DOCUMENTATIE

TEMA *3*

NUME STUDENT: RADU DARIUS - RAZVAN

GRUPA: 30225

# 

Contents

[1. Obiectivul temei 3](#_Toc100581557)

[2. Analiza problemei, modelare, scenarii, cazuri de utilizare 3](#_Toc100581558)

[3. Proiectare 3](#_Toc100581559)

[4. Implementare 6](#_Toc100581560)

[5. Rezultate 11](#_Toc100581561)

[6. Concluzii 12](#_Toc100581562)

[7. Bibliografie 12](#_Toc100581563)

# Obiectivul temei

Obiectivul temei a fost sa proiectam un sistem de management al comenzi unui depozit. Avem o interfata grafica de unde utilizatorul poate selecta daca vrea sa faca operatii pe clienti (sa adauge, modifice sau stearga clienti, cat si sa ii afisam pe toti), produse (sa adauge, modifice sau stearga clienti, cat si sa ii afisam pe toti) si comenzi (adaugare comanda, afisare toate comenzi si creare nota de plata).

# Analiza problemei, modelare, scenarii, cazuri de utilizare

Modelarea problemei a fost facuta in mare parte dupa exemplul alaturat prezentarii temei.

Use case: inserarea unui client in tabela.

Actor principal: utilizator

Pasi:

* Selectam butonul pentru operatiile pe tabela client.
* Selectam meniul pentru inserare
* Introducem datele clientului ce dorim a fi inserat
* Apasam butonul de insert
* Se afiseaza un mesaj daca inserarea s-a realizat cu succes.

# Proiectare

**Diagram

Description automatically generatedDiagrama de pachete** folosite in cadrul aplicatiei noastre poate fi obseravata mai sus. Diagrama prezinta dependentele pachetelor (view-ul este dependent de model, iar controller-ul este dependent atat de model, cat si de view) pachetul de BusinessLogic se foloseste doar de pachetele model si DataAccess, iar pachetul DataAccess se foloseste doar de pachetele Model si Connection.

Fiecare pachet contine clase ce au un anumit rol bine definit.

In pachetul „ModelPackage” regasim clasele Client, Product si Order, ce sunt modelele pentru client, produs si comanda. Acestea contin aceleasi campuri ca si tabelele din baza de date. Cu o mentiune: in clasa Order, nu stocam id-urile produsului si clientului, ci obiectul in sine.

In pachetul „ViewPackage” regasim clasele HomePage, ProductOperationsView, ClientOperationsView, OrderView. Acestea sunt GUI-urile, prin intermediul carora utilizatorul interactioneaza cu baza de date.

In pachetul „ControllerPackage” regasim clasele HomePageController, ProductOperationsController, ClientOperationsController, OrderController. Acestea se ocupa de controlul view-urilor.

In pachetul „BusinessLogicPackage” regasim clasele ClientBLL, ProductBLL, OrderBLL, ce se ocupa de logica din spate a aplicatiei. De asemenea, pachetul contine si un folder cu clase validator. Aceste clase valideaza datele ce urmeaza a fi stocate in baza de date.

In pachetul „DataAccessPackage” regasim clasele AbstractDAO, ClientDAO, ProductDAO, OrderDAO, ce au rolul de a se ocupa cu accesul la baza de date.

In pachetul „ConnectionPackage” regasim clasa ConnectionFactory, ce se ocupa cu conectarea sau deconectarea bazei de date.

**Diagrama UML:**

**Diagram

Description automatically generated**

* „ModelPackage”:

Clasa „Client” => reprezinta modelarea clientului, aceasta clasa contine id, nume, adresa, email si varsta. Aceste campuri sunt identice cu campurile din tabela client din baza de date.

Clasa „Produs” => reprezinta modelarea produsului, aceasta clasa contine id, nume, stoc si pret. Aceste campuri sunt identice cu campurile din tabela produs din baza de date.

