

**FUNDAMENTAL PROGRAMMING TECHNIQUES**

**ASSIGNMENT 3**

**ORDER MANAGEMENT**

**STUDENT: Balcanu Vlad-Andrei**

**Grupa: 30221**

**CUPRINS**

1. Obiectivul temei :…………………………………………………………3
2. Analiza problemei, modelare, scenarii, cazuri de utilizare:……………....3
3. Proiectare (decizii de proiectare, diagrame UML, structure de date, proiectare clase, interfete, relatii, packages, algoritmi, interfata utilizator):………………………………….3
4. Implementare:……………………………………………………………..5
5. Rezultate:………………………………………………………………….7
6. Concluzii:………………………………………………………………….13
7. Bibliografie:………………………………………………………………..13
8. **Obiectivul temei**

Obiectivul principal al acestei teme a fost proiectarea si implementarea unei aplicatii de management prin care utilizatorul/adminul poate procesa comenzile facute de clienti intr-un deposit impreuna cu o interfata grafica pentru a face utilizarea aplicatiei foarte usoara. Administratorul poate cauta, sterge, adauga si actualiza clientii, comenziile efectuate si produsele ce se gasesc in acest deposit si le poate vizualiza in orice pe moment pe toate. Rezultatele sunt stocate intr-o baza de date pentru a fi usor de stocat si organizat de catre aplicatie.

Obiectivile principale ale temei sunt:

* Folosirea unui desing de programare de tipul programare orientate pe obiect;
* Folosirea JavaDoc-ului pentru a documenta clasele si methodele si create fisierelor JavaDoc corespunzatoare;
* Folosirea unei baze de date relationale pentru a stoca datele folosite de aplicatie, minim trei tabele: Client,Product si Order;
* Creare unei interfete grafice care sa contina :
  + O fereastra pentru operatiile realizate pe clienti : adaugarea unui client nou , editarea unui client existent, stergerea unui client de baza de date, afisarea tuturor clientilor stocati in baza de date intr-o tabela (JTable);
  + O fereastra pentru operatiile realizate pe produse: adaugarea unui produs nou , editarea unui produs existent , stergerea unui produs din baza de edate, afisarea tuturor produselor din baza de date intr-o tabela (JTabel);
  + O fereastra pentru pentru creare comenzilor: utilizatorul va putea sa selecteze un produs existent, sa selecteze un client existent si sa insereze o cantitate de produse pe care clientul doreste sa le cumpere pentru a crea o comanda valida. In cazul in care nu sunt destule produse se va afisa un mesaj de stoc insufficient. Dupa ce comanda este finalizata se va decrementa stoc-ul;
* Folosirea tehnicilor de reflexie pentru a crea o metoda ce primeste o lista de obiecte si genreaza header-ul tabelei prin extragerea prin reflexie proprietatile obiectului si populeaza tabela cu valorile elementelor din lista.
* Documentatie bine pusa la punct;

Obiectivele secundare ale temei sunt :

* O arhitectura de tipul Layered Architecture(aplicati va contine cel putin 4 pachete : dataAccessLayer, businessLayer, model si presentation);
* Crearea unei chitante pentru fiecare comanda efectuata intr-un fisier de tip .txt sau .pdf;
* Folosirea tehnicilor de reflesie pentru a crea o clasa generica ce contine metodele pntru accesarea bazei de date : create object , edit object , delete object si find object. Interogarile folosite pentru accesarea bazei de date pentru un obiect specific vor fi generate prin reflexive;

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

Serviciile oferite de aceasta aplicatie sunt acelea de a realiza management-ul in timp real al unui deposit ce are produse, clienti si programeaza comenzi realizate de catre clienti. Toate acestea sunt realizate prin intermediul unui interfete grafice foarte simpla si utila pentru a face lucrul cu aceasta aplicatie foarte usor. Datele despre clienti , produse si comenzi trebuie sa fie cat mai usor de introdus in aplicatie de catre utilizator iar rezultatele sa fie cat mai concise si inteligibile pentru a pute afi analizate cu success.

Pasii pe care utilizatorul trebuie sa ii urmeze pentru a folosi aceasta aplicatie sunt: Daca se afla in fereastra pentru clienti , daca utilizatorul doreste sa vada toti clientii stocati momentan in baza de date va apasa pe butonul show Clients. Daca doreste sa caute un anume client sau sa stearga acesta va scrie in campul in care scrie “Introduce client id” id-ul clientului cautat si va apasa ori pe find Client ori pe Delete client. Daca utilizatorul doreste sa adauge un client nou va introduce datele despre acest client in campurile specificate si va apasa pe Add Client (id-ul clientului nou este generat automat). Daca utilizatorul doreste sa actualizeze un client existent acesta va introduce id-ul clientului dupa care va introduce noile date ale acestui client si va apasa pe butonul update Client..

