

Tehnici de Programare

Tema 4 – Food Delivery Management System

Profesor curs: Conf. Tudor Cioara Sudent: Varga Alexandra-Ioana

Profesor laborator: Dorin Moldovan Grupa: 30223

Cuprins

1. Obiectivul proiectului
2. Analiza problemei, modelare, scenarii, cazuri de utilizare
3. Proiectare (diagrama UML, structuri de date, clase, interfete, relatii, pachete, algoritmi)
4. Implementare si testare
5. Rezultate
6. Concluzii
7. Bibliografie
8. Obiectivul proiectului

Obiectivul principal al acestui assignment este de a realiza o aplicatie care sa implementeze un sistem de management au unui restaurant virtual. Este specificat ca sistemul sa contina 3 tipuri de utilizatori: **Administrator, Client** si respectiv **Employee**. Administratorul poate adauga, sterge sau modifica produse din meniul principal al restaurantului in care se stocheaza datele despre produse.Utilizatorul Client poate crea comenzi noi pentru o masa, adaugand elemente din meniu, pe baza carora poate calcula totalul si furniza factura pentru masa respectiva. Utilizatorul Employee este anuntat de fiecare data cand Client plaseaza o comanda.

Obiectivele secundare reprezinta etapele ce trebuie parcurse pentru realizarea obiectivului principal.

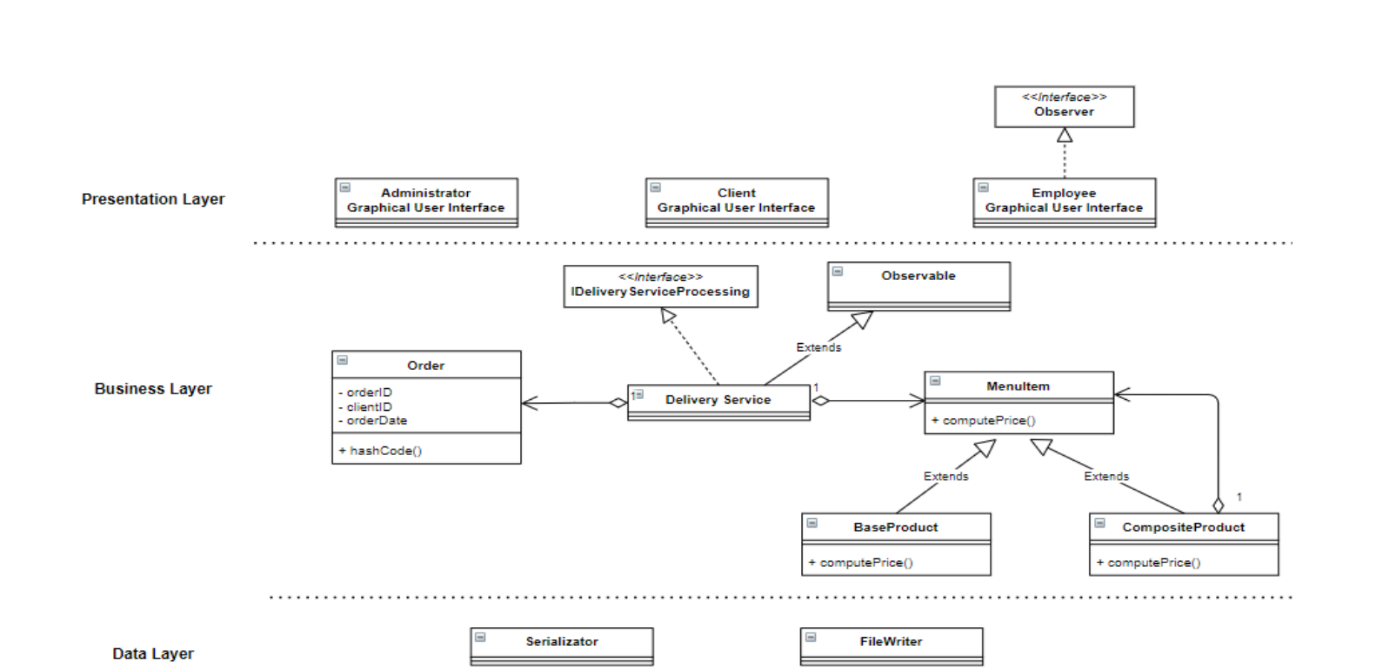
1. Analiza problemei, modelare, scenarii, cazuri de utilizare

