

Documentatie

Tema 3

~ Gestionare comenzilor~

Nume: Marinescu Roxana-Maria

Grupa: 30227

**Obiectivul temei:**

Cerința temei este de a crea o aplicație care gestioneaza comenzi intr-un magazin . Aplicația lucrează cu o bază de date unde utilizatorul poate sa introducă, să modifice sau să stearga clienți, produse si comenzi tinand cont de stocul de produse disponibil .

Tema trei de laborator are ca și obiectiv principal implementarea unui mod mai practic prin care user -ul poate sa lucreze cu tabele dintr-o baza de date. Aplicația trebuie sa respecte paradigmele de programare orientată pe obiect folosind ca limbaj de programare limbajul Java .

Obiectivele secundare ale temei sunt:

* Intelegerea corectă a cerinței și crearea unor scenarii de utilizare.
* Proiectarea unei soluții care să respecte toate cazurile introduse de user.
* Implementarea soluției și a interfeței grafice.
* Rezultatele finale și concluziile.

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

Analiza problemei a constat prin înțelegerea cerinței și a informațiilor pe care dorim să le primim la generarea unui rezultat. In primul rand, este necesară crearea unei baze de date și a unei conexiuni cu aceasta prin clasa ConnectionFactory. In al doilea rand, proiectul trebuie implementat folosind Layered Architecture , validatori, și o clasă abstractă (Data Access Object) care să conțina metodele necesare.

Aplicația lucreaza cu trei tabele: Client, Product și Order (fiecare avand diferite atribute cat si cate o cheie primară). Tabelul Client trebuie sa conțina date specific clienților ,cum ar fi: id, nume, prenume, email, la care am adaugat un validator pentru email (cu scopul de a verifica daca email ul introdus de user este unul valid). Tabelul Product trebuie sa conțina date specific produselor din shop, cum ar fi: id, nume, preț, cantitate, la care am adaugat validatori pentru pret (in cazul in care pretul e negativ se arunca o exceptie) si pentru cantitate (in cazul in care cantitatea e negativa se arunca o exceptie). Tabelul Order face legatura intre cele doua tabele de mai devreme (Client si Product) si continue datele specifice comenzilor, cum ar fi: id, idClient, idProduct , cantitate, la care am adaugat validatori pentru cantitatea comenzii la inserare si cantitatea comenzii dupa update (se va arunca o exceptie in cazul in care cantitatea de produs ceruta e mai mare decat cea de pe stoc).

*Pasii executiei fiecarui use-case:*

- User -ul poate sa introduca date in cele 3 tabele;

- User -ul alege tabelul in care vrea sa introduca datele, folosind interfata grafica;

- După ce au fost introduse datele, user -ul poate sa aleaga dintre cele 3 operatii de modificare a tabelelor si anume insert, delete si update si poate si sa vizualizeze informatiile din tabelul respectiv apasand pe select.

*Scenariu de utilizare:*

* Sa presupunem ca user -ul alege sa modifice tabelul Client:

