Order management

**1.Obiectivul temei**

Obiectivul principal al temei este proiectarea si implementarea unei aplicatii care sa administreze o baza de date printr-o interfata grafica dedicata. Aceasta aplicatie va furniza utilizatorului optiuni de manipulare a datelor din tabele (delete, add, update).

Sub-obiective:

* Analiza problemei si identificarea necesitatilor
* Proiectarea aplicatiei
* Implementarea aplicatiei
* Testarea aplicatiei

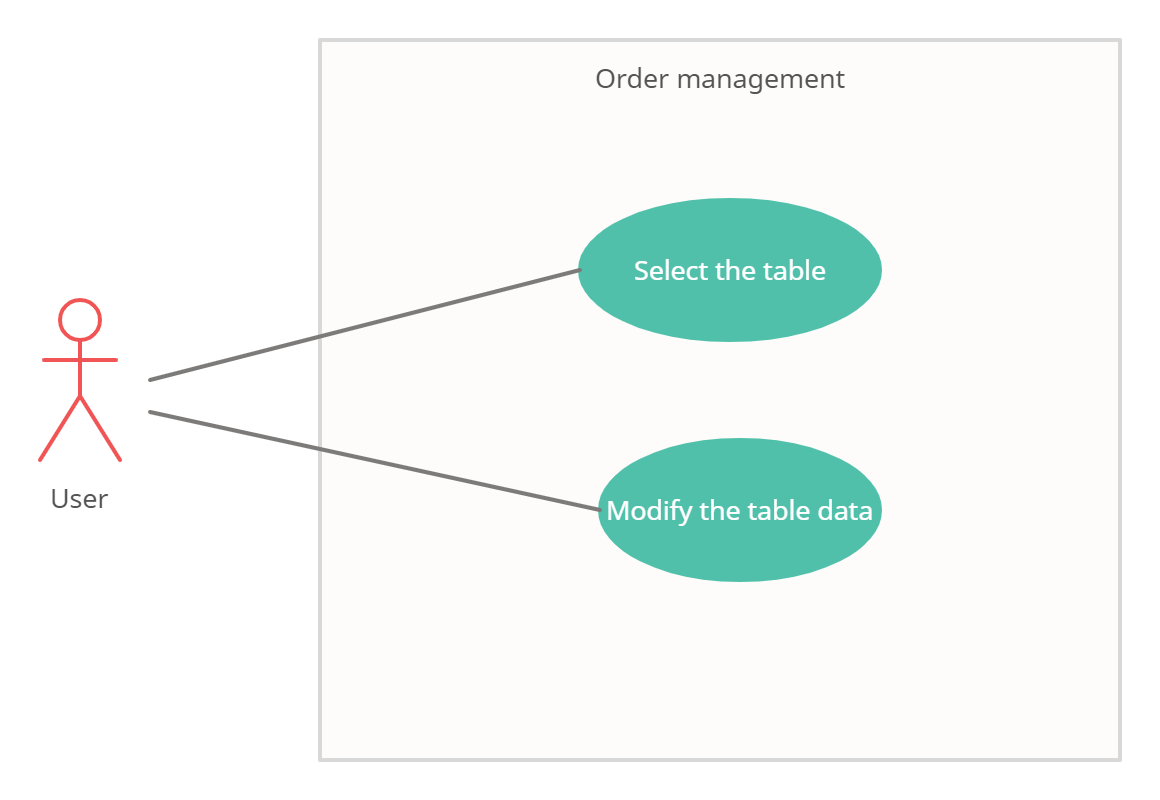
**2. Analiza problemei, modelare, scenariu, cazuri de utilizare**

Aplicatia va trebui sa creeze o conexiune cu o baza de date (MySQL).

Pentru fiecare tabel din baza de date respectiva se vor crea clase omonime in aplicatie cu exact aceleasi atribute (cu aceleasi dimensiuni). Astfel, tabelele vor putea fi afisate in interfata grafica dedicata, unde utilizatorul va putea face modificari intr-un mod simplist si intuitiv asupra bazei de date prin comenzi de add, delete si update. Baza de date din serverul mySQL va contine trei tabele: Client, Product si Order. In tabelul Client vom gasi toti clientii, fiecare dintre ei avand un id unic (primary key) si date de contact: numele (name) si adresa (address). In tabelul Product se vor gasi toate produsele aflate in stoc, identificate unic dupa un id (primary key) si continand detaliile necesare pentru clientii interesati de achizitionarea acestora: numele (name), pretul (price) si numarul de produse aflate in stoc (stock).

