*Universitatea Tehnică Cluj-Napoca*

*Facultatea de Automatică și Calculatoare*

**Order Management**

*Gabor Tania Loreana*

*Grupa:30224*

**1.Obiectivul principal**

Proiectarea și implementarea unui sistem de gestionare a comenzilor care are ca scop administrarea unui depozit și ține evidența clienților, produselor, comenzilor (creează noi comenzi).  Obiectivul principal al acestei aplicații este de a oferi proiectarea și implementarea unei aplicații pentru procesarea comenzilor clienților pentru un depozit.

Acesta poate fi descompus în următoarele obiective secundare:

* Crearea unei structuri de date utilizate pentru stocarea informațiilor despre clienți
* Crearea unei structuri de date utilizate pentru stocarea informațiilor despre produse
* Creearea clasei pentru implementarea operațiilor din baza de date pentru clienți, produse și comenzi,bonuri.
* Crearea unei conexiuni la o bază de date.
* Crearea unei baze de date pentru stocarea informațiilor despre fiecare client, produs și comandă,bonuri.
* Afisarea clientilor intr-un tabel, adaugarea, editarea si eliminarea unui client.
* Afișarea produselor într-un tabel, adăugarea, editarea și eliminarea un produs.
* Afișarea comenzilor într-un tabel, adăugarea ,editarea și eliminarea unei comenzi.
* Generarea după fiecare comandă a unui fișier de factură(sau bon) care conține informații despre acea comandă.

**2. Analiza problemelor**

În ziua de azi pe măsură ce depozitele devin din ce în ce mai mari și din ce în ce mai greu de gestionat și organizat, sistemele de administrare a magazinelor devin vitale în organizarea depozitelor și produselor din aceste depozite. Această temă include proiectarea unui sistem care să ajute managerii magazinelor să aibă acces mai ușor la datele lor, să gestioneze aceste date, prin adăugarea de clienți, editarea clienților, ștergerea clienților , tipărirea unei facturi pentru un anumit client, având o listă de clienți care se actualizează dinamic prin modificările efectuate de utilizator, adăugarea unui produs, eliminarea unui produs și editarea unui produs, având o listă de produse care se actualizează tot dinamic de către un utilizator reprezentat în acest caz de fisierul de input.Aplicația permite, de asemenea, utilizatorului să adauge sau să șteargă comenzi folosind o listă de comenzi actualizate dinamic.

Procesul de modelare este definit ca fiind procesul de împărțire a unei probleme mai mari în multe mai mici, care sunt mai ușor de înțeles și depanat. Acest lucru contribuie, de asemenea, la clarificarea unei idei abstracte. În dezvoltarea de software, modelarea este esențială pentru a construi o aplicație care are o structură complicată. Problema noastră, așa cum s-a menționat mai sus în paragraful „Obiectivul principal”, include un sistem de gestionare a datelor unui magazin. Pentru cazul minim pe care l-am dezvoltat în această aplicație, folosim doar 4 tipuri de elemente: clienți, comenzi ,facturi(bonuri) și produse. Această aplicație funcționează cu o bază de date care conține sistemul de tabele pe care tocmai l-am descris, de aceea, într-o primă abordare de gestionare a datelor, trebuie mai întâi să proiectăm structura bazei de date. Folosesc, în acest caz, 4 tabele, descrise anterior: clienți, comenzi , produse, fiecare având coloane care vor fi descrise în cele ce urmează. După faza de proiectare a bazei de date, vine faza de proiectare a claselor, unde am folosit o mapare de clase care arată ca baza de date. Pentru a face o legătură între acestea două, avem nevoie de o conexiune între acestea,adică de o clasă specială care să preia datele din baza de date.

**3.Utilitatea aplicației**

Aplicația software dezvoltată este una la nivel de bază, prin urmare, simulează doar un mod de a crea comenzi.Aplicația este utilă, totuși, pentru organizarea depozitelor simple, cum ar fi, de exemplu, cele ale unui supermarket, oferind utilizatorului posibilitatea de a adăuga, edita sau șterge cu ușurință elemente din tabelele de care are nevoie.

Această aplicație ar putea fi utilă, de exemplu, într-un magazin online pentru a face livrări. Am putea gestiona clienții care care cumpără lucruri de la magazinul online. De asemenea există tabelul care prezintă elementele pe care le are magazinul la un anumit moment. Clienții pot plasa comenzi, alegând ce produs vor să cumpere și

în cazul în care cantitatea de care acesta are nevoie nu este disponibilă, în factura generată se va afișa acest mesaj.Baza de date din spatele acestei aplicații are un rol important făcându-și update de fiecare dată când se efectuează o operație asupra acesteia.