Ex: id: 9

firstName: Ion

lastName: Ionescu

email: ion@gmail.com

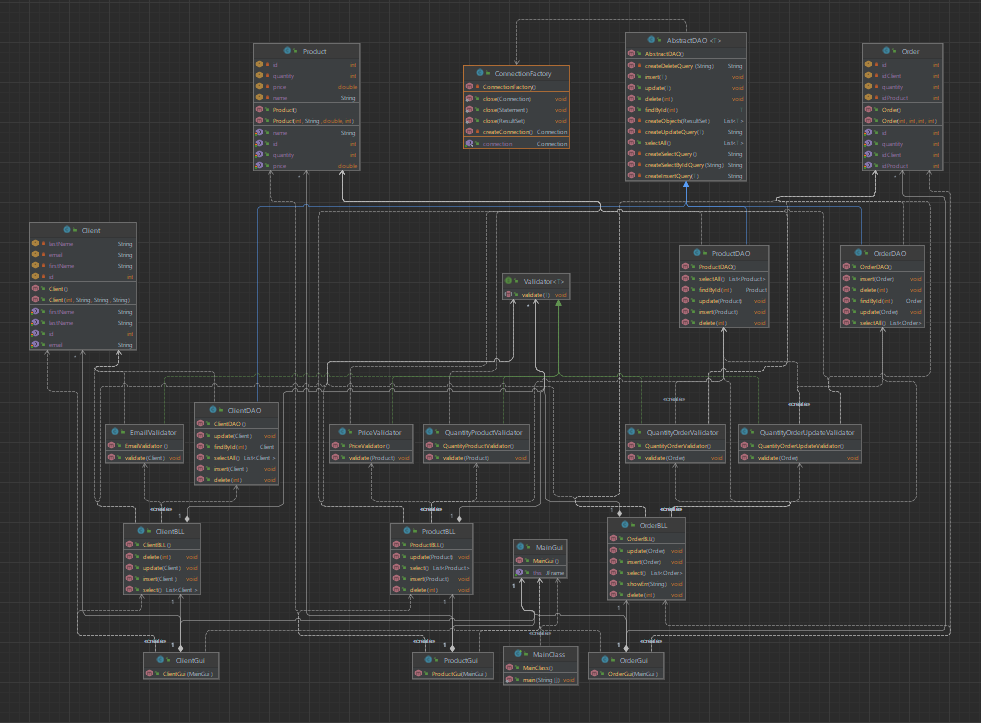
* User-ul alege sa introduca datele adaugate in tabelul Client, deci apasa pe butonul pentru Insert
* Pentru a vedea tabelul modificat, uder -ul o sa apese pe butonul Select care va deschide un frame cu tabelul si datele din acesta.

**Proiectare:**

Aplicatia a fost realizata in pachete, cum ar fi: presentation - contine clasele care definesc interfata grafica (MainGui, ClientGui, ProductGui, OrderGui) , bll (business logic layer) - contine clasele care incapsuleaza logica aplicatiei, adica : validatorii si clasele ClientBLL, ProductBLL si OrderBLL, connection - contine clasa ConnectionFactory care face conexiunea dintre baza de date si java, dao (Data Access Object ) - contine clasele care genereaza interogarile in baza de date, adica: AbstractDao, ClientDao, OrderDao si ProductDao , model - contine clasele care au ca si corespondent cate o tabela din baza de date (Client, Order sau Product), iar pachetul start – contine o singura clasa, si anume clasa MainClass care ruleaza programul .

Cat despre proiectarea în limbaj OOP, a fost urmarita crearea unei diagrame de pachete urmărind modelul Layered Architecture folosind Business Layer, Presentation Layer, Data Access Layer.

Diagrama UML a proiectului – aici se pot vedea mai amanuntit clasele, legaturile dintre clase, metodele, variabilele, cat si structura proiectului :



**Implementare:**

**Pachetul bll**

Pachetul bll (Business Logic Layer) contine atat validatori cat si 3 clase (Client, Product si Order) in care sunt aplicate metodele din DAO (Data Access Object) cu validatori.

**ClientBLL**

Clasa ClientBLL apeleaza metodele de insert, delete si update din ClientDAO si le adauga o verificare suplimentara raportata la validatori. Pe langa operatiile principale am mai creat o metoda select care, folosind tehnica de reflexie ia datele din baza de date si le pune intr-un tabel pentru ca acesta sa poata fi afisat in interfata . Aici am si o lista de validatori in care am bagat un obiect validator pentru email . Pentru metodele de insert si update se parcurge lista si se verifica validitatea mail -ului prin metoda validate care se afla in fiecare clasa de validator. De asemenea, in aceste metode se prind exceptiile aruncate de metoda validate.

**OrderBLL**

Clasa OrderBLL apeleaza metodele de insert, delete si update din OrderDAO si le adauga o verificare suplimentara raportata la validatori. Pe langa operatiile principale am mai creat o metoda select care, folosind tehnica de reflexie ia datele din baza de date si le pune intr-un tabel pentru ca acesta sa poata fi afisat in interfata . Aici am si o lista de validatori in care am bagat un obiect validator pentru cantitatea din stoc de dupa update respectiv cea de dupa insert . Pentru metodele de insert si update se parcurge lista si se verifica validitatea, prin metoda validate care se afla in fiecare clasa de validator. De asemenea, in aceste metode se prind exceptiile aruncate de metoda validate si se afiseaza intr-un MessageDialog.

