Documentație tema 3

Order Management

Sigoiu George

Grupa 30224

Cuprins

[1. Obiectivul temei 3](#_Toc70241117)

[a) Obiectivul principal 3](#_Toc70241118)

[b) Obiective secundare 3](#_Toc70241119)

[2. Analiza problemei, modelare, scenarii, cazuri de utilizare 4](#_Toc70241120)

[a) Cerințe funcționale 4](#_Toc70241121)

[b) Descriere cazuri de utilizare 4](#_Toc70241122)

[ Use-case Client/Product Window: 4](#_Toc70241123)

[ Use-case Order Window: 5](#_Toc70241124)

[3. Proiectare 6](#_Toc70241125)

[a) Diagrama UML 6](#_Toc70241126)

[b) Structuri de date 6](#_Toc70241127)

[c) Interfața utilizator 6](#_Toc70241128)

[4. Implementare 9](#_Toc70241129)

[a) Business Layer 9](#_Toc70241130)

[1) ClientBLL 9](#_Toc70241131)

[2) ProductBLL 9](#_Toc70241132)

[3) OrderBLL 9](#_Toc70241133)

[b) Data Access Layer 9](#_Toc70241134)

[c) Model 9](#_Toc70241135)

[d) Presentation Layer 9](#_Toc70241136)

[5. Rezultate 10](#_Toc70241137)

[6. Concluzii 11](#_Toc70241138)

[7. Bibliografie 12](#_Toc70241139)

# Obiectivul temei

## Obiectivul principal

Principalul obiectiv al proiectului se rezuma la crea o aplicație prin intermediul căreia să se proceseze comenzile clienților unui depozit, de către un administrator.

## Obiective secundare

|  |  |  |
| --- | --- | --- |
| Obiectiv secundar | Descriere | Capitol |
| Determinarea operațiilor | Descrierea operațiilor care trebuie implementate | [2](#_Cerințe_funcționale) |
| Dezvoltarea de use-case-uri și scenarii | Într-un sistem software un ‘use-case’ este o listă de acțiuni sau pași de eveniment care definesc în mod obișnuit interacțiunile dintre un rol ( cunoscut ca un actor în limbajul unificat de modelare ( UML ) ) și un sistem in atingerea unui obiectiv.  În cazul nostru, rolul este de a utiliza aplicația. | [2](#_Descriere_cazuri_de) |
| Alegerea structurilor de date | Structurile de date folosite pentru a duce la capăt obiectivul principal | [3](#_Structuri_de_date) |
| Interfața utilizator | Se descrie modul cum utilizatorul poate folosi aplicația | [3](#_Interfața_utilizator) |
| Împărțirea pe clase | Crearea mai multor clase pentru a ușura lucrul, clase pentru stocarea datelor, clase pentru efectuarea operațiilor, clase pentru interfața grafică | [4](#_Implementare) |
| Implementarea algoritmilor | Descrierea algoritmilor necesari efectuării operațiilor între polinoame | [4](#_Clasa_Operator) |
| Interfața grafică | Implementarea unei interfețe grafice care să afișeze rezultatele operațiilor cu polinoame | [4](#_Interfața_grafică_(clasa) |
| Testare | Vor fi descrise câteva scenarii de testare a operaților pe polinoame, folosind ca instrument un Junit Test. | [5](#_Rezultate) |