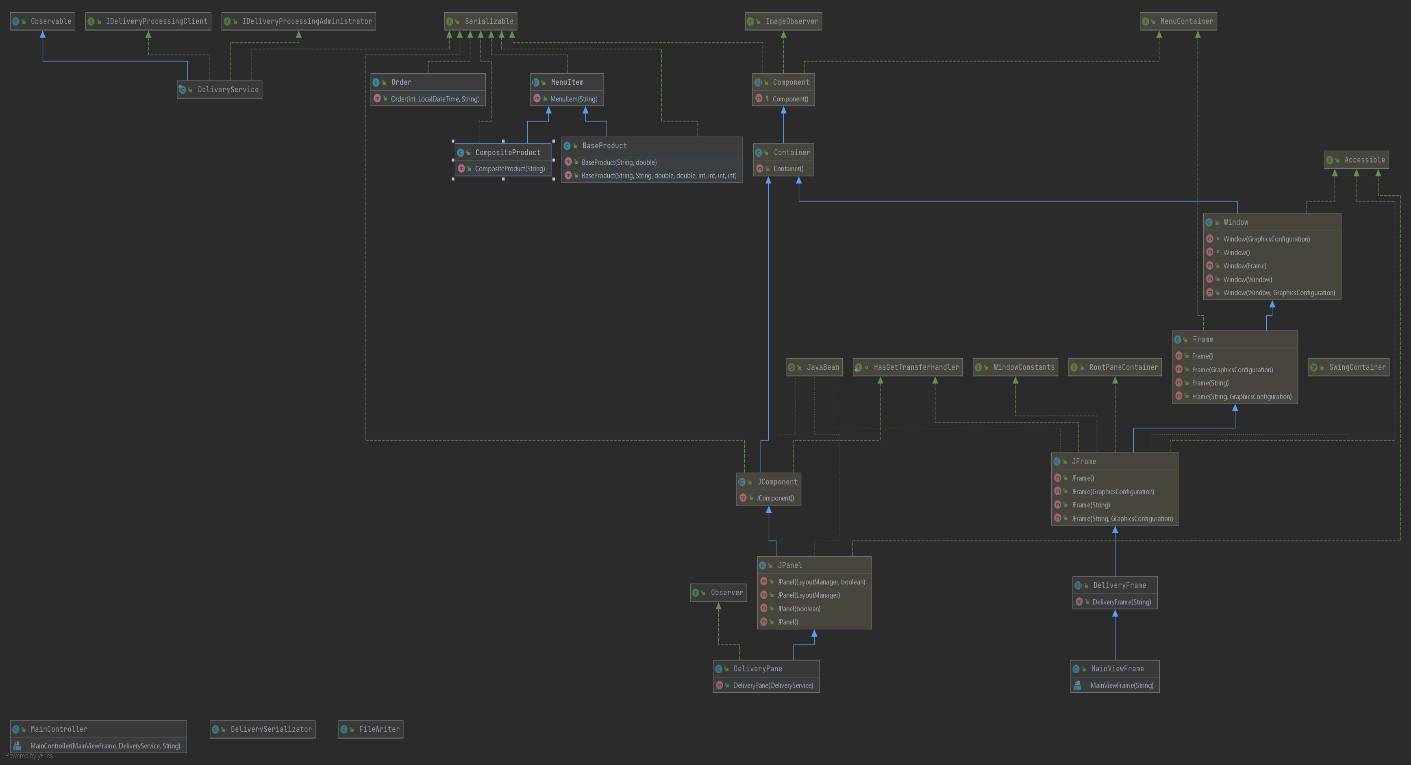
Principalele obiective sunt formulate prin modul de administrare al utilizatoriilor principali care sunt incadrati in acest sistem.Acestia sunt clasificati prin Administrator,Waitor si Chef. Pentru simplificarea implementarii, consideram ca sistemul Restaurant este utilizat de un singur user din fiecare tip enumerat, astfel putem evita folosirea unui proces de logare la utilizarea aplicatiei.

