Text

Description automatically generated

DOCUMENTAȚIE

TEMA 3-Orders Management Application

Buruian Cătălina

Grupa 30221

# 

**CUPRINS**

1. [Obiectivul temei 2](#_Toc97418559)

1. [Analiza problemei, modelare, scenarii, cazuri de utilizare 2](#_Toc97418560)
2. [Proiectare 2](#_Toc97418561)
3. [Implementare 2](#_Toc97418562)
4. [Rezultate 2](#_Toc97418563)
5. [Concluzii 2](#_Toc97418564)
6. [Bibliografie 2](#_Toc97418565)

**Capitolul 1:Obiectivul temei**

Obiectivul principal

Obiectivul principal al acestei teme a constat in implementarea unei aplicatii capabile sa gestioneze comenzile clientilor din cadrul unui depozit.

Obiective secundare

1. Analiza problemei si identificarea cerintelor (Capitolul 2)
2. Proiectarea aplicatiei (Capitolul 3)
3. Implementarea aplicatiei (Capitolul 4)
4. Testarea (Capitolul 5)

**Capitolul 2:Analiza problemei, modelare, scenarii, cazuri de utilizare**

Cerinte functionale:

1. Aplicatia ar trebui sa permita utilizatorilor sa adauge, sa editeze, sa stearga si sa vizualizeze clientii si produsele din depozit.
2. Aplicatia ar trebui sa permita utilizatorilor sa selecteze un client din baza de date, respectiv un produs si sa introduca o cantitate ca mai apoi sa creeze o comanda.
3. Aplicatia va afisa un mesaj pentru a atentiona utilizatorul in cazul introducerii unei cantitati mai mari decat cea prezenta in stoc.
4. Aplicatia va genera o chitanta pentru produsul cu cantitatea dorita si clientul selectat.

Use case 1: Adaugarea unui client

Actorul principal: Utilizatorul

Principalul scenariu de succes:

1. Utilizatorul introduce valori pentru id-ul clientului, numele, adresa si emailul acestuia.
2. Utilizatorul apasa pe butonul de „Add Client” pentru a adauga un client
3. Aplicatia valideaza datele introduse de utilizator si adauga clientul in baza de date daca datele au fost introduse corect.

Scenariu alternativ: Valori invalide pentru datele clientului

* Utilizatorul introduce valori invalide pentru emailul clientului
* Aplicatia afiseaza un mesaj de eroare si cere utilizatorului sa introduca valori valide
* Scenariul se intoarce la pasul 1.

Use case 2: Modificarea unui client

Actorul principal: Utilizatorul

Principalul scenariu de succes:

1. Utilizatorul selecteaza un client existent din tabel, iar datele sale vor fi afisate in textFields.
2. Utilizatorul modifica informatiile afisate in textFields si apasa pe butonul de „Update Client” pentru a modifica datele clientului

Use case 3: Stergerea unui client

Actorul principal: Utilizatorul

Principalul scenariu de succes:

1. Utilizatorul selecteaza un client existent din tabel, iar datele sale vor fi afisate in textFields.
2. Utilizatorul apasa pe butonul de „Delete Client” pentru a sterge clientul.

Diagram

Description automatically generated

Use case 4: Adaugarea unui produs

Actorul principal: Utilizatorul

Principalul scenariu de succes:

1. Utilizatorul introduce valori pentru id-ul produsului, numele, pretul si cantitatea acestuia.
2. Utilizatorul apasa pe butonul de „Add Product” pentru a adauga un produs
3. Aplicatia valideaza datele introduse de utilizator si adauga produsul in baza de date daca datele au fost introduse corect.

Use case 5: Modificarea unui produs

Actorul principal: Utilizatorul

Principalul scenariu de succes:

1. Utilizatorul selecteaza un produs existent din tabel, iar datele sale vor fi afisate in textFields.
2. Utilizatorul modifica informatiile afisate in textFields si apasa pe butonul de „Update Product” pentru a modifica datele produsului.

