Universitatea Technica Cluj-Napoca

Facultatea de Automatica si Calculatoare

Departamentul de Calculatoare

Semestrul 2

Tehnici de programare-Tema 4

Restaurant Managment

Danciu Maria Alexandra

Grupa:30222

An universitar 2019-2020

**Cuprins**

1. **Obiectivul temei**
2. **Analiza problemei,modelare,scenarii,cazuri de utilizare**
3. **Proiectare**
4. **Implementare si testare**
5. **Rezultate**
6. **Concluzii , dezvoltari ulterioare**
7. **Bibbliografie**
8. Obiectivul temei:

Obiectivul temei este implementarea unei aplicatii de gestiune a unui restaurant. Sistemul ar trebui sa aiba trei tipuri de utilizatori: administratorul, cheful si chelnerul. Admnistratorul poate sa adauge, sa stearga si sa modifice produse deja existente din meniu. Chelnerul poate sa creeze noi comenzi pentru o masa, sa adauge elemente de la meniu, sa calculeze nota si sa dea chitanta. Bucatarul este anuntat de fiecare data ce comanda a fost realizata anterior,pentru a efectua comanda si a verifica existenta produselor.Se considera ca nu este necesara existenta unui proces de login deoarece exista doar un singur administrator,un ospatar si un chef bucatar.

Cerinte:

1. Implementarea GUI-ului
2. Adaugarea frame-urilor pentru fiecare din cei 3 useri
3. Interschimbarea frame-urilor pentru ca fiecare user sa poata efectua operatiile destinate
4. Alegerea structurilor de date pentru a salva obiectele necesare efectuarii operatiilor
5. Adaugarea si implementarea componentelor si operatiilor pentru fiecare user
6. Generarea chitantelor in format txt.

**2. Analiza problemei**

Ca sa creeam o aplicatie care satisface toate cerintele mentionate mai sus, trebuie sa facem niste decizi de implementare. In primul rand trebuie sa ne gandim ce pattern si ce structuri de date vom folosi, in acest caz vom folosi “Composite Design pattern” si pentru implementare vom f olosi un HashMap.

Se va realiza o descriere a pattern-ului Composite. Este bazat pe un concept foarte natural. Conceptul produselor compuse din alte produse este unul foarte comun practic vizibil peste tot. Toate lucrile din viata noatra de zi cu zi sunt produse compuse din mai multe elemente. Acest pattern practic sugereaza crearea unor produse de baza care pot sa fie folosite in crearea unor produse compuse. Un produs compus este un element care contine mai multe elemente fie ele compuse sa de baza. Acest concept este foarte util pentru aplicatia noastra. Un element din meniu poate sa fie de tipul BaseProduct sau de tipul CompositeProduct.Se vor utiliza structurile de tip HashMap.Pentru a crea o legatura vom folosi campurile din order, ca si cheie pentru HashMap, iar ca valoarei lista de elemente care o sa o aiba fiecare comanda.

Un alt obiectiv de care trebuie sa tinem cont la crearea programul este faptul ca va trebui sa serializam elemente din clasa restaurant in cazul nostru meniul. Acest lucru se face prin implementarea interfetei Serializable pentru clasele la care dorim sa facem serializare, ceea ce se refera la scrierea datelor intr-un fisier de tipul “.ser". Serializarea reala a datelor este facuta de Java, si programatorul nu poate sa o vada prorpiu zis. In esenta ii spune limbajului Java sa scrie obiectele de tipul unei anumite clase intr-un fisier. De aici daca ii cerem lui Java sa ne scriem o clasa el o va face intr-un fisier .ser pe care il alegem noi. Daca dorim sa deserializam datele atunci vom avea nevoie sa citim efectiv datele cu formatul normal din acest fisier .