Mentinerea datelor in sistemul Restaurant functionaza pe baza de serializare/deserializare. Generarea notelor de plata pentru comenzi la anumite mese se face prin listarea de continut a comenzii in fisier text.

Utilizarea aplicatiei presupune in primul rand popularea meniului cu anumite produse.Administratorul poate alege numele, tipul si pretul produsului pe care doreste sa il introduca in meniu. Prin popularea continutului avem posibilatea si de a atribui diferite produse unor comenzi la o masa de clienti

1. Proiectare (diagrama UML, structuri de date, clase, interfete, relatii, pachete, algoritmi)





Arhitectura proiectului este stratificata (Layered Architecture), impartita in 3 diviziuni principale care sunt constituite din pachetele business logic, presentation si data access.Structura este urmata dupa sistemul de clase din diagrama de mai sus.

Pentru pornirea aplicatiei am introdus o interfata noua fata de scema de mai sus, Controller, pentru a putea manipula mai usor cele 3 interfete si pentru vizualizarea optiunilor dorite.

In Data Layer (data) obtinem accesul si la fisiere de tip “.ser”, prin clasa RestaurantSerializator, pentru serializare sau deserializare, dar si la folosirea fisierelor normale de tip “.txt” . In forma cea mai simpla serializarea obiectelor inseamna salvarea si restaurarea starii obiectelor. Obiectele oricarei clase care implementeaza interfata Serializable, pot fi salvate  intr-un stream(fluxde date) si restaurate  din acesta. Utilizand serializarea putem salva obictele create pe o platforma si le putem transmite prin retea pe un alt calculator care nu trebuie neaparat sa se ruleze pe aceasi platforma, totusi permite restauraea corecta a obiectelor.

Procesul invers, de citire a unui obiect serializat pentru a-i reface starea originala se numest deserializare.

Totodata am optat ca si pentru scrierea in interiorul fisierelor text sa folosesc o clasa predefinita, FileWriter care este menita sa scrie secvente de caractere sau bytes.

Urmatorul pachet semnificativ este reprezentat ca Business\_Layer (bll). Acest pachet contine setul de reguli care determina in ce forma vor fi datele stocate in interiorl aplicatiei, dar si manipularea acestora in program.Prima clasa semnificativa pe care o vom descrie va fi MenuItemcare reprezinta un obiect/produs ce il putem adauga in meniul virtual.Acesta este definit prin nume, tip si pret.Pe baza atributului “tip” am creat doua subclase ce extind MenuItem, astfel vom avea si BaseProductsi CompositeProduct. Este necesar de mentionat ca aceste clase sunt definite pentru a forma un Composite Design Pattern. Acest pattern ne permita sa tratam un grup de obiecte in acelasi mod ca o singura instanta din acelasi tip de obiect.

Interfata IrestaurantProcessing defineste principalele metode pe care le va indeplini clasa Restaurant. Clasa Restauranteste facuta pentru a define fiecare operatie legata FileWritermanipularea produselor de catre Administrator.

Ultimul pachet: presentation ce tine de Presentation\_Layer este legat de interfata grafica pentru utilizatorii principali. Acestui pachet apartin clasele AdministratorGraphicalUserInterface , WaiterGraphicalUserInterface, respectiv ChefGraphicalUserInterface. Care apeleaza in interiorul lor metodele corespunzatoare acestora si din clasa Restaurant unde putem manipula datele propriu-zise din interiorul sistemului.In clasa de interfata grafica doar setam interfata care trebuie sa fie functionala pentru a receptiona semnalele/comenziile utilizatorului la executia programului si sa produca un raspuns prin care putem obtine rezultatele asteptate.

1. Implementare si testare

Am inceput implementarea prin schitarea celor 4 interfete. In prima interfata Controller, am adaugat 3 butoane pentru fiecare dintre celelalte 3: AdministratorGraphicalUserInterface, WaiterGraphicalUserInterface, respectiv ChefGraphicalUserInterface.

