Universitatea Tehnica din Cluj-Napoca

Aprilie 2021

Tehnici de Programare

*~ Managementul comenzilor~*

Halas Denisa Izabela Grupa 30224

**Cuprins**

**Specificatiile problemei**…………………………………………….………………...…3

**Analiza Problemei**………………………………………………………………..…......3

Modelarea problemei……………..……………………………………….…..3

Use Case………………………………………………………………… ……5

**Design**……………………………………………………………………………………6

Diagrama de clase……………………………………………………………..6

Model: Client, Produs, Comanda…………………………………….………...7

Conexiune……………………………………………………………….………7

DAO: DAOClasaAbstracta, DAOClient, DAOProdus, DAOComanda…….....7

BLL: BLLClasaAbstracta, BLLClient, BLLProdus, BLLComanda……………8

MVC……………………………………………………………………………9

**Testare**…………………………………………………………………………………..10

**Rezultate**………………………………………………………………………………...10

**Concluzii**…………………………………………………………………………………11

**Webografie**………………………………………………………………………………12

# Specificatiile problemei

Considerati o aplicație de gestionare a comenzilor si pentru procesarea comenzilor clienților pentru un depozit. Bazele de date relaționale sunt utilizate pentru a stoca produsele, clienții și comenzile. În plus, aplicația folosește (minim) următoarele clase:

1. Clase specifice domeniului: Comandă, Client și Produs

2. Business Logic Classes(procesare specifică depozitului): Procesare comenzi,

Administrare depozit, Administrare client

3. Clase de prezentare: clase legate de GUI

4. Clase de acces la date: clase legate de accesul la baze de date

Alte clase și pachete pot fi adăugate pentru a implementa funcționalitatea completă a aplicației

# Analiza problemei

# Modelarea problemei

Prima și cea mai importantă concluzie extrasă din specificația problemei este că aplicația este, în principiu, un sistem bazat pe baze de date. Lucrul cu o bază de date implică posibilitatea de a manipula tabele și de a efectua operațiuni de selecție pe ele. Odată ce suntem capabili să efectuăm aceste operațiuni de bază, restul problemei implică doar dezvoltarea unui instrument ușor de folosit pentru utilizator.

Modelul bazei de date relaționale se refera la tabele și relații între acestea. Multe modele sunt cunoscute pentru sistemele de comandă, insa specificațiile implică o implementare directă. Cele 2 tabele fundamentale sunt Client și Produs. Acestea contin date obținute din lumea reală, fiecare tabel avand drept caracteristică principală un id. Id-ul este cel mai simplu mod de a distinge atat clienții, cat si produsele. Celelalte câmpuri (de exemplu: numele, adresa, email-ul, varsta, categoria sau cantitatea) nu sunt caracteristice care pot identifica în mod unic o linie din tabel deoarece, în situații din viața reală, putem găsi doi clienți cu același nume sau chiar două produse apartinand aceleiasi categorii (acest aspect este considerat astfel in mod intentionat, deoarece un magazin foarte organizat ar implica o scanare a identitatii fiecarui client in parte, dar si un cod unic de identificare pentru fiecare produs; in aceasta tema aceste date de identificare in mod unic sunt constituite de cheile primare din fiecare tabela, anume campurile id).

In cadrul unui magazine real consideram ca un client ar trebui să poată comanda unul sau mai multe produse care se gasesc in magazine (adica stocul sa fie sufficient).

Asadar, tabelele mentionate anterior nu sunt suficiente pentru a stoca toate datele de care avem nevoi. O rezolvare ar putea consta in adaugarea a cel puțin încă unui tabel care sa țina evidența particularităților unei singure comenzi. Astfel, o comandă este caracterizată de un id al comenzii, id-ul specific clientului care comanda și un id al produsului comandat.

Următorul tabel prezintă cele 3 entități existente in baza de date:

|  |  |
| --- | --- |
| Client | id, nume, adresa, email, varsta |
| Produs | id, nume, categorie, pret, cantitateProdus |
| Comanda | id, idClient, idProdus, cantitate |

Câmpul id (cheie primara) este fundamental pentru ca orice instrument de gestiune a bazelor de date îl necesită și acest lucru forțează proiectantul să utilizeze o implementare corectă pentru baza de date. Operațiile fundamentale pe care le presupune o comandă sunt:

1. Crearea unei corespondente intre client si comanda
2. Crearea a câte câmpuri solicita clientul, campuri ceconțin detalii despre produsele pe care dorește să le cumpere.
3. Finalizarea comenzii efectiata prin: validarea acesteia și actualizarea valorilor stocului.

