**DOCUMENTATIE**

**TEMA 3**

**Order management**

**Nume: Szabo Szilard**

**Grupa: 30229**

**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:**

Tema propusă constă în implementarea unei aplicații de gestionare a comenzilor pentru procesarea solicitărilor clienților. Scopul principal al aplicației este de a permite înregistrarea și manipularea datelor referitoare la clienți, produse și comenzile asociate acestora. Pentru a realiza acest lucru, este necesară utilizarea unei baze de date, formată din tabele corespunzătoare fiecărui element în parte. Fiecare tabel conține mai multe atribute care descriu detaliile necesare pentru o comandă.

În ceea ce privește clienții și produsele, aplicația trebuie să ofere următoarele funcționalități: inserarea unui nou client/produs, ștergerea unui client/produs, editarea datelor unui client/produs și posibilitatea de a vizualiza toate elementele de același tip într-un tabel.

Pentru comenzile propriu-zise, aplicația trebuie să permită crearea unei comenzi. Un client poate comanda un produs și poate specifica cantitatea dorită. În cazul în care cantitatea specificată este prea mare, aplicația trebuie să afișeze un mesaj de eroare.

Printre obiectivele temei se numără și conectarea aplicației la baza de date, precum și implementarea metodelor folosind tehnica Reflection. Reflection permite manipularea structurii, comportamentului și atributele obiectelor în timpul execuției. Cu toate acestea, trebuie avut în vedere că utilizarea excesivă a Reflection poate afecta performanța aplicației.

De asemenea, pentru a realiza proiectul, sunt necesare cunoștințe de SQL pentru interogarea bazei de date. Un alt obiectiv important este crearea unei interfețe ușor de utilizat, care să permită o experiență fluidă utilizatorilor. În plus, aplicația trebuie să fie flexibilă și să permită dezvoltări ulterioare pentru a adăuga funcționalități noi sau pentru a îmbunătăți cele existente.

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

Cerințele funcționale definesc funcționalitățile pe care aplicația trebuie să le îndeplinească, în timp ce cerințele non-funcționale se referă la aspecte calitative ale aplicației.

Cerința problemei constă în dezvoltarea unei aplicații de gestionare a comenzilor pentru procesarea solicitarilor clienților. Pentru a implementa această aplicație, este necesară utilizarea unei baze de date relaționale pentru stocarea informațiilor despre produse, clienți și comenzile asociate.

Structura aplicației este următoarea:

1. Pachetul "model": conține clasele de model care reprezintă entitățile aplicației. Atributele acestor clase sunt similare cu coloanele din baza de date. Numele claselor trebuie să corespundă cu numele tabelelor din baza de date.
2. Pachetul "logic": conține clasele responsabile de logica aplicației. Aceste clase manipulează obiectele din pachetul "model". Aici sunt implementate metodele necesare pentru crearea, modificarea și ștergerea comenzilor.
3. Pachetul "gui": conține clasa care formează interfața utilizatorului, atât partea de vizualizare, cât și conexiunea între logica aplicației și interfața vizuală. Angajatul poate vizualiza datele, introduce noi elemente, le poate modifica sau șterge și poate plasa noi comenzi. Aici sunt necesare componentele necesare pentru afișarea și manipularea acestor operații.
4. Pachetul "dao": conține clasele responsabile de accesarea bazei de date. Aceste clase furnizează operațiile CRUD (Create, Read, Update, Delete) asupra entităților. De exemplu, clasa "ProductDAO" oferă metode pentru salvarea, căutarea și ștergerea produselor din baza de date.
5. Pachetul "start": conține tehnica de implementare Reflection.
6. Pachetul "connection": stabileste conexiunea la baza de date.

Funcționarea aplicației este descrisă prin mai multe cazuri de utilizare. Utilizatorul aplicației este un angajat, iar aplicația trebuie să permită următoarele funcționalități:

* Adăugarea unui client nou
* Modificarea datelor unui client existent.
* Ștergerea unui client din sistem.
* Vizualizarea datelor despre clienți.
* Adăugarea unui produs nou
* Modificarea datelor unui produs existent.
* Ștergerea unui produs din sistem.
* Vizualizarea datelor despre produse.
* Plasarea unei comenzi, care implică selectarea unui produs, a unui client și specificarea cantității dorite.