JLabel lblNewLabel = **new** JLabel("Chose where to go:");

lblNewLabel.setBounds(17, 27, 115, 25);

frame.getContentPane().add(lblNewLabel);

JButton administrator = **new** JButton("Administrator");

administrator.addActionListener(**new** ActionListener() {

**public** **void** actionPerformed(ActionEvent a) {

AdministratorGraphicalUserInterface admin = **new** AdministratorGraphicalUserInterface();

frame.setVisible(**false**);

}

});

administrator.setBounds(26,60,120,23);

frame.getContentPane().add(administrator);

JButton chef = **new** JButton("Chef");

chef.addActionListener(**new** ActionListener() {

**public** **void** actionPerformed(ActionEvent a) {

ChefGraphicalUserInterface chef1 = **new** ChefGraphicalUserInterface();

frame.setVisible(**false**);

}

});

chef.setBounds(26,94,120,23);

frame.getContentPane().add(chef);

JButton waiter = **new** JButton("Waiter");

waiter.addActionListener(**new** ActionListener() {

**public** **void** actionPerformed(ActionEvent a) {

WaiterGraphicalUserInterface waiter1 = **new** WaiterGraphicalUserInterface();

frame.setVisible(**false**);

}

});

waiter.setBounds(26,128,120,23);

frame.getContentPane().add(waiter);

}

In AdministratorGraphicalUserInterface care implementeaza ActionListener am introdus metoda actionPerformed si in functie de butonul ales, administratorul va executa comanda dorita:

**public** **void** actionPerformed(ActionEvent e) {

// **TODO** Auto-generated method stub

**if**(e.getSource() == inapoi) {

frame.setVisible(**false**);

Controller start = **new** Controller();

start.frame.setVisible(**true**);

//start.initialize();

}

**if**(e.getSource() == bNewItem) {

**this**.adauga();

}

**if**(e.getSource() == bEdit) {

**this**.edit();

}

**if**(e.getSource() == bDelete) {

**this**.delete();

}

**if**(e.getSource() == bView) {

String meniu[][] = **new** String[restaurant.getSer().deserializationMenuItem().size()][2];

**for**(**int** i = 0; i < restaurant.getSer().deserializationMenuItem().size(); i++) {

meniu[i][0] = restaurant.getSer().deserializationMenuItem().get(i).getName();

meniu[i][1] = "" + restaurant.getSer().deserializationMenuItem().get(i).computePrice();

}

String column[] = {"Nume", "Pret"};

JTable t = **new** JTable(meniu,column);

t.setBounds(30,40,50,70);

JScrollPane scroll=**new** JScrollPane(t);

panel2.removeAll();

panel2.add(scroll);

frame2.add(panel2);

frame2.setVisible(**true**);

}

}

In WaiterGraphicalUserInterface avem metodele :

**public** **void** veziComenzi() {

String data[][] = **new** String[restaurant.getSer().deserializationOrders().size()][4];

**int** i = 0;

**for** (Map.Entry<Order, ArrayList<MenuItem>> j : restaurant.getSer().deserializationOrders()

.entrySet()) {

data[i][0] = "" + j.getKey().getOrderID();

data[i][1] = "" + j.getKey().getData();

data[i][3] = "" + j.getKey().getTable();

String s = "";

**for** (MenuItem h : j.getValue()) {

s += h.getName() + ", ";

}

s = s.substring(0, s.length() - 2);

data[i][2] = "" + s;

i++;

}

String column[] = { "Id", "Data", "Comanda", "Masa" };

JTable t = **new** JTable(data, column);

t.setBounds(100, 100, 100, 100);

JScrollPane scroll = **new** JScrollPane(t);

panel2.removeAll();

panel2.add(scroll);

frame2.add(panel2);

frame2.setVisible(**true**);

}

**public** **void** formareCombo1() {

**for** (**int** i = 0; i < **this**.restaurant.getSer().deserializationMenuItem().size(); i++) {

**this**.combo1.addItem(**this**.restaurant.getSer().deserializationMenuItem().get(i).getName());

}

}

**public** **void** adaugaProdusPeLista() {

**for** (**int** i = 0; i < **this**.restaurant.getSer().deserializationMenuItem().size(); i++) {

**if** (**this**.restaurant.getSer().deserializationMenuItem().get(i).getName().compareTo(**this**.combo1.getSelectedItem().toString()) == 0) {

