

**FACULTATEA DE AUTOMATICĂ ŞI CALCULATOARE**

ORDER MANAGEMENT

Documentație

Dorofte Andrei

Grupa 302210

Cuprins

1. Obiectivul temei

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

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

4. Implementare

5. Rezultate

6. Concluzii

7. Bibliografie

1. **Obiectivul temei**

Proiectati si implementati o aplicatie de simulare care sa analizeze comenzi date de o serie de clienti aflati intr-o baza de date. Comenzile sunt operate asupra unor produse.

O comanda poate fi observata atat in lumea reala cat si ca model virtual. Obiectivul principal al unei comenzi este de a oferi informatii atat despre clientul care o plaseaza, cat si despre produsul, cantitatea si pretul final al acesuia. Obiectivul unui sistem format dintr-o baza de date este de a oferi o evidenta clara, usor de citit si de modificat, a clientilor, comenzilor, produselor si a stocului acestora.

De asemenea, se por genera rapoarte detaliate cu privire la fiecare dintre elementele bazei de date, iar pentru fiecare comanda plasata a unui client prezent in baza de date, se va genera un raport cu detaliile comenzii acestuia, printre care cantitatea cumparata, produsul si pretul final al comenzii acestuia.

|  |
| --- |
| **Obiective secundare** |
| * Crearea claselor necesare |
| * Definirea metodelor de baza pentru fiecare clasa |
| * Crearea conexiunii cu baza de date |
| * Construirea interogarilor SQL |
| * Realizarea algoritmilor care prelucreaza interogarile SQL |
| * Introducerea generatorului de raport PDF |

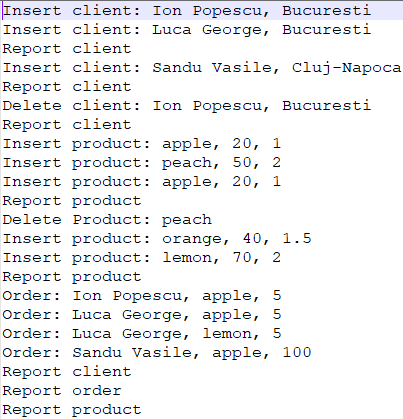
1. **Analiza problemei, modelare, scenarii, cazuri de utilizare**

Comenzile sunt prezente in oricare dintre magazinele in care intram zilnic. Niciun magazin sau depozit nu poate functiona fara o evidenta clara a produselor de pe stoc, sau a comenzilor care trebuie onorate clientilor. Astfel, o baza de date poate servi ca o metoda eficienta si usor de utilizat pentru o astfel de situatie.

Acest proiect are rolul de a simula un depozit/magazin, ai carui clienti sunt stocati intr-o baza de date. Asemenea sunt stocate si produsele, impreuna cu pretul si cantitatea prezenta pe stoc. Atunci cand se plaseaza o comanda, trebuie sa se verifice daca produsul se afla, intr-o cantitate suficienta pe stoc, iar daca da, sa se genereze comanda si sa se actualizeze stocul.

* Sistemul trebuie sa realizeze operatiile de inserare si stergere pentru client.
* Sistemul trebuie sa realizeze operatiile de inserare si stergere pentru produs.
* Sistemul trebuie sa realizeze generarea de raport pentru o comanda plasata cu succes de catre un client.
* Sistemul trebuie sa realizeze rapoarte pentru fiecare tabel din baza de date.

**Datele de intrare:**

Un fisier care sa contina pe cate un rand operatia care se doreste a fi realizata, dintre cele pomenite mai sus.

**Datele de iesire:**

Cate un raport este generat dupa fiecare operatie efectuata, pentru a verifica corectitudinea implementarii acestora.

