**Tehnici de programare**

**Tema 3:**

**Order Management**

Student: Pop Tudor Andrei

Grupa:20229

Data: 21.03.2019

Cuprins:

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

**Objective**

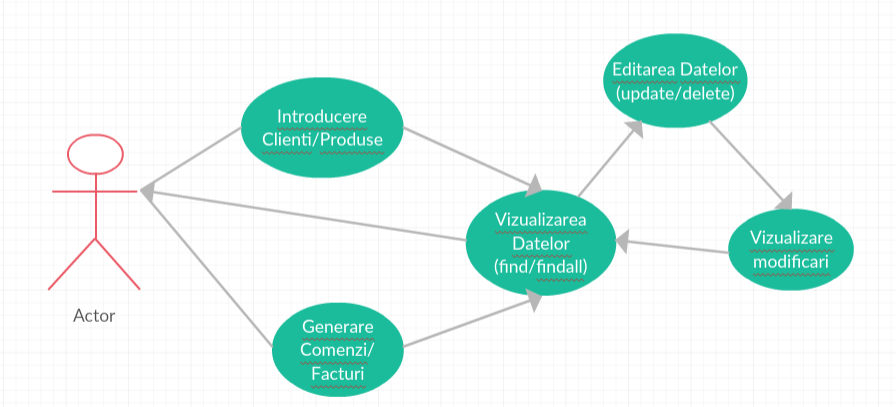
Consider an application OrderManagement for processing customer orders for a warehouse. Relational databases are used to store the products, the clients and the orders.

**Description**

Consider an application OrderManagement for processing customer orders for a warehouse. Relational databases are used to store the products, the clients and the orders. Furthermore, the application uses (minimally) the following classes: • Model classes - represent the data models of the application • Business Logic classes - contain the application logic • Presentation classes – classes that contain the graphical user interface • Data access classes - classes that contain the access to the database Other classes and packages can be added to implement the full functionality of the application. a. Analyze the application domain, determine the structure and behavior of its classes and draw an extended UML class diagram. b. Implement the application classes. Use javadoc for documenting classes. c. Use reflection techniques to create a method createTable that receives a list of objects and generates the header of the table by extracting through reflection the object properties and then populates the table with the values of the elements from the list: JTable createTable(List

• Graphical interface: o Window for client operations: add new client, edit client, delete client, view all clients in a table (JTable) o Window for product operations: add new product, edit product, delete product, view all product in a table (JTable) • Create a product order for a client: the application user will be able to select an existing product, select an existing client, and insert a desired quantity for the product to create a valid order. In case that there are not enough products, an under stock message will be displayed. After the order is finalized, the product stock is decremented. • Use relational databases for storing the data for the application, minimum three tables: Client, Product and Order. • Use reflection techniques to create a method that receives a list of objects and generates the header of the table by extracting through reflection the object properties and then populates the table with the values of the elements from the list • Documentation

2. Analiza problemei, modelare, scenarii, cazuri de utilizare

UML Case Diagram: 

Actorul – reprezinta utilizatorul, acesta introduce datele de intrare pentru simularea scenariului

Introducere client/produse – actorul va complete formularele care vor insera in baza de date noii client/produse doar daca datele sunt valide si daca nu incalca constrangerile prezente in baza de date. De exemplu foreinkey/primary key etc;

Generare comenzi/Facturi – in aceasta parte a scenariului actorul poate genera comenzi noi, respectiv pe baza comenzilor existente sau abia create poate genera facturi. Acestea sunt generate prin completarea unui formular pt comenzi sau doar a selectarii comenzii in cazul facturilor. Asemenea campului anterior, datele sunt verificate sa respecte contrangerile de la baza de date.

Vizualizarea datelor –actorul poate accesa unele din cele 2 comenzi disponibile pentru a selecta si vizualiza datele existente deja sau chiar introduse. Aditional cand o facture se genereaza se va genera intr-un fisier detaliile despre aceasta. Acest fisier va contine doar ultima factura generata. In principiu daca se incearca vizualizarea unor date inexistente, ca aparea un mesaj de eroare.

