**Student: Ardelean Adriana-Ioana**

**Grupa 302210**

**Tehnici de programare**

**Tema 3**

**Aplicatie de gestionare a bazelor de data**

**Continut:**

1. Obiectivul temei
2. Analiza problemei, modelare, scenario, cazuri de utilizare
3. Proiectare (decizii de proiectare, diagrame UML, structuride date, proiectare clase, interfete, relatii, packages, algoritmi, interfata utilizator)
4. Implementare
5. Rezultate
6. Concluzii
7. Bibliografie
8. **Obiectivul temei**

Creati o aplicatie care gestioneaza comenziile clientilor pentru un depozit. Bazele de date sunt folosite pentru a tine clientii, produsele si comenziile. Aplicatia trebuie structurata in pachete folosind o arhitectura pe straturi si ar trebui sa foloseasca urmatoarele clase:

* Model Classes – reprezinta modelele de date ale aplicatiei
* Business Logic Classes – contin logica aplicatiei
* Presentation classes- clasele care au legatura cu interfata grafica
* Daca Access Classes – clase care contin accesarea bazei de date

Printre cerinte mai trebuia si generat o documentatie javadoc, folosirea tehnicilor de reflexie, cat si creeare unei chitante, unui fisier pentru fiecare comanda sub format de text sau .pdf

1. **Analiza problemei, modelare, scenario, cazuri de utilizare**
   1. **Analiza problemei**

Aplicatia ar trebui sa permita utilizatorului sa insereze, stearga si updateze informatia din baza de date. Prin inserare se intelege adaugarea de client sau produs cu toate field-urile necesare, cat si generarea unei comenzi alegand atat clientul cat si produsul si cantitatea in care acesta eeste comandat. Prin stergere intelegem stergerea unui client, produs sau comanda (acest lucru s-ar intampla dupa ce comanda este livrata, angajatii depozitului pot sterge comanda din baza de date). Un produs nu poate fi sters daca exista comanda cu acest produs. La fel un client nu poate fi sters daca exista comanda cu acest client. Prin update se intelege schimbarea sau reactualizarea datelor pentru clienti sau produse. Se va introduce id-ul clientului sau produsului pentru care se face modificarea dupa care noile atribute ale clientului sau produsului, Aplicatia ar trebui sa genereze un fisier text sau pdf pentru fiecare comanda.

* 1. **Modelare**

Modelarea este activitatea de a face un concept abstract mai usor de inteles prin gasirea principalelor sale caracteristici. Acest proces include, de asemenea, descompunerea unei probleme complexe in probleme mai mici si mai simple, care vor fi mai usor de implementat.

Modelarea acestei probleme era in cerinte, prin separarea problemei in pachete, fiecare pachete avand parte lui in problema de baza. DAO-uri realizeaza operatiile pe bazele de date. Presentation realizeaza interfata grafica si functionalitatea ei.

* 1. **Scenario**

Un scenariu de success ar fi urmatorul:

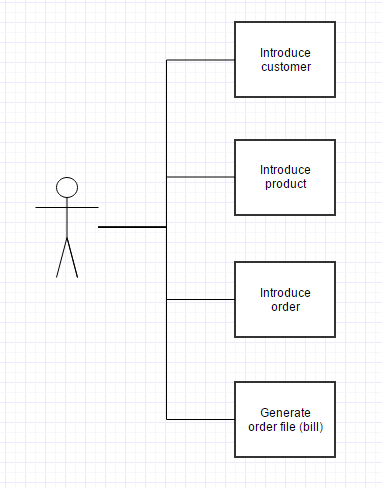
* Se introduce clienti, produse si comenzi
* Datele trebuie sa fie valide, varsta sa fie un numar natural mai mare de 16, sa existe produsul si clientul care face comanda si asa mai departe
* Cantitatea introdusa trebuie sa fie mai mica decat stocul
* Comanda va aparea in tabel

Un esec ar fi urmatorul:

* Se introduce o comanda
* Datele sunt invalide(varsta trebuie sa fie numar natural mai mare de 16, pretul si stocul trebuie sa fie numere naturale diferite de 0) sau cantitatea este mai mare decat stocul
* Se va genera un mesaj de eroare dupa care utilizatorul va trebui sa adauge alte input uri care sa fie valide
  1. **Cazuri de utilizare**

Diagrama pentru cazurile de utilizare sau diagrama de comportament descrie un set de actiuni care poate fi realizat de aplicatie in colaborare cu utilizatorii

Diagrama use case pentru aceasta aplicatie arata in felul urmator:



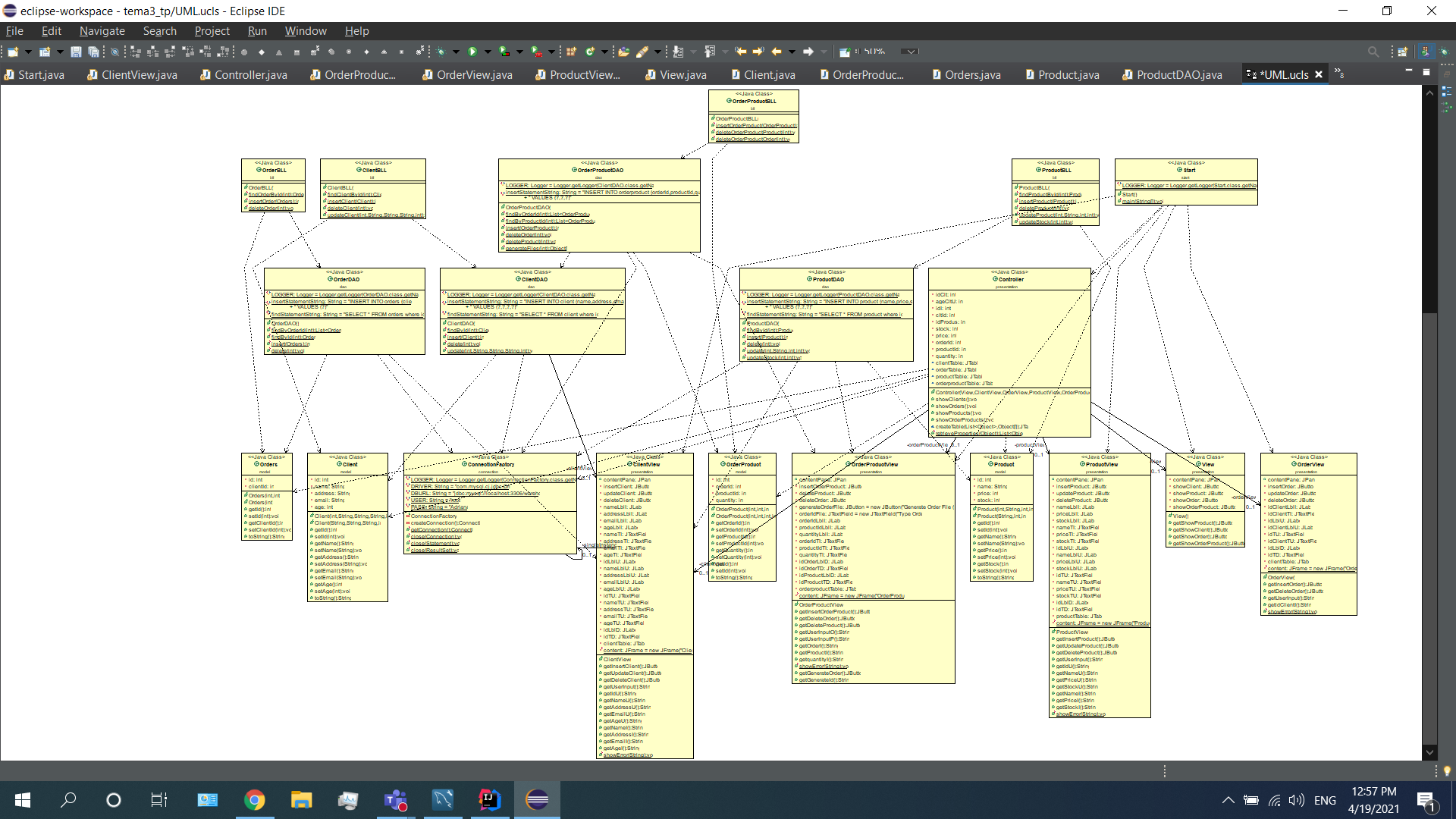
Un utilizator poate sa introduca un client, sa introduca produse cat si comenzi, acesta poate si sa stearga sau sa actualizeze datele despre client si produse. Tot asa un utilizator poate sa genereze un fisier reprezentand chitanta comenzii. In aceasta chitanta se va putea vedea numele produsului, pretul si cantitatea in care a fost cumparat, pretul total pentru fiecare tip de produs cat si pretul final al comenzii plasate.