Order Management

Insert client

Male icon

Delete client

Male icon

Insert product

Input data from file

Insert product

Output files

User

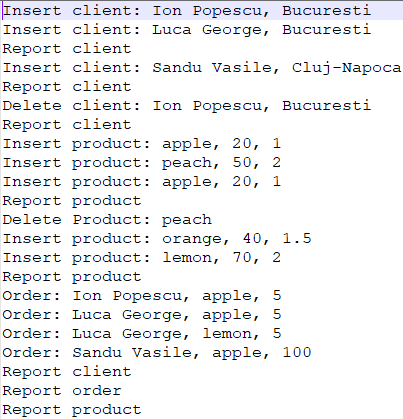
Delete product

User

Generate PDFs

Caz de utilizare:

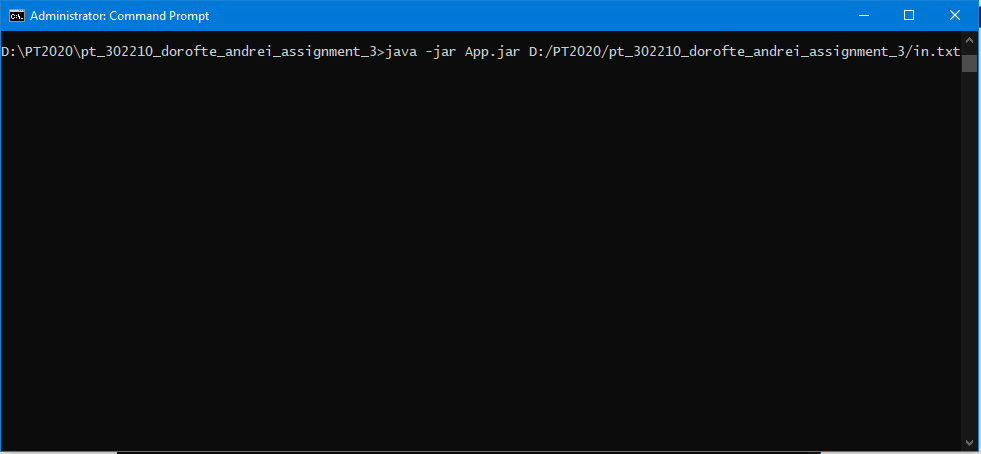
1. Utilizatorul introduce corespunzător datele de intrare in fisier.



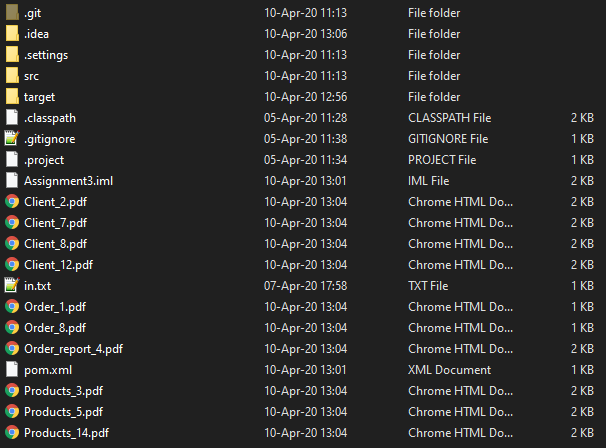
Fiecare rand reprezinta cate o operatie ce trebuie efectuata asupra bazei de date:

* Inserare client
* Stergere client
* Inserare produs
* Stergere produs
* Generare comanda
* Generare raport pentru unul din cele 3 tabele

1. Utilizatorul ruleaza aplicatia dintr-o fereastra command prompt, deschisa din folder-ul in care se afla aplicatia, utilizand comanda „java -jar App.jar ‚path’/in.txt ‚path’/out.txt”.



1. Dupa terminarea rularii aplicatiei, se verifica fisierele de iesire, de tip pdf, pentru rezultate.



1. **Proiectare (decizii de proiectare, diagrame UML, structuri de date, proiectare clase, interfețe, relații, packages, algoritmi, interfață utilizator)**

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:**

A picture containing text, map

Description automatically generated

Am încercat sa modularizez cerința cat de mult posibil, astfel rezultând clasele din diagrama de mai sus: ClientBLL, Generate\_Order, OrderBLL, ProductBLL, ReportBLL, ConnectionFactory, ClientDAO, OrderDAO, ProductDAO, Client, Order, Product, Controller, Start. Clasele au fost proiectate intuitiv, pentru a fi ușor de citit si de înțeles.

Pachetele create sunt bll, connection, dao, model, presentation, start, conform arhitecturii structurata pe straturi(layered architecture). Astfel, logica aplicatiei se afla in pachetul bll, iar in pachetul dao sunt implementate operatiile de baza cu care opereaza asupra tabelelor: insert, (update), delete.

Pachetul connection contine clasa ce creeaza conexiunea de fiecare data cand aceasta este necesara. Pachetul model contine cele 3 clase de baza ale proiectului, Client, Product si Order. Astfel, putem spune ca acestea sunt nucleul proiectului, intrucat acesta nu ar putea functiona fara ele.

