DOCUMENTATIE

TEMA *3*

NUME STUDENT:Grumazescu Silviu

GRUPA: 30221

# CUPRINS

[1. Obiectivul temei 3](#_Toc95297885)

[2. Analiza problemei, modelare, scenarii, cazuri de utilizare 3](#_Toc95297886)

[3. Proiectare 3](#_Toc95297887)

[4. Implementare 3](#_Toc95297888)

[5. Rezultate 3](#_Toc95297889)

[6. Concluzii 3](#_Toc95297890)

[7. Bibliografie 3](#_Toc95297891)

# Obiectivul temei

Obiectivul acestei teme este dezvoltarea unei aplicatii capabile de a procesa comenzile unor clienti pentru un depozit de produse, folosindu-ne de accestul catre o baza de date si respectand modelul Layered Arhitecture. Functiile minime ale aplicatiei:

* adaugarea, stergerea si editarea clientilor
* adaugarea, stergerea si editarea produselor
* crearea de comenzi folosind clienti si produse deja existente

# Analiza problemei, modelare, scenarii, cazuri de utilizare

Deoarece aplicatia va trebuie sa gestioneze atat interfata grafica a aplicatiei cat si comnicarea cu baza de date, este necesar sa adoptam un model de proiectare potrivit astfel incat codul sa fie organizat intr-un mod intuitiv si usor de modificat. Acestea fiind spuse este necesar ca aplicatia noastra sa respecte modelul Layered Arhitecture, impartind clasele in patru seturi:

* Clasele ce tin de interfata grafica
* Clasele model ce stocheaza informatia din bazele de date
* Clasele ce tin de logica aplicatiei
* Clasele care comunica cu baza de date

Pentru o implementare mai usoara ne vom folosi de technicile reflexiei din Java, astfel incat sa putem generaliza cat mai mult interogarile pe care le vom face catre baza de date. Pentru aceasta este necesar ca obiectele noastre din Model sa aiba exact aceleasi campuri ca atributele tabelelor din baza de date.

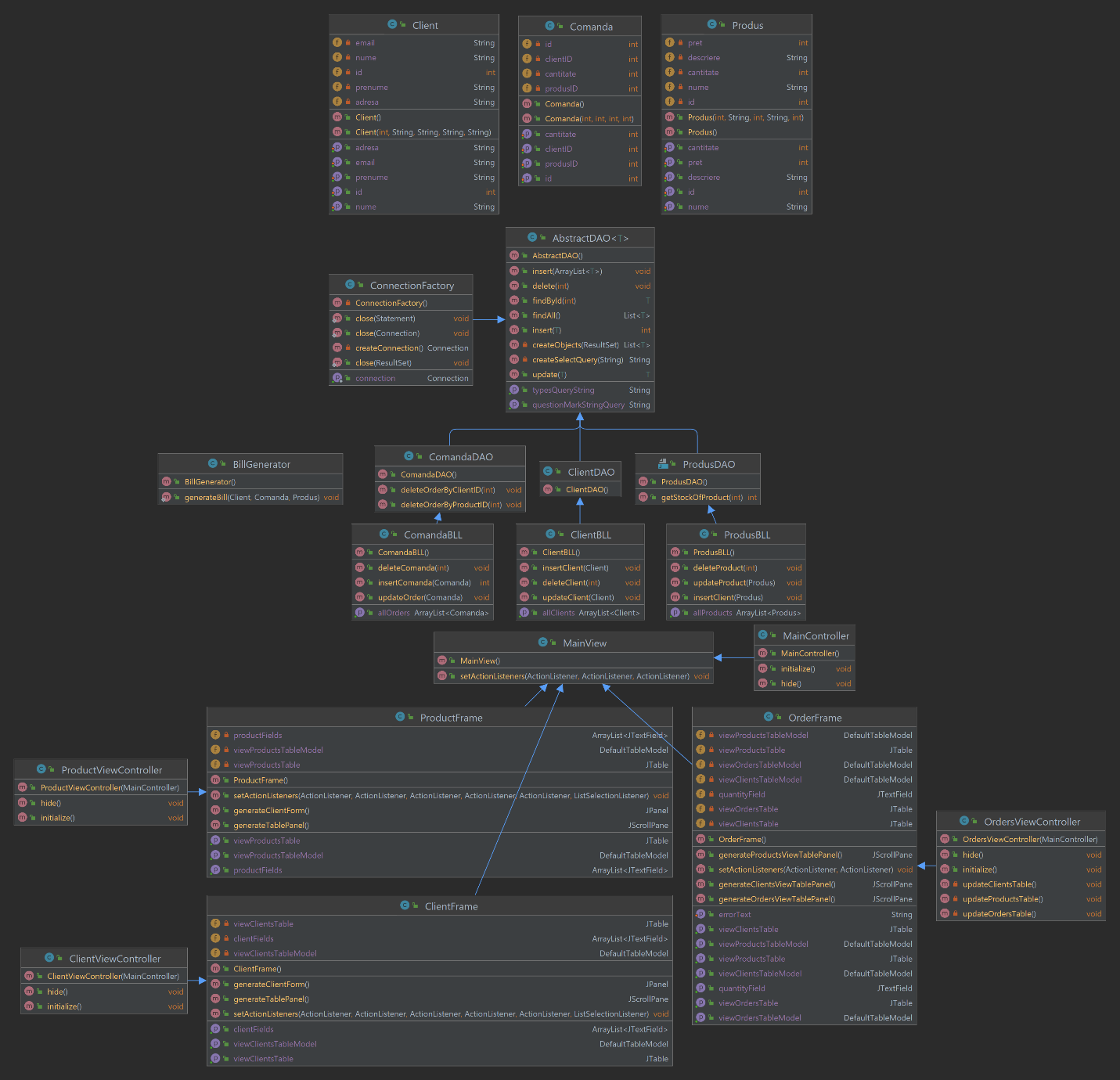
Un aspect important ce trebuie luat in considerare este validitatea informatiile introduse de utilizator in aplicatie, fiind necesara o verificare a datelor introduse inaintea adaugarii acestora in baza de date.