# Analiza problemei, modelare, scenarii, cazuri de utilizare

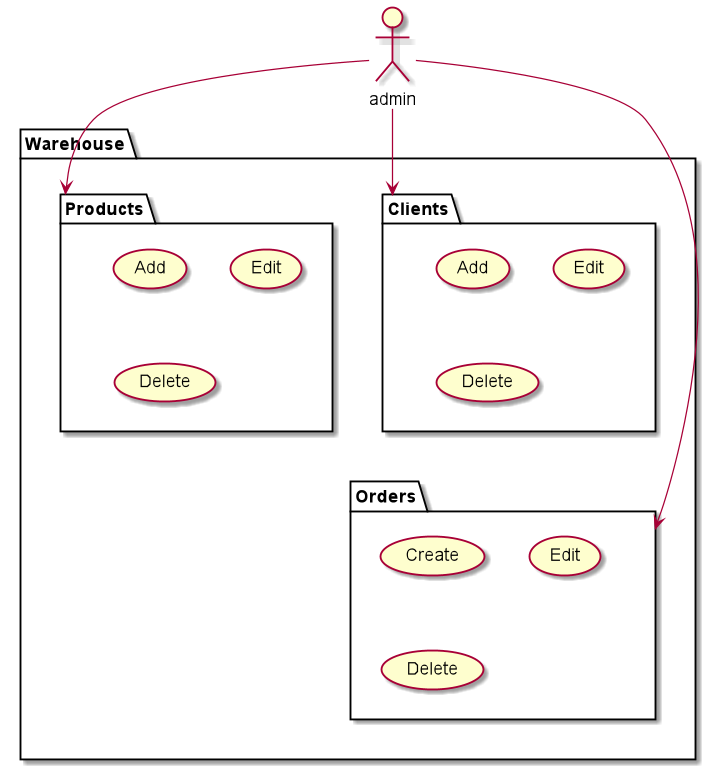
## Cerințe funcționale

Aplicația conține 4 ferestre: fereastra principală, în care se va alege una dintre variantele de a gestiona datele despre clienți, despre produse și de a gestiona comenzile clienților și alte 3 ferestre care reprezintă interfețele utilizator care permit realizarea gestiunilor anterior menționate.

Fiecare dintre ultimele 3 ferestre descrise mai sus conțin următoarele operații:

1. Adăugarea unui client / produs nou sau crearea unei comenzi noi.
2. Editarea datelor (aflate într-un tabel).
3. Ștergerea datelor din tabele.

## Descriere cazuri de utilizare



### Use-case Client/Product Window:

Administratorul, după ce a selectat una dintre cele 2 ferestre, va avea acces la operațiile de adăugare a unui client sau produs nou, de a edita valorile înregistrărilor din tabel și de a șterge clienți sau produse.

Pentru a adăuga un obiect ( client sau produs ) nou, trebuie specificate toate datele referitoare la acel obiect ( obligatoriu ), iar câmpul ID poate fi lăsat necompletat. Dacă câmpul ID nu este specificat la adăugare, obiectului respectiv i se va atribui un ID de către baza de date, altfel îi va fi atribuit ID-ul specificat. Dacă un câmp dintre cele obligatorii nu este completat, adăugarea nu va fi posibilă, iar administratorul va primi un mesaj de avertizare de tip pop-up prin care se specifică ca nu sunt completate toate câmpurile.

Pentru a edita un obiect este nevoie ca butonul „Edit” să fie apăsat pentru ca datele să poată fi editate și pentru a apărea butoanele de „Save”, „Cancel” și „Delete”. Butonul Save salvează datele actualizate în baza de date, butonul Cancel anulează modificările aduse de către administrator și afișează în tabel datele obiectelor din baza de la ultima actualizare a bazei de date. Câmpurile care pot fi modificate sunt toate înafară de câmpul ID.

Butonul Delete permite administratorul să șteargă înregistrări din baza de date. Datele pot fi șterse pe rând, selectându-se cu mouse-ul o singură înregistrare sau pot fi șterse mai multe odată dacă se selectează mai multe înregistrări folosind combinația de taste Ctrl + left click.

### Use-case Order Window:

Asemenea celorlalte 2 ferestre, fereastra Order permite administratorului să folosească aceleași operații, cu o mică diferență la adăugarea ( crearea ) unei noi comenzi. Administratorul trebuie să introducă clientul și produsul cumpărat de client, având posibilitatea de a alege un client sau un produs dintr-o listă cu clienți existenți în baza de date sau cu produse existente și trebuie să introducă cantitatea pe care clientul dorește să o cumpere. După ce aceste date au fost introduse, administratorul trebuie sa apeșe pe butonul Refresh pentru a actualiza comanda cu prețul unitar al obiectului și prețul total.

# Proiectare

## Diagrama UML

## Structuri de date

Ca și structuri de date, s-au definit clasele Client, Product, Order, care reprezintă cele 3 tabele din baza de date, clasele ClientBLL, ProductBLL și OrderBLL fac legătura între obiectele din java și înregistrările din baza de date, făcând, totodată, legătura între interfață și operațiile executate pentru ca informațiile să fie afișate în interfață, clasele ClientDAO, ProductDAO, OrderDAO și AbstractDAO fac legătura în aplicație și baza de date de unde aplicația afișează datele, iar clasele GUI, View, respectiv Controller reprezintă interfața grafică, respectiv interacțiunea cu butoanele din clasa grafică.

## Interfața utilizator

Interfața utilizator conține o fereastră principală din care se poate alege una dintre celelalte 3 ferestre predestinate gestiunii unui tabel din baza de date.

Graphical user interface

Description automatically generated

Toate cele 3 ferestre sunt construite pe baza aceleași idei de proiectare, dar există mici diferențe între ele.