Principalele proprietati ale aplicatiei sunt:

* Singurul actor este utilizatorul
* Acesta beneficiaza de functionalitatile aplicatiei prin interfata grafica dedicata
* Utilizatorul va putea introduce de la tastatura imputurile necesare pentru a modifica datele din tabele (add, update) si poate selecta datele pe care doreste sa le stearga (delete)
* Tabelele vor aparea in interfata grafica ca o reprezentare fidela a tabelelor din serverul MySQL
* Pentru fiecare tabel in parte va exista o vedere in interfata grafica de unde se vor putea face modificari



Caz de utilizare:

Actor principal: utilizatorul

Scenariu:

1. Utilizatorul alege tabelul pe care doreste sa-l vizualizeze
2. Daca utilizatorul doreste sa insereze date, acesta va trebui sa le furnizeze de la tastatura in campurile specific tabelului iar apoi sa apese butonul de add pentru a finaliza inserarea
3. Daca utilizatorul doreste sa modifice datele deja existente, acesta va selecta randul restectiv iar datele vor aparea in campurile interfetei grafice de unde se vor putea modifica. Pentru finalizare, utilizatorul va apasa butonul upate.
4. Daca utilizatorul doreste sa stearga un anumit rand, acesta va selecta randul respectiv si va apasa butonul delete

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

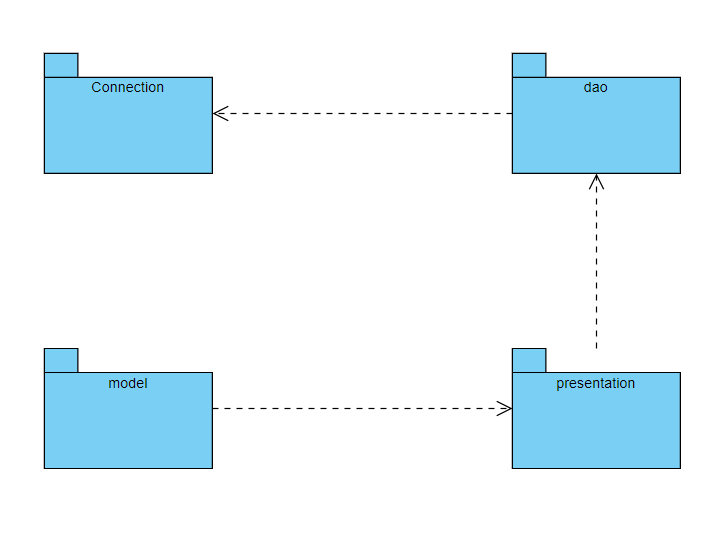
Proiectul va fi impartit in 4 pachete: connection, dao, model si presentation.

In pachetul connection se va realiza conexiunea proriu-zisa la baza de date MySQL.

In pachetul dao se vor realiza metodele pentru manipularea datelor din tabele.

In pachetul model se vor defini clase omonime cu numele tabelelor din serverul MySQL care vor contine atribute similare cu tipul datelor din tabele.

In pachetul presentation va fi realizata interfata grafica a aplicatiei prin care utilizatorul va putea face modificari asupra datelor din tabele.