Editare datelor – actorul are la dispozitie doua comenzi prin care poate actualize/sterge date din baza de date. Update si Delete. Comanda update va deschide o fereastra in care se cere identificarea obiectului dorit si de asemenea campurile al caror valori sa fie schimbate. Modificare unor date poate schimba si va schimba datele din alte tabele datorita constrangerilor aplciate asupa tabelelor. Pentru stergere se va selecta id-ul din tabela dorita si daca exista se va sterge. Atentie!! Similar actializarii, stergerea datelor de care depend alte tabele, va duce la stergerea datelor care folosesc acel camp. Efecetul este ireversibil deoarece nu exita backup pana acum.

Vizualizarea modificari –actorul poate accesa unele din cele 2 comenzi disponibile pentru a selecta si vizualiza datele existente deja sau chiar introduse. Aditional cand o facture se genereaza se va genera intr-un fisier detaliile despre aceasta. Acest fisier va contine doar ultima factura generata. In principiu daca se incearca vizualizarea unor date inexistente, ca aparea un mesaj de eroare.

Un posibil mod de utilizare:

1. Se introduc doi client.
2. Se introduc doua produse
3. Se genereaza comenzile dorite
4. Se genereaza facuturile pentru comenzile dorite
5. Se vizualizeaza datele introduse/ existente
6. Se modifica datele stocate cu scopul de a observa functionalitatea bazei de date
7. Se observa efectele aduse de modificari
8. Se poate continua cu punctul 1), sau libera explorare a aplicatie.

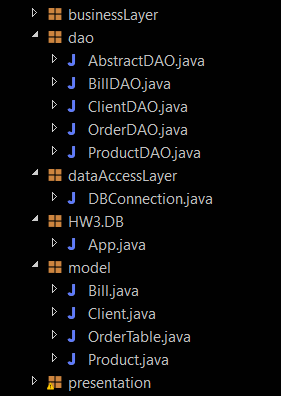
Campurile id trebuie sa fie unice pentru fiecare tabela (client, product etc) actualizarea sau introducerea datelor nu va avea loc. Fie apare popup-ul de eroare fie vor aparea in consola erori

Daca datele introduse sunt valide, acestea pot fi modificare/refolosite in alte operatii in limita perimisiunilor bazei de date.

Modificarea unor campuri din tabele poate duce si va duce la alterarea tabelelor care depend de valorize unor campuri. Simila stergerea unui rand poate duce la stergerea altor date din alte tabele de exemplu (stergere client -> stergere comanda -> stergere factura) .

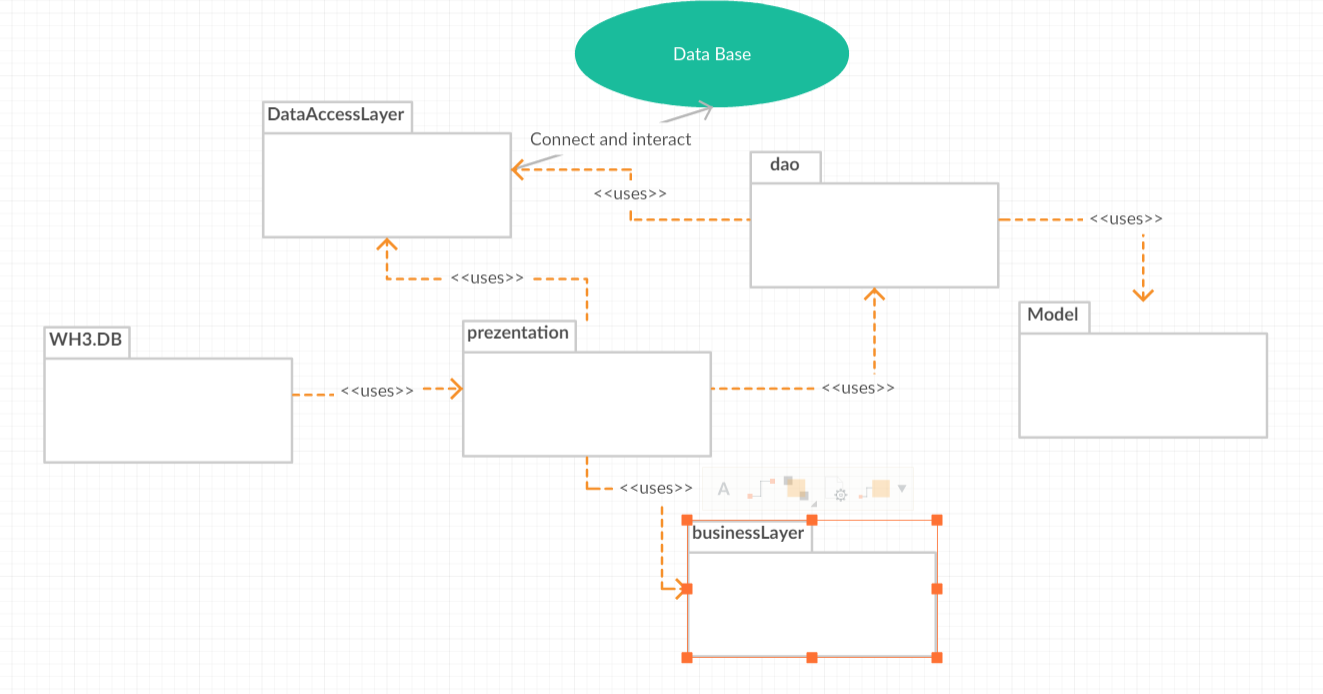
3. Proiectare (decizii de proiectare, diagrame UML, structuri de date, proiectare clase, interfete, relatii, packages, algoritmi, interfata utilizator)