Situatii ce ar putea fi luate in considerare sunt:

* Utilizatorul a introdus un cuvant intr-un camp in care se asteapta un numar.
* Utilizatorul nu are un format de adresa de email valid.

# Proiectare

Clasele sunt organizate dupa cele patru seturi discutate la capitolul 2: Model classes, Business Logic classes, Presentation classes si Data acces classes. Cele patru layere sunt putin dependente una de cealalta, astfel garantand modificarea fara dificultate a logicii din spatele oricarui layer fara nevoia modificarii restului programului. Diagrama UML a claselor este urmatoarea:



**Data acces layer:**

Acest layer asigura conexiunea aplicatiei cu baza de date si include clasele: ConnectionFactory, AbstractDAO, ClientDAO, ProdusDAO, ComandaDAO.

ConnectionFactory:

Clasa ConnectionFactory initializeaza conexiunea cu baza de date, in ea fiind stocate adresa serverului catre baza de date, username-ul si parola de inregistrare. Aceasta clasa are de asemenea rolul de a inchide atat conexiunea catre baza de date in momentul in care nu mai este necesara, cat si statement-urile si resultset-urile generate in timpul ei.

AbstractDAO

Aceasta clasa implementeaza operatiile principale care vor fi realizate pe baza de date, cu ajutorul tehnicii reflexiei generandu-se automat interogarile corespunzatoare in functie de clasa care extinde aceasta clasa abstracta. Operatiile implementate sunt insert, delete, update, findById si findAll.

ComandaDAO

Clasa comandaDAO mosteneste toate operatiile implementate in AbstractDAO, iar pe langa acestea au fost implementate si doua metode ce sterg anumite comenzi din baza de date in functie de ID-ul unui produs sau al unui client

ProdusDAO

Clasa ProdusDAO mosteneste toate operatiile implementate in AbstractDAO, iar pe langa acestea a fost implementata o metoda ce returneaza atributul cantitate al unui anumit produs in functie de ID.

Business logic layer

Acest layer se ocupa cu logica din spatele aplicatiei, si se foloseste de clasele din Data acces layer pentru a comunica cu baza de date. Clasele componente sunt: ClientBLL, ProdusBLL, ComandaBLL.

**Model**

Pachetul model defineste prin clasele sale toate structurile de date cu care va lucra aplicatia, astfel ele coincid cu tabelele prezente in baza de date. Pentru a ne putea folosi de tehnica reflexiei, este important ca denumirea field-urilor definite in clase sa fie aceeasi cu atributele tabelelor din baza de date. Clasele definite in pachetul Model sunt: Client, Produs, Comanda

**Presentation**

Pachetul Presentation contine toate clasele care tin de interfata grafica a aplicatiei, asta incluzand atat Frame-urile definite cat si clasele Controller ale acestora. Aplicatia contine patru ferestre, o fereastra pentru gestionarea clientilor, una pentru gestionarea produselor, una pentru gestionarea comenzilor si o fereastra principala care face trimitere catre celelalte trei.