1. **Proiectare (decizii de proiectare, diagrame UML, structuri de date, proiectare clase, interfete, relatii, packages, algoritmi, interfata utilizator)**
   1. **Decizii de proiectare**

Sunt multe decizii de luat in proiectarea acestei aplicatii, datorita libertatii de alegere date la implementarea acestui proiect. De exemplu, la implementarea bazei de date, pe langa cele 3 tabele obligatorii (client, product si orders) am ales sa mai adaug un tabel OrderProducts, care leaga tabelele orders si product pentru a exista o relatie many to many. Am ales si sa testez daca datele sunt valide folosind try catch uri in controller deoarece mi s-a parut mai la indemana decat sa fac validatoare prin Validators. Pentru a genera fisierul text pentru comenzi, am folosit un buffer pentru a genera fisierul iar mai apoi sa scrie date din tabel. Preluarea datelor este implementata cu ajutorul unei metode care extrage printr un query informatia necesare sim ai apoi o pune, o stocheaza intr un obiect de tip matrice.

* 1. **Diagrame UML**

UML este prescurtarea de la “Unified Modeling Language” si este notatia internationala standard pentru analiza si proiectarea orientata pe obiecte. Exista 2 categorii mari de diagrame UML, diagramele de structura si diagramele de comportament. Diagrama de clasa, o diagrama de structura, descrie structura unui sistem prin evidentierea claselor din sistem, a atributelor lor si a relatiilor dintre clase.



* 1. **Structuri de date**

Am folosit structura colectia List de unde am folosit ArrayList pentru partea de reflective pentru a adauga valorile in JTable. Am folosit ArrayList si pentru a putea stoca toate orderproduct-urile care au acelasi productId sau orderId.

* 1. **Proiectare clase**

Pentru fiecare tabel din baza de date, avem o clasa care are ca field uri numele coloanelor din baza de date. Deci avem clasele : Client, Order, Product si OrderProduct. Client reprezinta tabelul client din baza de date si are field urile : id, name, address, email si age. Order are id si idClient. Product are id, name, price si stock. OrderProduct are id, orderId, productId si quantity. OrderProduct este o clasa create pentru a simula relatia many to many dintre produse si comenzi, aceasta comanda este folosita pentru ca pe o comanda sa poata fi mai multe produse. Am folosit MVC (model, view, controller) pentru a implementa interfata grafica. View-ul are componentele de java swing iar butoanele vor fi returnate ca mai apoi sa putem implementa ascultatori in controller. Clasa ConnectionFactory face legatura dintre aplicatie si baza de date utilizata, in cazul nostru MySQL.

* 1. **Interfete**

Pentru implementarea acestui proiect nu am folosit nicio interfata, deoarece nu am avut nevoie de ele. Toate clasele au metode diferite, chiar daca au acelasi nume, functionalitatea si paremetrii sunt diferiti.

* 1. **Relatii**

Asocierea reprezinta faptul ca obiectele unui lucru sunt conectate la obiecte ale altui lucru, dar nu exista proprietate. Dependența reprezinta faptul ca o clasa foloseste operatii din alta clasa sau foloseste variabile sau argumente din cealalta clasa. Intre business layer si model sunt multe relatii. Fiecare element din business layer are un element asociat in model. Pentru ClientBLL avem Client, pentru ProductBLL avem product si asa mai departe. Exista multe relatii de dependenta intre clase, pentru ca clasele sunt foarte bine conectate. Majoritatea claselor sunt dependente de clasa connection factory deoarece toate metodele care gestioneaza baza de date trebuie sa aiba un obiect de tipul clasei respective pe care sa se poata opera. Intre controller si restul claselor de asemenea sunt multe relatii de dependenta, deoarece sunt depozitate valori din toate clasele pentru ca acestea sa fie puse in tabele. Controller-ul este de asemenea dependent de View deoarece acesta implementeaza functionalitatea butoanelor.

* 1. **Pachete**

Dupa cum este cerut si in indicatia de munca avem cele 4 pachete obligatorii: dataAccesLayer, businessLaayer, model si presentation. De asemenea am adaugat si un pachet start care contine doar clasa main care porneste aplicatia. Data access layer-ul reprezinta conecxiunea dintre baza de date si aplicatie, business layer reprezinta partea administrativa a aplicatiei, aici se pot sterge, adauga sau updata date din baza de date, prin metodele definite. Modelul contine practice un model al fiecarui tabel, cu coloanele pe post de field-uri, deci in acest pachet avem o clasa pentru fiecare tabel: client, orders, product and orderproduct. Pachetul presentation contine interfata grafica care va fi prezentata mai jos in sectiunea 3.8.

