M

**Sistem de gestionare a comenzilor**

**Documentatie**

**Facultatea: Automatica si Calculatoare**

**Roman Alexandru | Grupa 30228**

**CUPRINS**

1. **Obiectivul si descrierea proiectului**
2. **Analiza problemei, modelare, scenarii, cazuri de utilizare**
3. **Proiectare**
4. **4.Implementare**
5. **Rezultate**
6. **Concluzii**

**1**. **Obiectivul si descrierea proiectului**

Un Sistem de gestionare al depozitelor este o soluţie software care oferă vizibilitate asupra tuturor stocurilor unei companii şi care gestionează efectuarea operaţiilor din lanţul de aprovizionare

Este un sistem care gestioneaza comenzile efectuate asupra unui depozit de produse. Obiectivul acestui proiect este de a putea procesa comenzile unor clienti, a verifica stocul produselor si a crea o chitanta(bill) care contine detaliile comenzii sau un mesaj care descrie eroarea intalnita. Deasemenea se pot crea rapoarte asupra depozitului care contin informatiile necesare despre clienti,produse, etc.

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

Pentru acest proiect, vom avea nevoie de o baza de date numita warehouse. Baza de date va avea 4 tabele, si anume client, product, orders, report. Tabela client este pentru clientii care vin si pleaca din depozit, in tabela product vom stoca produsele, in tabela orders vom tine minte comenzile iar in tabela report vom tine minte un reprtId si data la care s-a facut reportul. Legatura cu baza de date va fii facute de catre un jdbc driver. Pentru a putea face comenzi, mai intai trebuie sa existe clientul respectivul produsul din comanda in baza de date. De acceea in fisierul de intrare trebuie sa exista niste insertClient si insertProduct. Ideea sistemului este ca orice client care vine sa poata face o comanda si sa primeasca o chitanta pentru ea sau un mesaj de eroare in caz ca nu sunt suficiente produse in stoc.

**3. Proiectare (decizii de proiectare, diagrame UML, structuri de date, proiectare clase, interfete, relatii, packages, algoritmi, interfata utilizator)**

Ca si decizie de proiectare am decis sa structurez proiectul in 5 pachete : BusinessLayer, Connection, DataAccessLayer, Model, Presentation.

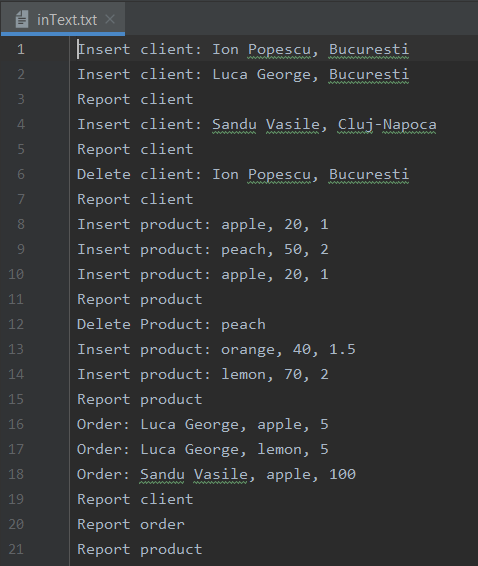
* + Pachetul ***BusinessLayer*** este format din claselecare se ocupa de controlul aplicatiei si anume clasa Controller si MainClass(care contine metoda main si instanteaza clasa Controller).
  + Pachetul ***Connection*** este format dintr-o singura clasa si anume clasa ConnectionFactory. Aceasta clasa creaza conexiunea catre baza noastra de date
  + Pachetul ***Model*** contine clasele Client, Product, Order, Report si Command. Cele 4 sunt niste clase simple care contin variabilele instanta pentru obiectele respective.
  + Pachetul ***DataAccessLayer*** contine urmatoarele si anume ClientDAO, ProductDAO, OrderDAO, ReportDao. Dao vine de la Data Access Object si se refera la un pattern care ne ajuta sa izolam BusinessLayerul de un persistence layer (relational database) folosind un API adica un java application programming interface. Un API contine toate clasele ce fac parte dintr-un JDK( Java development kit).
  + Pachetul ***presentation*** contine 3 clase: Bill, BillValidator si PdfReport. Dupa cum putem observa aceste 3 clase sunt responsabile pentru crearea outputului adica prezentarii aplicatiei. Clasa Bill este foloste pentru a create chitante pentru comenzile clientilor. Despre BillValidator vom vorbii putin mai jos. Clasa PdfReport este responsabile pentru a crea raporturi ale tabelelor in format .pdf.