Clasele ClientViewController, OrdersViewController si ProductViewController asigura legatura ferestrelor cu restul logicii aplicatiei, respectiv cu clasele din Business logic layer.

# Implementare

**Clasa AbstractDAO**

**Metode**

* public List<T> findAll() – are rolul de a returna o lista de obiecte de corespund tutoror elementelor dintr-o tabela. Singurul parametru necesar este tipul clasei ce a extins clasa AbstractDAO, interogarea SQL fiind generata automat luand toate atributele clasei respective prin tehnica reflexiei.
* public T findById(int id) – similara cu metoda findAll() dar returneaza doar un obiect din tabela corespunzatoare in functie de id-ul primit ca parametru
* public int insert(T t) – insereaza in tabela corespunzatoare din baza de date un obiect de tip T. Metoda returneaza deasemenea id-ul generat de baza de date in urma inserarii acesteia, aspect util pentru cateva metode din Business logic layer.
* public void insert(ArrayList<T> elements) – o metoda Overload pentru metoda insert care poate primi ca parametru o lista de obiecte pentru inserare.
* public T update(T t) – aceasta metoda modifica valorile tuplei corespunzatoare field-ului id din cadrul obiectului t.
* public void delete(int id) – sterge tupla cu id-ul primit ca parametru din tabela corespunzatoare clasei ce mosteneste aceasta metoda.
* public String getTypesQueryString() – aceasta metoda genereaza un string in care sunt plasate numele field-urile clasei T, intr-un format potrivit pentru generarea interogarilor SQL
* public String getQuestionMarkStringQuery() – genereaza un string cu formatul (?,?,…?) numarul semnelor de intrebare fiind numarul de field-urile ale clasei T

**Clasa ComandaDAO extends AbstractDAO**

**Metode**

* public void deleteOrderByClientID(int ClientID) – aceasta metoda acceseaza baza de date si sterge doar tupla care are atributul ClientID egal cu valoarea primita ca parametru
* public void deleteOrderByProductID(int productID) - aceasta metoda acceseaza baza de date si sterge doar tupla care are atributul productID egal cu valoarea primita ca parametru

**Clasa ProdusDAO extends AbstractDAO**

**Metode**

* public int getStockOfProduct(int id): aceasta metoda returneaza atributul cantitate din tupla care are atributul id egal cu valoarea primita ca parametru.

**Clasa ConnectionFactory**

**Metode**

* private Connection createConnection(): returneaza conexiunea cu baza de date
* private static Connection getConnection(): metoda statica ce returneaza o referinta la conexiunea catre baza de date
* public static void close(Connection connection): incearca inchiderea conexiunii si prinde eroarea SQL in caz de eroare.

**Clasa ClientBLL**

**Metode**

* public ArrayList<Client> getAllClients(): se foloseste de clasa ClientDAO pentru a returna intr-o lista de clienti toate tuplele din tabela Clienti.
* public void deleteClient(int id): se foloseste de clasa ClientDAO pentru a sterge un client cu un anumit id din tabela Client din baza de date. Inainte de a sterge un client metoda sterge si toate comenzile acestuia pentru a nu ramane cu informatii incorecte in baza de date.
* public void insertClient(Client c): insereaza in baza de date un client primit ca parametru
* public void updateClient(Client c): actualizeaza tuplele unui client deja existent

**Clasa ComandaBLL**

**Metode**

* public ArrayList<Comanda> getAllOrders: returneaza o lista de obiecte corespunzatoare tuturor tuplelor din tabela Comanda
* public void deleteComanda(int id): se foloseste de clasa ComandaDAO pentru a sterge o comanda cu un anumit id din tabela Comanda din baza de date.
* public void insertComanda(Comanda c): insereaza in baza de date o comanda primita ca parametru. Inainte de introducerea acesteia in baza de date se verifica daca exista destule produse in stoc. In cazul in care nu exista, se va returna valoarea -1. In caz contrar se va insera comanda in baza de date, se va decrementa stocul produsului si se va genera in format .txt o factura a comenzii.
* public void updateComanda(Client c): actualizeaza tuplele unei comenzi deja existente