Clasa „Order” => reprezinta modelarea comenzii, aceasta clasa contine id, client, produs si cantitate. Aceste campuri sunt ca si in baza de date, cu exceptia faptului ca, in baza de date stocam doar id-ul clientului si al produsului pt realizarea relatiei dintre tabele, dar in java stocam intreg obiectul.

* „ViewPackage”:

Clasa „HomePage” => reprezinta un frame, ce contine 3 butoane, fiecare buton va deschide un alt frame, in functie de ce operatii vrem sa facem si pe ce tabele le vom face.

Clasa „OrderView” => reprezinta frame-ul unde vom face operatiile pe tabela comanda (vom crea o comanda si o inseram in tabela, afisam toate comenziile si creem o factura cu tot ce s-a comandat).

Clasa „ProductOperationsView” => reprezinta frame-ul unde vom face operatiile pe tabela produs (vom insera produsa, edita produse, sterge produse si afisa toate produsele).

Clasa „ClientOperationsView” => reprezinta frame-ul unde vom face operatiile pe tabela client (vom insera clienti, edita clienti, sterge clienti si afisa toti clientii).

* „ControllerPackage”:

Clasa „HomePageController” => reprezinta controllerul pentru frame-ul „HomePage”.

Clasa „OrderController” => reprezinta controller-ul pentru frame-ul „OrderView”

Clasa „ClientOperationsController” => reprezinta controller-ul pentru frame-ul „ClientOperationsView”.

Clasa „ProductOperationsController” => reprezinta controller-ul pentru frame-ul „ProductOperationsView”.

* „BusinessLogicPackage”:

Clasa „ClientBLL” => reprezinta clasa ce se ocupa cu validarile clientilor si apelurile de metode ce opereaza pe tabela client, in baza de date.

Clasa „ProductBLL” => reprezinta clasa ce se ocupa cu validarile produselor si apelurile de metode ce opereaza pe tabela product, in baza de date.

Clasa „OrderBLL” => reprezinta clasa ce se ocupa cu validarile comenzilor si apelurile de metode ce opereaza pe tabela order, in baza de date.

Interfata „Validator” => contine metoda valitate(), ce valideaza un obiect trimis ca parametru

Clasa „AgeValidator” => implementeaza interfata Validator si valideaza varsta clientilor.

Clasa „AmountValidator” => implementeaza interfata Validator si valideaza cantitatea comenziilor.

Clasa „EmailValidator” => implementeaza interfata Validator si valideaza email-ul clientilor.

Clasa „PriceValidator” => implementeaza interfata Validator si valideaza pretul produselor.

Clasa „StockValidator” => implementeaza interfata Validator si valideaza stocul produselor.

* „DataAccessPackage”:

Clasa „AbstractDAO” => este o clasa generica ce contine toate metodele (find, insert, update, delete) ce pot fi realizate pe orice tabela din baza de date.

Clasa „ClientDAO” => este o clasa ce extinde AbstractDAO. Nu are alte metode.

Clasa „ProductDAO” => este o clasa ce extinde AbstractDAO. Nu are alte metode.

Clasa „OrderDAO” => este o clasa ce extinde AbstractDAO. Are unele metode in plus, ce sunt pentru operatii specifice doar pe tabela order.

* „ConnectionPackage”:

Clasa „ConnectionFactory” => este o clasa ce deschide sau inchide o conexiune cu baza de date.

**Structuri de date folosite**:

ArrayList<> sau List<> pentru stocarea de clienti, produse sau comenzi.

Map<Filed, Object> pentru stocarea de atribute si valorile lor, in procesul de reflection.

Map<Client, ArrayList<Order>> pentru stocarea

# Implementare

Metode din pachetul ModelPackage:

1. Clasa „Client”:

Avem 4 constructori, pentru 4 scenarii diferite: unul ce nu primeste niciun parametru; unul ce primeste ca parametru id (acest constructor este folosit atunci cand vrem sa stergem comenziile ce au fost facute de un anumit client. In procesul de stergere avem nevoie doar de id-ul clientului, deci vom crea un obiect de tip client ce are setat doar id-ul respectiv, in functie de care dorim sa stergem comanda); unul cu toate atributele mai putin id-ul (pentru scenariul in care dorim sa inseram un client, id-ul se genereaza automat, deci nu avem de unde stii care va fi id-ul clientului pana ce nu este inserat); unul cu toate atributele.