Mai jos avem diagrama UML a proiectului. Stiu, contine foarte multe clase. Ceea ce ne intereseaza pe noi dinaceasta diagrama UML sunt relatiile dintre clase. Putem observa exact 2 relatii si anume agregare si dependenta. Relatiile de agregare se pot observa la clasele Bill si Command fara de clasa Controller deoarece aceasta contine clasele respective . Observam faltul ca clasa Controller formeaza cele mai multe relatii. Restul relatiilor dintre clase sunt relatii puternice, de dependenta. Acest lucru inseamna ca modificarea uneia poate aduce schimbari in cealalta.



Folosirea structurilor de date este restransa deoarece toate produsele,clientii, etc vor fii stocate intr-o baza de date. Tot ce am avut nevoie este un ArrayList care contine o lista de obiecte de tip Command. Asta inseamna ca vom memora comezile intr-un arrayList si apoi le vom stoca in baza de date. Am folosit structura de date pentru a structura codul intr-un mod usor de inteles si a nu alglomera metodele.

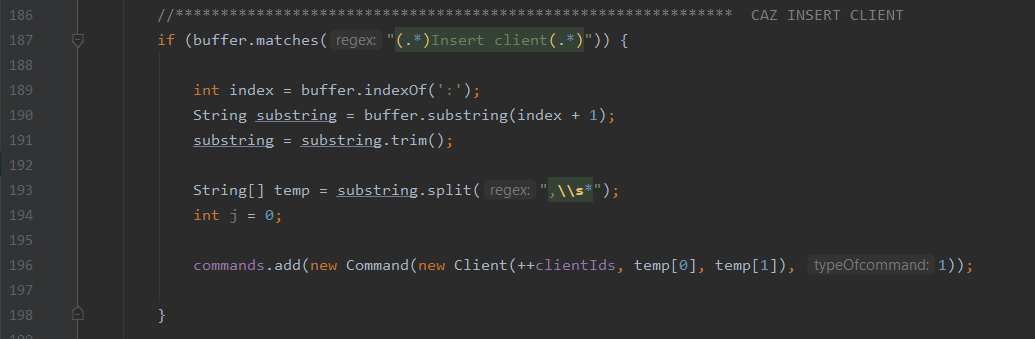
Proiectul nu are o interfata grafica. Acesta primeste informatiile dintr-u fisier de intrare .txt si apiseaza rezultatele obtinute atat in consola cat si in fisiere .pdf asociate comenzilor respectiv rapoartelor. La fiecare comanda sau raport citit din fisierul de intrare se va crea cate un .pdf.



**4. Implementare**

La capitolul implementare voi explica cum am implementat clasele enumerate mai sus.

Clasa ***Main:*** *e*ste clasa care contine metoda main(). Aceasta instantiaza clasa Coltroller si trateaza exceptiile SQLException si DocuentException printr-un bloc try-catch.

Clasa ***Controller:*** nu are parametrii, instantiaza ArrayListul de comenzi si apoi foloseste clasa FileReader pentru a deschide fisierul de intrare inText.txt iar apoi apeleaza metoda fileParser care parcurge fisierul caracter cu caracter. Cand ajunge la final de linie apeleaza metoda formatare() care compara stringul dat(linia din fisier) cu expresii de tip Regex pentru a determina ce tip de operatie este si ii memoreaza informatiile ca si comanda de tip Command in ArrayList. 

Desigur, pentru ca citirea sa aiba efect informatiile vor trebuii scrise ca in poze de mai sus, cea cu fisierul .txt. Trebuie scrise in aceiasi structura cu “:” si apoi separate prin virgula cu sau fara spatii.