Serializare= transformarea unui obiect ˆıntr-o secvent¸˘a de octet¸i, din care s˘a poat˘a fi ref˘acut ulterior obiectul original. Procesul invers, de citire a unui obiect serializat pentru a-i reface starea original˘a, se nume¸ste deserializare. Referint¸ele care construiesc starea unui obiect formeaz˘a oˆıntreag˘a ret¸ea de obiecte.

Utilitatea serializarii

Mecanism simplu de utilizat pentru salvarea si restaurarea datelor.

• Persistent¸a obiectelor

• Compensarea diferentelor intre sisteme de operare

• Transmiterea datelor in retea

• RMI (Remote Method Invocation) • Java Beans - asigurarea persistent¸ei componentelor.

USER CASE

Adauga produse

Sterge produse

Editeaza produse

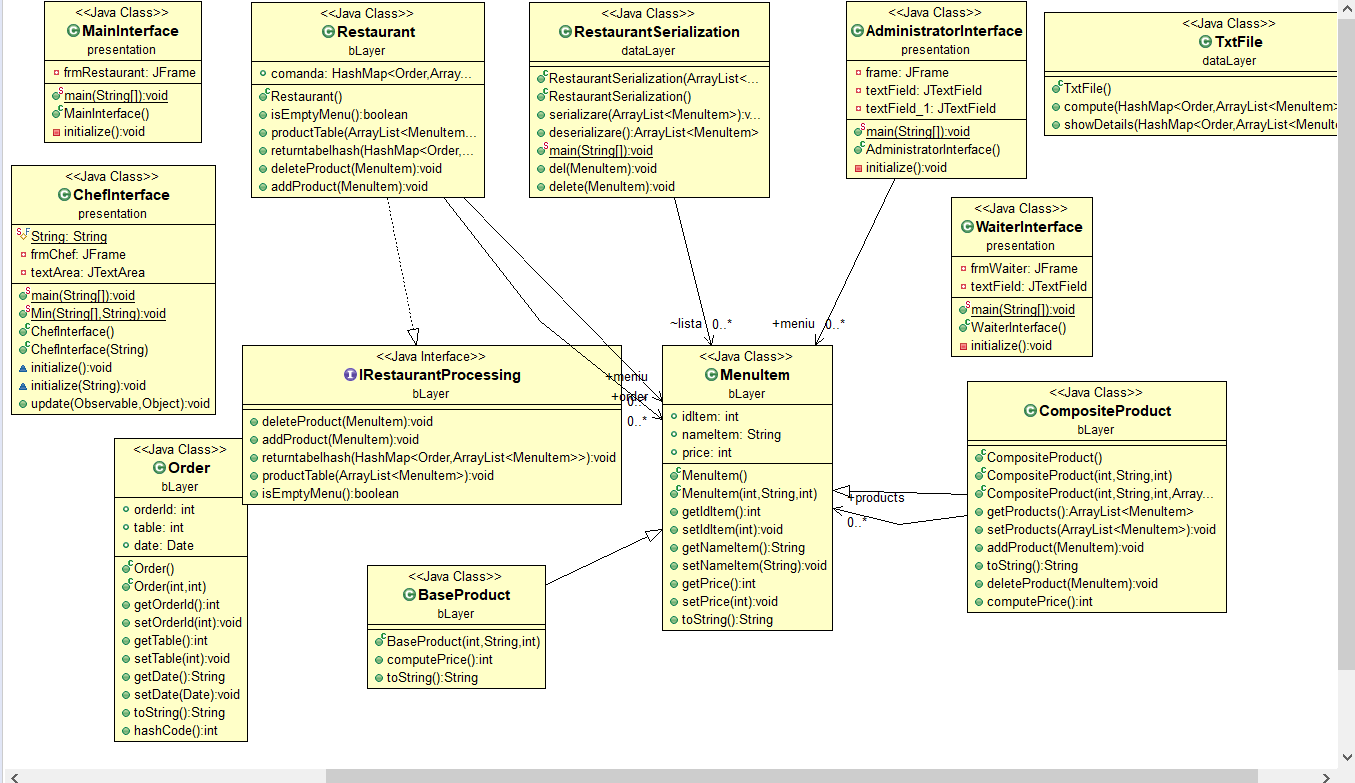
Creare comanda

Creare chitanta+calculare suma totala

Notificare+pregatirea preparatelor

Diagrama UML

Se prezinta mai jos diagrama UML a proiectului:



**4.Implementarea**

In continuarea o sa va descriu implementarea fiecarei clase si ce metode am folosit si ce field-uri am definit.