Metoda toString(): am modificat metoda generata automat, pentru ca afisarea clientilor sa fie dupa un model propriu, aceasta metoda o folosesc pentru ca vreau sa introduc clientii intr-o lista (componenta de fronend) pentru a se afisa in frame.

Getters and setters => folosite pentru a returna, respectiv a seta, atributele obiectului.

1. Clasa „Product”:

Avem 4 constructori, pentru 4 scenarii diferite: unul ce nu primeste niciun parametru; unul ce primeste ca parametru id (acest constructor este folosit atunci cand vrem sa stergem comenziile de un anumit produs. In procesul de stergere avem nevoie doar de id-ul produsului, deci vom crea un obiect de tip produs ce are setat doar id-ul respectiv, in functie de care dorim sa stergem comanda); unul cu toate atributele mai putin id-ul (pentru scenariul in care dorim sa inseram un produs, id-ul se genereaza automat, deci nu avem de unde stii care va fi id-ul produsului pana ce nu este inserat); unul cu toate atributele.

Metoda decrementStock(int amount) => este folosita pentru a decrementa stocul produsului dupa ce a fost facuta o comanda.

Metoda toString(): am modificat metoda generata automat, pentru ca afisarea produselor sa fie dupa un model propriu, aceasta metoda o folosesc pentru ca vreau sa introduc produsele intr-o lista (componenta de fronend) pentru a se afisa in frame.

Getters and setters => folosite pentru a returna, respectiv a seta, atributele obiectului.

1. Clasa „Order„:

Avem 3 constructori pentru 3 scenarii diferite: unul fara niciun parametru; unul ce primeste toate atributele, mai putin id-ul (pentru cazul in care dorim sa inseram comanda); unul ce primeste toate atributele.

Metoda toString(): am modificat metoda generata automat, pentru ca afisarea comenziile sa fie dupa un model propriu, aceasta metoda o folosesc pentru ca vreau sa introduc comenziile intr-o lista (componenta de fronend) pentru a se afisa in frame.

Getters and setters => folosite pentru a returna, respectiv a seta, atributele obiectului.

Metode din pachetul ConnectionPackage:

1. Clasa „ConnectionFactory”:

Metoda createConnection() => creaza o conexiune cu baza de date.

Metoda getConnection() => returneaza conexiunea cu baza de date.

Metodele close() => aceste metode sunt folosite pentru a inchide ResultSet-urile, Statement-urile si Conexiunile cu baza de date.

Metode din pachetul DataAccessPackage:

1. Clasa „AbstractDAO”:

Metoda retrieveProperties() => aceasta metoda este folosita pentru a crea un Map, ce are ca si keys field-urile din clasa parametru generic si ca si values valorile din acele campuri.

Metoda createObjects() => aceasta metoda este folosita pentru a crea o lista de obiecte de tipul parametrului generic, din resultset-ul trimis ca parametri. Aceasta metoda va parcurge resultset-ul trimis ca parametru si va seta fiecare camp dintr-o instanta a clasei parametrului generic cu valoarea din result set.

Metoda createSelectQuery => aceasta metoda creaza un string, ce reprezinta un sql query, ce returneaza toate intrarile din tabela trimisa ca parametru generic, in functie de un field ce il trimitem ca parametru.

Metoda createSelectAllQuery => aceasta metoda creaza un string, ce reprezinta un sql query, ce returneaza toate intrarile din tabela trimisa ca parametru generic.

Metoda createInsertQuery => aceasta metoda creaza un string, ce reprezinta un sql query, ce insereaza un obiect in tabela trimisa ca parametru generic. Metoda se foloseste de map-ul returnat de metoda retrieveProperties, pentru ca nu stim exact cate campuri are obiectul ce dorim a fi inserat.