O imagine care conține diagramă, linie, schiță, desen

Descriere generată automat

1. **Proiectare:**

Programarea orientată pe obiecte (OOP) este o paradigmă de programare care se bazează pe conceptul de obiecte - unități de programare care înglobează funcționalități asociate. OOP permite manipularea obiectelor pentru a realiza diverse sarcini în cadrul unui proiect. Această abordare utilizează conceptul de încapsulare a datelor, care permite ascunderea detaliilor interne ale unui obiect, asigurând astfel protecția datelor. Prin urmare, construirea unui program eficient și ușor de întreținut se bazează pe structura și organizarea logică a datelor conform principiilor OOP.

În cadrul proiectului nostru, am utilizat patru clase care reprezintă modelul aplicației și corespund tabelelor din baza de date. Aceste clase definesc structura și funcționalitățile necesare pentru implementarea temei, respectând astfel tehnicile de programare OOP.

De asemenea, am utilizat tehnica Reflection în acest proiect. Reflection ne permite să lucrăm cu clase, interfețe, atribute și metode la momentul rulării programului, fără a cunoaște numele acestora în prealabil. Această tehnică ne oferă posibilitatea de a crea noi obiecte, de a apela metode și de a accesa sau modifica valorile atributelor. Reflection este extrem de utilă în diverse situații. De exemplu, în proiectul nostru, am folosit Reflection pentru a accesa baza de date. Am creat metode Reflection care realizează operații de inserare, ștergere, editare și interogare, specifice oricărui tabel din baza de date. Astfel, nu a fost necesară implementarea acestor metode pentru fiecare tabel în parte, ci am putut folosi aceleași metode Reflection pentru toate tabelele.

Prin utilizarea principiilor OOP și a tehnicii Reflection, am reușit să dezvoltăm o aplicație care respectă structura și organizarea logica a datelor, facilitând astfel implementarea temei și permițând extensibilitatea și întreținerea ușoară a programului.

Proiectul a fost împărțit in 5 pachete:

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

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

**Pachetul gui:** 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 dao:** - conține clasele care accesează baza de date. Acestea furnizează operații CRUD asupra entităților.

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

Diagrama de pachete:

O imagine care conține diagramă, Plan, Dreptunghi, linie

Descriere generată automat

1. **Implementare:**

* Clasa Client este definită cu patru atribute: id (unic), nume, adresă și email. Această clasă conține doar metode setter și getter pentru a accesa și actualiza valorile atributelor. Clasa Client corespunde tabelului "client" din baza de date.
* Clasa Product are trei atribute: id (unic), nume și cantitate. Similar clasei Client, această clasă oferă doar metode setter și getter pentru gestionarea atributelor. Clasa Product corespunde tabelului "product" din baza de date.
* Clasa Order are patru atribute: id (unic), id client, id produs și cantitate. Această clasă, la fel ca celelalte, furnizează metode setter și getter. Clasa Order corespunde tabelului "order" din baza de date.
* Clasa AbstractDAO reprezintă o clasă generică în Java, care conține metode principale pentru operațiile CRUD pe baza de date. Această clasă servește ca o bază pentru clasele DAO specifice pentru diverse entități într-o aplicație. Utilizarea reflecției permite obținerea tipului entității gestionate prin argumentul de tip al clasei generice. Această abordare facilitează implementarea operațiilor CRUD pentru diferite entități fără duplicarea codului. Clasele DAO care extind clasa AbstractDAO sunt: ProductDAO, ClientDAO și OrderDAO. În cazul clasei OrderDAO, metoda de inserare a fost suprascrisă pentru a crea și insera un obiect de tip Bill în baza de date, precum și pentru a actualiza cantitatea de produse din tabelul "product" în momentul plasării comenzii.
* Clasele ClientLogic, ProductLogic și OrderLogic manipulează obiectele din pachetul model. Aceste clase apelează metodele din clasele DAO pentru a efectua operațiuni specifice asupra datelor.