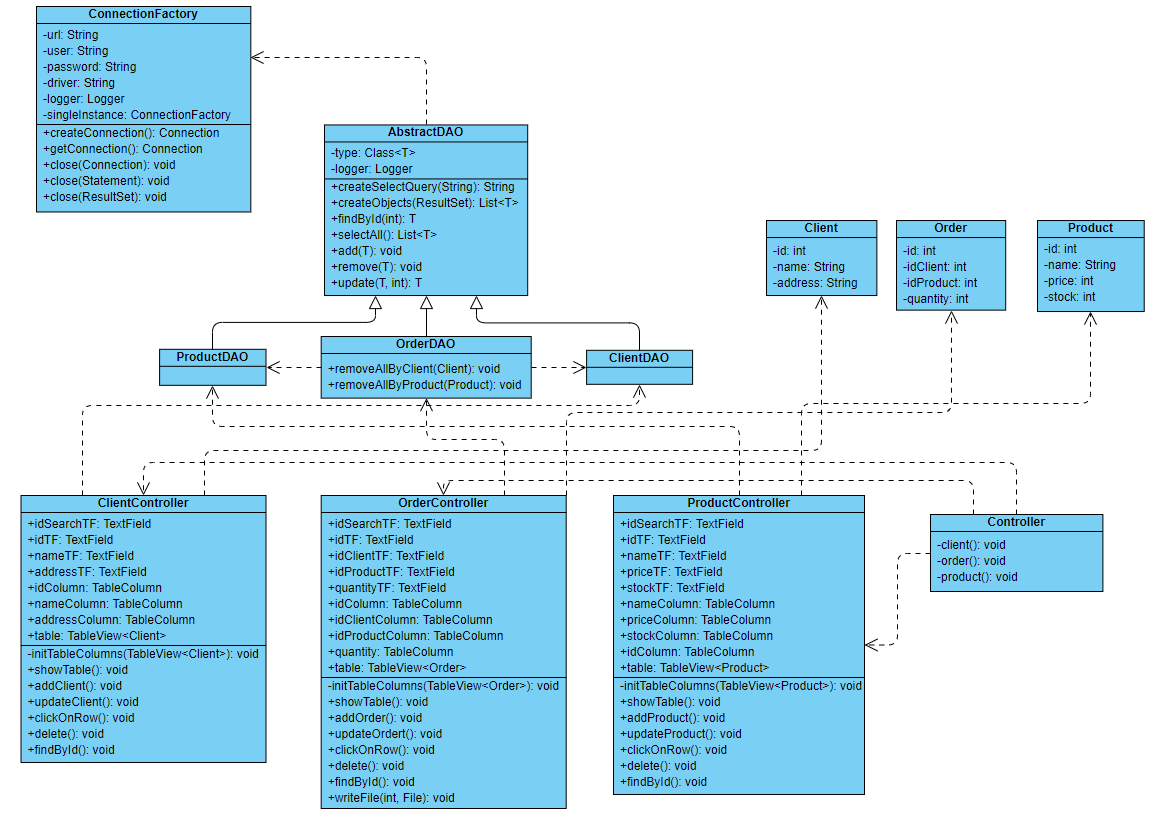
In primul rand, aplicatia trebuie sa creeze o legatura cu baza de date. Aceasta legatura se va face in pachetul connection in clasa ConnectionFactory.

In al doilea rand, trebuie create clase care vor fi asociate cu tabelele din baza de date. Astfel, in pachetul model se vor crea trei clase omonime cu tabelele existente si avand aceleasi atribute: Client, Product si Order.

Pe urma, in pachetul dao, se vor implementa functionalitatile propriu-zise ale aplicatiei. Aceasta impelmentare va fi una generalizata prin tehnica reflectarii astfel incat ea va putea fi folosita de catre orice clasa reprezentativa a unui tabel din baza de date. Astfel, clasa AbstractDAO va contine functionalitatile aplicatiei (metode pentru inserarea, actualizarea, stergerea etc. datelor din tabele), iar clasele ClientDAO, ProductDAO si OrderDAO vor extinde aceasta clasa particularizand functionalitatile pentru fiecare tabel in parte.

In final, in pachetul presentation se va gasi impelmentarea interfetei grafice.

Diagrama UML pentru clase:



**4. Implementare**

***Clasa ConnectionFactory:***

Ca atribute, aceasta clasa contine un Logger folosit pentru cazul in care apar exceptii si patru String-uri (URL, USER, PASSWORD, DRIVER) care ne ajuta sa ne conectam la baza de date.

private static final Logger *LOGGER* = Logger.*getLogger(*ConnectionFactory.class.getName*())*;  
 private final String URL = "jdbc:mysql://localhost:3306/order\_management?useSSL=false";  
 private final String USER = "root";  
 private final String PASSWORD = "1234";  
 private final String DRIVER = "ConnectionFactory";

Metoda createConnection() va fi folosita pentru crearea unei conexiuni cu baza de date, folosind atributele clasei.