In continuare, in constructorul clasei se itereaza ArrayListul commands cu un for each. Vom vorbii mai sus despre clasa Command, ea are niste flaguri care sunt setate pe true. Se verifica acele flaguri si in functie de flagul setat pe true se apeleaza o metoda de insert/delete din clasele DAO din DataAccessLayer. Daca comanda este order atunci se folosesc pe langa insertOrder clasele Bill si BillValidator despre care vom vorbii putin mai jos. Daca comanda este raport atunci vom folosii metodele din pdfReport pentru a crea fisierele .pdf necesare.

Clasa ***ConnectionFactory:*** creeaza conexiunea dintre baza de date si java. Constructorul clasei face load obiectului asociat clasei jdbc.Driver. Aceasta clasa contine o variabila clasa singleInstance care apeleaza constructorul. Variabila fiind statica apeleaza constructorul declarat private, asadar vom avea o singura instanta a clasei.

Clasa ***ClientDAO:*** Aceasta clasa contine metodele de insert, delete, display pentru clienti. Avem trei variabile clasa si anume logger care poate fii folosit pentru a afisa diferite mesaje, insertStatementString care contine un query in sql in format String pentru insert client iar al treilea este un String pentru delete client. Cand se apeleaza metodele de insert sau delete prima data se face conexiunea la baza de date cu ajutorul clasei ConnectionFactory. Apoi, folosind prepareStatement() inseram campurile necesare in baza de date. Cu metoda display() se afiseaza pe ecran continutul tabelei Client. O metoda display() exista in toate cele 4 clase din DataAccessLayer si se apeleaza toate la finalul constructorului din clasa Controller pentru a se vizualiza tabalele.

Clasa ***ProductDAO:*** La fel ca si ClientDAO, aceasta clasa contine metodele de insert, delete, display dar pentru produse. Avem din nou variabila logger precizata mai sus, si trei String-uri care contin query-urile in sql pentru a insera/sterge un produs respectiv pentru a face un update la acel produs in caz ca produsul exista deja. Cand se executa metoda insertProduct() se apeleaza metoda verify pe acel produs de tip Product pentru a verifica daca produsul exista deja in bada de date caci nu trebuie inserat de 2 ori. In cazul in care este deja se face update in loc de insert. Exista si metoda display() pentru afisarea in consola a tabelului Product

Clasa ***OrderDAO:*** contine trei variabile clasa si anume logger si stringurile insertStatementString pentru a face insert la comanda respectiv updateStatementString pentru a face update produselor comandate. Clasa contine 2 metode si anume makeOrder folosita pentru insert si Display() pentru afisarea pe ecran. In Controller inainte sa fie apelata metoda este folosita clasa BillValidator pentru a valida comanda iar apoi dupa aceasta metoda se va crea o factura prin clasa Bill.

Clasa ***ReportDAO:*** contine variabilele de clasa logger si insertStatementString si medota insertRaport pentru a insera in tabelul Report care contine un reportId si inca un camp cu data si ora cand a fost facut raportul. Dupa ca a fost apelata metoda in Controller se foloseste clasa PdfReport pentru a afisa raportul in forma de tabel.

De asmenea in cele 4 clase despre care am vorbit anterior se afiseaza pe ecran number of rows affected.

Clasa ***Bill:*** este folosita pentru a crea chitanta pentru o comanda. Ea are un constructor care apeleaza metoda statica makeOrderPdf pentru a crea o chitanta sau metoda msjBill acre va afisa un anumit mesaj in fisierul .pdf. Fisierele in format .pdf suntcreate cu ajutorul bibliotecii IText5 Mesajul este generat de clasa BillValidator despre care vom vorbii in continuare.

Clasa ***BillValidator:*** contine un costructor si trei metode. Constructorul apeleaza metoda validate() care verifica daca clientul si produsul sunt ambele pe stoc apoi verifica dac stocul este sufucient pentru a satisface cerintele comenzii. Metoda returneaza un String care va fii mesajul generat in urma verificarilor.Daca mesajul este “ok” atunci nu au aparut erori altfel va fiii returnat mesajul corespunzator. Metoda validate() foloseste de asemenea metoda returnRs pentru a obtine continutul unui anumit tabel si metoda searchForPrice care gaseste pretul unui anumir produs aflat in comanda facuta de client.

