**DOCUMENTATIE**

**TEMA 3**

**Order managemaent**

**Nume: Ilovan Mara Gabriela**

**Grupa: 30222**

**CUPRINS**

1. Obiectivul temei ............................................................................................. 3

2. Analiza problemei, modelare, scenarii, cazuri de utilizare ............................ 3

3. Proiectare ........................................................................................................ 5

4. Implementare .................................................................................................. 7

5. Rezultate ......................................................................................................... 8

6. Concluzii ......................................................................................................... 9

7. Bibliografie .................................................................................................... 10

1. **Obiectivul temei:**

Aceasta tema a avut ca scop principal implementarea unei aplicații de gestiune a comenzilor pentru procesarea comenzilor date de clienți. O comanda trebuie sa aibă asociat un client si un produs. Pentru fiecare element: client, produs, comanda avem nevoie de o baza de date alcătuita din tabele cu numele corespunzătoare elementelor pentru care sunt. Fiecare tabel are mai multe atribute cu datele necesare unei comenzi. Astfel, pentru clienți si produse trebuie sa putem realiza următoarele operații: inserarea unui nou client/produs, ștergerea unui client/produs, editarea datelor uni client/produs, dar si posibilitatea de a vedea toate elementelor de același tip intr-un tabel. In cea ce privește comanda: trebuie sa avem posibilitatea de creare a unei comenzi: un client comanda un produs, acesta poate alege cantitatea, iar daca cantitatea este prea mare va aparea un mesaj de eroare.

Alte obiective ale temei au fost: conexiunea cu baza de date, realizarea de metode folosind reflection. Folosind reflection, putem manipula structura, comportamentul si atributele obiectelor in timpul execuției. Deși aceasta tehnica are multe beneficii, folosita in exces poate afecta performanta aplicației. Totodată, pentru realizarea proiectului au fost necesare cunoștințe sql, pentru interogarea unei baze de date. Ar putea fi considerate obiective si: reușirea implementării unei interfețe ușor de utilizat si posibilitatea unor dezvoltări ulterioare.

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

Cerințele funcționale sunt acelea care descriu funcționalitățile pe care programul trebuie sa le îndeplinească, in timp ce cerințele non funcționale sunt cerințele care descriu programul.

Cerința problemei: realizarea unei aplicații de management al comenzilor pentru procesarea comenzilor clienților. Este necesara utilizarea unei baze de date relaționale pentru stocarea produselor, clienților si comenzilor. Aplicația trebuie realizata conform structurii:

1. Pachetul Model: conține clasele model reprezentând modelele de date ale aplicației. Atributele acestora vor fi identice cu cele din baza de date. De asemenea, numele claselor trebuie sa fie ca numele tabelelor.
2. Pachetul BuisinessLogic: conține clasele care formează logica aplicației. Acestea manipulează obiecte din Model. Astfel aici vom avea metode pentru crearea, modificarea si ștergerea comenzilor.
3. Pachetul Presentation: conține clase care alcătuiesc interfața, atât partea de vizualizare, cat si conexiunea intre logica si partea vizuala. Astfel, este permis unui angajat sa vizualizeze datele, sa introducă elemente noi, sa le modifice sau chiar sa le șteargă, putând plasa comenzi noi. Astfel au fost nevoie de componentele necesare pentru vizualizarea acestor operații.
4. Pachetul DataAccess: - conține clasele care accesează baza de date. Acestea furnizează operații CRUD asupra entităților. De exemplu ProductDAO care oferă metode pentru salvarea, căutarea si ștergerea produselor din baza de date.

Aceasta ar fi o abordare simplă a proiectării arhitecturii aplicației Orders Management. Desigur, design-ul detaliat va depinde de cerințele specifice ale aplicației și de tehnologiile utilizate. Alte aspecte care descriu aplicația sunt: aceasta trebuie sa ofere o performanta buna si sa răspundă la interacțiunile utilizatorului, aceasta trebuie sa fie, totodată, ușor de utilizat, aceasta ar trebui sa execute comenzile rapid.

Funcționarea aplicației e descrisa de mai multe cazuri de utilizare. In ceea ce privește aplicația de gestiune a comenzilor utilizatorul este un angajat, iar aplicația trebuie sa ii permită sa: introducă clienți noi, sa modifice datele unui client, sa îl șteargă sau chiar sa vizualizeze aceste date. Aceleași operații vor fi disponibile si pentru produs. Când vine vorba de comanda, angajatul poate selecta un produs, un client, cantitatea, plasând astfel o noua comanda. Plasarea unei comenzi noi va determina realizarea unei facturi, aceasta nu poate fi vizualizata doar de cineva care are acces la baza de date a magazinului/depozitului.