private Connection createConnection*()* throws SQLException*{* Connection connection;  
 try *{* connection = DriverManager.*getConnection(*URL, USER, PASSWORD*)*;  
  
 return connection;  
 *}* catch *(*SQLException e*) {  
 LOGGER*.log*(*Level.*WARNING*, "An error occurred while trying to connect to the database"*)*;  
 e.printStackTrace*()*;  
 *}* return null;  
 *}*

Metoda close() va fi folosita pentru inchiderea conexiunii dupa ce s-a executat operatia dorita pe tabele.

public static void close*(*Connection connection*) {* if *(*connection != null*) {* try *{* connection.close*()*;  
 *}* catch *(*SQLException e*) {  
 LOGGER*.log*(*Level.*WARNING*, "An error occurred while trying to close the connection"*)*;  
 *}  
 }  
 }*

Metoda close() mai este folosita (suprascrisa) pentru a inchide un statement SQL.

public static void close*(*Statement statement*) {* if *(*statement != null*) {* try *{* statement.close*()*;  
 *}* catch *(*SQLException e*) {  
 LOGGER*.log*(*Level.*WARNING*, "An error occurred while trying to close the statement"*)*;  
 *}  
 }  
 }*

Metoda close() mai este folosita (suprascrisa) pentru a inchide un ResultSet primit dupa efectuarea query-ului SQL.

public static void close*(*ResultSet resultSet*) {* if *(*resultSet != null*) {* try *{* resultSet.close*()*;  
 *}* catch *(*SQLException e*) {  
 LOGGER*.log*(*Level.*WARNING*, "An error occurred while trying to close the ResultSet"*)*;  
 *}  
 }  
 }*

***Clasa AbstractDAO:***

Ca atribute, aceasta clasa contine un Logger folosit pentru cazul in care apar exceptii si un atribut Class<T> type pentru a particulariza functionalitatiile acestei clase altor clase (ClientDAO, OrderDAO, ProductDAO).

private static final Logger *LOGGER* = Logger.*getLogger(*AbstractDAO.class.getName*())*;  
 private final Class*<*T*>* type;

Metoda createSelectQuery() creeaza un String ce contine comanda SQL pentru selectarea elementelor dintr-un tabel anume. Aceasta metoda va fi folosita in metoda findById pentru a crea comanda SQL.

private String createSelectQuery*(*String field*)  
 {* StringBuilder sb =new StringBuilder*()*;  
 sb.append*(*"SELECT \* FROM "*)*;  
 sb.append*(*"order\_management."*)*;  
 sb.append*(*type.getSimpleName*()*.toLowerCase*())*;  
 sb.append*(*" WHERE "*)*.append*(*field*)*.append*(*" = ?"*)*;  
 return sb.toString*()*;  
 *}*

Metoda createObjects() creeaza o lista de obiecte avand ca elemente rezultatele unei comenzi SQL.

private List*<*T*>* createObjects*(*ResultSet resultSet*)  
 {* List*<*T*>* list = new ArrayList*<*T*>()*;  
 try*{* while*(*resultSet.next*())  
 {* T instance = type.newInstance*()*;  
 for*(*Field field: type.getDeclaredFields*())  
 {* Object value = resultSet.getObject*(*field.getName*())*;  
 PropertyDescriptor propertyDescriptor = new PropertyDescriptor*(*field.getName*()*, type*)*;  
 Method method = propertyDescriptor.getWriteMethod*()*;  
 method.invoke*(*instance,value*)*;  
 *}* list.add*(*instance*)*;  
 *}  
 }* catch *(*SQLException | IllegalAccessException | InstantiationException | IntrospectionException | InvocationTargetException throwables*) {* throwables.printStackTrace*()*;  
 *}* return list;  
 *}*

Metoda findById() returneaza un “rand” dintr-un tabel care are id-ul egal cu cel transmis ca parametru.

public T findById*(*int id*)  
 {* Connection connection = null;  
 PreparedStatement statement = null;  
 ResultSet resultSet = null;  
 String query = createSelectQuery*(*"id"*)*;  
 System.*out*.println*(*query*)*;  
 try*{* connection = ConnectionFactory.*getConnection()*;  
 statement = connection.prepareStatement*(*query*)*;  
 statement.setInt*(*1,id*)*;  
 resultSet = statement.executeQuery*()*;  
   
 return createObjects*(*resultSet*)*.get*(*0*)*;  
 *}* catch *(*SQLException e*) {  
 LOGGER*.log*(*Level.*WARNING*, "dao/AbstractDAO: " + e.getMessage*())*;  
 *}*finally *{* ConnectionFactory.*close(*resultSet*)*;  
 ConnectionFactory.*close(*statement*)*;  
 ConnectionFactory.*close(*connection*)*;  
 *}* return null;  
 *}*