**ProductBLL**

Clasa ProductBLL apeleaza metodele de insert, delete si update din ProductDAO si le adauga o verificare suplimentara raportata la validatori. Pe langa operatiile principale am mai creat o metoda select care, folosind tehnica de reflexie ia datele din baza de date si le pune intr-un tabel pentru ca acesta sa poata fi afisat in interfata . Aici am si o lista de validatori in care am bagat un obiect validator pret si cantitate . Pentru metodele de insert si update se parcurge lista si se verifica validitatea , prin metoda validate care se afla in fiecare clasa de validator. De asemenea, in aceste metode se prind exceptiile aruncate de metoda validate.

**Validators:**

Avem o interfata numita validator ce este implementata de clasele EmaiValidator, PriceValidator, QuantityOrderUpdateValidator, QuantityOrderValidator , QuantityProductValidator care valideaza daca informatia introdusa de utilizator in textfield-uri indeplineste formatul necesar.

**EmailValidator**

In clasa EmailValidator se verifica daca email-ul introdus este unul care respecta formatul standard al unei adrese de mail folosind un regex. In caz contrar de va arunca o exceptie care e prinsa in ClientBLL.

**PriceValidator**

In clasa PriceValidator se verifica daca pretul produsului introdus in textfield e negativ. In cazul asta se va arunca o exceptie care e prinsa in ProductBLL.

**QuantityOrderUpdateValidator**

In clasa QuantityOrderUpdateValidator se verifica daca cantitatea ramasa in stoc inainte de insert e mai mica decat cantitatea comenzii updatate. In caz contrar se va arunca o exceptie care e prinsa in OrderBLL si se afiseaza un mesaj.

**QuantityOrderValidator**

In clasa QuantityOrderValidator se verifica daca cantitatea ramasa in stoc dupa comanda e negativa. In cazul asta se va arunca o exceptie care e prinsa in OrderBLL si se afiseaza un mesaj.

**QuantityProductValidator**

In clasa QuantityProductValidator se verifica daca cantitatea produsului introdus in textfield e negativ. In cazul asta se va arunca o exceptie care e prinsa in ProductBLL.

**Pachetul dao**

Contine o clasa abstracta DAO care este alcatuita din metodele necesare pentru a face operatii pe tabele. ClientDAO, ProductDAO si OrderDAO implementeaza aceasta clasa si folosesc aceste metode.

**AbstractDAO**

Clasa AbstractDAO implementeaza metode generale pentru clasele din model si corespondentele acestora prin intermediul tehnicii de reflexie. Se folosește tehnica de reflexie în Java pentru a nu fi nevoie să scriem mai multe metode similare, particularizate pentru fiecare clasă ce modelează obiectele aplicației. În acest sens, a fost definit un tip generic asociat clasei AbstractDAO. Fiecare operatie pe tabel este alcatuita din doua metode, una care selecteaza folosind SQL datele din tabel si una care gaseste in ce tabel trebuie introduse, sterse sau updatate. În acest pachet se folosește tehnica reflexiei, așa încât introducerea, ștergerea și updatarea în baza de date se fac folosind aceleași metode, indiferent de tipul obiectului instanță al unei clase din pachetul model. Operațiile care sunt specifice fiecărei clase din model sunt implementate prin metode în afara clasei abstracte AbstractDAO.

**ClientDAO**

Clasa ClientDAO extinde clasa AbstractDAO si foloseste metodele selectAll, findById, update, insert si delete din AbstractDAO.

**ProductDAO**

Clasa ProductDAO extinde clasa AbstractDAO si foloseste metodele selectAll, findById, update, insert si delete din AbstractDAO.

**OrderDAO**

Clasa OrderDAO extinde clasa AbstractDAO si foloseste metodele selectAll, findById, update, insert si delete din AbstractDAO.

**Pachetul model**

In pachetul model sunt cele 3 clase Client, Product si Order cu atributele lor care corespund cu tabelele facute in baza de date.