In final, pachetul controller contine clasa in care am implementat metoda cu ajutorul careia se citeste, linie cu linie, fisierul text de intrare, iar pachetul start este Main-ul aplicatiei.

1. **Implementare**

**Clasa ClientBLL**

Aceasta clasa contine logica de functionare a operatiilor care se pot executa asupra unui client, adica inserare si stergere. Aici se verifica daca clientul este deja prezent in baza de date, si daca da, se afiseaza un mesaj de eroare. De asemenea, daca se doreste stergerea unui client inexistent in baza de date, se afiseaza un mesaj de eroare aferent pe ecran.

**Clasa Generate\_Order**

Aceasta clasa contine metoda care genereaza fisierul PDF aferent comenzii, in caz ca aceasta este valida. Validitatea comenzii se verifica in OrderBLL. Fisierul PDF se genereaza cu ajutorul plugin-ului ITextPDF. Se creeaza cate un chunk, dupa care cate un paragraf separat pentru acesta, pentru a evita suprapunerea nedorita a acestora in documentul generat, iar apoi se adauga in acesta. Am folosit o variabila declarata static, pentru a putea genera de fiecare data un fisier pdf nou, pentru a evita suprascrierea acestuia, in caz ca in fisierul text de intrare se afla mai multe cereri de comenzi ale clientilor.

**Clasa OrderBLL**

Aceasta clasa contine logica de functionare a plasarii unei comenzi. Se verifica daca produsul se afla pe stoc, daca stocul este suficient si daca clientul se afla deja in baza de date. Daca una din aceste conditii nu este indeplinita, se afiseaza un mesaj de eroare pe ecran aferent conditiei neindeplinite de aceasta sau se genereaza un document pdf in care se scrie faptul ca stocul nu este suficient.

**Clasa ProductBLL**

Aceasta clasa contine logica de functionare a operatiilor care se pot executa asupra unui produs, adica inserare, stergere. Daca cumva produsul se afla deja in baza de date, se considera un update al cantitatii acestuia, iar daca nu, se introduce in baza de date, insotit de cantitatea citita din fisierul de intrare. Daca se incearca stergerea unui produs ce nu se afla deja in baza de date, se va afisa un mesaj de eraore aferent pe ecran.

**Clasa ReportBLL**

Aceasta clasa contine logica generarii unui raport al unui tabel prezent in baza de date. Daca se doreste generarea unui raport, in functie de comanda citita din fisierul text de intrare, se va apela metoda potrivita, care va genera raportul dorit, sub forma de tabel, asa cum este prezent in baza de date. Am folosit o variabila declarata static, pentru a putea genera de fiecare data un fisier pdf nou, pentru a evita suprascrierea acestuia, in caz ca in fisierul text de intrare se afla mai multe cereri de generare raport pentru tabelele din baza de date.

**Clasa ConnectionFactory**

Aceasta clasa este responsabila de crearea si inchiderea conexiunii cu baza de date. De asemenea, exista o metoda care realizeaza inchiderea unui ResultSet si o metoda care inchide un Statement, ce sunt mai depart utilizate in metodele care lucreaza cu acestea.

**Clasa ClientDAO**

Aceasta clasa implementeaza metodele pentru adaugarea, stergerea, respectiv afisarea tutror clientilor din baza de date. Se foloseste de logica care verifica daca clientul se afla deja in baza de date din clasa ClientBLL.

**Clasa OrderDAO**

Aceasta clasa implementeaza metodele pentru generarea unei comenzi, respectiv afisarea tuturor comenzilor din baza de date. Se foloseste de logica care verifica daca produsul si clientul exista in baza de date si daca exista stoc suficient din clasa OrderBLL.

**Clasa ProductDAO**

Aceasta clasa implementeaza metodele pentru adaugarea, stergerea, actualizarea cantitatii, respectiv afisarea tutror produselor din baza de date. Se foloseste de logica, care decide daca se efectueaza insert sau update, din clasa ProductBLL.

**Clasa Client**

Aceasta clasa face parte din nucleul aplicatiei. Fara existenta ei, aplicatia nu ar putea functiona. In aceasta clasa se definesc atributele unui client, impreuna cu gettere si settere, dar si o metoda toString.