Metoda selectAll() returneaza o lista cu toate elementele unui tabel. Aceasta metoda va fi folosita pentru a afisa tabelele in interfata grafica.

public List*<*T*>* selectAll*() {* Connection connection = null;  
 PreparedStatement statement = null;  
 ResultSet resultSet = null;  
 String query = "SELECT \* FROM order\_management." + type.getSimpleName*()*.toLowerCase*()* + ";";  
 System.*out*.println*(*query*)*;  
  
 try *{* connection = ConnectionFactory.*getConnection()*;  
 statement = connection.prepareStatement*(*query*)*;  
 resultSet = statement.executeQuery*()*;  
  
 return createObjects*(*resultSet*)*;  
 *}* catch *(*SQLException e*) {  
 LOGGER*.log*(*Level.*WARNING*, "dao/AbstractDAO: " + e.getMessage*())*;  
 *}* finally *{* ConnectionFactory.*close(*resultSet*)*;  
 ConnectionFactory.*close(*statement*)*;  
 ConnectionFactory.*close(*connection*)*;  
 *}* return null;  
 *}*

Metoda add() adauga in tabelul dorit ur rand nou. Elementele adaugate vor fi date aplicatiei prin interfata grafica.

public void add*(*T t*) {* Connection connection = null;  
 PreparedStatement statement = null;  
 ResultSet resultSet = null;  
  
 //INSERT INTO `order\_management`.`client` (`id`, `name`, `address`) VALUES ('7', 'g', 'ga');  
  
 String s = "INSERT INTO " + "`" + "order\_management" + "`" + "." + "`" + type.getSimpleName*()*.toLowerCase*()* + "`" + " (";  
  
 try *{* Class cls = Class.*forName(*"sample.model." + type.getSimpleName*())*;  
 Field*[]* fields = cls.getDeclaredFields*()*;  
 int i=0;  
 while*(*i < fields.length*) {* fields*[*i*]*.setAccessible*(*true*)*;  
 s = s + "`" + fields*[*i*]*.getName*()* + "`";  
 if *(*i < fields.length - 1*) {* s = s + ", ";  
  
 *}* else *{* s = s + ") ";  
 *}* i++;  
 *}* s = s + "VALUES ( ";  
  
 i=0;  
 while*(*i < fields.length*) {* fields*[*i*]*.setAccessible*(*true*)*;  
 try *{* s = s + "\'" + fields*[*i*]*.get*(*t*)* + "\'";  
 *}* catch *(*IllegalAccessException e*) {* e.printStackTrace*()*;  
 *}* if *(*i < fields.length - 1*) {* s = s + ", ";  
  
 *}* else *{* s = s + ") ";  
 *}* i++;  
 *}* s = s + ';';  
  
 *}* catch *(*ClassNotFoundException e*) {* e.printStackTrace*()*;  
  
 *}* try *{* System.*out*.println*(*s*)*;  
  
 connection = ConnectionFactory.*getConnection()*;  
 statement = connection.prepareStatement*(*s*)*;  
 statement.execute*()*;  
  
 *}* catch *(*SQLException e*) {  
 LOGGER*.log*(*Level.*WARNING*, "dao/AbstractDAO: " + e.getMessage*())*;  
 *}* finally *{* ConnectionFactory.*close(*resultSet*)*;  
 ConnectionFactory.*close(*statement*)*;  
 ConnectionFactory.*close(*connection*)*;  
 *}  
 }*

Metoda remove() va fi folosita penru a sterge din tabel un rand selectat din interfata grafica.

public void remove*(*T t*) {* Connection connection = null;  
 PreparedStatement statement = null;  
 ResultSet resultSet = null;  
  