Metoda createUpdateQuery => aceasta metoda creaza un string, ce reprezinta un sql query, ce face update unui obiect din tabela trimisa ca parametru generic. Metoda se foloseste de map-ul returnat de metoda retrieveProperties, pentru ca nu stim exact cate campuri are obiectul ce dorim a fi editat.

Metoda createDeleteQuery => aceasta metoda creaza un string, ce reprezinta un sql query, ce face delete unui obiect din tabela trimisa ca parametru generic.

Metoda findAll => aceasta metoda executa query-ul creat de metoda createSelectAllQuery, apoi resultset-ul il converteste intr-o lista de obiecte de tipul parametrului generic prin apelarea functiei createObjects.

Metoda findById => aceasta metoda executa query-ul creat de metoda createSelectQuery, apoi resultset-ul il converteste intr-o lista de obiecte de tipul parametrului generic prin apelarea functiei createObjects si returneaza primul element din lista.

Metoda insert => aceasta metoda executa query-ul creat de metoda createInsertQuery, apoi seteaza id-ul obiectului trimis ca parametru si il returneaza inpoi.

Metoda update => aceasta metoda executa query-ul creat de metoda createUpdateQuery, apoi returneaza obiectul inapoi.

Metoda delete => aceasta metoda executa query-ul creat de metoda createDeleteQuery, apoi returneaza obiectul inapoi.s

1. Clasa „ClientDAO”: clasa goala, am crat-o doar pentru a extinde clasa AbstractDAO si a trimite parametrul generic ca fiind clasa Client.
2. Clasa „ProductDAO”: clasa goala, am crat-o doar pentru a extinde clasa AbstractDAO si a trimite parametrul generic ca fiind clasa Product.
3. Clasa „OrderDAO”:

Metoda createSelectByIdsQuery => aceasta metoda creaza un string, ce reprezinta un sql query, ce returneaza toate intrarile din tabela order, in functie de id-ul produsului si id-ul clientului.

Metoda findByIds => aceasta metoda executa query-ul creat de metoda createSelectByIdsQuery, apoi resultset-ul il converteste intr-o lista comenzi prin apelarea functiei createObjects.

Metoda findByIdProduct => aceasta metoda executa query-ul creat de metoda createSelectQuery, apoi resultset-ul il converteste intr-o lista de comenzi prin apelarea functiei createObjects. Aceasta metoda returneaza toate comenziile ce au id-ul produsului egal cu cel primit ca si parametru, deoarece createSelectQuery, va crea un select query in functie de field-ul produs.

Metoda findByIdClient => aceasta metoda executa query-ul creat de metoda createSelectQuery, apoi resultset-ul il converteste intr-o lista de comenzi prin apelarea functiei createObjects. Aceasta metoda returneaza toate comenziile ce au id-ul clientului egal cu cel primit ca si parametru, deoarece createSelectQuery, va crea un select query in functie de field-ul client.

Metode din pachetul BusinessLogicPackage:

1. Clasa ClientBLL:

Metoda findAll => aceasta metoda apeleaza metoda findAll din clasa ClientDAO, pentru a returna toti clientii din baza de date, in cazul in care tabela este goala, se va afisa un mesaj de informare.

Metoda insert => aceasta metoda apeleaza metoda insert din clasa ClientDAO, pentru a insera un client in baza de date. Intainte de a-l insera, vom valida clientul cu toti validatorii specifici clientului. Daca nu se reuseste vom trimite o exceptie.

Metoda update => aceasta metoda apeleaza metoda update din clasa ClientDAO, pentru a edita un client din baza de date. Intainte de a-l edita, vom valida clientul cu toti validatorii specifici clientului. Daca nu se reuseste vom trimite o exceptie.

Metoda delete => aceasta metoda apeleaza metoda delete din clasa ClientDAO, pentru a sterge un client din baza de date. Inainte de a sterge clienti, trebuie sa stergem toate comenziile ce au campul client egal cu id-ul clientului pe care vrem sa il stergem. Daca nu se reuseste vom trimite o exceptie.