Pentru proiectare respectat propunerea din barem: Layered Architecture (the application will contain at least four packages: dataAccessLayer, businessLayer, model and presentation), similar conditia Use relational databases for storing the data for the application, minimum three tables: Client, Product and Order.

****

Pachetul dao se ocupa cu mare parte a logicii aplicatiei, se poate spune ca are rol de businessLayer (but that’s none of my business)

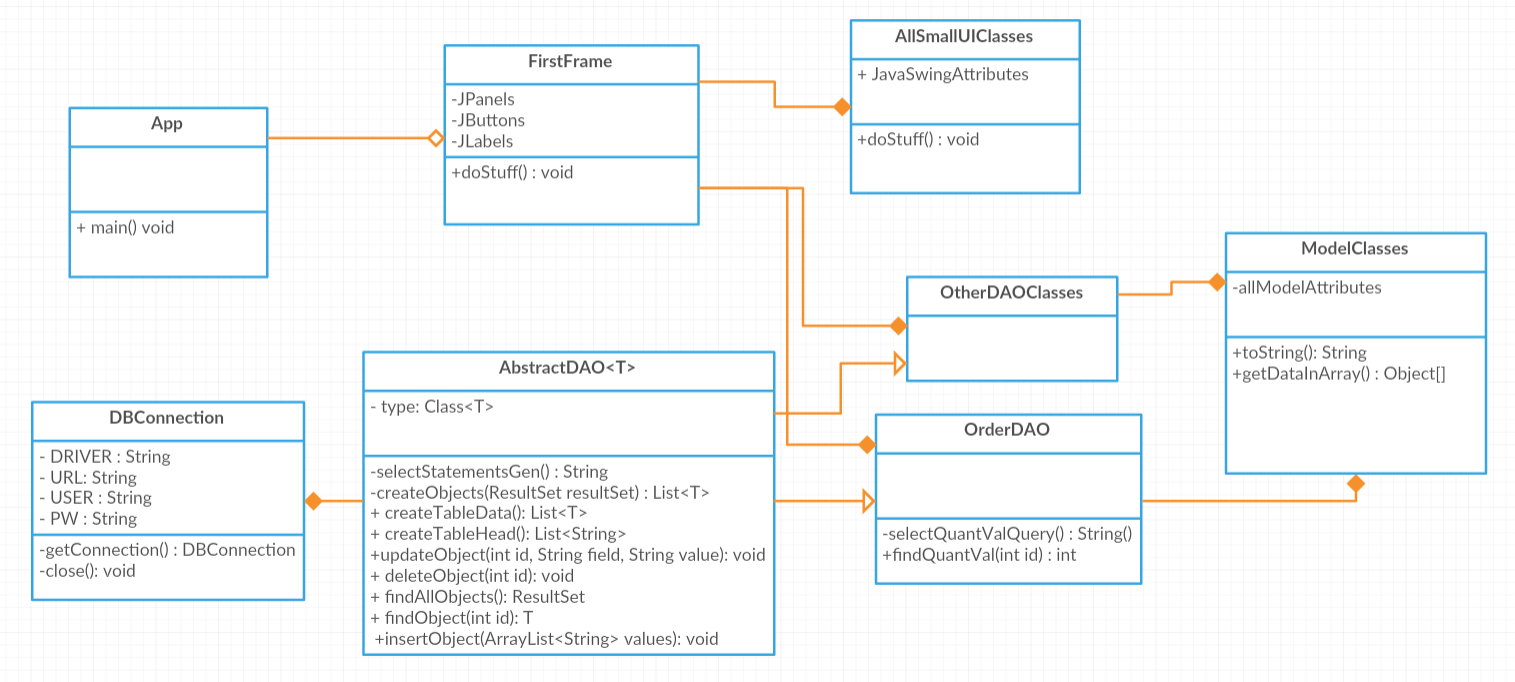
**Package Diagram:**



Pachetul WH3.DB contine o clasa ce implementeaza metoda main(), care porneste interfata grafica. Interfata grafica este centrul programului, deoarece seleteaza si foloseste direct sau indirect fiecare functionalitate a celorlalte pchete.

Pachetul dataAccessLayer asigura conexiunea cu baza de date pentru toata aplicatia conform exemplelor aduse la laborator. Prin intermediul claselor din pachetul dao, datele din baza de date pot fi manipulate, actualizara, introduse si sterse. Mare parte din logica aplicatiei este concentrate in clasele pachetului dao, respcetiv AbstractDAO.

**Class Diagram:**

****

Dupa cum se poate observa din ilustratia de mai sus, Toate clasele DAO mostenesc calasa AbstractDAO pentru a prelua metodele generice aduse de aceeasta, dar aceste clase sunt dependente prin compozitie de clasele model. Aceste clase model sunt reprezentarea tabelelor din bazei de date global la nivelul intregrii aplicatii

4. Implementare

Varianta acutala acestei aplicatii este rezultatul unui sir lung de incercari si esuari in care am incercat diferite metode, diferite abordari,variante asemanatoare cu cele prezentate la laborator ( poate chiar curs ) testate pe o perioada de 3 saptamani, recomandari de pe diferite site-uri cum ar fi StackOverflow/GeeksForGeeks, youtube si asa mai departe.

Codul poate contine diferite bucati de cod commentate, variabile nefolosite, metode neapelate sau care pur si simplu nu am nicio aplicabilitate toate din incercariile anterioare care pentru un motiv sau altul nu si-au gasit loc in variant “finala”

Clasa APP : in interiorul aceste clasa se afla metoda “public static void main(String[] args)” in interiorul acesteia declar un obiect din clasa “FirstWindow” si apelez cu acel obiect metoda “perform()”. Desigur, acest apel este inconjurat de o structura try/catch pentru a intercepta eventualele exceptii ce inevitabil pot aparea (Fara try/catch deoarece aceaesta structura isi are locul acum in “FirstWindow” )