MenuItem

Aceasta clasa reprezinta obiectele care se vor afla in meniu. Atributele idItem.nameItem si price sunt folosite pentru a descrie caracteristicile unui item din meniu si se mentioneaza ca aceasta clasa implementeaza Serializabil deoarece se va realiza serializarea elementelor din meniu.

Metodele:

-setter,getter,toString

**public** **int** getIdItem() {

**return** idItem;

}

**public** **void** setIdItem(**int** idItem) {

**this**.idItem = idItem;

}

**public** String getNameItem() {

**return** nameItem;

}

**public** **void** setNameItem(String nameItem) {

**this**.nameItem = nameItem;

}

**public** **int** getPrice() {

**return** price;

}

**public** **void** setPrice(**int** price) {

**this**.price = price;

}

@Override

**public** String toString() {

**return** "MenuItem [idItem=" + idItem + ", nameItem=" + nameItem + ", price=" + price + "]";

}

BaseProduct

Este o clasa care reprezinta produsele de baza cu care se creaza un produs compus. Se mentioneaza ca aceasta clasa extinde MenuItem.Clasa are un singur constructor ce apeleaza prin super constrcutorul clasei pe care o extinde,adica MenuItem.

**public** BaseProduct(**int** idProduct,String nameProduct,**int** priceProduct) {

**super**(idProduct,nameProduct,priceProduct);

}

CompositeProduct

Aceasta clasa reprezinta ideea de produse compuse din alte produse aici se materializeaza conceptul de “Composite Pattern". Se mentioneaza ca aceasta clasa extinde MenuItem si contine un field care este o lista de MenuItems, deci aceasta lista poate contine atat BaseProduct cat si CompositeProduct. In aceasta clasa implementam urmatoarele metode:

- getItems() : ArrayList<MenuItem> aceasta metoda este una de tip getter pentru lista

**public** CompositeProduct(**int** id, String name, **int** pret,ArrayList<MenuItem> product) {

**super**(id, name, pret);

**this**.products=product;

}:

acesta este un cunstructor care primeste ca parametru id,nume si o lista de produse de tip baseProduct(MenuItem)

- addItem(MenuItem menuItem) : void este o functie care adauga in lista curenta un noua menuItem si Aduna la pretul curent noul pret al item-ului adaugat

**public** **void** deleteProduct(MenuItem item) {

products.remove(item);

}

--Sterge un item din lista de produse

**public** **int** computePrice() {

**int** computePrice=0;

**for**(MenuItem item:products) {

computePrice=computePrice+item.getPrice();

}

**return** computePrice;

}

--Calculeaza pretul pentru lista de produse

Order

Aceasta clasa reprezinta conceptul de comanda de la restaurant. O comanda se constituie prin atributele numarul mesei -pentru distingerea clientilor, orderId-id-ul comenzii si data, combinatia carora este un identificator unic. In aceasta clasa implementam urmatoarele metode:

- setOrderID(intorderID) : void aceasta este o metoda de tipul setter pentru field-ul id

-setTable(int table) : void aceasta este o metoda de tipul setter pentru field-ul table

-setDate(Date date) : void aceasta este o metoda de tipul setter pentru field-ul date

-getOrderId() : int aceasta este o metoda de tipul getter pentru field-ul id

-getTable() : int aceasta este o metoda de tipul getter petrnu field-ul table

-getDate() : int aceasta este o metoda de tipul getter pentru field-ul date

-hashCode() : int aceasta este o metoda care returneaza codul de hash pentru comanda curenta

**public** Order() {

}

///constructor

**public** Order(**int** ID,**int** t) {

orderId=ID;

table=t;

**this**.date=**new** Date();

}

Am utilizat 2 constructori,unul default iar in celalat primeste ca si parametru atributele comenzii.