**this**.produse.add(**this**.restaurant.getSer().deserializationMenuItem().get(i));

**if** (**this**.text.getText().isEmpty() == **false**)

**this**.text.setText(**this**.text.getText() + ", " + **this**.combo1.getSelectedItem().toString());

**else**

**this**.text.setText(**this**.text.getText() + "" + **this**.combo1.getSelectedItem().toString());

}

}

}

**void** newOrder() {

**if** (text.getText().compareTo("") != 0) {

ArrayList<MenuItem> items = **new** ArrayList<MenuItem>();

**for** (MenuItem i : **this**.produse) {

items.add(i);

}

**try** {

**int** n = Integer.*parseInt*(mese.getSelectedItem().toString());

**this**.restaurant.createNewOrder(**new** Order(*index*, **new** Date(), Integer.*parseInt*(mese.getSelectedItem().toString())),items);

} **catch** (IllegalArgumentException e) {

JPanel panel1 = **new** JPanel();

JTextField text1 = **new** JTextField("Introduceti numarul mesei: ");

JFrame frame1 = **new** JFrame();

panel1.add(text1);

frame1.add(panel1);

frame1.setSize(100, 40);

frame1.setDefaultCloseOperation(8);

frame1.setVisible(**true**);

}

**this**.*combo2*.addItem(*index*);

*index*++;

**this**.text.setText("");

**this**.produse.clear();

}

}

**public** **void** creareNotaDePlata() {

**int** n = Integer.*parseInt*(*combo2*.getSelectedItem().toString());

*combo2*.removeItem(n);

**for** (Map.Entry<Order, ArrayList<MenuItem>> order : restaurant.getSer().deserializationOrders().entrySet()) {

**if** (order.getKey().getOrderID() == n) {

restaurant.generateBill(order.getKey());

}

}

}

Care ne ajuta sa manipulam comenzile pe care le va primi ospatarul si sa multumim clientii pe cat posibil.

In ChefGraphicalUserInterface se face mereu update-ul comenziilor pentru a sti ce mancare sa prepare si timpul in care a fost facuta comanda.

In continuare am inceput sa lucrez in pachetul bll. Am facut clasele Order, CompositeProduct si BaseProduct.

Am implementat interfata IRestaurantProcessing:

**public** **interface** IRestaurantProcessing {

**public** **void** createNewMenuItem(MenuItem menu);

**public** **void** deleteMenuItem(String deleteMenu);

**public** **void** editMenuItem(String name, **double** price);

**public** **void** createNewOrder(Order order, ArrayList<MenuItem> items);

**public** **double** computePrice(Order order);

**public** **void** generateBill(Order order);

}

Cu Metodele folosite in clasa Restaurant, unde se intampla majoritatea operatiilor cerute de atre client. Putem sa spunem ca aceasta clasa este un fel de creier al acestui proiect.

**public** **class** Restaurant **extends** ~~Observable~~ **implements** IRestaurantProcessing {

**private** RestaurantSerializator ser = **new** RestaurantSerializator();

**private** Map<Order, ArrayList<MenuItem>> tables = ser.deserializationOrders();

**private** ArrayList<MenuItem> meniu = ser.deserializationMenuItem();

**private** FileWriter f;

**public** Restaurant() {

**this**.tables = **new** HashMap<>();

**this**.meniu = **new** ArrayList<>();

}

**public** **boolean** isTrue() {

**if**(**this** **instanceof** Restaurant) {

**return** **true**;

} **else**

**return** **false**;

}

@Override

**public** **void** createNewMenuItem(MenuItem menu) {

// **TODO** Auto-generated method stub

**assert** menu != **null**;

**assert** isTrue();

**try** {

meniu.add(menu);

ser.serializationMenuItem(meniu);

}**catch** (Exception e) {

System.***out***.println("Nu s-a putut introduce in meniu");

}

**assert** isTrue();

**assert** menu != **null**;

}

@Override

**public** **void** deleteMenuItem(String deleteMenu) {

// **TODO** Auto-generated method stub

**assert** deleteMenu != "";

**assert** isTrue();

**try** {

**for**(**int** i=0; i<meniu.size(); i++) {

**if**(meniu.get(i) **instanceof** BaseProduct) {

**if**(((BaseProduct) meniu.get(i)).getName().compareTo(deleteMenu)==0)

meniu.remove(i);