Clasa FirstWindow: in interiorul acestei clasa construiesc efectiv interfata grafica si implementez clasa “ProjectListener”. Constructorul acestei clase da forma aplicatiei, un obiect ”JFram” care contine restul obiectelor intr-un “GridLayout”. Fereastra in sine are rol de a procura si valida parametrii care trebuie sa dicteze cum decurge rularea programului. O data cu apasarea butoanelor care reprezinta comenzile pe care userul de poate executa, se va construi o fereastra de de tipul respective care va fi specializata in executarea unui singur sql query cu anumite date introduse de utilizator.

Clasa JFram: Sunt specializare in executare unui singur querry dictat de datele introduse

Clasa AbstractDAO: reprezinta marea majoritatea a logicii aplicatiei care este transmisa mai departe claselor care o mostenesc. Este construita dupa modelul care se afla in resuresele din utcn\_dsrl. Am mai adaugat dupa nevoie alte metode, am modificat pe cele din exemplu pentru a se mula peste modul de proiectare regasit in aplicatie.

Metodele creatQuery:

private String createSelectQuery(String field) {

StringBuilder ret = new StringBuilder();

ret.append("SELECT ");

ret.append(" \* ");

ret.append(" FROM ");

ret.append( type.getSimpleName() );

ret.append(" WHERE " + field + " =?");

return ret.toString();

}

private String createSelectAllQuery()

private String createDeleteQuery(int rowId)

private String createUpdateQuery(int rowId, String field)}

private String createInsertQuery(int fieldsNumber)

Aceste metode creaza query-urie care vor fi executate in continuare de metodele care fac legatura dintre baza de date si functionalitate programului. Acestea sunt urmatoarele:

public void insertObject(ArrayList<String> values)

public T findObject(int id)

public ResultSet findAllObjects()

public void deleteObject(int id)

public void updateObject(int id, String field, String value)

functionalitatea acestora poate fi dedusa foarte usor din numele lor: insert: insereaza in baza de date, find : caut din baza de date dupa un id dat, findAll: gaseste toate datele unui table, delete: sterge din baza de date dupa un anumit id. updateObj: va actualize un singur camp la un moment dat cu o valoare de pe campurile care trebuie completate in fereastra corespunzatoare acestei functionalitati.