Daca utlizatorul se afla in fereastra pentru produse , pentru a cauta sau sterge un produs va introduce id-ul acestuia in campul corespunzator si va apasa pe butonul de Find Product . Daca doreste sa vada toate produsele stocate momentan in aplicatie va apasa pe butonul show Orders. Daca doreste sa adauge un nou produs va complete campurile corespunzatoare dupa care va apasa pe butonul de add product ( id-ul produsului se va incrementa automat). Daca doreste sa actualizeze un produs existent va introduce id-ul produsului pe care doreste sa il actualizeze dupa care va introduce datele noi despre acest produs si va apasa pe butonul update Product.

Daca utilziatorul se afla in fereastra pentru comenzi , pentru a cauta sau sterge o camanda va introduce id-ul acesteia in campul corespunzator si va apasa pe butonul find Order. Daca doreste sa vada toate produselee stocate momentan in aplicatie va apasa pe butonul show Orders. Daca doreste sa adauge o noua comanda va complete campurile corespunzatoare dupa care va apasa pe butonul place order(id-ul comenzii se va incrementa automat). Daca doreste sa actualizeze o comanda existenta va introduce id-ul comenzii pe care doreste sa o schimbe dupa care va introduce noile date despre acea comanda si va apasa pe butonul update Order.

1. **Proiectare (decizii de proiectare , diagrama UML , structure de date, proiectare clase , interfete , relatii , packages , algoritmi, interfata utilizator)**

Pentru proiectarea acestei aplicatii s-a utilizat procedeul de design Layered Architecture ce separa aplicatia in 4 parti mari. The presentation Layer ce contine toate clasele responsabile pentru prezentarea aplicatiei (GUI) .The Model Layer ce contine in general toate clasele ce sunt structurate dupa tabelele folosite in baza de date. The Data Access layer ce contine in general clasele si deobicei clase de reflexive (DAO). Business Layer ce contine clase si incapsuleaza logica aplicatiei.

Clasele folosite sunt definitorii pentru fiecare parte din Layered Architecture.

In pachetul presentation avem clasele ClientGUI, ProductGUI si OrdersGUI in care am realizat codul pentru generarea fiecarei ferestre din interfata grafica pe care utilizatorii o folosesc pentru a manipula datele stocate si de asemenea codul prin care se realizeaza extragerea informatiilor si afisarea informatiilor. Aici mai avem clasa Controller care functioneaza ca o clasa Main a aplicatiei. In aceasta doar am instantiate cele 3 interfete grafice si de aici rulam aplicatia.

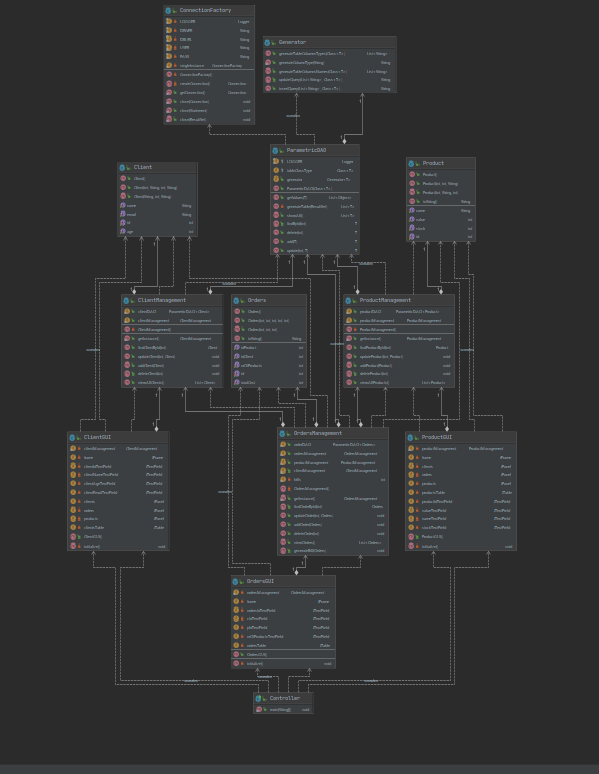
In pachetul Model se gasesc clasele Client, Orders si Product ce sunt conturate exact ca tabelele din baza de date in care se stocheaza datele. Acestea contin doar atributele celor 3 tabele.

In pachetul connection avem clasa ConnectionFactory care realizeaza conectarea aplicatiei java la baza de date utilizata.

In pachetul dataAccessLayer avem clasa ParametricDao ce reprezinta clasa prin care se realizeaza reflexivitatea din aceasta aplicatie.