Clasa ***PdfReport:***  contine un constructorcare in functie de un intreg primit ca parametru executa una dintre metodele: reportClient pentru a crea un raport .pdf cu tabelul Client, impreuna cu data si ora la care a fost facut raportul, reportProduct pentru a crea un raport .pdf cu tabelul Product, impreuna cu data si ora la care a fost facut raportul, sau metoda reportOrder pentru a crea un raport .pdf cu tabelul Order impreuna cu data si ora la care s-a facut raportul. Aceste trei metode folosesc de asemenea metoda getTableResultSet care returneaza ResultSet-ul corespunzator metodei Select \* fom tableName.

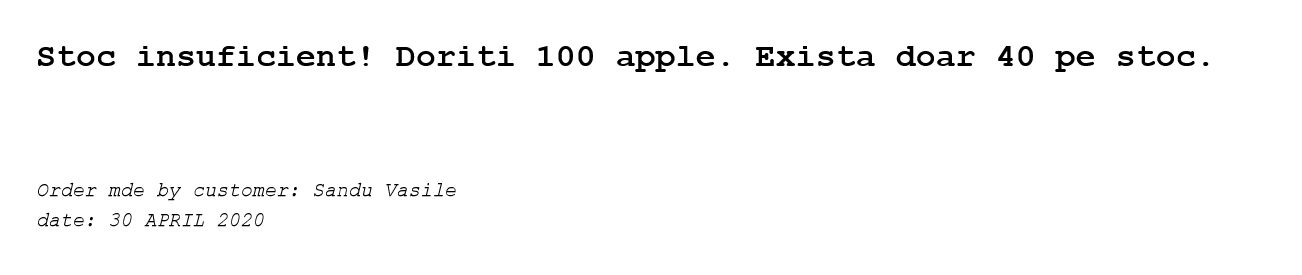
Clasele ***Client, Order, Product, Report:*** dupa cum am mentionat mai sus sunt doar niste clase care contin variabilele instanta necesare.

Clasa ***Command:*** contine obiectul ce trebuie memorat(comanda) si cate un flag pentru fiecare operatie posibila. Flagurile sunt setate pe false initial dar flagul corespunzator devine true cand este apelat constructorul clasei Command.

**5. Rezultate**

La finalul rularii programului vom obtine baza de date completa a depozitului, si vom obtine toate chitantele primite de catre clienti pentru comenzi, precum si reporturile in cu tabelele de pe parcursul simularii.

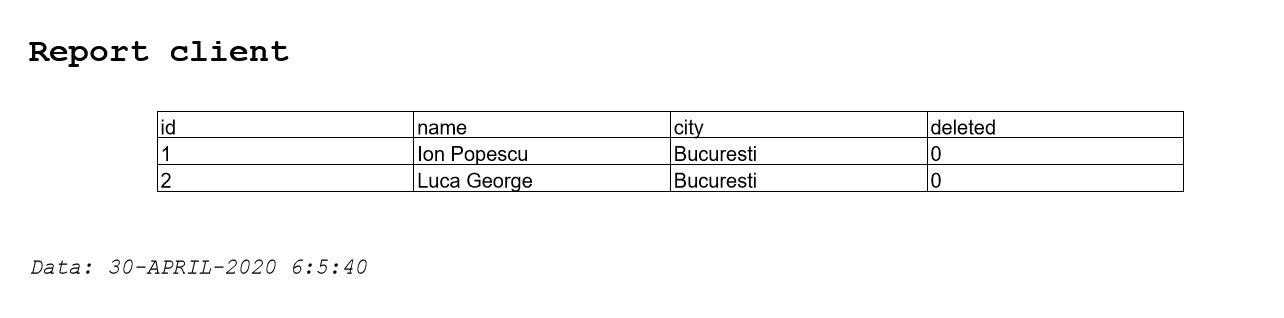
Daca vom intampina o eroare precum stoc insuficient va aparea un fisier .pdf precum cel de mai jos in care clientul Sandu Vasile incearca sa comande mai multe mere decat sunt pe stoc