 String s = "DELETE FROM order\_management." + type.getSimpleName*()*.toLowerCase*()* + " WHERE `id` = ";  
 try *{* Field field = Class.*forName(*"sample.model." + type.getSimpleName*())*.getDeclaredField*(*"id"*)*;  
 field.setAccessible*(*true*)*;  
 s = s +"\'" + field.get*(*t*)*+"\'";  
 *}* catch *(*NoSuchFieldException | IllegalAccessException | ClassNotFoundException e*) {* e.printStackTrace*()*;  
 *}* try *{* System.*out*.println*(*s*)*;  
  
 connection = ConnectionFactory.*getConnection()*;  
 statement = connection.prepareStatement*(*s*)*;  
 statement.execute*()*;  
  
 *}* catch *(*SQLException e*) {  
 LOGGER*.log*(*Level.*WARNING*, "dao/AbstractDAO: " + e.getMessage*())*;  
 *}* finally *{* ConnectionFactory.*close(*resultSet*)*;  
 ConnectionFactory.*close(*statement*)*;  
 ConnectionFactory.*close(*connection*)*;  
 *}  
 }*

Metoda update() va fi folosita pentru a actualiza un rand selectat din interfata grafica.

public T update*(*T t, int id*)* throws ClassNotFoundException, IllegalAccessException *{* Connection connection = null;  
 PreparedStatement statement = null;  
 ResultSet resultSet = null;  
  
 String s="UPDATE order\_management." + type.getSimpleName*()*.toLowerCase*()* + " SET ";  
 Class cls2 = Class.*forName(*"sample.model." + type.getSimpleName*())*;  
 Field*[]* fields = cls2.getDeclaredFields*()*;  
 int i=0;  
 while*(*i < fields.length*) {* fields*[*i*]*.setAccessible*(*true*)*;  
 if*(*i<fields.length-1*)* s = s + "`" + fields*[*i*]*.getName*()* + "` = '" +fields*[*i*]*.get*(*t*)*+ "', ";  
 else  
 s = s + "`" + fields*[*i*]*.getName*()* + "` = '" +fields*[*i*]*.get*(*t*)*+ "'";  
  
 i++;  
 *}* s = s + " WHERE *(*`id` = '" + id + "'*)*; ";  
  
 try *{* System.*out*.println*(*s*)*;  
  
 connection = ConnectionFactory.*getConnection()*;  
 statement = connection.prepareStatement*(*s*)*;  
  
 statement.execute*()*;  
  
 *}* catch *(*SQLException e*) {  
 LOGGER*.log*(*Level.*WARNING*, "dao/AbstractDAO: " + e.getMessage*())*;  
 *}* finally *{* ConnectionFactory.*close(*resultSet*)*;  
 ConnectionFactory.*close(*statement*)*;  
 ConnectionFactory.*close(*connection*)*;  
 *}* return t;  
 *}  
  
}*

***Clasa OrdertDAO:***

Desi clasele ClientDAO, ProductDAO si OrderDAO sunt generalizate in clasa AbstractDAO, vom avea nevoie de inca doua functionalitati pentru clasa OrderDAO.

Metoda removeAllByClient() va cauta in tabelul Order toate randurile specifice unui anumit client si le va sterge. Aceasta functionalitate este necesare atunci cand dorim sa stergem un client din baza de date (daca un client este sters atunci toate datele despre el trebuie sa dispara). Pe de alta parte aceasta este si o conditionare a legaturii primary key – foreign key.

public void removeAllByClient*(*Client client*)  
 {* List*<*Order*>* ordersList = this.selectAll*()*;  
 for *(*Order order: ordersList*) {* if*(*client.getId*()* == order.getIdClient*())* remove*(*order*)*;  
 *}  
 }*

Acelasi lucru se aplica si pentru tabelul Product.

public void removeAllByProduct*(*Product product*)  
 {* List*<*Order*>* ordersList = this.selectAll*()*;  
 for *(*Order order: ordersList*) {* if*(*product.getId*()* == order.getIdProduct*())* remove*(*order*)*;  
 *}  
 }*

***Clasa ClientController:***

Aceasta clasa are ca atribute elementele necesare pentru afisarea interfetei grafice: TextField-uri, un TableView<Client> si TableColum-uri.

public TextField idSearchTF;  
 public TextField idTF;  
 public TextField nameTF;  
 public TextField addressTF;  
 public TableColumn idColumn;  
 public TableView*<*Client*>* table;  
 public TableColumn nameColumn;  
 public TableColumn addressColumn;

Metoda initTableColums() creeaza cloanele necesare tabelului Client.