Scenariul principal:

* Angajatul selectează pentru ce vrea sa realizeze operații: client, produs sau comanda.
* In funcție de ce alege, acestea va alege o alta opțiune: de exemplu adăugarea unui nou produs.
* Aplicația va deschide componente noi in care detaliile produsului vor fi introduse.
* Acesta va introduce numele, email-ul si adresa acestuia.
* Angajatul apasă OK.
* Aplicația va adăuga produsul la baza de date.

Acest scenariu se va schimba in funcție de ce alege angajatul, fiecare opțiune având alți pași.

O imagine care conține schiță, desen, diagramă, Schiță

Descriere generată automat

O imagine care conține text, captură de ecran

Descriere generată automat

1. **Proiectare:**

Programarea orientata pe obiecte este o paradigma de programare ce are la baza conceptul obiectelor- unități de programare care conțin funcționalități asociate. Astfel, este permisa manipularea de obiecte pentru crearea mai multor sarcini in cadrul unui proiect. Acest tip de programare folosește conceptul de încapsulare al datelor, care permite ascunderea detaliilor unui obiect, fapt ce duce la protejarea datelor. Pentru a construi un program eficient, ușor de întreținut, cu o structura clasica si o organizare logica a datelor la prima tema vom folosi principii OOP.

Astfel, pentru acest proiect am folosit 4 clase care constituie modelul aplicației, tabelele sin baza de date. Aceste clase conturând ideea temei si fiind specifice tehnicii de programare alese.

Totodată, la acest proiect am folosit reflection, tehnica ce ne permite sa lucram cu clase, interfețe, atribute si metode la rulare, fără sa știm numele acestora. Este posibila instantierea de noi obiecte, apelarea de metode si obținerea sau schimbarea valoarea atributelor. Aceasta tehnica este foarte folositoare. De exemplu la acest proiect am folosit reflection la accesarea bazei de date. Astfel am făcut metode reflection care inserează, șterge, editează si interogările specifice care pot fi folosite pentru orice tabel din baza de date. Astfel, nu a fost nevoie sa implementam aceste metode de fiecare data.

Proiectul a fost împărțit in 4 pachete:

**Pachetul Model:** conține clasele model reprezentând modelele de date ale aplicației.

**Pachetul BuisinessLogic:** conține clasele care formează logica aplicației. Acestea manipulează obiecte din Model.

**Pachetul Presentation:** conține clase care alcătuiesc interfața, atât partea de vizualizare, cat si conexiunea intre logica si vizuala. Angajatul are astfel, posibilitatea sa vizualizeze datele, sa introducă elemente noi, sa le modifice sau chiar sa le șteargă, putând plasa comenzi noi.

**Pachetul DataAccess:** - conițe clasele care accesează baza de date. Acestea furnizează operații CRUD asupra entităților.

**Pachetul Connection:** - realizează conexiunea cu baza de date.

1. **Implementare:**

**Clasa Client:** aceasta clasa are 4 atribute : nume, id (unic), adresa, email, iar ca metode aceasta clasa cortine doar settere si gettere. Aceasta clasa ii corespunde tabelului din baza de date intitulat „client”.

**Clasa Product:** aceasta clasa are 4 atribute : nume, id (unic), numar produse, preț, iar ca metode aceasta clasa conține doar settere si gettere. Aceasta clasa ii corespunde tabelului din baza de date intitulat „product”.

**Clasa Order:** aceasta clasa are 5 atribute : nume, id (unic), id client, id produs, preț, iar ca metode aceasta clasa conține doar settere si gettere. Aceasta clasa ii corespunde tabelului din baza de date intitulat „order”.

**Clasa Bill:** aceasta clasa este o clasa imutabila si a fost construita folosind Java record. Aceasta are 3 atribute: id client, pret, id produs si id, iar ca metode are doar gettere. Aceasta clasa ii corespunde tabelului intitulat „bill”.