**public** String getDate() {

Date dNow = **new** Date( );

SimpleDateFormat ft =

**new** SimpleDateFormat ("E yyyy.MM.dd 'at' hh:mm:ss a zzz");

**return** ft.format(dNow);

}

Clasa getDate-afiseaza un format standard de tip an-luna-zi ora-minute-secunde pentru inregistrarea comenzii.

IRestaurantProcessing

Aceasta este o interfata, nu vom descrie aici toate functionalitatile acestor functii insa le vom descrie in clasa restaurant, iar aici doar vom aminti antetele functiilor. Antetele metodelor sunt urmatoarele:

**public** **void** deleteProduct(MenuItem item);

**public** **void** addProduct(MenuItem item);

**public** **void** returntabelhash(HashMap<Order,ArrayList<MenuItem>> comanda);

**public** **void** productTable(ArrayList<MenuItem> meniu);

**public** **boolean** isEmptyMenu();

Restaurant

Aceasta clasa ne reprezinta restaurantul propriu zis, aceasta clasa se afla in pachetul de business ca si toate celelalte clase de mai sus. In aceasta clasa extindem Observable si implementam IRestaurantProcessing. Ca si atribute avem un hashMap care face legatura intre meniu si comenzile clientilor, o lista de MenuItem care reprezinta totalitea produselor din meniu, o lista de Order care reprezinta toate order-urile date la un restaurant, de asemenea se precizeaza ca am verificat folosindu-ne de post conditii si pre conditii si de o metoda de tipul isWellFormed. Metodele acestei clase sunt:

- Restaurant() acesta este un constructor care populeaza prin serializare meniul restaurantului dintr-un fisier “.ser" dat de utilizator .

- createItem(String name, int price) : void aceasta metoda adauga un BaseProduct in meniul de produse, in implementarea curenta am decis sa nu il las pe utilizator sa comande BaseProduct, pentru ca am simtit ca e mai potrivit sa mai realistic sa poata sa comande doar CompositeProduct

- deleteItem(MenuItem item) : void aceasta metoda sterge un obiect din meniul actual dupa id-ul acestuia,nume si pret

- createOrder(int table, ArrayList<MenuItem> order) : void aceasta metoda ii permite unui chelner sa creeze o noua comanda la o anumita masa

- createBill(int table) : void aceasta metoda creaza o chitanta pentru o comanda de la o anumita masa

- getCustomerOrders() : HashMap<Order, ArrayList<MenuItem>> aceasta metoda este un getter pentru HashMap

- getMenuItems() : ArrayList<MenuItem> aceasta metoda este o metoda de tip getter pentru meniul restaurantului

- getOrders() : ArrayList<Order> aceasta metoda este una de tip getter si este folosita pentru a returna comenzile actuale ale restaurantului.

**public** **void** productTable(ArrayList<MenuItem> meniu)

Metoda afiseaza intr-un tabel elementele din meniu dupa efectuarea serializarii,se extrag atributele "Id produs","Denumire produs","Pret produs","Compozitie"

Se verifica instanta obiectului,pentru a putea fi reprezentat.

Ex Base Product/Composite Product

**public** **void** returntabelhash(HashMap<Order,ArrayList<MenuItem>> comanda)

Aceasta metoda afiseaza de asemenea un tabel ,dar de aceasta data vor fi reprezentate comenzile.Metoda primeste ca si parametru un HashMap de tip<Order, ArrayList<MenuItem>>

Unde sunt stocate comenzile.