}

**if**(meniu.get(i) **instanceof** CompositeProduct) {

**if**(((BaseProduct) meniu.get(i)).getName().compareTo(deleteMenu)==0)

meniu.remove(i);

}

}

**this**.ser.serializationMenuItem(meniu);

}**catch**(Exception e) {

System.***out***.println("Nu s-a putut sterge ");

}

**assert** isTrue();

**assert** deleteMenu != "";

}

@Override

**public** **void** editMenuItem(String name, **double** price) {

// **TODO** Auto-generated method stub

**assert** name != "";

**assert** isTrue();

**try** {

**for**(**int** i=0; i<meniu.size(); i++) {

**if**(meniu.get(i) **instanceof** BaseProduct) {

**if**(((BaseProduct) meniu.get(i)).getName().compareTo(name)==0)

((BaseProduct) meniu.get(i)).setPrice(price);

}

**if**(meniu.get(i) **instanceof** CompositeProduct) {

**if**(((CompositeProduct) meniu.get(i)).getName().compareTo(name)==0)

((CompositeProduct) meniu.get(i)).setPrice(price);

}

}

**this**.ser.serializationMenuItem(meniu);

}**catch**(Exception e) {

System.***out***.println("Nu s-a putut modifica");

}

**assert** isTrue();

**assert** name != "";

}

@Override

**public** **void** createNewOrder(Order order, ArrayList<MenuItem> items) {

// **TODO** Auto-generated method stub

**assert** order != **null**;

**assert** isTrue();

**try** {

**this**.tables.put(order, items);

ser.serializationOrders(tables);

}**catch**(Exception e) {

System.***out***.println("Nu s-a putut introduce comanda");

}

**assert** isTrue();

**assert** order != **null**;

}

@Override

**public** **double** computePrice(Order order) {

// **TODO** Auto-generated method stub

**double** suma = 0;

**for**(Map.Entry<Order, ArrayList<MenuItem>> i : **this**.getSer().deserializationOrders().entrySet()) {

**if**(i.getKey().getOrderID() == order.getOrderID()) {

**for**(**int** j = 0; j < i.getValue().size(); j++) {

suma += i.getValue().get(j).computePrice();

}

}

}

**return** suma;

}

@Override

**public** **void** generateBill(Order order) {

// **TODO** Auto-generated method stub

FileWriter f=**new** FileWriter();

f.getBill(order);

}

**public** Map<Order, ArrayList<MenuItem>> getTables() {

**return** tables;

}

**public** **void** setTables(Map<Order, ArrayList<MenuItem>> tables) {

**this**.tables = tables;

}

**public** RestaurantSerializator getSer() {

**return** ser;

}

**public** ArrayList<MenuItem> getMeniu() {

**return** meniu;

}

}

In pachetul data am implementat clasele FileWriter care scrie in fisier txt nota de plata si detaliile clientului ( ce a consumat, cat costa, masa la care a stat)

**public** **class** FileWriter {

**public** **void** getBill(Order order)

{

ArrayList<MenuItem> items=order.getItem();

**try** {

PrintWriter output = **new** PrintWriter("BillNo" + order.getOrderID() + ".txt");

output.println("--NOTA DE PLATA--");

output.println();

output.println("data: "+order.getData());

//output.println();

output.println("Masa "+order.getTable());

output.println("Comanda " + order.getOrderID());

output.println();

output.println("Produsele comandate sunt:");

output.println();

**for**(**int** i=0;i<items.size();i++)

{

**if**(items.get(i) **instanceof** BaseProduct)

output.println("produs: "+((BaseProduct) items.get(i)).getName()+" -> pret: "+((BaseProduct) items.get(i)).getPrice());

**if**(items.get(i) **instanceof** CompositeProduct)

output.println("produs: "+((CompositeProduct) items.get(i)).getName()+" -> pret: "+((CompositeProduct) items.get(i)).getPrice());

}

output.close();

} **catch** (Exception e) {

e.printStackTrace();

System.***out***.println(e.getMessage());

}

}

}

si clasa RestaurantSerializator unde se realizeaza serializarea si deserializarea informatiilor.