**Clasa AbstractDAO:** este clasa care conține metode principale cu ajutorul cărora realizam operațiile CRUD pe baza de date. Clasa AbstractDAO este o clasă generică în Java, care servește ca o clasă de bază pentru clasele DAO (Data Access Object) specifice pentru diverse entități într-o aplicație. Această clasă implementează metode generale pentru accesul la date într-o bază de date relațională. Clasa este definită cu parametrul generic <T>, care reprezintă tipul entității pentru care este creată clasa DAO. Clasa utilizează reflecția pentru a obține tipul entității pe care o gestionează. Acest lucru este realizat prin accesarea argumentului de tip (type) al clasei generice. Astfel, clasa poate fi utilizată pentru orice tip de entitate specificat în momentul extinderii clasei AbstractDAO. Aste, am creat metode specifice pentru interogarea cu baze de date, dar si metode care executa aceste interogări pe clase generice. De exemplu metoda findAll() realizează o interogare către baza de date pentru a obține toate înregistrările pentru entitatea specificată. Rezultatele sunt transformate în obiecte de tipul entității utilizând metoda createObjects(ResultSet resultSet). Această clasă servește ca o schelet pentru implementarea claselor DAO specifice pentru diverse entități în aplicația de gestionare a comenzilor. Metodele generice și utilizarea reflecției facilitează implementarea operațiilor CRUD (Create, Read, Update, Delete) pentru diferite entități fără a fi necesară duplicarea codului. Celelalte clase DAO care extind clasa Abstract sunt: **ProductDAO, ClientDAO, OrderDAO.** Pentru obținerea funcționalităților dorire doar in cazul OrderDAO a fost nevoie de suprascrierea metodei de insert întrucât la inserarea unei comenzi trebuia sa cream un obiect de tip Bill, si totodată, sa inserăm acest obiect in baza de date, dar si editarea numărului de produse din tabelul „product” când comanda este efectuata.

**Clasele ClientBLL, ProductBLL, OrderBLL** manipulează obiectele din pachetul model: astfel aici sunt apelate metodele din clasele DAO.

Pentru realizarea interfeței grafice am avut nevoie de o multitudine de clase pentru a reda toate funcționalitățile dorite. Astfel, putem: adaugă, șterge, modifica clienti si produse sau plasa comenzi. Toate aceste operații au avut nevoie de Label-uti si TextField-uri noi, dar si butoane.

1. **Rezultate:**

In urma rulării vom putea obține rezultatele dorite. Pentru a ajunge la îndeplinirea tuturor funcționalităților a fost nevoie sa testez fiecare caz, încercare de inserare de multe obiecte, editarea acestora, ștergerea, dar si vizualizarea acestora.

In urma rulării vom putea vedea următoarele rezultate:

O imagine care conține text, captură de ecran, Font, proiectare

Descriere generată automat O imagine care conține text, captură de ecran, Font, număr

Descriere generată automat

O imagine care conține text, captură de ecran, Font, număr

Descriere generată automat

O imagine care conține text, captură de ecran, număr, Font

Descriere generată automat

Modificările făcute in interfața, vor putea fi văzute in MySQL, unde a fost creat baza de date. Daca am inserat un nou produs acesta va putea fi văzut in tabel, iar daca l-am șters acesta va fi șters.

1. **Concluzii:**

Aceasta tema mi-a oferit oportunitatea de a exersa POO si de a învață diverse concepte si tehnici noi pentru mine precum: legătura java si baze de date relaționale, metodele Reflection si reamintirea elementelor SQL.

Astfel, prin dezvoltarea acestei aplicații am avut multe de învățat. Am înțeles importanța separării componentelor în pachete, cum ar fi Model, Business Logic, Presentation și Data Access. Am învățat importanta implementării și utilizării claselor model pentru reprezentarea datelor, astfel încât acestea să corespundă cu structura tabelelor din baza de date. Am învățat utilizarea clasei AbstractDAO pentru a realiza operațiile CRUD (Create, Read, Update, Delete) asupra entităților din baza de date. Am folosit Reflection pentru a obține informații despre entitatea pe care o gestionăm și pentru a apela metodele adecvate în funcție de operația dorită. Totodată, am învățat importanta testării și validării funcționalităților implementate, asigurându-ne că aplicația funcționează corect în diferite scenarii și că rezultatele obținute sunt conforme cu așteptările.

Această experiență ne-a ajutat să înțelegem mai bine structura și organizarea unui proiect software și să dobândim cunoștințe esențiale în domeniul dezvoltării aplicațiilor de gestionare a comenzilor și a bazelor de date. Deși la început am întâlnit probleme la înțelegerea construirii metodelor folosind reflection, la final simt ca am învățat foarte multe făcând aceasta tema.

1. **Bibliografie:**

<https://www.baeldung.com/javadoc>

<https://jenkov.com/tutorials/java-reflection/index.html>

<https://stackoverflow.com/>