In pachetul businessLayer gasim alte doua pachete numite management si Trivia. In management gasim clasele ClientManagement, OrdersManagement si ProductManagement in care sunt metodele ce folosesc reflexivitatea din clasa ParametricDao pentru a prelucra datele din baza de date. In pachetul Trivia gasim clasa Generator<T> ce este o clasa ajutatoare in care generam interogarile de insert si update si avem si metodele de extragere a tipului de data a fiecarei coloane dintr-o tabela si o metoda pentru extragerea numelor fiecarei coloane dintr-o tabela.

**DIAGRAMA UML A CLASELOR :**

****

Structurile de date utilizate au fost : Liste si ArrayList-uri pentru stocarea sirurilor de caractere , a obiecteleor de tip Produs, Client sau Orders.

Au fost folosite pachetele necesare pentru implementarea interfetei grafice (import java.awt.EventQueue; import javax.swing.JFrame ;import java.awt.CardLayout; import javax.swing.JPanel; import javax.swing.JTextField; import javax.swing.JButton;import java.awt.event.ActionListener; import java.awt.event.ActionEvent; import java.sql.SQLException; import java.util.List; import javax.swing.JTable; import javax.swing.table.DefaultTableModel;) si pachete din java.lang. pentru realizarea reflexiei , java.sql. pentru realizarea conexiunii cu baza de date , java.util. pentru a putea folosii Obiecte , Loggere si Liste si java.beans pentru introspectie.

1. **Implementare**

In acest capitol vor fi explicate in amanunt aspectele si implementarile pentru fiecare clasa in parte.

In pachetul presentation avem clasele ClientGUI, OrdersGUI si ProductGUI ce au rolul de a realiza interactiunea dintre utilizator si functionalitatile aplicatiei. Prima data in cele 3 clase sunt instantiate toate componentele avand un nume , dimensiune si pozitionare pe Panel si pe monitor. Au fost create cate un JButton pentru fiecare actiune ce este suportata de aplicatie (cautare dupa id , stergere dupa id , introducerea unui obiect nou , actualizarea unui obiect deja existent sau afisarea tuturor obiectelor aceste 5 comenzi fiind disponibile pentru fiecare fereastra in parte). Mai exista un JTable pe fiecare fereastra pentru afisarea in cazul cautarii unui obiect sau afisarii tuturor obiectelor. Se mai regasesc TextField-uri in care utilizatorul sa poata introduce datele necesare pentru actiunea dorita de acesta.

In fiecare clasa se gaseste un constructor de tip clasaGUI() . In acesti constructori se apeleaza metoda initialize() ce creaza un frame nou si adauga toate componentele pe cele 3 ferestre create.

In aceste 3 clase mai sunt prezenti actionListeneri pe fiecare buton ce realizeaza extragerea datelor din interfata grafica si cu algoritmul de executare si apelare a metodelor din clasa businessLayer pentru a prelua datele extrase si a le prelucra si trimite in aplicatie pentru a fi folosite la manipularea bazei de date.

In pachetul presentation se mai gaseste clasa Controller care contine metoda Main a aplicatiei si in care doar se apeleaza constructorii celor 3 interfete grafice pentru a porni aplicatia.

In pachetul Model vom gasi cele 3 clase , Client , Orders si Product ce au rol de a contura in aplicatie obiectele de aceste tip pentru a se sincroniza cu cele 3 tabele din baza de date , aceste clase avand exact acelasi nume si aceleasi atribute , in aceeasi ordine si la fel denumite ca si coloanele din tabela corespunzatoare din baza de date. In aceste clase vom gasi constructorul fiecarei clase gol deoarece avem nevoie de acesta cand lucram cu reflexivitate si trebuie sa creem instante ale obiectului si va explicat in cele ce urmeaza de ce. In fiecare dintre aceste 3 clase se mai gasesc gettere si settere pentru fiecare idn atributele lor precum si o metoda suprascrisa de toString pentru afisarea obiectului cu toate caracteristicile sale.

In pachetul dataAccessLayer avem doar clasa ParametricDAO<T> ce este o clasa generica si este responsabila pentru realizarea accesului la baza de date si contine metodele prin care se realizeaza interogarile pe aceasta baza de date pentru fiecare clasa , interogarile fiind generate dinamic prin reflexie.

Avem constructorul clasei public ParametricDAO(Class<T> tableClassType) prin care se seteaza tipul clasei/tabelei folosite in acest moment de catre DAO la cel transmis ca parametru.

Metoda public List<Object> getValues(T object) throws IllegalAccessException este folosita pentru a extrage valorile unui obiect/rand din clasa trimisa si returneaza o lista de obiecte ce contine aceste valori extrase.

Metoda private List<T> generateTable(ResultSet rs) realizeaza interogarea bazei de date si crearea clasei/tabelei in urma rezulatelor obtinute. Aceasta metoda returneaza lista de obiecte obtinuta ce reprezinta tabelul din baza de date dat in formatul aplicatiei java.