**Clasa Order**

Aceasta clasa face parte din nucleul aplicatiei. Fara existenta ei, aplicatia nu ar putea functiona. In aceasta clasa se definesc atributele unei comenzi, impreuna cu gettere si settere, dar si o metoda toString.

**Clasa Product**

Aceasta clasa face parte din nucleul aplicatiei. Fara existenta ei, aplicatia nu ar putea functiona. In aceasta clasa se definesc atributele unui produs, impreuna cu gettere si settere, dar si o metoda toString.

**Clasa Controller**

Aceasta clasa implementeaza metoda read(), cu ajutorul careia se face citirea din fisierul de intrare. Citirea se face cu BufferedReader, pentru a putea opera cu fiecare linie din fisier in parte. Astfel, se realizeaza split-uri in functie de structura fisierului de intrare. In cazul de fata, a fost nevoie doar de split-uri cu caracterele „:” si „ ” (spatiu). In acest fel se identifica comanda data de utilizator, se extrag informatiile necesare (ex: nume, adresa, nume produs, cantitatea stocului produsului, pretul produsului, numele clientului pentru care se genereaza comanda, produsul dorit de client si cantitatea acestuia), si se apeleaza mai departe metoda aferenta operatiei cerute.

**Clasa Start**

Aceasta clasa implementeaza metoda Main a aplicatiei. Aici se apeleaza doar metoda read() din clasa Controller, cu parametrul args[0], pentru a se putea rula direct din Command Prompt, cu paramentrul potrivit.

1. **Rezultate**

Pentru cazurile de testare am testat aplicatia pentru un fisier text de intrare, care contine cel putin o comanda din cele implementate in proiect, pentru a putea observa rularea si corectitudinea implementarii. De asemenea, am realizat si un fisier .jar, pe care l-am atasat in proiect, si care poate fi rulat cu parametrii cailor pentru fiserul de intrare si cel de iesire.

Astfel, se creeaza clienti si se genereaza cate un raport al bazei de date a clientilor dupa ficare operatie de adaugare. Apoi se introduc produse, se face update unuia dintre ele, dupa care unul din ele este sters. Se genereaza mai apoi un raport al bazei de date pentru produse, se plaseaza 2 comenzi, dupa care, in final, se genereaza cate un raport pentru clientii, produsele si comenzile prezente in baza de date.

Rezultatele din etapa de testare arata ca aplicatia este modelata corect, iar algoritmul de generare a comenzilor, cat si cel de gestiune a clientilor si a produselor, este functional.

1. **Concluzii**

In concluzie, sunt de părere ca acest proiect m-a ajutat sa îmi aprofundez cunoștințele in tot ce înseamnă limbajul Java, implementarea paradigmelor OOP, si dezvoltarea unui algoritm eficient, care sa satisfacă obiectivul unui proiect.

Aceasta tema simuleaza situatii din viata de zi cu zi, intrucat, oriunde avem nevoie sa mergem, deseori nu ne dam seama ca, de fapt, in spatele oricarui magazin sau depozit, exista o baza de date de dimensiuni mari, si o aplicatie de gestiune a acesteia, similara cu cea implementata in acest proiect.

Chiar daca simularea este corecta, ea nu poate prevedea toate situatiile neasteptate din realitate. In aceasta tema am invatat sa lucrez cu plugin-ul ITextPDF, sa generez un document PDF utilizand limbajul Java, dar si sa il folosesc pentru a putea comunica cu o baza de date existenta pe dispozitiv. De asemenea, am aprofundat generarea si configurarea unui fisier .jar si rularea din linia de comanda, folosind parametri.

In ceea ce privește dezvoltarea ulterioara a programului:

* Se poate imbunatati performanta algoritmilor
* Se pot implementa operatii de update asupra tabelelor din baza de date.
* Utilizarea abstractizarii claselor care contin operatii asupra tabelelor din baza de date (folosirea arhitecturii de tip reflection)
* Se poate configura fisierul .jar astfel incat parametrii fisierelor de intrare si de iesire sa fie doar in.txt si out.txt si sa se poata rula doar cu ajutorul comenzii „java -jar App.jar in.txt out.txt”

1. **Bibliografie**

* <https://stackoverflow.com/>
* <http://docs.oracle.com/>
* <https://www.geeksforgeeks.org/>
* youtube.com - tutoriale