Use case 6: Stergerea unui produs

Actorul principal: Utilizatorul

Principalul scenariu de succes:

1. Utilizatorul selecteaza un produs existent din tabel, iar datele sale vor fi afisate in textFields.
2. Utilizatorul apasa pe butonul de „Delete Product” pentru a sterge produsul.

Diagram

Description automatically generated

Use case 6: Crearea unei comenzi

Actorul principal: Utilizatorul

Principalul scenariu de succes:

1. Utilizatorul selecteaza un produs existent din tabel, respectiv un client care sa cumpere o anumita cantitate din acel produs.
2. Utilizatorul adauga cantitatea dorita in textField.
3. Utilizatorul apasa pe butonul „Create order” pentru a crea comanda.
4. Utilizatorul apasa pe butonul „Generate Bill” pentru a genera o chitanta.

Scenariu alternativ: Valori invalide pentru cantitate

* Utilizatorul introduce valori invalide pentru cantitate
* Aplicatia afiseaza un mesaj de eroare si cere utilizatorului sa introduca valori valide
* Scenariul se intoarce la pasul 2.

Diagram

Description automatically generated

**Capitolul 3: Proiectarea**

**Arhitectura conceptuala**

Modelul conceptual al arhitecturii se bazeaza pe impartirea in pachete care despart partea de Presentation, Business, Data Acces si Model.

Presentation Layer – contine clase care definesc partea de interfata si controller

Business Layer- contine clase care incapsuleaza logica aplicatiei

Data Acces Layer- contine clase care contribuie la conectarea cu baza de date si interogarile

Model- contine clase mapate la tabelul din baza de date

**Diagram

Description automatically generated**

**Modelul architectural utilizat: MVC**

Model-view-controller (MVC) este un model arhitectural utilizat în ingineria software. Succesul modelului se datorează izolării logicii de business față de considerentele interfeței cu utilizatorul, rezultând o aplicație unde aspectul vizual sau/și nivelele inferioare ale regulilor de business sunt mai ușor de modificat, fără a afecta alte nivele.

**Model**

Această parte a controlatorului manipulează operațiunile logice și de utilizare de informație (trimisă dinainte de către rangul său superior) pentru a rezulta de o formă ușor de înțeles.

Modelul va fi impartit in 3 modele: Client, Product, Order.

**View**

Acestui membru al familiei îi corespunde reprezentarea grafică, sau mai bine zis, exprimarea ultimei forme a datelor: interfața grafică ce interacționează cu utilizatorul final. Rolul său este de a evidenția informația obținută până ce ea ajunge la controlor.

View-ul va fi impartit in 4: View (pentru pagina principala), ClientView(pentru client), ProductView(pentru produs), OrderView(pentru order), ShowOrders(pentru comenzi).

**Controller**

Cu acest element putem controla accesul la aplicația noastră. Pot fi fișiere, scripturi (eng. scripts) sau programe, in general orice tip de informație permisă de interfață. În acest fel putem diversifica conținutul nostru de o formă dinamică și statică, în același timp.

Controller-ul va fi impartit in 4: Controller-ul principal (controleaza celelalte controllere), ControllerClient(pentru client), ControllerProduct(pentru produs), ControllerOrder(pentru order).

**MVC Pattern**

Diagram

Description automatically generatedDiagram

Description automatically generated

**Diagrama UML de clase**

Unified Modeling Language sau UML pe scurt este un limbaj standard pentru descrierea de modele si specificatii pentru software. UML a fost la bază dezvoltat pentru reprezentarea complexității programelor orientate pe obiect, al căror fundament este structurarea programelor pe clase, și instanțele acestora ( numite și obiecte ). Cu toate acestea, datorită eficienței și clarității în reprezentarea unor elemente abstracte, UML este utilizat dincolo de domeniul IT. Așa se face că există aplicații ale UML-ului pentru management de proiecte, pentru business Process Design etc.

Graphical user interface, application

Description automatically generated

Diagram

Description automatically generated

**Structuri de date folosite**

**Liste**

O lista reprezinta o colectie ordonata de elemente. Intr-o lista sunt permise elementele duplicate. Permite un control mult mai precis asupra pozitiei unui element prin intermediul indexului acelui element.