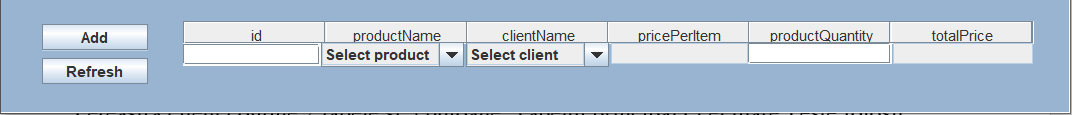
Graphical user interface

Description automatically generated

Fereastra client conține 2 tabele și 5 butoane. Tabelul principal ( cel mare ) este folosit pentru a afișa datele din baza de date, date referitoare la înregistrările clienților. Tabelul de mai jos este folosit pentru adăugarea unui nou client în baza de date, trebuie completate acele câmpuri, apoi se apasă butonul Add. Pentru a edita datele din baza de date trebuie întâi apăsat butonul Edit, pentru a activa editarea tabelului, apoi se modifică datele respective, iar pentru a salva modificările trebuie apăsat butonul Save și pentru a anula modificările, butonul Cancel. Butonul Delete va fi disponibil doar după ce butonul Edit a fost acționat, acesta permite ștergerea rândurilor selectate din tabel.

Fereastra pentru produse prezintă același design, cu singura modificare a numelor câmpurilor din baza de date.

O diferență mai mare apare însă la fereastra pentru gestiunea comenzilor, la partea de adăugare/creare a unei comenzi noi.



Administratorul bazei de date poate selecta un produs dintr-o listă de produse existente în baza de date, la fel poate selecta un client deja existent în baza de date, iar cantitatea trebuie introdusă de la tastatură. După ce cele 3 câmpuri anterior reamintite au fost selectate, trebuie apăsat butonul Refresh pentru a actualiza câmpurile pricePerItem și totalPrice.

# Implementare

## Business Layer

Layer-ul conține 3 clase care reprezintă tabelele din baza de date sub formă de obiecte în Java. Aceste clase fac legătura între informațiile afișate în interfața grafică și operațiile executate în spate. Toate metodele din aceste clase apelează metodele din ClientDAO, ProductDAO sau OrderDAO.

### ClientBLL

public Client findClientById(int id)

* Metodă care caută un client în baza de date după id-ul său și returnează clientul respectiv dacă există

public List<Client> findAll()

* Metodă care caută toți clienții din baza de date și returnează o listă cu aceștia.

public Client insert(Client c)

* Metodă care introduce un client nou în baza de date.

public Client update(Client c)

* Metodă care actualizează un client din baza de date.

public void deleteClientById(int id)

* Metodă care șterge un client din baza de date.

public Client findClientByName(String name)

* Metodă care returnează un client, căutat după nume.

### ProductBLL

public Product findProductById(int id)

* Metodă care returnează un produs cu id-ul egal cu cel dat ca parametru.

public List<Product> findAll()

* Metodă care returnează sub formă de listă toate produsele din baza de date.

public Product insert(Product p)

* Metodă care adăugă un produs nou în baza de date.

public void deleteProductById(int id)

* Metodă care șterge un produs din baza de date în funcție de id-ul său.

public Product findProductByName(String name)

* Metodă care returnează un produs al cărui nume este dat ca parametru.

public Product update(Product p)

* Metodă care actualizează un produs din baza de date.

### OrderBLL

public Order findOrderById(int id)

* Metoda care caută o comandă dupăa id și returnează comanda respectivă dacă există.

public List<Order> findAll()

* Metodă care returnează o listă de comenzi, reprezentând toate comenzile din baza de date.

public Order insert(Order o)

* Metodă care adăugă o comandă nouă în baza de date.

private void generateBill(Order o)

* Metodă care generează factura pentru o comandă nou adăugată.

public void deleteOrderById(int id)

* Metodă care șterge o comandă din baza de date.

public Order update(Order o)

* Metodă care actualizează câmpurile unei comenzi.

## Data Access Layer

Layer-ul reprezintă legătura dintre baza de date și aplicație. ClientDAO, ProductDAO și OrderDAO extind clasa AbstractDAO. Aceste clase realizează toate metodele claselor din business layer.

