**Documentație**

Disciplina

Tehnici de programare

**Tema 3**

**Order Management**

**Pîrvu Andreea Gabriela**

An academic:2019-2020

Cuprins

1. Obiectivul temei
2. Analiza problemei, modelare, scenarii, cazuri de utilizare
3. Proiectare
4. Implementare
5. Concluzii
6. Bibliografie

**1.Obiectivul temei**

* 1. **Obiectivul principal**

Scopul principal este realizarea unei aplicații de gestionare a comenzilor pentru procesarea comenzilor clienților pentru un depozit. Bazele de date relaționale sunt utilizate pentru a stoca produsele, clienții și comenzile. Definiți, proiectați și implementați un sistem de programe de utilitate (exemple: rapoarte pentru stocuri, totaluri, filtre etc.).

Aplicația oferă posibilitatea de a efectua anumite calcule pe datele deținute de baza de date, cum ar fi:

Adăugarea unui client , a unui produs sau a unei comenzi

Ștergerea unui client sau a unui produs

Generarea unei facturi pentru clienții care au făcut o comandă precum și actualizarea stocului după efectuarea comenzii. De asemenea, daca cantitatea pe care clientul o cere nu este disponibila acesta primește un Pdf cu mesajul „Out of stock. Stock available:” și este anunțat cu privire la stocul disponibil in depozit.

Generarea rapoartelor sub forma de tabel a tuturor clienților, produselor și comenzilor înregistrate în baza de date.

* 1. **Obiective secundare**

Pentru îndeplinirea scopului principal:

1. Crearea claselor necesare;
2. Stabilirea caracteristicilor și comportamentelor obiectelor necesare;
3. Implementarea metodelor necesare unei clase;
4. Extragerea informațiilor din fișiere.txt;
5. Procesarea datelor in baza de date
6. Generarea pdf-urilor
7. **Analiza problemei, modelare, scenarii, cazuri de utilizare**

**2.1 Analiza problemei**

Un sistem de gestionare a comenzilor are multe aplicații utile în viața reală, fiind integrat pe toate site-urile online care au orice fel de comandă și livrare. Acest lucru oferă clienților posibilitatea de a alege dintr-o varietate de produse puse la dispoziție pentru a fi achiziționate de la distanță prin intermediul unor clicuri. Îmbunătățește și ușurează interacțiunea dintre producător și client, deoarece procedura este mult mai rapidă și mai convenabilă pentru ambele părți.

Așa cum am menționat anterior, site-urile online care au ceva de vândut sau oferit trebuie să utilizeze o astfel de aplicație, astfel încât proprietarul unei astfel de afaceri poate beneficia de aceasta și poate solicita ca acest program să fie dezvoltat în funcție de nevoile sale.

În spatele aplicației în sine, trebuie să avem o bază de date care să conțină toate datele prelucrate, stocate și disponibile pentru a fi utilizate sau accesate oricând.

Pentru a interacționa cu utilizatorul, programul solicită comenzile prin intemediul unui fisier de intare, astfel atunci cand în fișier găsește comanda Insert client: nume client, adresa, in baza de date se va insera clientul respective. Analog și la Insert Product. De asemenea atunci când în fișier se găsește comanda Delete client sau Delete Product, clientul respectiv și produsul respective sunt șterși din baza de date. Atunci când în fișier se găsește comanda Order: nume, produs, cantitate, atunci programul va genera un Pdf cu factura clientului care a efectuat comanda. Alte comenzi care pot fi găsite în fișier sunt Report client, Report order, Report product, în cazul acestora se genereaza un Pdf cu un tabel cu toți clienții, toate produsele, respectiv toate comenzile din baza de date.

**2.2 Modelarea**

Pentru gestionarea unui sistem de gestionare a comenzilor, trebuie să ne creăm clasele în conformitate cu tot ceea ce este necesar pentru a deține informațiile pe care le presupune un astfel de calcul. Prin urmare, programul constă în primul rând dintr-o clasă numită „Client”, una numită „Produs” și o altă clasă numită „Comandă” care, practic, creează o legătură între celelalte două. În plus, avem clase care se ocupă de calculele bazelor de date realizate pe obiectele instantaneate de aceste clase.

De asemenea avem clasa care realizeaza parcurgerea fișierului de intrare pe baza căruia se apeleaza metodele necesare.