**public** **class** RestaurantSerializator {

**public** **void** serializationMenuItem(ArrayList<MenuItem> item) {

**try** {

FileOutputStream fileOut = **new** FileOutputStream("Restaurant.ser");

ObjectOutputStream out = **new** ObjectOutputStream(fileOut);

out.writeObject(item);

out.close();

fileOut.close();

System.***out***.printf("Serialized data is saved in Restaurant.ser\n");

} **catch** (IOException e) {

e.printStackTrace();

}

}

**public** ArrayList<MenuItem> deserializationMenuItem() {

ArrayList<MenuItem> items = **new** ArrayList<MenuItem>();

**try** {

FileInputStream fileIn = **new** FileInputStream("Restaurant.ser");

ObjectInputStream in = **new** ObjectInputStream(fileIn);

items = (ArrayList<MenuItem>) in.readObject();

in.close();

fileIn.close();

**return** items;

} **catch** (IOException e) {

//e.printStackTrace();

**return** items;

} **catch** (ClassNotFoundException e) {

System.***out***.println("Eroare la deserializare");

e.printStackTrace();

**return** items;

}

}

**public** **void** serializationOrders(Map<Order, ArrayList<MenuItem>> tables) {

**try** {

FileOutputStream fileOut = **new** FileOutputStream("Orders.ser");

ObjectOutputStream out = **new** ObjectOutputStream(fileOut);

out.writeObject(tables);

out.close();

fileOut.close();

System.***out***.printf("Serialized data is saved in Restaurant.ser\n");

} **catch** (IOException e) {

e.printStackTrace();

}

}

**public** HashMap<Order, ArrayList<MenuItem>> deserializationOrders() {

HashMap<Order, ArrayList<MenuItem>> orders = **new** HashMap<Order, ArrayList<MenuItem>>();

**try** {

FileInputStream fileIn = **new** FileInputStream("Orders.ser");

ObjectInputStream in = **new** ObjectInputStream(fileIn);

orders = (HashMap<Order, ArrayList<MenuItem>>) in.readObject();

in.close();

fileIn.close();

**return** orders;

} **catch** (IOException e) {

//e.printStackTrace();

**return** orders;

} **catch** (ClassNotFoundException e) {

System.***out***.println("Eroare la deserializare");

e.printStackTrace();

**return** orders;

}

}

}

1. Rezultate

Rezultatele sunt obtinute pe loc, de obicei in TextFields sau JPanels, componente ale interfetei grafice pe care o accesam. Depinde de fiecare si ceea ce va cere consumatorul sau clientul ospatarului. Se pot verifica si fisierele text generate in fisierul proiectului.

1. Concluzii

Am invatat ce inseamna serializarea si deserializarea si cum se pot aborda in implementarea unui proiect. Astfel pot incapsula si reciti o clona exacta a unui Object.Datorita implementarii mai detaliate a interfetei grafice fata de alte assignment-uri. Am exersat implementarea interfetelor grafice, chiar daca as fi vrut sa le fac mai aspectuase, mai colorate si mai atragatoare, din pacate timpul nu a fost de partea mea si a trebuit sa ma rezum la o interfata mai simpla si banala. Sper ca aplicatia mea sa mearga si probabil imi va fi utila in viitor la mai multe proiecte sau chiar alte materii.

1. Bibliografie

* <https://ms.sapientia.ro/~manyi/teaching/oop/oop_romanian/curs15/curs15.html>
* <https://www.youtube.com>
* <http://docs.oracle.com/javase/8/docs/technotes/guides/language/assert.html>
* <http://javarevisited.blogspot.ro/2012/01/what-is-assertion-in-java-java.html>
* <http://stackoverflow.com/questions/11415160/how-to-enable-the-java-keywordassert-in-eclipse-program-wise>
* <http://www.tutorialspoint.com/java/java_serialization.htm>
* <https://www.baeldung.com/java-serialization>
* <https://www.geeksforgeeks.org/serialization-in-java/>
* <https://docs.oracle.com/javase/8/docs/api/java/io/Serializable.html>