**Clasa ProductBLL**

**Metode**

* public ArrayList<Product> getAllProducts(): se foloseste de clasa ProductDAO pentru a returna intr-o lista de produse toate tuplele din tabela Produs.
* public void deleteProduct(int id): se foloseste de clasa ProductDAO pentru a sterge un un produs cu un anumit id din tabela Produs din baza de date. Inainte de a sterge un produs metoda sterge si toate comenzile ce contin produsul pentru a nu ramane cu informatii incorecte in baza de date.
* public void insertProduct(Product p): insereaza in baza de date un produs primit ca parametru
* public void updateProduct(Product p): actualizeaza tuplele unui produs deja existent

**Clasa BillGenerator**

O clasa utilitara ce are rolul de a genera o factura in urma plasarii unei comenzi

**Metode**

* public static void generateBill(Client client, Produs produs, Comanda comanda): metoda stocheaza intr-un fisier denumit „order(orderId).txt” datele despre o anumita comanda cum ar fi informatii despre client, informatii despre produs, si suma totala de plata.

**Claselse Client, Produs, Comanda**

**Atribute**

Au ca field-uri atributele din tabelele corespunzatoare clasei:

Client: int id, String nume, String prenume, String email, String adresa;

Produs: int id, String nume, int pret, String descriere, int cantitate

Comanda: int id, int clientID, int produsID, cantitate

**Metode**

Toate metodele din aceste clase sunt de tip setter sau getter pentru field-uri.

**Clasa MainController**

Face legatura dintre fereastra principala si logica aplicatiei

**Constructori**

* public MainController(): initializeaza clasele controller pentru celelalte trei ferestre si clasa Frame a ferestrei principale

**Clase**

* public class ClientViewListener implements ActionListener: implementeaza functia actionListener actionata in urma apasarii butonului Clienti din fereastra principala. Metoda ascunde fereastra principala si initializeaza fereastra ClientFrame
* public class ProductViewListener implements ActionListener: implementeaza functia actionListener actionata in urma apasarii butonului Produse din fereastra principala. Metoda ascunde fereastra principala si initializeaza fereastra ProductFrame
* public class OrdersViewListener implements ActionListener: implementeaza functia actionListener actionata in urma apasarii butonului Comenzi din fereastra principala. Metoda ascunde fereastra principala si initializeaza fereastra OrderFrame

**Clasa ClientViewController**

**Constructori**

* public ClientViewController(MainController controller): initializeaza frame-ul ClientFrame si adauga clasele listener pentru butoanele si tabelele din interfata.

**Metode**

* public class ApplyAddClientListener implements ActionListener: implementeaza functia actionListener actionata in urma apasarii butonului addClientButton din fereastra ClientFrame. Metoda extrage datele din tabele, le insereaza in baza de date folosindu-se de metoda insertClient din clasa ClientBLL
* public class EditClientListener implements ActionListener: implementeaza functia actionListener actionata in urma apasarii butonului editClientButton din fereastra ClientFrame. Metoda extrage datele din obiectele JtextField si actualizeaza tupla din baza de date corespunzatoare liniei selectate in tabela din ClientFrame
* public class DeleteClientListener implements ActionListener: implementeaza functia actionListener actionata in urma apasarii butonului deleteButton din fereastra ClientFrame. Metoda sterge tupla din baza de date corespunzatoare elementului selectat in tabela.
* public class TableSelectionListener implements ActionListener: implementeaza functia valueChanged actionata in urma schimbarii selectiei din tabel. Odata ce selectia este schimbata obiectele JtextField vor avea setat ca text informatia corespunzatoare elementului selectat
* public void initialize(): metoda ce afiseaza fereastra si in acelasi timp actualizeaza informatia din tabelele din ClientFrame apeland la metodele din ClientBLL
* public void hide(): ascunde fereastra

**Clasa ProductViewController**

**Constructori**

* public ProductViewController(MainController controller): initializeaza frame-ul ProductFrame si adauga clasele listener pentru butoanele si tabelele din interfata.

**Metode**

* public class ApplyAddProductListener implements ActionListener: implementeaza functia actionListener actionata in urma apasarii butonului addProductButton din fereastra ProductFrame. Metoda extrage datele din tabele, le insereaza in baza de date folosindu-se de metoda insertProduct din clasa ProductBLL
* public class EditProductListener implements ActionListener: implementeaza functia actionListener actionata in urma apasarii butonului editProductButton din fereastra ProductFrame. Metoda extrage datele din obiectele JtextField si actualizeaza tupla din baza de date corespunzatoare liniei selectate in tabela din ProductFrame
* public class DeleteProductListener implements ActionListener: implementeaza functia actionListener actionata in urma apasarii butonului deleteButton din fereastra ProductFrame. Metoda sterge tupla din baza de date corespunzatoare elementului selectat in tabela.
* public class TableSelectionListener implements ActionListener: implementeaza functia valueChanged actionata in urma schimbarii selectiei din tabel. Odata ce selectia este schimbata obiectele JtextField vor avea setat ca text informatia corespunzatoare elementului selectat
* public void initialize(): metoda ce afiseaza fereastra si in acelasi timp actualizeaza informatia din tabelul din ProductFrame apeland la metodele din ProductBLL
* public void hide(): ascunde fereastra