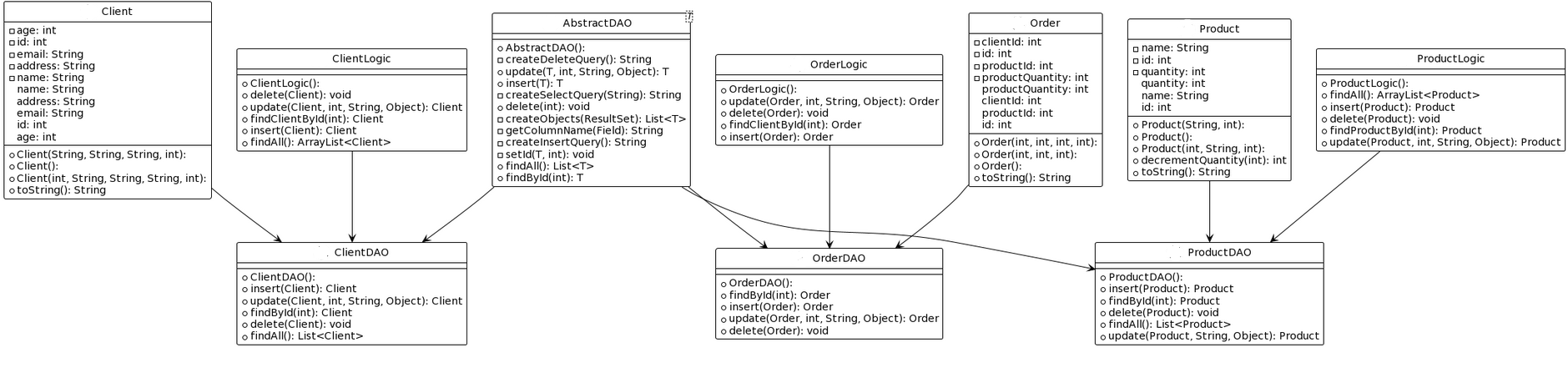
Pentru realizarea interfeței grafice, au fost utilizate multiple clase pentru a implementa funcționalitățile dorite. Interfața permite adăugarea, ștergerea și modificarea clienților și produselor, precum și plasarea comenzilor. Pentru aceste operații, s-au folosit etichete (label-uri), câmpuri de text (text fields) și butoane.

Structura modulară a aplicației este folosită pentru a separa funcționalitățile și responsabilitățile diferitelor componente, ceea ce aduce mai multă claritate, flexibilitate și ușurință în dezvoltarea și întreținerea aplicației.

Prin utilizarea structurii modulare, fiecare componentă (pachet și clasă) își are propria responsabilitate și funcționalitate bine definită. Aceasta permite dezvoltatorilor să lucreze în paralel la diferite părți ale aplicației fără a interferea unii cu ceilalți. De exemplu, un dezvoltator poate lucra la partea de interfață grafică, în timp ce altul se ocupă de logica aplicației sau accesul la baza de date.

Prin separarea logică a funcționalităților în module diferite, se obține și reutilizabilitatea. Modulele pot fi folosite în alte proiecte similare sau pot fi extinse cu ușurință pentru a adăuga noi funcționalități. De exemplu, pachetul dao poate fi utilizat în alte aplicații care necesită acces la o bază de date relațională, iar clasa AbstractDAO poate fi extinsă pentru a gestiona alte entități.

Pe lângă avantajele de dezvoltare, structura modulară facilitează și mentenanța aplicației. În cazul în care este necesară o modificare sau o remediere a unei funcționalități, dezvoltatorii pot interveni asupra componentei respective fără a afecta întregul sistem. Aceasta duce la o gestionare mai ușoară a codului și reducerea riscului de a introduce erori în alte părți ale aplicației.



1. **Rezultate:**

In urma rularii programului meu se obtine:

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

Descriere generată automat

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

Descriere generată automat

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

Descriere generată automat

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

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:**

Această temă mi-a oferit o oportunitate excelentă de a exersa programarea orientată pe obiecte (POO) și de a învăța diverse concepte și tehnici noi pentru mine, cum ar fi integrarea Java cu bazele de date relaționale, utilizarea metodelor Reflection și reamintirea aspectelor legate de limbajul SQL.

Pe parcursul dezvoltării acestei aplicații, am avut multe de învățat. Am înțeles importanța separării componentelor în pachete distincte, precum Model, Business Logic, Presentation și Data Access. Am învățat să utilizez clasele model pentru a reprezenta datele într-un mod coerent cu structura tabelelor din baza de date. De asemenea, am înțeles importanța implementării și utilizării clasei AbstractDAO pentru a efectua operațiile CRUD (Create, Read, Update, Delete) asupra entităților din baza de date. Am folosit metoda Reflection pentru a obține informații despre entitatea pe care o manipulăm și pentru a apela metodele adecvate în funcție de operația dorită. Totodată, am învățat să testăm și să validăm funcționalitățile 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 ne-a adus cunoștințe esențiale în domeniul dezvoltării aplicațiilor de gestionare a comenzilor și a bazelor de date. Deși am întâmpinat inițial dificultăți în înțelegerea construirii metodelor folosind reflection, consider că, în final, am acumulat cunoștințe valoroase prin realizarea acestei teme.

1. **Bibliografie:**

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

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

<https://stackoverflow.com/>

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

https://dev.mysql.com/doc/workbench/en/wb-admin-export-import-management.html