* **Cerințe funcționale:**

Sistemul de gestionare a comenzilor trebuie sa functioneze corect.

aceasta se intelege ca realizarea unei comenzi trebuie sa fie corecta, in functie de clientii si produsele din baza de date, si reprezentata sub forma asteptata de utilizator.

De asemenea, informatiile transmise de utilizator prin fisierul de intrare trebuie interpretate corect , stocate corespunzator.

* Cerinte:

- inserarea și ștergerea clienților;

- inserarea și ștergerea produselor;

- realizarea corecta a unei comenzi precum și generarea unei facturi cu prețul total sau avertizarea clientului atunci când produsul nu este disponibil;

- utilizarea metodei de Reflection;

- respectarea cantitatii unui produs;

- generearea pdf-urilor cu rapoarte privind tabelele din baza de date.

**2.3 Use-cases**

Entry conditions: utilizatorul a introdus comenzile pe care vrea să le efectueze asupra aplicației din fișer și poate vizualiza apoi pdf-urile pentru a vedea facturile clienților precum și conținutul bazei de date.

Exit conditions: programul insereaza, sterge, reinnoieste datele unui client/ produs sau realizeaza o comanda (și generează factura corespunzătoare acesteia).

Flow of events:

1. Introducerea datelor în fișierul de intrare

Comenzile pe care programul le suporta sunt:

Insert client: nume, adresa;

Report client;

Delete client: nume, adresa;

Insert product: nume, cantitate, preț;

Report product;

Delete product: nume;

Order: numele clientului, numele produsului, cantitatea;

2) Se rulează programul

3) Se vizualizează rezultatele în pdf-urile care s-au generat

1. **Proiectare**

Programarea Orientata pe Obiecte este o paradigmă de programare care utilizează obiecte și interacțiuni între acestea pentru a modela arhitectura unui program.

Aplicația creată respectă principiile OOP:

-orice obiect are un tip (este o instanță a unei clase);

-încapsulare;

-un obiect nu poate modifica starea alui obiect decât prin intermediul metodelor;

-abstractizarea (fiecare obiect are rolul unui actor abstract care iși poate modifica valoarea, poate realiza acțiuni și poate comunica cu alte obiecte din sistem fără a-și dezvălui implementarea facilităților);

-obiectele pot comunica între ele;

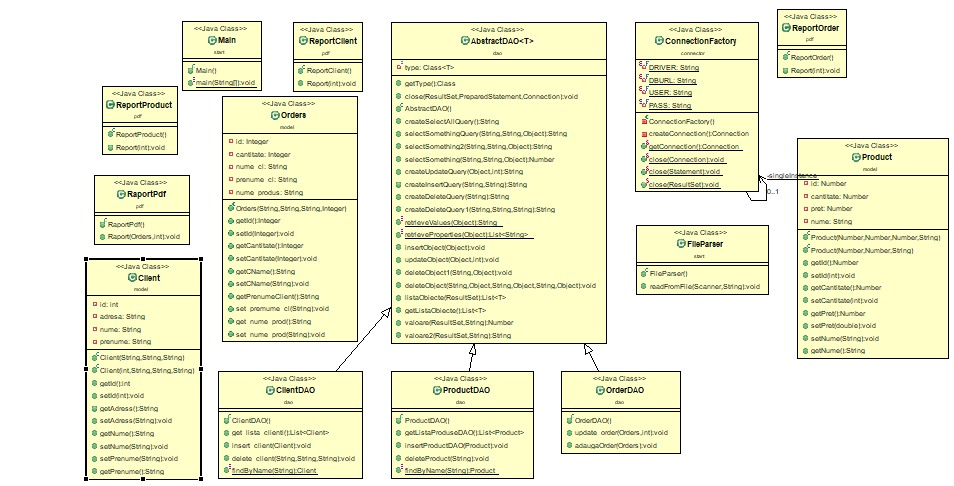
Aplicația lucrează cu comenzi a căror funcționalitate este procesată de o baza de date.