Metoda public List<T> showAll() throws SQLException imi realizeaza interogarea de tip Select \* from din tabela/clasa trimisa si imi genereaza tabelul corespunzator bazei de date intr-o lista de obiecte pe care mi-o returneaza.

Metoda public T findById(int id) throws SQLException realizeaza interogarea Select \* from where id= prin care imi cauta un obiect din una din cele 3 tabele . Aceasta metoda genereaza tabela coresupunzatoare si imi cauta in aceasta elemental cu id-ul corespunzator dupa care il returneaza. Daca obiectul nu exista se va arunca o exceptie.

Metoda public T delete(int id) throws SQLException realizeaza executarea interogarii Delete from where id= prin care se sterge din baza de date obiectul cu id-ul dorit. Aceasta metoda imi returneaza instanta obiectului sters.

Metoda public T add(T object) throws SQLException realizeaza interogarea de tip Insert Into table cu ajutorul clasei Generator in care am realizat metode pentru simplificarea acestei interogari. Aceasta metoda imi adauga in tabela dorita obiectul trimis si imi returneaza obiectul introdus.

Metoda public T update(int id, T object) throws SQLException realizeaza interograrea de tip Update Table cu ajutorul clasei Generator in care am realizat metode pentru simplificarea acestei interogari. Aceasta metoda imi actualizeaza un obiect al carui id il trimitem ca parametru cu un obiect trimis ca parametru ce contine detaliile actualizate in tabela corespunzatoare acestuia. Imi returneaza obiectul actualizat la final.

In pachetul connetion se regaseste clasa ConnectionFactory ce este responsabila pentru crearea conexiunii la baza de date folosite.

Construcotrul clasei private ConnectionFactory() ne creaza driver-ul conexiunii.

Metoda private Connection createConnection() ne creaza conexiunea la baza de date si ne returneaza un obiect de tip connection.

Metoda public static Connection getConnection() ne returneaza o instanta de tipul Connection.

Metoda public static void close(Connection connection) are rolul de a opri conexiunea cu baza de date.

Metoda public static void close(Statement statement) are rolul de a opri interogarea curenta.

Metoda public static void close(ResultSet resultSet) are rolul de a opri setul de rezultate curent.

In pachetul businessLayer avem 2 pachete : Management si Trivia.

In pachetul Management se regasesc clasele ClientManagement , OrdersManagement si ProductManagement ce au ca rol realizarea legaturii dintre baza de date si aplicatie prin utilizarea clasei ParametricDAO si a reflexivitatii sale. In cele 3 clase gasim instantiate cate un obiect de tipul ParametricDAO<T> cu parametrul fiind cel sugerat de clasa in care ne aflam (pentru ClientManagement vom avea ParametricDAO<Client> clientDAO) folosit pentru a apela prin reflexivitate metodele de manipulare a bazei de date gata implementate. Tot in aceste clase mai avem un obiect de tipul clasei (ClientManagement clientManagement=null) pentru a apela metodele cu acesta.

In fiecare dintre cele 3 clase gasim Constructorul acestora in care se instantia obiectul de tip ParametricDAO<T> pentru a se putea lucra cu acesta ( ex: ClientManagement.clientDAO = new ParametricDAO<>(Client.class)) , si o metoda de getInstance pentru a instantia Obiectul clasei.

In fiecare dintre cele 3 clase se gasesc metodele de prelucrare a datelor tabelelor corespunzatoare in care este folosita reflexivititatea. De exemplu in metoda public Client findClientById(int id) throws SQLException se apeleaza clientDAO.findById(id); care va apela metoda de gasire dupa id din clasa ParametricDAO folosind ca parametru tipul clasei trimise adica in acest exemplu tipul Client.

In clasa de tip OrdersManagement de mai gaseste in plus fata de restul claselor metoda public void generateBill(Orders orders) ce imi creaza un fisier text pentru fiecare noua comanda efectuata ce reprezinta chitanta acestei comenzi.

In pachetul Trivia se regaseste clasa ajutatoare parametrica Generator<T> in care sunt definite metode pentru interogarea tabelelor si crearea tipurilor si valorilor coloanelor tabelelor .

Avem metoda public static String generateColumnType(String receivedType) care primeste tipul adnotat al coloanei din tabela si returneaza tipul ei sub forma de String.

In aceasta clasa se gaseste metoda public List<String> generateColumnsTypes(@org.jetbrains.annotations.NotNull Class<T> type) care imi genereaza tipurile fiecarei coloane a tabelului specificat.

Mai avem metoda public List<String> generateTableColumnsNames(Class<T> type) prin care generam numele fiecarei coloane din tabelul specificat.