1. **Exemplu de funcționare a aplicației**

Să luăm în considerare un exemplu de funcționare a aplicației;

NOTE IMPORTANTE:

* Citirea din fisier a numelui clientului se face în două variabile string.
* Citirea din fisier a variabilei cantitate conține doar numere reale.
* Citirea din fisier a variabilei preț conține doar numere reale.
* Citirea din fisier a variabilei adresă se face într-un string.
* Citirea din fisier a denumirii produsului se face într-un string.
* Aplicația nu permite noi comenzi care conțin produse pentru care cantitatea e<0.
* Toate modificările aduse tabelelor sunt actualizate instantaneu și în baza de date.

Să luăm în considerare un scenariu în care un utilizator(voi considera drept utilizator fișierul de input) dorește să adauge un client, să adauge două produse și să plaseze o comandă cu o cantitate existentă q din primul produs.

Pași:

- Utilizatorul (reprezentat prin fișierul de input) introduce corect datele.

-Aplicația parsează stringul introdus în numele clientului pe care dorește să-l adauge și adresa acestuia.

- Clientul este adăugat în listă , dar și în tabela corespunzătoare din baza de date.

- Aplicația parsează următorul string introdus în numele produsului pe care dorește să-l adauge ,cantitatea și prețul acestuia.

-Introduce datele produsului corespunzător în tabela și in listă.

-Se procedează la fel și pentru al doilea produs cu specificația că, aplicația verifica ca în baza de date să nu fie introdus un produs cu același numa, în acest caz făcând update doar la cantitate.

-Utilizatorul plasează comanda.

-Aceasta este introdusă in tabela de comenzi, cu specificația că daca clientul a fost șters din tabela de clienți se va șterge și din comanda plasată de acesta din tabela de comenzi , dar și din lista aceasta.

- Cantitatea q este scăzută din cantitatea totală a primului produs.

1. **Design**

**5.1 Baza de date**

Baza de date a fost construită asemeni tabelelor din pachetul model, pe lângă câmpurile corespunzătoare mai conține și o cheie primară reprezentată printr un număr întreg și setată ca auto-increment.

**Parola pentru accesul în baza de date este : parola123@**

**Username-ul pentru accesul în baza de date este:root**

drop database if exists warehouse;

create database if not exists warehouse;

use warehouse;

SET GLOBAL time\_zone = '+3:00';

create table if not exists customer(

id int unique auto\_increment,

nameClient varchar(20),

adress varchar(20),

primary key(id));

create table if not exists orders(

id int unique auto\_increment,

nameClient varchar(20),

nameProdus varchar(20),

quantity int,

primary key(id));

create table if not exists products(

id int unique auto\_increment,

nameProdus varchar(20),

quantity float,

price float,

primary key(id));

create table if not exists bills(

id int unique auto\_increment,

nameProdus varchar(20),

nameClient varchar(20),

totalPrice float,

primary key(id));

**5.2 Pachetul Model**

Pachetul model conține definiția claselor utilizate pentru procesarea datelor din baza de date adică clasa de clienți, clasa de comandă , clasa de produse și clasa de facturi.

- Clasa de clienți are 2 câmpuri: numele clientului (conține și prenumele clientului )și adresa. De asemenea conține 1 constructor cu toate câmpurile. De asemenea conține getters și setters pentru fiecare variabilă din clasă.

- Clasa de comenzi are 3 câmpuri: numele produsului, numele clientului și cantitatea

produsului. Aceasta conține, de asemenea 1constructor cu toate câmpurile. De asemenea conține getters și setters pentru fiecare variabilă din clasă.

- Clasa de produse are 3 câmpuri: , numele produsului,

cantitatea produsului, prețul produsului. Aceasta conține, de asemenea, 1 constructor cu toate câmpurile. De asemeneaonține getters și setters pentru fiecare variabilă din clasă.

- Clasa defacturi are 3 câmpuri: , numele produsului,

numele clientului, prețul produsului. Aceasta conține, de asemenea, 1 constructor cu toate câmpurile. De asemenea,conține getters și setters pentru fiecare variabilă din clasă.

**5.3. Pachetul bussinessLayer**

Pachetul bussinessLayer conține o singură clasă, managerul, care este obligatoriu si este un mediator între celelalte 2 straturi ale aplicației baza de date și fișierul de input. Stratul de intrare în cazul nostru este reprezentat de clasele model și de fișierul de input. Această clasă manager conține 4 liste: lista de clienți, lista de comenzi,lista de produse și lista de bonuri.Conține un constructor care are drept argumente un obiect de tipul generator de pdf,listele de clienți , produse,comenzi și facturi, conexiunea care face legatura cu package-ul Connection Factory.