* **Diagrama UML**

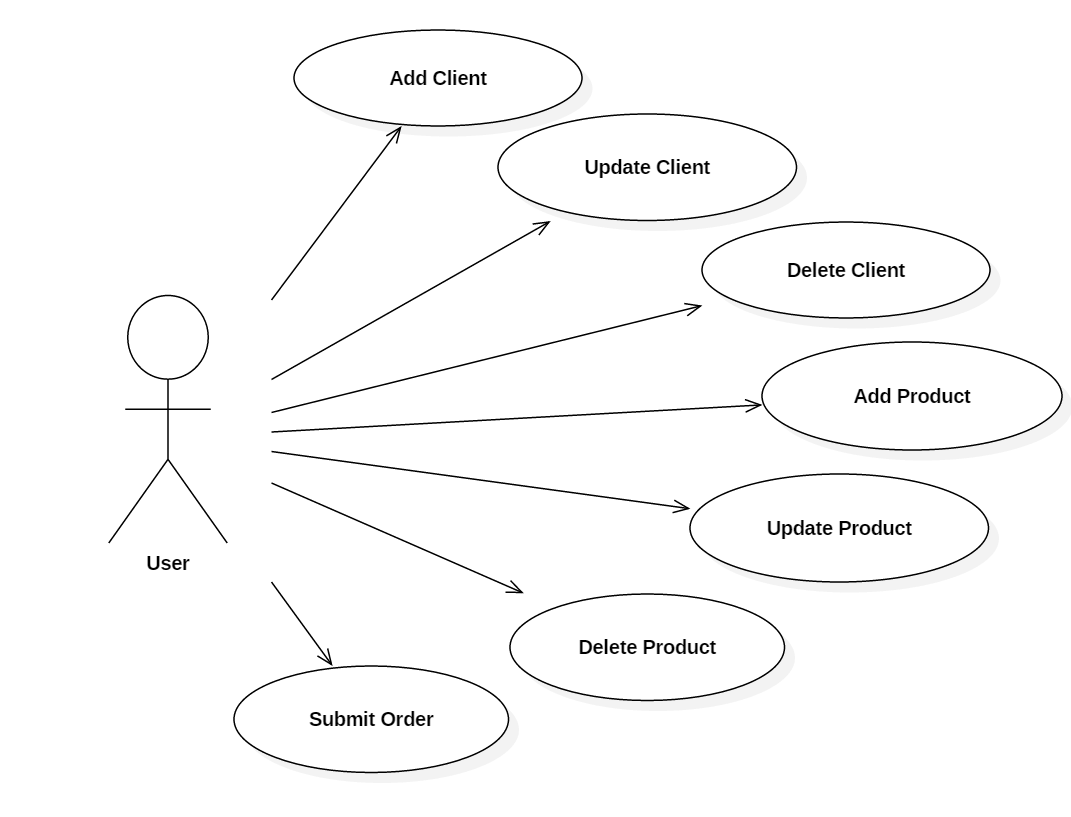
UML este un limbaj comun utilizat pentru a descrie, specifica, proiecta și documenta procesele, structura și comportamentul de afaceri existente sau noi, a artefactelor sistemelor software.

Pentru a arăta designul programului, am ales să creez și să prezint diagrama clasei, deoarece este cea mai specifică și mai utilizată diagrama UML folosită în general. Este blocul principal al oricărei soluții orientate pe obiecte. Prezintă clasele dintr-un sistem, atributele și operațiile fiecărei clase și relația dintre fiecare clasă. Prin urmare, toate detaliile de implementare necesare despre legăturile făcute între clase și modul în care se folosesc reciproc pot fi citite din aceasta.

În majoritatea instrumentelor de modelare, o clasă are trei părți, nume în partea de sus, atribute în mijloc și operații sau metode în partea de jos.



Use cases diagram

****

* **Structuri de date**

Dupa cum am spus, aplicatia este creata pe baza interactiunii dintre utilizator cu clientii si produsele. Astfel, principalele structuri de date cu care se opereaza sunt structura Client si structura Product.

Structura Product are urmatoarele atribute : private Number id; private Number cantitate, private Number pret, private String nume.

Clientul are ca si proprietati private Integer id, private String adresa, private String nume și private String prenume.

O comanda are ca si atribute : private Integer id, private Integer cantitate, private String nume\_cl, private String prenume\_cl, private String nume\_produs.

1. **Implementare**

* **Clasa Client**

Această clasă este una dintre principalele, deoarece inițiază obiectele Client necesare pentru popularea uneia dintre tabelele bazei de date. Acestea au ca atribute id-ul, numele, prenumele și adresa. Mai mult, acesta are doi constructori, cu următoarele argumente:

public Client(String adresa,String nume, String prenume)

public Client(int id,String adresa,String nume, String prenume)

* **Clasa Product**

Alături de clasa Client, aceasta se află în centrul programului, creând obiectele Produs care sunt o parte principală a procesului de comandă. Atributele prezentate aici sunt id-ul, numele produsului, cantitatea de pe stoc și prețul produsului. Clasa are, de asemenea, doi constructori, cu următoarea formă:

public Product (Number id,Number cantitate,Number pret,String nume)

public Product (Number cantitate,Number pret,String nume)

Ca metode, avem setter și getter pentru toate atribute.