Metoda public String updateQuery(List<String> columns, Class<T> type) realizeaza interogarea de tip update , returnandu-mi un string ce contine textul pentru realizarea interogarii in SQL.

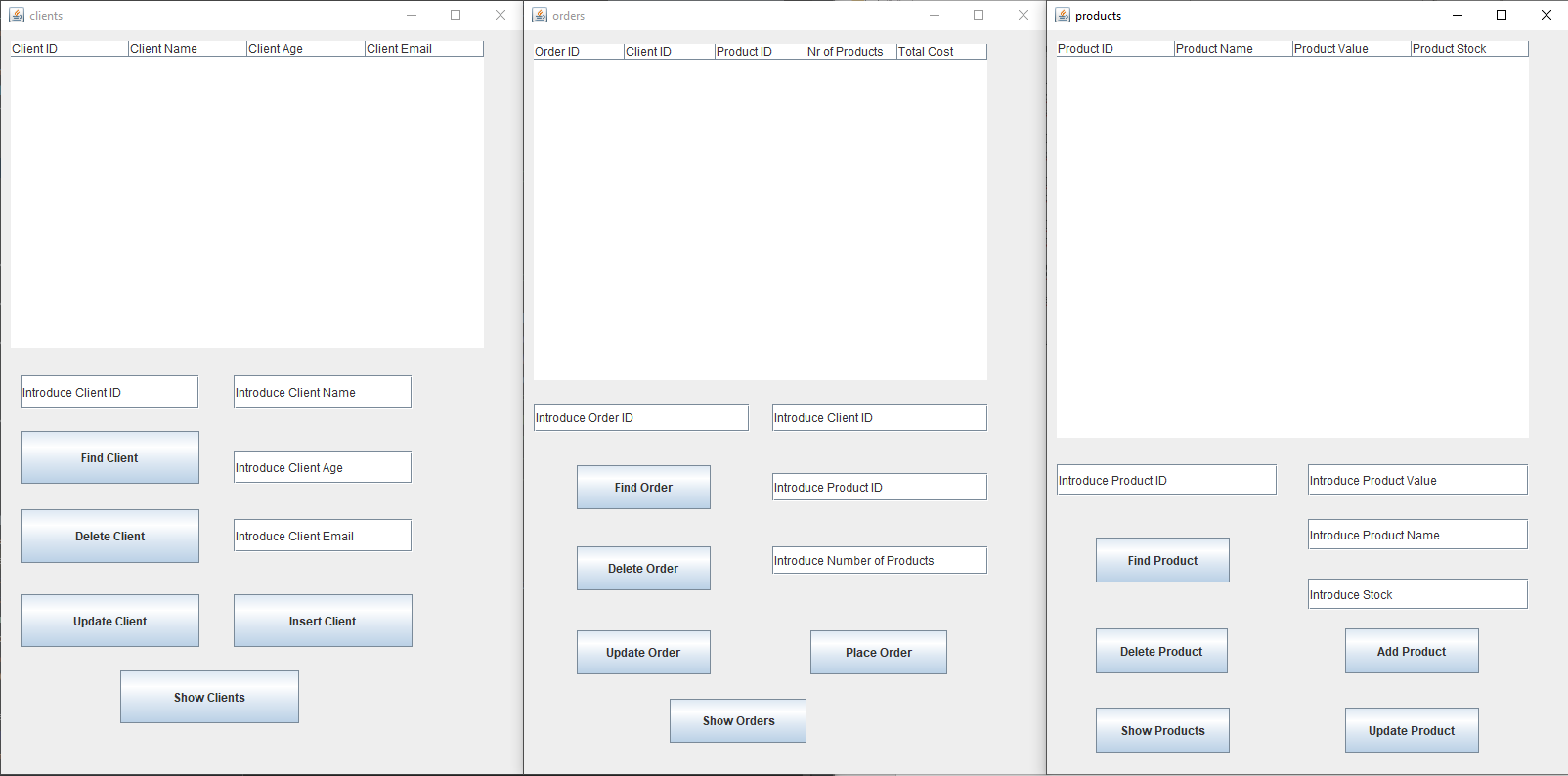
Metoda public String insertQuery(List<String> columns, Class<T> type) realizeaza interogarea de tip inset , returnandu-mi un string ce contine textul pentru realizarea interogarii in SQL.