Metoda private List<T> createObjects(ResultSet resultSet) {

List<T> list = new ArrayList<T>();

try {

while (resultSet.next()) {

@SuppressWarnings("deprecation")

T instance = type.~~newInstance~~();

for (Field field : type.getDeclaredFields()) {

field.setAccessible(true);

Object value = resultSet.getObject(field.getName());

PropertyDescriptor propertyDescriptor = new PropertyDescriptor(field.getName(), type);

Method method = propertyDescriptor.getWriteMethod();

System.out.println(field.toString());

method.invoke(instance, value);

}

list.add(instance);

}

} catch(){

}

Aceasta metoda folosete tehnica reflection pentru a ganera obiectele echivalente din aplicatie cu randurile tabelelor bazei de date. Ea sta la baza inca unei metode care obtine numele campurilor claselor pentru generearea coloanelor tabelului care apare la obtiunea findAll.

Clasa OrderDAO: Mosteneste clasa AbstractDAO, iar in plus fata de aceasta si fata de restul claselor dao, mai are cateva metode specifice care au rol in a updata tabela product de fiecare deta cand o noua comanda se plaseaza.

Implementeaza metodele:

private String selectQuantValQuery()

public int findQuantVal(int id)

Clasa public class CreateBillWin extends JFrame: mentionez aceasta clasa pentru a castiga cuvinte in plus la documentatie si pentru ca este relevanta deoarece pentru generarea facturii se face pe baza id-ului unei comenzi. Contructorul aceste clasa va genera tabelul (head + data ) folosind tehnica reflectiei conform cerintei minime. Constructorul executa un select statement pe fiecare tabela a bazei de date, obtinand astfel automat datele necesare completarii campurilor noilor facturi, iar de la user foloseste doar id-ul comenzii la care se doreste generarea facturii.

Tot in aceasta clasa se genereaza fisierul txt care reprezinta ultima factura generate de aplicatie

FileWriter wr = new FileWriter("LastEmittedBill.txt");

wr.write(bd.findObject(order.getOrdertableId()).toString();

wr.flush();

wr.close();

5. Rezultate

Conform asteptariilor, programul functioneaza bine, afisaza rezultatele dorite, ferestrele de erori apar cand trebuie, conexiunea cu baza de date functioneaza.

Datele pot fi introduse, pot fi vizualizate, modificate, sterse si utilizare de celetalte tabele ale bazei de date.

In anumite situatii se mai propaga excepti al caror stacktrace este afisat in consola, de asemenea apar anumite mesaje in consola care au un rol de debugging si nimic mai mult. Dar din lipsa timpului nu am mai apucat sa le elimin. Nu afecteaza functionalitatea programului.

6. Concluzii

- documentatia s-a putut scrie in ultima zi (\*noapte 2:15 AM, Test la EGC imediat dupa lab-ul de PT (y) )

- proiectul se poate inbunatatii la fiecare pas: optimizare, design, implementarea operatiilor pe mai multe threaduri, oprirea threadurilor trebuie revizuita

- codul contine multe ramasite de la incercarile anterioare, incluziv clase care nu au niciun impact asupra logicii in sine, dat daca sunt eliminate apar exceptii, iar programul refuza sa functionze

-am invatat multe despre cum functioneaza functia reflection-ul, desi nu la un nivel la care m-as putea descurca fara exemplul pus de dumneavoastra ca material ajutator

- reflection-ul este foarte foarte sensibil la numele metodelor, la tipurile de date, la contructorii care apar in clase, va arunca exceptii din motive neaspatate (raportat la nivelul me de cunostinte despre tehnica). Aceste exceptii si rezolvarea lor au fost foarte frustrante si au luat un timp considerabul de implementat.

7. Bibliografie

1) <https://en.wikipedia.org/wiki/Polynomial_long_divisio> -- pentru impartire

2) <https://creately.com/diagram-type/use-case> -- pentru diagrame

3) <https://stackoverflow.com/> -- pentru restul problemelor

4) Toate linkurile care apar in materialul ajutator