private void initTableColumns*(*TableView*<*Client*>* tableView*)  
 {* idColumn = new TableColumn*<>(*"Id"*)*;  
 idColumn.setMinWidth*(*60*)*;  
 idColumn.setCellValueFactory*(*new PropertyValueFactory*<>(*"id"*))*;  
  
 nameColumn = new TableColumn*<>(*"Name"*)*;  
 nameColumn.setMinWidth*(*100*)*;  
 nameColumn.setCellValueFactory*(*new PropertyValueFactory*<>(*"name"*))*;  
  
 addressColumn = new TableColumn*<>(*"Address"*)*;  
 addressColumn.setMinWidth*(*100*)*;  
 addressColumn.setCellValueFactory*(*new PropertyValueFactory*<>(*"address"*))*;  
  
 tableView.getColumns*()*.addAll*(*idColumn,nameColumn,addressColumn*)*;  
 *}*

Metoda showTable() afiseaza toate datele din tabelul din baza de date in tabelul din interfata grafica. Aceasta metoda foloseste metoda selectAll() din AbstractDAO pentru a face rost de datele din tabel.

public void showTable*()  
 {* table.getItems*()*.clear*()*;  
 table.getColumns*()*.clear*()*;  
 table.setEditable*(*true*)*;  
 initTableColumns*(*table*)*;  
  
 List*<*Client*>* clients = clientDAO.selectAll*()*;  
 final ObservableList*<*Client*>* data = FXCollections.*observableArrayList(*clients*)*;  
 table.setItems*(*data*)*;  
 *}*

Metoda addClient() adauga un nou client bazat pe datele introduse de utilizator in campurile din interfata grafica. Aceasta metoda foloseste functionalitatea metodei add() din AbstractDAO pentru a adauga un nou client in tabel.

public void addClient*()  
 {* int id = Integer.*parseInt(*idTF.getText*())*;  
 String name = nameTF.getText*()*;  
 String address = addressTF.getText*()*;  
  
 Client client = new Client*()*;  
 client.setId*(*id*)*;  
 client.setName*(*name*)*;  
 client.setAddress*(*address*)*;  
  
 clientDAO.add*(*client*)*;  
 showTable*()*;  
 *}*

Metoda updateClient() actualizeaza un client selectat din interfata grafica. Aceasta metoda foloseste functionalitatea metodei update() din AbstractDAO pentru a actualiza un client in tabel.

public void updateClient*()* throws IllegalAccessException, ClassNotFoundException *{* int id = Integer.*parseInt(*idTF.getText*())*;  
 String name = nameTF.getText*()*;  
 String address = addressTF.getText*()*;  
  
 Client client = new Client*()*;  
 client.setId*(*id*)*;  
 client.setName*(*name*)*;  
 client.setAddress*(*address*)*;  
  
 clientDAO.update*(*client, id*)*;  
 showTable*()*;  
 *}*

Metoda clickOnRow() introduce datele unui rand selectat din interfata grafica in TextField-uri.

public void clickOnRow*()  
 {* Client client = table.getSelectionModel*()*.getSelectedItem*()*;  
 idTF.setText*(*Integer.*toString(*client.getId*()))*;  
 nameTF.setText*(*client.getName*())*;  
 addressTF.setText*(*client.getAddress*())*;  
 *}*

Metoda delete() sterge un client selectat din interfata grafica. Aceasta metoda foloseste functionalitatea metodei remove() din clasa AbstractDao.

public void delete*() {* Client client = table.getSelectionModel*()*.getSelectedItem*()*;  
  
 OrderDAO orderDAO = new OrderDAO*()*;  
 orderDAO.removeAllByClient*(*client*)*;  
 clientDAO.remove*(*client*)*;  
 showTable*()*;  
 *}*

Metoda findById() foloseste metoda omonima din clasa AbstractDAO pentru a gasi un anumit client. Id-ul clientului care se doreste a fi gasit este introdus in TextField-ul aferent si apoi se apasa butonul respectiv pentru a afisa rezultatul in tabel.

public void findById*() {* int id = Integer.*parseInt(*idSearchTF.getText*())*;  
 Client client = clientDAO.findById*(*id*)*;  
 table.getItems*()*.clear*()*;  
  