Implementarea mentionata anterior are la baza 2 instrumente principale, anume MySQL și Java care ar trebui să comunice intre ele cat mai simplu cu putinta. Prin urmare, pentru a modela problema cât mai clar posibil, este util sa subliniez ca intre un tabel din MYSQL si o clasa din Java exista o echivalența incontestabila. Fiecare clasă din pachetul model al aplicației trebuie să corespundă unei entități din baza de date, deținând aceleași câmpuri și, asa cum este evident, aceleași tipuri ale câmpurilor). Dacă această echivalenta nu este realizată corect și cu atenție, programul nu este capabil să răspundă corect cerințelor. Fiecare cerință pe care o are creatorul unei baze de date corespunde unei așa-numite interogări. O interogare este reprezentata de solicitarea unor date/ informatii dintr-unul sau mai multe tabele apartinand bazei de date. Aceste interogări sunt efectuate prin limbaj SQL, acesta fiind un limbaj special conceput pentru gestionarea datelor păstrate într-o bază de date relațională. Regulile sale sunt destul de simple și afirmațiile pot fi ușor adaptate folosind manualul de cod. Dintr-un alt punct de vedere, comunicarea Java-MySQL trebuie făcută prin conectarea la server ori de câte ori este necesară o operație, urmata imediat de închiderea serverului. După aceea colaborarea este oarecum bazată pe șiruri, ceea ce înseamnă că interogările vor fi șiruri luate de catre MySQL care își va efectua propria operație și va întoarce rezultatele necesare (in cazul in care este necesar; în unele cazuri operațiunile sunt doar actualizări și nu necesită întoarcere).

Pentru o modelare correcta a problemei, au fost luate în considerare atat regulile menționate mai sus, cat si unele particularități ale MySQL. De la mașina Java trebuie, de asemenea, să trimitem câteva proprietăți aplicației in care se gaseste baza de date, astfel încât să poată fi identificata corect și unic baza de date dorită. Acest lucru se face prin declararea proprietăților acesteia, cum ar fi adresa URL-ul a bazei de date, serverul, utilizatorul de server și parola.

Problema folosește un model stratificat pentru implementare, ceea ce înseamnă o separare corectă între seturi de clase care au proprietăți comune. În acest sens, avem nevoie de 2 straturi pentru comunicarea bazei de date:

• Strat pentru conexiune

• Strat pentru acces la obiect

Stratul de conexiune este direct, ceea ce înseamnă că implementează conexiunea directa la la baza de date dorita. Stratul de acces la obiecte reprezintă portiunea de proiect care se ocupă de operațiunile de bază, anume inserarea, ștergerea, actualizarea și selecția pe baza anumitor criterii. Este folosit stratul de conexiune pentru a putea accesa baza de date și, din acel moment, folosește interogări pentru a actualiza / obține informații din baza de date a depozitului. Stratul de bază este stratul model care implementează obiectele care corespund tabelelor. Acest strat oferă acces la obiecte și, de asemenea, posibilitatea de a modifica caracteristicile acestora. Stratul de logică de afaceri (bll) trece de la partea bazei de date la partea reală a aplicației. Folosește în principal stratul de acces pentru a opera datele primite și pentru a efectua operațiunile solicitate de utilizator. De asemenea, se ocupă de validarea intrărilor și a unor date de ieșire cum ar fi un generator de bonuri pentru comenzi.

# Use Case

Diagram

Description automatically generated with medium confidenceLogo, company name

Description automatically generated



Logo, company name

Description automatically generated

A picture containing text, clipart

Description automatically generated

Graphical user interface

Description automatically generated with low confidence

A picture containing text, clipart

Description automatically generated

Logo, company name

Description automatically generated

Logo, company name

Description automatically generated

Logo

Description automatically generated

Graphical user interface

Description automatically generated

Logo, company name

Description automatically generated

Graphical user interface, application

Description automatically generatedLogo

Description automatically generated

A picture containing graphical user interface

Description automatically generatedUtiliatorul poate efectua diverse operatii, in functie de clasa asupra careia se efectueaza actiunea. Astfel pentru un client se pot efectua urmatoarele:

* adaugare de client nou (necesita totodata completarea manuala a campurilor cu datele personale ale fiecraui client nou adaugat)
* cautarea unui client (se face in functie de id, camp ce trebuie de asemenea completat)
* modificarea unui client (se face cu datele obtinute din cautarea anterioara, urmata de modificarea campului droit de catre utilizator)
* stergerea unui client (se face pe baza id ului)
* afisarea tabelului continand clientii de la momentul curent