Clasa manager conține metode pentru adăugarea de elemente la fiecare tip de listă .După adăugarea obiectelor la listele locale urmează adăugarea lor în baza de date prin apelarea metodei de introducerea obiectului specific .Acest lucru se face printr-o interogare realizata din Java.Clasa aceasta conține, de asemenea, opțiuni de editare a elementelor din fiecare tip de listă:adăugare,ștergere.Conține de asemenea și metode pentru generarea de rapoarte particulare pentru ficare obiect.În aceste metode creeaza un obiect de tipul GeneratePdf() si apelează metoda pentru acel obiect specific.Această clasă are o importanță foare mare, substanțială în acest proiect fiind legătura dintre cele 2 straturi sau porțiuni.Aici se implementează logica care are drept scop obținerea modificărilor asupra listelor din java dar și asupra tabelelor din baza de date.

**5.3. Pachetul dataAccess**

Pachetul dataAcces este o idee foarte bună în ceea ce privește

conexiunea la baza de date pentru folosirea modelului Singleton implementat de o“fabrică de conexiuni”. Acest pachet conține clasa asa numită ConnectionFactory creează o conexiune la baza de date după care clasa manager operează cu tabela de care are nevoie în acel moment. În acest mod, la nivelul aplicației există o singură conexiune de bază de date globală.În această clasă am setat numele bazei de date, numele userului si parola.

**5.3. Pachetul presentation**

În acest pachet am inclus două clase:GeneratePdf și ReadFile.În clasa GeneratePdf am creat metode pentru generarea diferitelor tipuri de rapoarte.Am declarat un obiect de tipul Document și două fonturi pentru capetele de tabel respectiv restul datelor din tabele. Această clasă conține un constructor fără parametrii,iar metodele din aceasta sunt apelate în clasa Manager.Clasa ReadFile conține o metodă foarte importanta readFile() care preia fiecare String din fișierul de input și îl parsează în mai multe.Am folosit o listă de stringuri pentru a decide ce metodă trebuie folosită și pentru a identifica cazul în care mă aflu.

**5.3. Pachetul start**

Pachetul Start conține clasa Start care pornește întreaga aplicație. Acesta conține metoda main.În aceasta am creat un obiect de tipul Manager și un obiect de tipul ReadFile care primește drept parametru arg[0] care reprezintă un string, fișierul de intrare.

La finalul implementării proiectului am adăugat comentariile specifice pentru JavaDoc și am generat acest fișier precum si executabilul jar care primește drept argument fișierul din care se face citirea datelor.

**Concluzii**

Obiectivul principal al proiectului a fost crearea unui sistem care să poată efectua operațiuni pe o bază de date pentru a gestiona o colecție de clienți, produse , comenzi și facturi.și, de asemenea, să gestioneze plasarea comenzilor și generarea unei facturi pentru fiecare comandă. Procesul de dezvoltare a implicat învățarea următoarelor abilități:

* Citirea dintr-un fișier de input a datelor
* Parsarea datelor în mai multe substringuri.
* Proiectarea și dezvoltarea unui proiect folosind mai multe pachete
* Aplicarea paradigmelor OOP
* Crearea unei structuri de date (clasă) pentru reprezentarea clienților, produselor ,facturilor și
* Lucrul cu conexiunile la baze de date și crearea bazelor de date
* Fixarea cunoștintelor legate de limbajul SQL

**7.Rezultatul**

Rezultatul acestui proiect este o aplicație care conține gestionarea unor probleme de marketing ,aceasta satisface nevoile unui depozit standard cu produse și clienți, comenzi și facturi ,făcând conexiunile la baza de date și obținând în același timp date despre conținutul bazei de date.

**8.Posibile îmbunătățiri**

Implementarea unor operațiuni mai eficiente în baza de date,dezvoltarea structurii bazei de date pentru extensii suplimentare ale aplicației . S-ar putea implementa o interfață grafică pentru această aplicație care să conțină câmpuri de text

pentru adăugarea datelor.Această aplicație s-ar putea extinde gestionând mai multe tabele sau prin adăugarea diferitelor niveluri de utilizator la aplicație, adăugarea mai multor detalii in ceea ce privește clienții,adăugarea mai multor detalii despre produse; ,afisarea detaliilor clientului si detaliile produsului intr-o interfață grafică, adăugarea posibilității de a adăuga mai multe produse în aceeași comandă.

**9.Bibliografie**

[https:// www.baeldung.com/java-pdf-creation](https://www.baeldung.com/java-pdf-creation)

<https://dzone.com/articles/layers-standard-enterprise>

<https://ibytecode.com/blog/jdbc-mysql-create-database-example/>

<https://mkyong.com/jdbc/how-to-connect-to-mysql-with-jdbc-driver-java/>

Tehnici de programare - Prelegerea prof. univ. Ioan SALOMIE