 List*<*Client*>* clientsList = new ArrayList*<>()*;  
 clientsList.add*(*client*)*;  
 final ObservableList*<*Client*>* data = FXCollections.*observableArrayList(*clientsList*)*;  
 table.setItems*(*data*)*;  
 *}*

***Clasa OrderController:***

Metoda printSelectedOrder() creeaza un fisier .txt cu numele format din id-ul comenzii si id-ul clientului care a comandat-o.

public void printSelectedOrder*()* throws IOException, SQLException *{* int id = Integer.*parseInt(*idTF.getText*())*;  
 int idClient= Integer.*parseInt(*idClientTF.getText*())*;  
  
 String path = System.*getProperty(*"user.dir"*)* + "\\bills\\";  
 String fileName = id + "\_" + idClient + ".txt";  
 path += fileName;  
  
 File file = new File*(*path*)*;  
 if*(*file.exists*())  
 {* file.delete*()*;  
 *}* file.createNewFile*()*;  
 System.*out*.println*(*fileName*)*;  
 System.*out*.println*(*path*)*;  
 writeFile*(*id,file*)*;  
 *}*

Iar metoda writeFile() scrie in fisierul creat. Aceasta creeaza un query pentru a face rost de toate informatiile necesare facturii.

public void writeFile*(*int id, File file*)* throws IOException, SQLException *{* Connection connection = null;  
 PreparedStatement statement = null;  
 ResultSet rs = null;  
 String query =  
 "SELECT "+  
 "`order`.`id` AS `Order ID`, `client`.`name` AS `Customer Name`, `client`.`address` AS `Address`, "+  
 "`product`.`id` AS `Product ID`, `product`.`name` AS `Product Name`, `product`.`price` AS `Product Price`, `order`.`quantity` AS `Quantity purchased` "+  
 "FROM *((*`order` JOIN `client` ON *((*`order`.`idClient` = `client`.`id`*)))* JOIN `product` ON *(((*`order`.`idProduct` = `product`.`id`*)* "+  
 " AND *(*`order`.`id` = ?*))))*";  
 try *{*  
 connection = ConnectionFactory.*getConnection()*;  
 statement = connection.prepareStatement*(*query*)*;  
 statement.setInt*(*1,id*)*;  
 rs = statement.executeQuery*()*;  
  
 *}* catch *(*SQLException e*) {  
 LOGGER*.log*(*Level.*WARNING*, "dao/OrderController: " + e.getMessage*())*;  
 *}* if*(*rs==null*)  
 {* System.*out*.println*(*"No order found!"*)*;  
 return;  
 *}* FileWriter fw = new FileWriter*(*file*)*;  
 BufferedWriter out = new BufferedWriter*(*fw*)*;  
  
 rs.next*()*;  
 out.write*(*"Order ID: " + rs.getString*(*1*)* + "\n"*)*;  
 out.write*(*" Customer Name: " + rs.getString*(*2*)* + "\n"*)*;  
 out.write*(*" Address: " + rs.getString*(*3*)* + "\n"*)*;  
 out.write*(*" Product Name: " + rs.getString*(*5*)* + "\n"*)*;//4  
 out.write*(*" Product Price: " + rs.getString*(*6*)* + "\n"*)*;//5  
 out.write*(*" Quantity: " + rs.getString*(*4*)* + "\n"*)*;//6  
  
 out.write*(*"\n"*)*;  
  
 out.flush*()*;  
 out.close*()*;  
 ConnectionFactory.*close(*rs*)*;  
 ConnectionFactory.*close(*statement*)*;  
 ConnectionFactory.*close(*connection*)*;  
 *}*

In rest, aceasta clasa + clasa ProductController contin aceleasi metode prezentate in ClientController.

**5. Rezultate**

Aplicatia functioneaza, dar mai sunt unele erori pe care nu le-am rezolvat. Aceasta indeplineste cu success functionalitatile cerute in cerinta temei.

**6. Concluzie**

In timpul implementarii acestui proiect a trebuit sa caut informatii legate de metoda reflectarii si modul de conexiune cu o baza de date. Cum am spus sim ai sus, mai sunt unele erori pe care nu le-am rezolvat dar per total aplicatie functioneaza conform asteptarilor.

**7. Bibliografie**

* <https://www.oracle.com/technical-resources/articles/java/javareflection.html>
* <https://www3.ntu.edu.sg/home/ehchua/programming/java/JDBC_Basic.html>