In acest tabel se vor nota atributele "ID order","Data comenzii","Masa","Continut comanda"

Exemplu reprezentare table:

**public** **void** returntabelhash(HashMap<Order,ArrayList<MenuItem>> comanda) {

// Restaurant rez=new Restaurant();

JFrame f = **new** JFrame();

// Frame Title

f.setTitle("Comenzi");

String[] captabel= {"ID order","Data comenzii","Masa","Continut comanda"};

System.***out***.println("Am comanda de lungime"+comanda.size());

Object[][] datetabel=**new** Object[comanda.size()][4];

java.util.Iterator<Entry<Order, ArrayList<MenuItem>>> it = comanda.entrySet().iterator();

**int** i=0;

**while** (it.hasNext()) {

String s="";

HashMap.Entry pair = (HashMap.Entry)it.next();

System.***out***.println(pair.getKey() + " = " + pair.getValue());

Order order=(Order) pair.getKey();

datetabel[i][0]=order.getOrderId();

datetabel[i][1]=order.getDate();

datetabel[i][2]=order.getTable();

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

produse=(ArrayList<MenuItem>) pair.getValue();

**for** (**int** k=0;k<produse.size();k++)

{

s+=produse.get(k).getNameItem()+" ";

}

datetabel[i][3]=s;

i++;

it.remove();

}

JTable tabel=**new** JTable(datetabel,captabel);

tabel.setBounds(30, 40, 200, 300);

// adding it to JScrollPane

JScrollPane sp = **new** JScrollPane(tabel);

f.add(sp);

// Frame Size

f.setSize(500, 200);

// Frame Visible = true

f.setVisible(**true**);

}

TxtFile

Aceasta clasa este folosita la scrierea obiectelor in contextul serializari. Metodele acestei clase sunt:

**public** **void** compute( HashMap<Order,ArrayList<MenuItem>> comanda)aceasta metoda este folosita pentru a scrie intr-un fisier de tip txt chitanta aplicata in urma unei comenzi

RestaurantSerializator

Aceasta clasa este folosita la citirea obiectelor in contextul serializari. Metodele acestei clase sunt:

**public** **void** serializare(ArrayList<MenuItem> lista)

Serializarea este procedeul prin care datele încapsulate în instanța unei clase sunt trimise și primite cu ajutorul unui flux de I/O. Clasele care realizează acest lucru sunt [java.io.ObjectOutputStream](http://docs.oracle.com/javase/7/docs/api/java/io/ObjectOutputStream.html) și [java.io.ObjectInputStream](http://docs.oracle.com/javase/7/docs/api/java/io/ObjectInputStream.html) iar metodele cele mai utilizate sunt *writeObject(Object)* și *readObject()*.

**public** ArrayList<MenuItem> deserializare()

Deserializarea este procedeul invers,de extragere a datelor din fisier si returnarea lor ,in acest caz de tip ArrayList<MenuItem>

MainClass

Aceasta este prima clasa care se incarca din program. Metodele acestei clase sunt:

- main(String[] args) : void aceasta metoda este metoda principala din main si aceasta porneste programul, de astemena aceasta transmite numele unui fisier ca parametru, daca acesta se ofera

Pachetul presentation contine cele 4 GUI-uri

MainInterface

Aceasta clasa reprezinta interfata de inceput din care utilizatorul poate sa isi aleaga ce optiuni doreste. Optiunile sunt:Administrator,Chef,Waiter(ospatar).