* **Clasa Orders**

Clasa „Orders” are ca atribute id-ul comenzii, numele și prenumele clientului care efectuează comanda, numele produsului comandat și cantitatea acestuia. Avem constructorul:

public Orders(String nume\_cl,String prenume\_cl,String nume\_produs,Integer cantitate)

Pe lângă aceasta, am implementat programe getter pentru toate atributele.

* **Clasa Connection**

Cu ajutorul acestei clase, creem conexiunea cu baza de date, pentru a putea efectua operațiile de insert, delete, update din codul Java.

* **Clasa AbstractDAO**

Această clasă va fi moștenită de clasele ProductDAO, ClientDAO, OrderDAO. Aceasta conține metodele neceare formarii querry-urilor pentru a putea efectua operații pe baza de date. Astfel, conține querry-uri pentru operații de insert, delete, update.

* **Clasa ClientDAO**

Această clasă moștenește clasa AbstractDAO și este responsabilă pentru toate calculele făcute pe tabela „client” din baza noastră de date numită „Magazin” care stochează toate datele despre clienții noștri capabili să efectueze o comandă.

Face conexiunea la baza de date și execută toate querry-urile cerute de programul nostru (insert, delete și cautarea unui client dupa nume și lista clientilor din baza de date). Prin urmare, avem următoarele metode:

public List<Client> get\_lista\_clienti()

public void insert\_client(Client c)

public void delete\_client(String nume\_client,String prenume,String adresa)

public static Client findByName(String name)

Nu avem niciun constructor implementat, deoarece clasa are un scop mai funcțional. Utilizăm în principal metodele sale într-un mod static.

* **Clasa ProductDAO**

Este similar cu clasa ClientDAO, fiind responsabil pentru toate calculele făcute pe tabelul „produs” din baza noastră de date, stocând toate informațiile suplimentare despre produsele disponibile pentru a fi comandate în depozitul nostru.

Face conexiunea la baza de date și execută toate querry-urile cerute de programul nostru (insert, delete și cautarea unui client dupa nume și lista clientilor din baza de date). Prin urmare, avem următoarele metode:

public List<Product> getListaProduseDAO()

public void insertProductDAO(Product p)

public void deleteProduct(String numeProdus)

public static Product findByName(String name)

Nu avem niciun constructor implementat, deoarece clasa are un scop mai funcțional. Utilizăm în principal metodele sale într-un mod static.

* **Clasa OrderDAO**

Similar cu ClientDAO și ProductDAO, implementează toate metodele necesare pentru a lucra la tabelul „comanda” din baza de date. Are mai puține calcule de făcut și, prin urmare, mai puține metode:

public void update\_order(Orders ord, int idOrd)

public void adaugaOrder(Orders ord)

* **Clasa RaportPdf**

În această clasă se generează pdf-ul cu facture clientului atunci când acesta face o comandă. Acest pdf conține data zilei în care s-a efectuat comanda, numele clientului, produsele comandate, prețul totatal al comenzii, precum și un mesaj.

De asemenea, dacă clientul efectuează o comandă, iar produsul nu este în stoc, se va genera un pdf cu mesajul corespunzător și va fi anunțat de stocul disponibil al depozitului.

* **Clasele ReportClient, ReportOrder și ReportProduct**

În aceste clase se genereaza pdf-uri cu conținutul tabelelor Client, Order, respectiv Product.

* **Clasa FileParser**

Cu ajutorul acestei clase preluam datele din intrare din fisierul.txt care conține comenzi. Am parcurs fisierul si atunci cand intâlnește striguri corespunzatoare unui șablon de comanda, programul apeleaza metoda corespunzătoare și astfel se efectueaza operațiile în baza de date.

* **Pachete și fișiere**

Am folosit 5 pachete “connector”, “dao”, “pdf”, “start” și “model”.

În pachetul “model” sunt prezente clasele principale ale programului pe baza cărora putem efectua operații: Client, Product, Orders.