### AbstractDAO<T>

private String createSelectQuery(String field)

* Metodă care creează interogarea din baza de date pentru a afișa toate înregistrările dintr-un anumit tabel al bazei de date.

public List<T> findAll()

* Metodă care returnează o listă de obiecte ce reprezintă toate înregistrările dintr-un anumit tabel din baza de date.

public T findById(int id)

* Returnează un obiect care are un anumit id.

private List<T> createObjects(ResultSet resultSet)

* Metodă care creează o listă de obiecte pe baza rezultatului unei interogări din baza de date.

public T findByName(String field, String value)

* Metodă care returnează un obiect care pe un anumit câmp care o anumită valoare, câmpul și valoarea sunt date ca parametrii.

private String createInsertQuery(T t)

* Metodă care creează interogarea pentru adăugarea unei înregistrări noi.

public T insert(T t)

* Metodă care execută adăugarea unei noi înregistrări în baza de date.

private String createUpdateQuery(T t)

* Metodă care creează interogarea pentru actualizarea unui obiect din baza de date.

public T update(T t)

* Metodă care execută actualizarea obiectului.

public void deleteById(int id)

* Metodă care șterge o înregistrare din baza de date.

private String createDeleteQuery(String s)

* Metodă care creează interogarea pentru ștergerea unei înregistrări din baza de date.

### ClientDAO

public Client findByName(String name)

* Metodă care caută un client în baza de date, folosind o metodă din AbstractDAO, personalizând parametrii metodei respective.

### ProductDAO

public Product findByName(String name)

* Metodă care caută un produs în baza de date, folosind o metodă din AbstractDAO, personalizând parametrii metodei respective.

### OrderDAO

public int getOrderId(Order o)

* Metodă care caută în baza de date comanda cu anumite valori și returnează id-ul comenzii.

## Model

Acest package conține cele 3 tipuri de obiecte existente în baza de date și ajută la stocarea informațiilor din baza de date în structuri ( clase ) care conțin atribute identice cu câmpurile din baza de date. Clasele sunt următoarele:

1. Client
2. Product
3. Order

Aceste clase conțin ca metode doar getter-e și setter-e.

## Presentation Layer

Clasele care alcătuiesc acest layer reprezintă interfața grafică.

Clasa GUI reprezintă meniul principal de unde administratorul alege care tabel din baza de date vrea să-l gestioneze. Acesta poate alege între clienți, produse și comenzi. Această interfață conține 3 butoane cu funcția descrisă mai sus (de a alege un tabel pentru gestionare).

Clasa View reprezintă interfața grafică prin intermediul căreia administratorul modifică, adaugă sau șterge datele din baza de date. Pentru fiecare dintre cele 3 tabele interfața este aceeași, diferă doar numele câmpurilor din tabele și metoda de adăugare a comenzilor în baza de date, explicată la [secțiunea anterioară](#_Interfața_utilizator). Această clasă conține doar 5 butoane, 2 scroll pane-uri, unul pentru fiecare tabel și câte un tabel în fiecare scroll pane.

Clasa Controller este mecanismul de funcționare al fiecărui buton din interfața grafică. Clasa conține alte clase membru care implementează Action Listener și Mouse Listener pentru a descrie funcționalitatea fiecărui buton.

# Rezultate

Funcționarea aplicației a fost testată în totalitate, fiecare buton al fiecărei interfețe a fost testat. S-a testat inițial pe baza de date goală, fiind neintroduse date în aceasta, funcționarea operațiilor de adăugare în baza de date și adăugarea în baza de date cu date deja existente în ea. Modificarea a fost testată pe fiecare câmp al bazei de date, mai puțin pe câmpul ID, acest câmp nu trebuie modificat pentru funcționarea corectă a aplicației. De asemenea, ștergerea a fost testată, prima data pe cate un obiect, apoi și pentru ștergerea mai multor înregistrări odată.

# Concluzii

Aplicația funcționează corect și este foarte ușor de folosit pentru gestiunea unui depozit. Ca o dezvoltare anterioară se poate realiza opțiunea ca un client să cumpere mai multe produse, în prezent poate cumpăra doar un produs. O altă dezvoltare ar putea fi ca fiecare client să-și creeze un cont propriu și să-și comande produsele dorite.

Realizând această temă am reușit să învăț să lucrez cu bazele de date folosind Java, am învățat că lucrul cu „reflection techniques” face ca codul scris să fie mai puțin dacă se scrie o dată cod generalizat pentru mai multe obiecte decât să se scrie cod aproape identic pentru mai multe obiecte și de asemenea am învățat să lucrez cu „layered architectures”.

# Bibliografie

• Connect to MySql from a Java application

o <https://www.baeldung.com/java-jdbc>

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

• Layered architectures

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

• Reflection in Java

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