Interfata List descrie liste de elemente indexate si are disponibile implementari precum cea utilizata in cadrul acestei teme, si anume ArrayList.

ArrayList: permite acces aleator la elementele listei; obtinearea oricarui element din lista se va face in timp constant; in schimb, adaugarea sau stergerea unui element in interiorul listei necesita deplasarea elementelor care se gasesc dupa pozitia la care se executa operatia.

**Capitolul 4: Implementarea**

**Descrierea claselor si metodelor din cadrul claselor**

Implementarea s-a bazat pe crearea unei conexiuni cu baza de date si executarea de statements pentru a efectua diferite operatii CRUD. Pentru realizarea conexiunii s-a folosit clasa ConnectionFactory din cadrul pachetului connection. Această clasă conține numele driverului (inițializat prin reflecție), locația bazei de date (DBURL) și utilizatorul și parola pentru accesarea serverului MySQL. Conexiunea la DB va fi plasată într-un obiect Singleton\*. Clasa conține metode de creare a unei conexiuni, obținerea unei conexiuni active și închiderea unei conexiuni, unui statement sau a unui resultSet.

Maparea in tabele se realizeaza cu ajutorul unor clase care au aceleasi field-uri ca si in baza de date. Astfel, clasele Client, Order, Product contin date esentiale care vor fi regasite in tabele. De asemenea, contin settere, gettere si constructori.

Clasa EmailValidator valideaza emailul clientului prin metoda validate(Client client) care verifica daca emailul respecta un anumit pattern.

Clasele ClientBLL, OrderBLL, ProductBLL reprezinta logica de business, in cadrul lor se apeleaza metodele de add, insert, update, findById, findAll din cadrul claselor ClientDAO, OrderDAO, ProductDAO.

Prin intermediul clasei ReflexionExample se evidentiaza tehnica reflexiei. Reflexia este o caracteristică a limbajului de programare Java. Permite unui program Java care se execută să examineze sau să „introspecteze” asupra lui însuși și să manipuleze proprietățile interne ale programului. De exemplu, este posibil ca o clasă Java să obțină numele tuturor membrilor săi și să le afișeze, fapt observat in clasa ReflexionExample.

Clasele ClientDAO, OrderDAO, ProductDAO extind clasa AbstractDAO <T > unde se definesc toate operatiile realizabile pe tabele si mostenesc metodele necesare pentru fiecare. Clasa AbstractDAO<T> defineste operatiile add, insert, update, findById, findAll pentru tipul generic <T> care poate fi Client, Product sau Order. De asemenea, clasa contine metode care creeaza statementuri care vor fi executate pe tabelele din baza de date, in functie de operatiile apelate si de tabela referinta.

**Descrierea GUI**

**View**

Graphical user interface, application

Description automatically generated

View-ul principal contine butoane care acceseaza paginile pentru clienti, comenzi sau produse.

**ClientView**

Table

Description automatically generated

ClientView contine textFileds pentru introducerea datelor unui client precum id, nume, email, adresa. Butoanele pentru potentialele operatii pe care poate sa le execute utilizatorul pe baza de date : Add Client, Update Client, Delete Client. De asemenea, in cadrul paginii este afisat tabelul din baza de date, iar cu butonul refresh table putem vedea tabelul actualizat in urma efectuarii unei operatii.

In partea de jos exista un buton Back care permite revenirea la pagina principala.

**ProductView**

Table

Description automatically generated

ProductView contine textFileds pentru introducerea datelor unui produs precum id, nume, pret, cantitate. Butoanele pentru potentialele operatii pe care poate sa le execute utilizatorul pe baza de date : Add Product, Update Product, Delete Product. De asemenea, in cadrul paginii este afisat tabelul din baza de date, iar cu butonul refresh table putem vedea tabelul actualizat in urma efectuarii unei operatii.

In partea de jos exista un buton Back care permite revenirea la pagina principala.