În pachetul “pdf” sunte prezente clasele care genereaza pdf-uri RaportPdf (genereaza facture), ReportClient(generează datele din tabelul Client din baza de date), ReportOrder(generează datele din tabelul Orders din baza de date), ReportProduct(generează datele din tabelul Product din baza de date).

Nu am folosit interfață grafică, însă interacțiunea cu utilizatorul se realizează prin intermediul fișierelor de intrare. Astfel utilizatorul introduce comenzile pe care vrea sa le efectueze asupra

Comenzile pe care programul le suporta sunt:

-Insert client: nume, adresa;

-Report client;

-Delete client: nume, adresa;

-Insert product: nume, cantitate, preț;

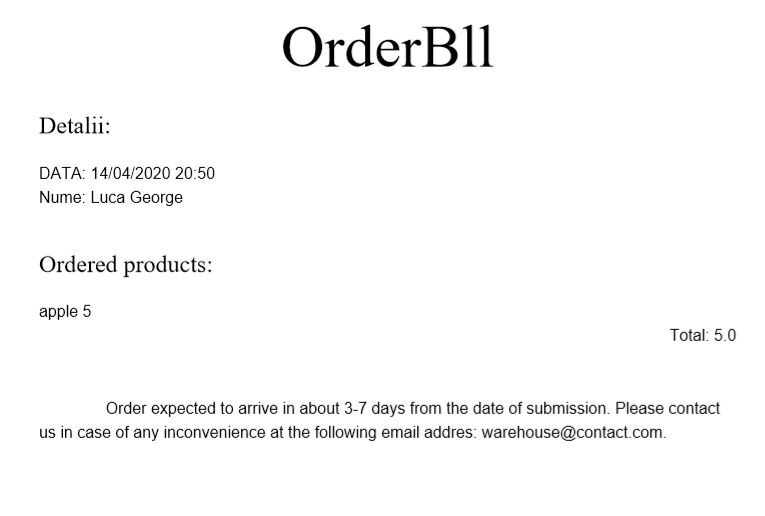
-Report product;

-Delete product: nume;

-Order: numele clientului, numele produsului, cantitatea;

* **Rezultate**

După trimiterea unei comenzi, programul se termină prin generarea unei facturi în format .pdf care arată astfel:

****

Pentru operațiunile făcute direct pe clienți și produs, putem vedea toate datele noastre scriind comanda Report client sau Report product sau Report order.

1. **Concluzii**

Combinarea cunoștințelor din bazele de date cu conceptele și regulile de programare orientată pe obiecte a fost o provocare cu un rezultat autocompătător. Văzând toți algoritmii și ideile se reunesc într-o singură unitate de lucru reprezintă implementarea și toată munca grea care duce la finalizarea unui proiect.

Specificația problemei s-a prezentat ca un subiect interesant, cu multe posibilități. Scrierea algoritmilor a ajuns să fie mai complexă atunci când avem de-a face cu conexiunea cu baza de date, deoarece practic calculăm date dintr-un alt program (MySQL Workbench).

* Cunostinte dobandite

În primul rând, cred că această misiune m-a ajutat să înțeleg mult mai bine conceptele de programare orientată pe obiecte, precum și să folosesc Java pentru prima dată în sincronizare cu o bază de date.

Am aflat despre biblioteci noi, cum ar fi conectorul mysql și itext, care păreau a fi foarte interesante de utilizat. A trebuit să descopăr toate metodele disponibile și utilizarea lor pentru a genera un fișier .pdf direct de la Eclipse.

* Ca și dezvoltări ulterioare:

Există încă mult spațiu pentru îmbunătățiri pentru a face programul mai eficient. Unii dintre algoritmi ar putea fi optimizati și pot fi implementate într-un mod diferit. Pe intrare pot fi impuse mai multe constrângeri, ceea ce înseamnă că se pot gestiona mai multe excepții și se pot crea mai multe operații pentru datele din tabele. Tehnicile de reflecție pot fi utilizate pentru a face codul mai compact și general, nefiind necesară implementarea acelorași metode pentru fiecare clasă din pachetul DAO. De asemenea, aplicației îi lipsește o interfață pentru ca interacțiunea cu utilizatorul să fie una mult mai “prietenoasă”.

1. .**Bibliografie**

1. <http://users.utcluj.ro/~jim/OOPE/>

2. <http://stackoverflow.com/>

3. <https://docs.oracle.com/javase/tutorial/java/javaOO/index.html>

4. <http://beginnersbook.com/2013/03/packages-in-java/>

5. <http://www.javatpoint.com/>