**Clasa OrderViewController**

**Metode**

* public class CreateOrderButtonListener implements ActionListener: implementeaza functia actionListener actionata in urma apasarii butonului createOrderButton din fereastra OrderFrame. Metoda determina id-ul produsului si al clientului selectat in tabelele din OrderFrame iar apoi insereaza rezultatul folosind clasa ComandaBLL. Daca rezultatul returnat de metoda insertComanda este -1 atunci se afiseaza un mesaj de eroare. In caz contrar se reactualizeaza informatiile din tabele.
* public class TableSelectionListener implements ActionListener: implementeaza functia valueChanged actionata in urma schimbarii selectiei din tabel. Odata ce selectia este schimbata obiectele JtextField vor avea setat ca text informatia corespunzatoare elementului selectat
* public void updateOrdersTable(): actualizeaza datele din tabelul comenzilor
* public void updateClientsTable(): actualizeaza datele din tabelul clientilor
* public void updateProductsTable(): actualizeaza datele din tabelul produselor

**Interfata**

**MainView**

**Graphical user interface

Description automatically generated with low confidence**

**ClientView**

**Graphical user interface, application

Description automatically generated**

**ProductView**

**Graphical user interface

Description automatically generated**

**OrdersView**

**Table

Description automatically generated**

# Rezultate

Pentru testarea programului am creat 9 clienti si 8 produse, iar apoi am creat 5 comenzi. Fisierele generate in urma crearii comenzilor sunt urmatoarele:

* order28.txt:

FACTURA COMANDA NR. 28

--------------------------

Client:

Nume: Maria

Prenume: Ioana

Email: mariaioana5020@yahoo.com

Adresa livrare: Iasi, str. Mihai Eminescu nr. 19

--------------------------

Produse: Cuie x 5

Total: 5 x 1 lei = 5

* order29.txt:

FACTURA COMANDA NR. 29

--------------------------

Client:

Nume: Costel

Prenume: Vamesu

Email: costikka.vamesu@gmail.com

Adresa livrare: Radaseni, str. Viilor nr.2

--------------------------

Produse: Chinga de ancoraj x 2

Total: 2 x 100 lei = 200

* order31.txt:

FACTURA COMANDA NR. 31

--------------------------

Client:

Nume: Matei

Prenume: Corvin

Email: mateiush420@yahoo.com

Adresa livrare: Cluj-Napoca, str. Augusting Presecan nr. 6 bl. T2

--------------------------

Produse: Tabla zincata x 5

Total: 5 x 34 lei = 170

* order32.txt:

FACTURA COMANDA NR. 32

--------------------------

Client:

Nume: Mircea

Prenume: Diaconu

Email: mircea.diaconu@protonmail.com

Adresa livrare: Bucuresti, str. Victoriei nr. 8

--------------------------

Produse: Ciocan x 2

Total: 2 x 20 lei = 40

* order 33.txt:

FACTURA COMANDA NR. 33

--------------------------

Client:

Nume: Laura

Prenume: Matei

Email: laura.matei@yahoo.com

Adresa livrare: Constanta, Costinesti, str. Delfinului nr.2

--------------------------

Produse: Stivuitor electric x 1

Total: 1 x 21000 lei = 21000

# Concluzii

In concluzie din aceasta tema am invatat ca tehnica reflexiei poate fi extrem de utila in contextul comunicarii cu o baza de date, capacitatea de a vedea in timpul executiei unei aplicatii structura unei clase oferindu-ne foarte multe posibilitati de a ne optimiza codul si de a evita repetarea acestuia.

# Bibliografie

* <https://jenkov.com/tutorials/java-reflection/index.html>
* <https://dzone.com/articles/layers-standard-enterprise>
* <https://www.baeldung.com/java-jdbc>
* <https://www.baeldung.com/javadoc>
* <https://www.baeldung.com/java-write-to-file>
* <https://www.baeldung.com/raw-types-java>
* https://www.geeksforgeeks.org/class-getdeclaredfields-method-in-java-with-examples/