1. **Rezultate**

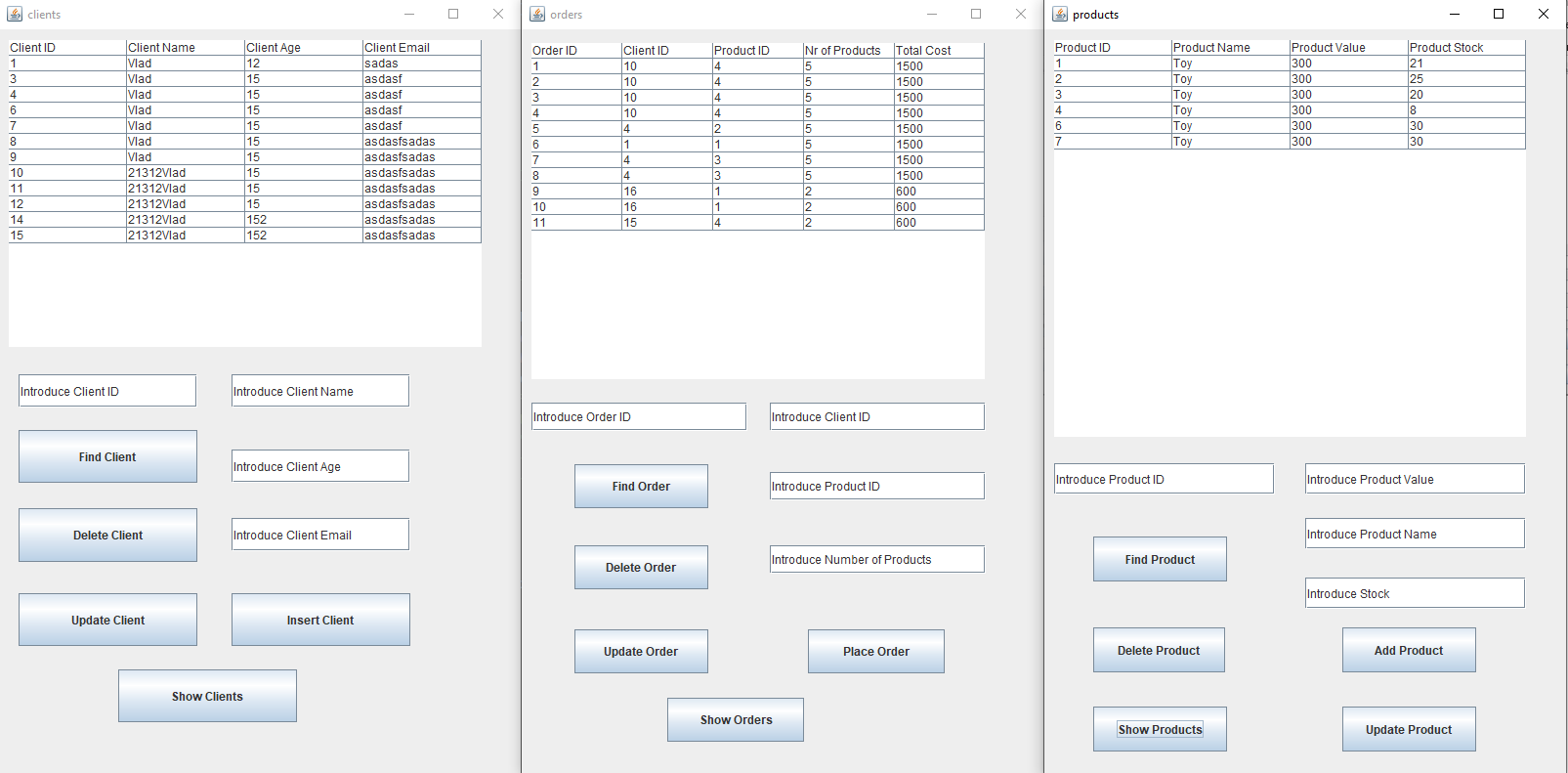
Pentru fiecare comanda noua efectuata se creaza o noua chitanta cu numarul comenzii in care sunt scrise datele despre comanda efectuata.

In interfata grafica se pot vizualiza rezultatele fiecarei operatii efectuate.