1. Clasa ProductBLL:

Metoda findAll => aceasta metoda apeleaza metoda findAll din clasa ProductDAO, pentru a returna tpate produsele din baza de date, in cazul in care tabela este goala, se va afisa un mesaj de informare.

Metoda insert => aceasta metoda apeleaza metoda insert din clasa ProductDAO, pentru a insera un produs in baza de date. Intainte de a-l insera, vom valida produsului cu toti validatorii specifici produsului. Daca nu se reuseste vom trimite o exceptie.

Metoda update => aceasta metoda apeleaza metoda update din clasa ProductDAO, pentru a edita un produs din baza de date. Intainte de a-l edita, vom valida produsul cu toti validatorii specifici produsului. Daca nu se reuseste vom trimite o exceptie.

Metoda delete => aceasta metoda apeleaza metoda delete din clasa ProductDAO, pentru a sterge un produs din baza de date. Inainte de a sterge produse, trebuie sa stergem toate comenziile ce au campul produs egal cu id-ul produsului pe care vrem sa il stergem. Daca nu se reuseste vom trimite o exceptie.

1. Clasa OrderBLL:

Metoda findAll => aceasta metoda apeleaza metoda findAll din clasa OrderDAO, pentru a returna toate comenziile din baza de date, in cazul in care tabela este goala, se va afisa un mesaj de informare.

Metoda insert => aceasta metoda apeleaza metoda insert din clasa OrderDAO, pentru a insera o comanda in baza de date. Intainte de a o insera, vom valida comanda cu toti validatorii specifici comenzii. Daca nu se reuseste vom trimite o exceptie.

Metoda update => aceasta metoda apeleaza metoda update din clasa OrderDAO, pentru a edita o comanda din baza de date. Intainte de a o edita, vom valida comanda cu toti validatorii specifici comenzii. Daca nu se reuseste vom trimite o exceptie.

Metoda delete => aceasta metoda apeleaza metoda delete din clasa OrderDAO, pentru a sterge o comanda din baza de date. Daca nu se reuseste vom trimite o exceptie.

1. Clasele de validatori contin metoda validate, ce valideaza datele ce urmeaza a fi inserate in baza de date. Este cate un validator pentru fiecare camp ce trebuie validat.

Metode din pachetul ControllerPackage:

1. Clasa „HomePageController”:

Metoda controlOrder => aceasta metoda inchide frame-ul homepage si deschide un OrderView, pentru realizarea operatiilor pe tabela Order

Metoda controlProduct => aceasta metoda inchide frame-ul homepage si deschide un ProductOperationsView, pentru realizarea operatiilor pe tabela product

Metoda controlProduct => aceasta metoda inchide frame-ul homepage si deschide un ClientOperationsView, pentru realizarea operatiilor pe tabela client

1. Clasa „ClientOperationsController”:

Metoda controlBack => inchide fereastra curenta si ne intoarce la homepage.

Metoda controlInsert => ia informatiile despre client din view, creaza un obiect de tipul client si il insereaza in tabela folosind metoda insert din BLL. Se afiseaza un mesaj daca insertul s-a reusit cu succes sau nu.

Metoda controlUpdate => ia intai id-ul clientului pe care vrem sa il modificam, apoi noile informatii despre client din view, creaza un obiect de tipul client si il editeaza folosind metoda update din BLL. Se afiseaza un mesaj daca update-ul s-a reusit cu succes sau nu.

Metoda controlDelete => ia intai id-ul clientului pe care vrem sa il stergem, din view, apoi creem un obiect de tipul client ce contine id-ul respectiv si apelam functia delete din BLL. Se afiseaza un mesaj daca stergerea s-a realizat cu succes sau nu. De asemenea se face un double check daca chiar dorim sa stergem acel client sau nu.

1. Clasa „ProductOperationsController”:

Metoda controlBack => inchide fereastra curenta si ne intoarce la homepage.

Metoda controlInsert => ia informatiile despre produs din view, creaza un obiect de tipul product si il insereaza in tabela folosind metoda insert din BLL. Se afiseaza un mesaj daca insertul s-a reusit cu succes sau nu.