Pentru un produs actiunile sunt exact aceleasi ca si la client, in sensul ca un nou produs poate fi adaugat, unul existent poate fi cautat, modificat sau sters si se poate afisa lista de produse existente la momentul current. Pentru efectuarea unei comenzi sunt necesari alti pasi: se completeaza campul corespunzator id ului clientului care va plasa comanda, se completeaza campul corespunzator id ului produsului/ produselor care vor si comandate si totodata cantitatea acestora. Exista un buton pentru afisarea comenzii si unul pentru printarea bonului.

Simulare adaugare client nou

Actor principal: Utilizatorul

Rulare finalizata cu succes:

• Utilizatorul lansează cu succes aplicația.

• Utilizatorul selecteaza tab-ul „Client” și se deschide meniul clientului.

• Se completează câmpurile necesare cu informații: nume, adresă, e-mail și varsta.

• Utilizatorul apasa butonul + de la adaugare client.

• Datele sunt introduse cu succes în baza de date și utilizatorul poate derula în jos pentru a vedea că noul client a fost adăugat.

Dacă utilizatorul introduce date în format greșit, de exemplu emailul cu model eronat se va semnala o eroare și utilizatorul va putea reintroduce datele.

Simulare creare comanda noua

Actor principal: Utilizatorul

Rulare finalizata cu succes:

• Utilizatorul lansează cu succes aplicația.

• Utilizatorul completeaza campul aferent clientului cu id0ul clientului care va plasa o comanda

• Utilizatorul completeaza campul aferent produsului cu id-ul produsului pe care il doreste.

• Utilizatorul completeaza campul aferent produsului cu cantitatea produsului pe care il doreste.

• Utilizatorul apasă butonul ,,+” afferent produsului

• Utilizatorul apasă butonul „Arata comanda”

• Utilizatorul apasă butonul „Printeaza bon”

Dacă utilizatorul dorește să obțină mai multe produse la comandă decât stocul corespunzător, este semnalată o eroare de subestocare.

# Design

# Diagrama de clase

# Model: Client, Produs, Comanda

În acest pachet, fiecare clasă reprezinta un obiect lumea reală. Clasa client definește cuvântul real persoană care dorește să facă o comandă și este păstrat în baza de date a sistemului.

Clasa de produs este minimalizată și conține doar numele, categoria, pretul și cantitatea produsului și poate fi dezvoltată în continuare (prin adăugarea de câmpuri precum producătorul și multe alte caracteristici).

Comanda reprezinta o relație de one-to-many intre client și produs. Totodata un client poate avea cât mai multe comenzi posibil, dar o comandă este definită de un singur client.

# Conexiune

Clasa principală a acestui pachet implementează conexiunea cu baza de date. Metodele sale se bazează exclusiv pe crearea conexiunii și închiderea acesteia. Atributele sunt cele ale serverului bazei de date:

Clasa are o metodă pentru crearea conexiunii, dar mai important, trei tipuri de închidere: închiderea instrucțiunilor (legate de interogare), închiderea conexiunii (închiderea efectivă a conexiunii cu baza de date) și închiderea setului de rezultate (închiderea rezultatului interogării executate).

Graphical user interface, text, application

Description automatically generated

# DAO: DAOClasaAbstracta, DAOClient, DAOProdus, DAOComanda

Acest pachet contine 4 clase, dintre care cea mai importanta este chala abstracta ce implementeaza toate metodele utilizate ulterior de catre celelalte 3 clase. In clasa abstracta sunt practice toate metodele necesare adaugarii, modificarii, cautarii si stergerii din baza de date a unui element. Asadar, este nevoie si de metode aditionale de transformare a campurilor si a valorilor acestora in stringuri.

# BLL: BLLClasaAbstracta, BLLClient, BLLProdus, BLLComanda

Acest pachet face simplu legatura intre cee ace ma insemna legarea de interfata si pachetul mentionat anterior, DAO. La fel ca mai devreme, aici se gasesc 4 clase: o clasa abstracta ce are metode de adaugare, mofificare, stergere si gasire, apelate din DAO, dar si alte 3 clase ce o extend, cate una pentru fiecare clasa din model.