**Client**

Clasa Client are ca atribute id (care reprezinta id -ul clientului), firstName, lastName si email. De asemenea, s-au generat getter-ele si setter-ele necesare.

**Product**

Clasa Product are ca atribute id (care reprezinta id -ul produsului) , name(numele produsului) , price (pretul produsului) si quantity (care reprezinta cantitatea disponibila in shop a produsului). De asemenea, s-au generat getter-ele si setter-ele necesare.

**Order**

Clasa Order are ca atribute id (care reprezinta id -ul comenzii) , idClient (care reprezinta id -ul clientului care comanda), idProduct (care reprezinta id -ul produsului comandat) si quantity (care reprezinta cantitatea de produs solicitata de client). De asemenea, s-au generat getter-ele si setter-ele necesare.

**Pachetul presentation**

Pachetul presentation contine partea de interfata grafica si actionListeneri, cu clasele: MainGui (frame principal) si ClientGui, ProductGui, OrderGui (frame-uri secundare).

**MainGui**

MainGui e frame-ul prinicpal al proiectului si contine 3 butoane, care deschid frame-uri separate, pentru prelucrarea tabelelor din baza de date (client, order, product).

**ClientGui**

In ClientGui se pot insera, sterge si updata informatii despre un client, cat si vizualiza ce avem in tabelul client al bazei de date apasand butonul select (care deschide un frame care contine tabelul).

**ProductGui**

In ProductGui se pot insera, sterge si updata informatii despre un produs, cat si vizualiza ce avem in tabelul product al bazei de date apasand butonul select .

**OrderGui**

In OrderGui se pot insera, sterge si updata informatii despre o comanda, cat si vizualiza ce avem in tabelul order al bazei de date apasand butonul select.

**Pachetul connection**

**ConnectionFactory**

Clasa ConnectionFactory realizează conexiunea la baza de date folosind o singură instanță de obiect Connection, prin implementarea modelului arhitectural Singleton Pattern. Prin clasa de connection se realizeaza conectarea cu baza de datele si toate informatiile deja introduse.

***Clasa MainClass :***

In clasa MainClass am creat si instantiat un obiect de tip gui , care reprezinta interfata grafica principala in care user -ul urmeaza sa aleaga ce tabel doreste a vizualiza.

**Rezultate:**

După rezolvarea erorilor din cod și ajungerea la forma finala a proiectului, am testat introducand si modificand date despre clienti, produse si comenzi . In fiecare tabela se pot verifica operatiile de inserare, actualizare si stergere, atat pentru clienti cat si pentru produse si comenzi. Pentru fiecare comanda se poate observa faptul ca in cazul in care se insereaza sau se face update la o comanda, iar cantitatea de produse introdusa nu e mai mica decat cantitatea produselor din stoc atunci apare un mesaj de eroare care anunta ca nu exista atatea produse in stoc. Un lucru pe care l-as adauga proiectului ar fi ca in loc sa trebuiasca sa apas pe butonul de Select pentru a deschide un alt frame care afisaza tabelul, sa fac tabelul in acelasi frame si sa isi dea refresh in momentul in care apas pe insert, delete sau update. Un alt lucru pe care l-as modifica ar fi aspectul total al interfetei grafice, l-as face ceva mai complex si mai atragator.

**Concluzii:**

Proiectul ar trebui să funcționeze pentru orice caz valid pe care user -ul îl introduce.

Prin aceasta tema am dobandit cunostinte cu privire la utilizarea modelelor arhitecturale Layers, folosirea tehnicii de reflexie in Java, precum si crearea javadoc-ului pentru documentarea claselor . De asemenea, am reusit sa inteleg cum sunt accesate datele dintr- o baza de date pentru a putea sa lucram cu ele direct dintr-o interfata in java . Pe langa asta, am invatat sa utilizez validatori, foarte utili in cazurile in care user -ul introduce in textfield -uri date care nu corespund.

**Bibliografie:**

*UML Diagram:* [*https://youtu.be/v8G7FWdiwqs*](https://youtu.be/v8G7FWdiwqs)

MySQL: <https://www.jetbrains.com/help/idea/connecting-to-a-database.html>

Presentation: <https://dsrl.eu/courses/pt/materials/A3_Support_Presentation.pdf>