Metoda controlUpdate => ia intai id-ul produsului pe care vrem sa il modificam, apoi noile informatii despre produs din view, creaza un obiect de tipul product si il editeaza folosind metoda update din BLL. Se afiseaza un mesaj daca update-ul s-a reusit cu succes sau nu.

Metoda controlDelete => ia intai id-ul produsului pe care vrem sa il stergem, din view, apoi creem un obiect de tipul product ce contine id-ul respectiv si apelam functia delete din BLL. Se afiseaza un mesaj daca stergerea s-a realizat cu succes sau nu. De asemenea se face un double check daca chiar dorim sa stergem acel produs sau nu.

1. Clasa „OrderController”:

Metoda controlBack => inchide fereastra curenta si ne intoarce la homepage.

Metoda controlSubmit => ia id-ul produsului, id-ul clientului si cantitatea comenzii din view, creaza un obiect de tipul Order si il insereaza in tabela folosind metoda insert din BLL. Se afiseaza un mesaj daca insertul s-a reusit cu succes sau nu. De asemenea inseram comanda inserata (daca s-a inserat cu succes), intr-o lista pentru a putea face bill-ul in caz ca este necesar.

Metoda controlBill => din lista de comenzi creata in timpul rularii aplicatiei, cream un map<Client, ArrayList<Order>>, pentru a stii pt fiecare client, care sunt comenziile sale. Dupa ce cream map-ul, afisam intr-un fisier text bill.txt, nota de plata pentru fiecare client.

Metode din pachetul ViewPackage:

1. Clasa „HomePage”:

Graphical user interface, text, application

Description automatically generated

Contine doar metode ce seteaza ActionListeners pentru butoane.

1. Clasa „ClientOperationsView”:

Graphical user interface, application

Description automatically generated

Contine metode ce seteaza ActionListeners pentru butoane, metode ce returneaza valorile din textField-uri gata convertite in tipul de date pe care il dorim in controller.

Metoda setTable => este o metoda ce populeaza tabelul cu informatiile din baza de date.

Metoda setList => este o metoda ce populeaza cele doua liste de unde selectam ce client dorim sa il editam sau stergem.

1. Clasa „ProductOperationsView”:

Graphical user interface, application

Description automatically generated

Contine metode ce seteaza ActionListeners pentru butoane, metode ce returneaza valorile din textField-uri gata convertite in tipul de date pe care il dorim in controller.

Metoda setTable => este o metoda ce populeaza tabelul cu informatiile din baza de date.

Metoda setList => este o metoda ce populeaza cele doua liste de unde selectam ce produs dorim sa il editam sau stergem

1. Clasa „OrderView”:

Graphical user interface, application

Description automatically generated

Contine metode ce seteaza ActionListeners pentru butoane, metode ce returneaza valorile din textField-uri gata convertite in tipul de date pe care il dorim in controller.

Metoda setTable => este o metoda ce populeaza tabelul cu informatiile din baza de date.

Metoda setList => este o metoda ce populeaza cele doua liste de unde selectam ce client si ce produs dorim sa introducem in comanda.

# Rezultate

Aplicatia ruleaza bine, fara nicio eroare. Fiecare functionalitate este insotita cu un mesaj ce ne va spune daca operatia s-a realizat cu succes sau s-a petrecut o eroare a utilizatorului. De asemenea se genereaza bill-ul pentru fiecare client ce a facut comenzi in timpul rularii aplicatiei, iar datele sunt corecte.

# Concluzii

Tema nu a fost dificila din punct de vedere al implementarii, dar a fost lunga. Multe clase au fost implementate din cauza cresterii nivelului de abstractizare si de protectie. Am avut nevoie de mult mai multe nivele de securitate, deoarece lucram cu o baza de date.

De asemenea mi-a placut foarte mult ca am invatat sa structuram o aplicatie in acest mod, cu control, bussiness si data access. Consider ca aceasta tema a fost foarte utila, atat din cauza faptului ca am exersat interactiunea cu baza de date din codul java, cat si din cauza invatarii unei structuri bine definite de cod.

# Bibliografie