# MVC

Acesta este principiul folosit tocmai pentru ca am ales sa lucrez pe un proiect JavaFX. Asadar a fost nevoie sa scriu codul pentru interfata, sa completez o clasa, ”Controller”, cu legaturile intre interfata si operatii, dar si un main in care am lansat practice aceasta aplicatie.

Interfata consta intr-o fereastra complexa ce are 4 tab-uri. Primul tab este acela care permite adaugarea, cautarea, modificarea si stergerea unui client. Cel de-al doilea tab este similar primului doar ca se refera la produs, iar cel de-al treilea este destinat comenzilor. Utilizatorul poate selecta un client, un produs si scrie o cantitate, iar dup ace apasa butonul de creare comanda aceasta va aparea pe cel de-al patrulea tab, alaturi de lista comenzilor anterioare. Mai mult decat atat, ultima comanda este mereu printata si pe un bon fiscal, un fisier ce se modifica in proiect de fiecare data cand este nevoie.

Graphical user interface, text, application

Description automatically generated

# Testare

Implementarea a fost realizată într-o manieră treptată. În primul rând, conexiunea la baza de date era prioritatea. După ce acesta a fost efectuata, generarea de interogări a fost următoarea problemă. Folosirea tehnicilor de reflecție a făcut posibilă testarea parțială a aplicației pentru fiecare tabel.

Aplicația în versiunea sa finală a fost testată pentru mai multe tipuri de operatii. O privire asupra functionarii aplicației este prezentată în următoarele cadre:

Gestionarea comenzilor a fost testată prin urmărirea bazei de date, dar și prin evaluarea conținutului final al fișierului pdf generat.

# Rezultate

Aplicația rezultată este una foarte ușor de manevrat de catre utilizator pentru a gestiona un magazin. Utilizatorii nu sunt obligați să înțeleagă particularitățile de implementare ale bazei de date, reusind sa tina evidența modificărilor foarte ușor. Operațiunile, care au fost destul de dificil de gestionat în cod, sunt aici simple clicuri sau completari de campuri de text. De asemenea utilizatorul nu este nevoit să furnizeze id-uri pentru elementele sale, acestea generandu-se automat.

Aplicația este, de asemenea, foarte utilă in cee ace priveste furnizarea tabelelor. În multe cazuri, actualizarea unei baze de date poate fi văzută prin reîmprospătarea ferestrelor sau chiar închiderea și deschiderea acestora. In cadrul acestei aplicatii totul se întâmplă în timp real și se poate observa că la fiecare eveniment care se întâmplă în baza de date, aplicația o semnalizează și prin actualizarea tabelelor. Operațiunile sunt, de asemenea, foarte ușor de utilizat.

# Concluzii

Această aplicație nu a constituit prima mea experiență cu sistemele de baze de date relaționale, dar a fost cu siguranta prima experienta atat de ampla privind gestionarea bazelor de date Java. Mi s-a părut destul de simplu să execut interogări în mediul Java, insa o problemă foarte importantă abordată de aceasta tema a fost lucrul cu tehnici de reflecție. A fost un foarte bun exemplu de utilizare a acestora pentru abstractizare și minimizarea lucrului efectuat prin scrierea repetata a codului. Părea destul de imposibil până la aceasta tema să obțin detalii despre un obiect al carui tip nu imi este cunoscut. Totodata interfata grafica a fost un punct obligatoriu de bifat, astfel ca experiența mea cu interfețele grafice s-a mai dezvoltat.

Dezvoltarea aplicației trebuie să înceapă prin actualizarea structurii bazei de date. Sunt necesare mai multe tabele (cum ar fi producătorul, adresa, țara etc.). Mai mult, aplicația ar fi putut folosi o abordare în 2 perspective: perspectiva administratorului și perspectiva utilizatorului. Utilizatorul ar putea fi identificat printr-un nume de utilizator și o parolă pe lângă celelalte atribute ale sale. Mai mult, administratorul și utilizatorul ar putea colabora prin comenzi, ceea ce înseamnă că un client ar putea solicita o comandă, în timp ce administratorul ar putea informa utilizatorul cu privire la starea stocurilor și a prețurilor.

# Webografie

1. <https://docs.oracle.com/javase/7/docs/api/java/sql/package-summary.html>
2. <https://docs.oracle.com/javase/tutorial/jdbc/basics/processingsqlstatements.html>
3. <https://stackoverflow.com/>
4. <https://bitbucket.org/utcn_dsrl/pt-layered-architecture.git>
5. <https://dzone.com/articles/creating-pdf-documents-with-apache-pdfbox-2><https://pdfbox.apache.org/>