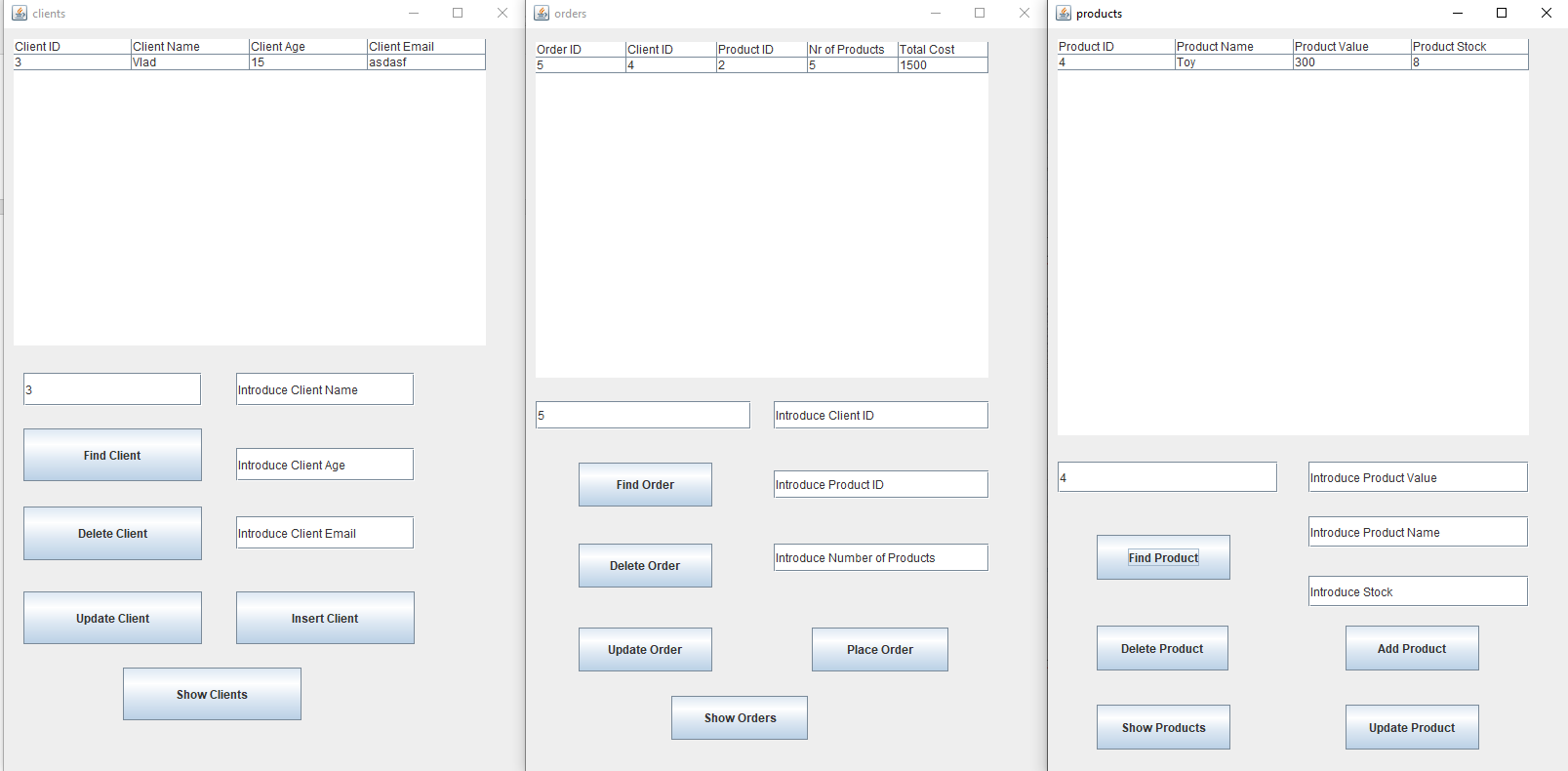
Interfata grafica la pornire :



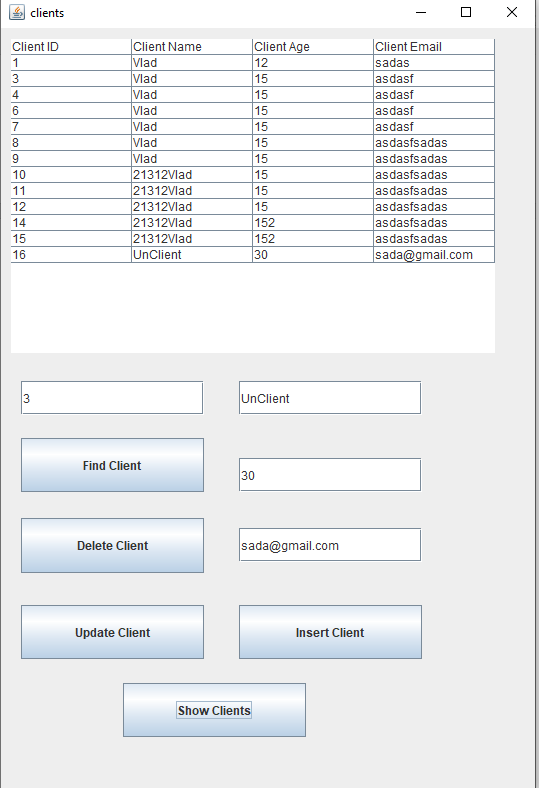
Interfata grafica cu toate elementele afisate:



