<T>, T fiind un tip generic înlocuit cu tipul potrivit de fiecare clasă atunci când se face moștenirea.

**Clasa AbstractDAO**

Este clasa în care se realizează accesul la datele stocate în baza de date. Ca atribute , clasa are un LOGGER static ( același pentru toate instanțele clasei AbstractDAO ), acest logger este instanțiat de fiecare clasă în mod diferit când se face o instanță a uneia din clasele ce moștenesc clasa AbstractDAO și un obiect generic ce va fi instanțiat la apelarea constructorului cu clasa ce apelează constructorul.

Metoda createObjects(ResulSet resultSet ) primește ca argument un obiect ce conține datele preluate din baza de date. Acestă metodă creează obiecte de aceeași clasă cu clasa ce apelează metoda și le stochează într-o listă ce va fi returnată. În acestă metodă se pot arunca mai multe excepții în caz de eșuează crearea obiectelor. În caz de eșec va fi afișat un Warning prin intermediul Logger-ului.

Metodele createSelectQuery, createDeleteQuery, createInsertQuery si createUpdateQuery generează un string reprezentând instrucțiunea potrivită. Aceste metode sunt apelate în cadrul metodelor ce extrag sau inserează date în baza de date, stringul primit este utilizat pentru crearea unui statement pentru a fi executat asupra bazei de date.

Metodele findByID, findByName și findAll extrag informații din baza de date. În cazul primei metode datele căutate trebuie să corespundă unui Id furnizat ca parametru, asemănător și pentru a 2-a metodă ce caută înregistrările după un nume transmis ca parametru. Metoda findAll() returnează toate înregistrările unei tabele, numele tabelei este preluat automat la apelul funcției în cadrul acesteia cu ajutorul tipul generic. Astfel că fiecare clasă poate obține date prin apelul acestor metode doar din tabelul corespunzator acelei clase.

Metoda insert(T t) primește un obiect de aceeași clasă cu obiectul ce apelează metoda și realizează inserarea în baza de date în tabelul corespunzător acelei clase.Această metodă returnează id-ul la care a fost facută inserarea în caz de succes sau -1 în caz de eroare. Metoda update(T t,int id) se comportă asemănător doar că înlocuiește înregistrarea de la id-ul specificat ca parametru cu informațiile stocate în obiectul ’t’, această metodă fiind folosită pentru actualizarea datelor. Metoda update() returnează un boolean ,cu valorea true, ce semnfică dacă s-a produs actualizarea datelor sau false în caz contrar.

Metoda delete(int id) șterge înregistrarea de la id-ul specificat ca și parametru din tabele corespunzătoare clasei ce apelează această funcție, returnând o valoare de tip boolean care semnifică dacă ștergerea a avut succes sau nu.

# 5. Rezultate

Un exemplu de date de intrare/ieșire poate fi considerat următorul:

Date de intrare:

Insert client: Ion Popescu, Bucuresti

Insert client: Luca George, Bucuresti

Report client

Insert product: apple, 20, 1

Insert product: peach, 50, 2

Order: Luca George, apple, 5

Date de ieșire:

Fișierul PDF pentru raportul clienților :

ID Name Address

1 Ion Popescu Bucuresti

2 Luca George Bucuresti

Fișierul PDF pentru raportul asupra comenzii efectuate:

Ordered: 5 x apple

Price per unit: 1.0

Total : 5.0

Customer: Luca George

# 6. Concluzii

Din această temă am invățat să lucrez cu fișiere de tip PDF, să generez JavaDoc și să folosesc metoda Reflection. De asemenea am înțeles mult mai bine cum să gestionez legătura dintre o aplicație Java și o baza de date modelată în MySql, cât și generarea DumpFile-urilor.

# 7. Bibliografie

* <http://coned.utcluj.ro/~salomie/PT_Lic/4_Lab/Assignment_3/Assignment_3_Indications.pdf>
* <http://tutorials.jenkov.com/java-reflection/index.html>
* <https://www.baeldung.com/javadoc>
* https://dzone.com/articles/layers-standard-enterprise
* https://dev.mysql.com/doc/workbench/en/wb-admin-export-import-management.html o
* https://dev.mysql.com/doc/refman/5.7/en/using-mysqldump.html