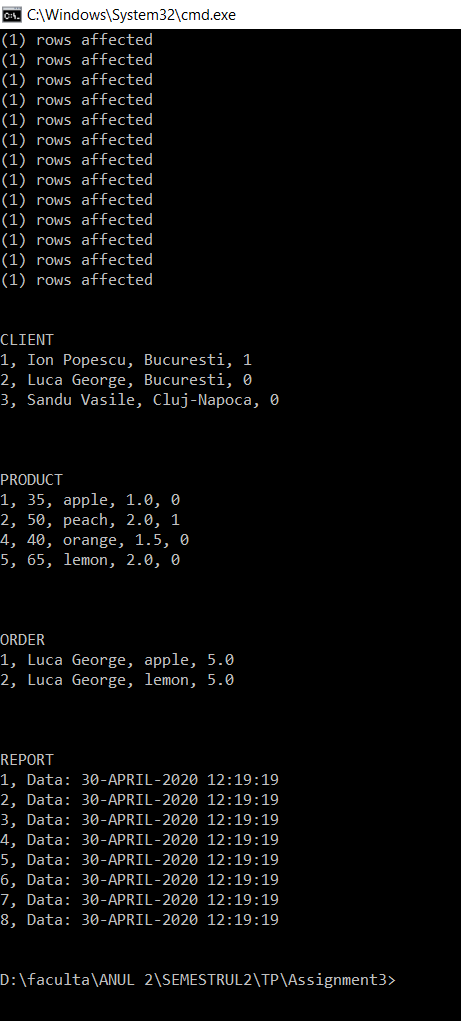
In caz de succes se va genera o factura ca in imaginea de mai jos care contine numele produsului, cantitatea si suma totala, precum si numele clientului care a facut comanda impreuna cu data.



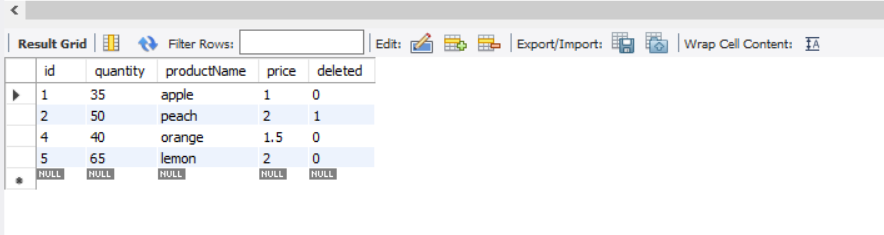
Tabela report are o coloana cu data la care s-a facut comanda. Aceasta este utila pentru stocarea datei si orei cand s-a facut un raport.



Mai jos este un screenshot cu consola care afiseaza(pe langa faptul ca afiseaza si fisierul in consola) randurile afectate si tabelele la final de executie.



Putem vedea, de asemenea si in baza de date rezultatele executiei ca in screenshotul de mai jos:



Pentru a rula proiectul din jar se ruleaza cu comanda “java –jar Assignment3.jar

Proiectul se gaseste pe int [**PT2020\_30228\_Roman\_Alexandru\_assignment1**](https://gitlab.com/alecs29/pt2020_30228_roman_alexandru_assignment1) pe GitLab si contine clasele in java, fisierele dump pentru crearea bazei de date, fisierele html javadoc si fisierele .pdf adica chitantele si reporturile.

**6. Concluzii si posibilitati de dezvoltare ulterioara**

In concluzie, pot zice ca dupa fiecare proiect imi aprofundez cunostintele in java tot mai mult. Din proiectul asta am invatat cateva tehnici precum si conectarea la o baza de date cu un jdbc, sa creez un DAO si folosirea bibliotecii IText5 pentru crearea fisierelor .pdf in java.

Proiectul are multe posibilitati de dezvoltare ulterioara. Prin adaugarea mai multor tabele si cateva imbunatatiri ar putea manageria vanzarile online ale unui magazin deoarece proiectul oferta chitante pentru comenzi si actualizeaza si baza de date. In tabelul cu client s-ar putea face insert atunci cand clientul intra pe site si delete atunci cand iese sau tinut minte o vreme cuu un flag activ care sa se seteze pe false atunci cand clientul nu mai e activ. O modalitate buna de a implementa un sistem asemenator cu mai multe tabele ar fii folosirea unei clase generice pentru operatiile de insert, delete, etc.