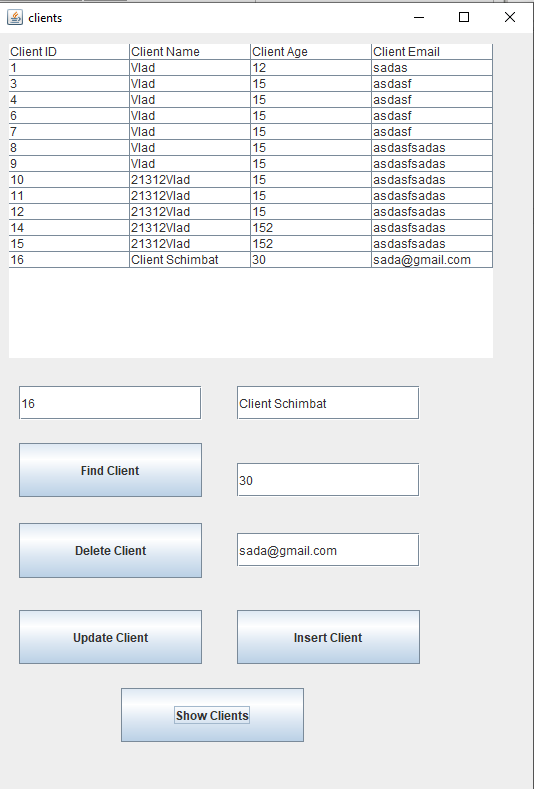
Interfata grafica cand cautam un element :



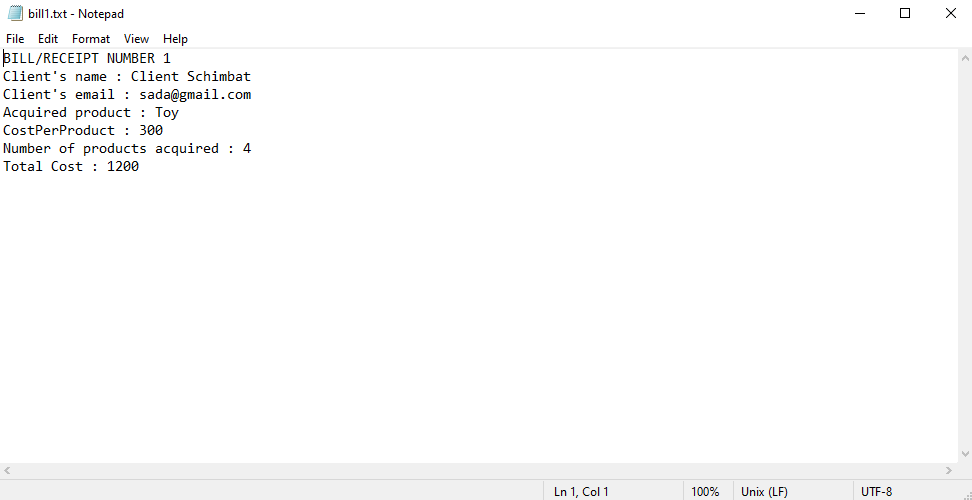
Dupa adaugarea unui nou element in una dintre cele 3 ferestre se poate apasa pe butonul de Show (Clients , Orders ,Products) pentru a vizualiza aparitia noului element in una dintre cele 3 ferestre. Am introdus clientul nou cu numele UnClient ce are 30 de adi si o adresa de mail la intamplare si am apasat pe butonul Show Clients.



Daca dorim sa actualizam un client introducem id-ul clientului si noile date ale acestuia dupa care apasam pe Update Client si dupa pe Show Clients pentru a observa schimbarea. Am schimbat numele clientului 16:



Fisierul text de creare a unei comenzi noi :



1. **Concluzii**

Dupa realizarea acestei teme am invatat cum sa conectez o aplicatie Java la o baza de date si sa lucrez cu aceasta si am invatat concepte importante despre Reflexivitate si despre Layered Architecture.

Dezvoltari ulterioare :

* Adaugarea de informatii suplimentare pentru fiecare dintre cele 3 tabele/clase ( adresa clientului , data efectuarii comenzii , Culoare/Detalii despre produs);
* Adaugarea de actiuni suplimentare pentru utilizator ( cautarea tuturor comenzilor realizate de un singur client , cautarea tuturor comenzilor dintr-o anumita data);
* Clasificarea Produselor in mai multe tipuri;
* Imbunatatirea vizuala a Interfetei gracice;

1. **Bibliografie**

* Conectarea MySQL cu Aplicatia Java : <https://www.baeldung.com/java-jdbc>si <https://www.baeldung.com/java-jdbc>;
* Layered Architecture : <https://dzone.com/articles/layers-standard-enterprise>;
* Reflection in Java : <http://tutorials.jenkov.com/java-reflection/index.html>;
* JavaDoc: <https://www.baeldung.com/javadoc> ;
* SQL dump file generation : <https://dev.mysql.com/doc/workbench/en/wb-admin-export-import-management.html>;