**OrderView**

Graphical user interface, table

Description automatically generatedOrderView contine cele doua tabele de clienti, respectiv produse existente. Utilizatorul poate selecta un client din primul tabel, respectiv un produs din al doilea tabel si poate introduce in textField-ul din stanga sus cantitatea dorita in care se comanda un produs. Prin apasarea butonului create Order se va crea o comanda, iar prin apasarea butonului generate bill se va genera o chitanta intr-un .txt file unde se pot vedea date despre client, produs si pretul total. Prin apasarea butonului Show orders va aparea o fereastra cu un istoric al tuturor comenzilor generate. In partea de jos exista un buton Back care permite revenirea la pagina principala.

**ShowOrders**

Table

Description automatically generated

ShowOrders contine un tabel cu un istoric al tuturor comenzilor efectuate pana in prezent. Este salvat id-ul clientului, id-ul produsului, cantitatea si este generat automat un id pentru comanda.

**Capitolul 5: Rezultate**

Adaugarea unui client

Introducem datele pentru un nou client si apasam pe butonul add.

Table

Description automatically generated

Vizualizam tabela actualizata si observam ca clientul a fost introdus cu succes.

Table

Description automatically generated

Stergerea unui client

Selectam un client din tabel si apasam pe butonul delete.

Table

Description automatically generated

Vizualizam tabela actualizata si observam ca clientul a fost sters cu succes.

Table

Description automatically generated

Update Client

Selectam un client din tabel, ii modificam datele in cadrul textFieldurilor si apasam pe butonul update.

Table

Description automatically generated

Vizualizam tabela actualizata si observam ca clientul a fost modificat cu succes.

Table

Description automatically generated

In cazul paginii cu produsele, pasii se realizeaza exact ca in cazul clientilor, astfel efectuandu-se adaugarea, modificarea sau stergerea unui produs.

Crearea unei comenzi

Se selecteaza un client din primul tabel, un produs din al doilea tabel, se completeaza textFieldul cu cantitatea dorita care nu trebuie sa depaseasca cantitatea existenta in stoc si se apasa pe butonul Create Order.

Graphical user interface, table

Description automatically generated

Dupa apasarea butonului de Create Order va fi creata o noua linie in tabelul cu comenzi. Toate comenzile pot fi afisate apasand butonul Show orders.

Table

Description automatically generated

Observam ca comanda a fost adaugata cu succes, astfel incat se decrementeaza cantitatea produsului pentru care a fost facuta comanda.

Table

Description automatically generated

In cazul introducerii unei valori pentru cantitate mai mari decat cantitatea existenta in stoc se va afisa o caseta dialog cu un mesaj de avertizare pentru utilizator.

Graphical user interface

Description automatically generated

Apasand butonul Generate Bill se va genera o chitanta pentru client cu produsul cumparat, cantitatea, adresa de livrare si pretul pe care trebuie sa il plateasca in total.

Graphical user interface

Description automatically generated

Chitanta va fi scrisa intr-un .txt file.

Graphical user interface, text, application, email

Description automatically generated

Table

Description automatically generated

**Capitolul 6: Concluzii**

In concluzie, aceasta tema m-a ajutat sa aprofundez anumite aspect legate de OOP, cat sa-mi si reamintesc notiunile invatate semestrul trecut. A fost o buna oportunitate pentru mine sa ma familizarizez cu limbajul de programare Java, in special cu tehnici Java precum reflexia, folosirea tipului generic, conexiunea cu baza de date si crearea de java doc.

Dezvoltari ulterioare: Functionalitatea butonului de refresh

**Capitolul 7: Bibliografie**

<https://www.jetbrains.com/help/idea/working-with-code-documentation.html#generate-javadoc>

<https://www.oracle.com/technical-resources/articles/java/javareflection.html>

<https://docs.oracle.com/javase/tutorial/java/generics/types.html>

<https://dzone.com/articles/layered-architecture-is-good>

<https://www.javatpoint.com/example-to-connect-to-the-mysql-database>