* 1. **Interfata utilizator**

Interfata utilizator a fost construita cu ajutorul elementelor Java Swing (butoane, label-uri, textfield-uri, panel-uri). Interfata este usor de utilizat deoarece nu necesita cunostinte in programare pentru a putea fi utilizata. Datele de intrare pentru fiecare operatie (inserare, stergere, update) o sa fie introduse in textfield-uri care au in fata lor un label specific. Operatia va fi realizata apasand un buton denumit corespunzator. Au fost create 5 ferestre, una din care se va alege tabelul, si mai apoi una pentru fiecare tabel. In fiecare fereastra utilizata pentru tabele exista butoane specifice pentru operatii. In fereastra pentru orderproduct exista si un buton Generate care genereaza o chitanta pentru acea comanda.

1. **Implementare si testare**

Aplicatia a fost implementata in limbajul de programare Java, folosind IntelliJ. Pentru interfata grafica, butoanele, textfield-urile label-urile si toate componentele au fost adaugate prin cod, fara a folosi Window Builder sau alte plugin-uri de drag and drop. Am folosit ascultatori pentru toate componentele necesare, ca de exemplu click-ul de buton. Testarea este foarte diferita fata de proiectul anterior, unde am folosit Junit. Am folosit blocuri try-catch pentru prinde orice tip de eroare care ar putea aparea, ca de exemplu introducerea de date gresite, stergerea de produse sau comenzii daca exista OrderProduct cu ele. Un exemplu ar fi ca varsta trebuie sa fie un numar natural mai mare de 16, pretul si stoc ul trebuie sa fie numere naturale si asa mai departe. Una din cele mai dificile parti de inteles si implementat a fost utilizarea tehnicilor de reflexive, pentru a crea metode si clase abstracte care sa functioneze indiferent de tipul obiectului, adica indiferent daca e client, comanda sau produs. Aceste informatii nu sunt stiute pana la compilare.

1. **Rezultate**

Am reusit sa implementez aceasta aplicatie. Nu am folosit metode cu mai mult de 30 de linii si nici clase cu mai mult de 300. Am folosit interfata grafica cat si o perspectiva orientata pe obiect. Dupa multa munca si multe testari facute, am reusit sa creez o aplicatie pentru procesarea comenziilor penntru un depozit. Aceasta lasa utilizatorul sa introduca client si produse cu care mai apoi sa genereze comenzi. Asa cum ar fi mai aproape de realitate, pe o comanda se pot pune mai multe tipuri de produse atata timp cat este destul stoc disponibil. Am realizat si o documentatie de tip Javadoc in care am descris pe scurt parametrii unei metode, ce returneaza o metoda cat si o scurta descriere a metodelor. . . . . . . . . . . . . . . . . . . . . . . . . . . . .

1. **Concluzii**

Ca si in tema anterioare, cunostiintele noastre in accest domeniu nu au prea existat. Cunostiintele legate de donectarea bazei de date la aplicatie si gestionarea ei nu au prea existat, dar dupa ce am invatat si inteles cum functioneaza a devenit destul de usor de implementat, desi a fost nevoie de multa munca si testare. Un mare ajutor a fost exemplul dat in pdf, unde era deja facuta clasa Connection Factory, cat si implementata functia de find by id. Aceasta functie a fost un punct bun de start pentru restul metodelor care fac operatii legate de baza de date. Tehnicile de reflexie sunt destul de greu de inteles, deoarece marea parte a timpului nu stii nimic. In aceasta aplicatie, de exemplu cand cream un JTable, nu stim nimic despre field-urile tabelului si nici ce contine, nu stim cate coloane are tabelul si nici ce tipuri de date contine.

* 1. **Dezvoltari ulterioare**

Exista o multitudine de dezvoltari ulterioare al acestui proiect, precum adaugarea de noi field-uri tabelelor precum metoda de plata pentru comanda sau numarul de telefon al clientului. O alta dezvoltare ar fi adaugarea de noi tabele, de exemplu pentru a se stii starea comenzii. In acest moment utilizatorul este atat manager-ul depozitului cat si client. O dezvoltare ar fi diferentierea acestor roluri, creare de conturi separate atat pentru client, cat si pentru manager, fiecare avand permisiuni diferite, clientul sa poata sa comande, dar nu sa stearga produse, iar manager-ul sa se ocupe doar de gestionarea produselor si a comenzilor.

1. **Bibliografie**

Probleme de sintaxa

<http://stackoverflow.com/>

Tehnici de reflexie

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

Informatii legate de ArrayList

<https://docs.oracle.com/javase/8/docs/api/java/util/ArrayList.html>