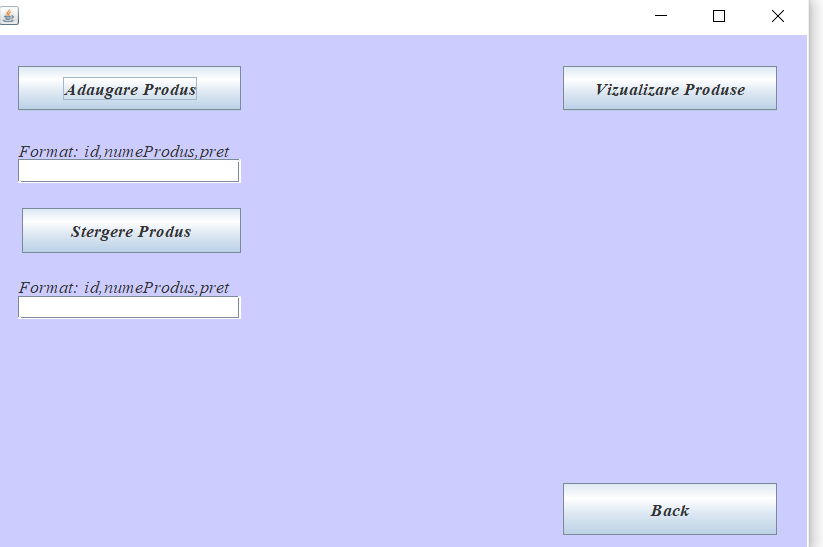
Functionalitatile sunt diferite in functie de ce selectam.

- initialize() : void aceasta metoda este folosita pentru a incarca elemente de UI pentru pagina de inceput

AdministratorInterface

Aceasta clasa reprezinta interfata de administrator. Aceasta clase are un numar foarte mare de field-uri, de gettere si setter pentru actionListener-urile butoanelor, doarece aceste au un numar foarte mare, insa o functionealitate foate similiara nu o sa le mai precizam in acest caz, cu mentiunea ca au aceasi functionalitate cu metodele similare din clasele descrise mai sus.

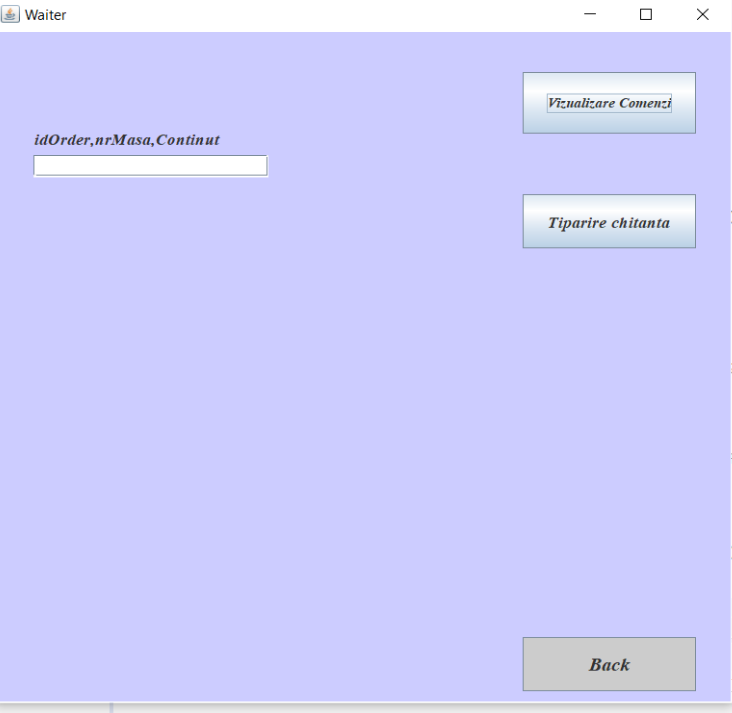
- initialize() : void aceasta este metoda principala si cea mei mare din aceasta clasa, aceasta metoda este folosita pentru a descrie GUI-ul din administrator si pentru a seta fiecare proprietate a obiectelor



Interfata contine butoane destinate adaugarii unui produs,stergerii si vizualizarii meniului.Butonul back realizeaza revenirea la interfata principala pentru a schimba tipul utilizatorului.

WaiterInterface

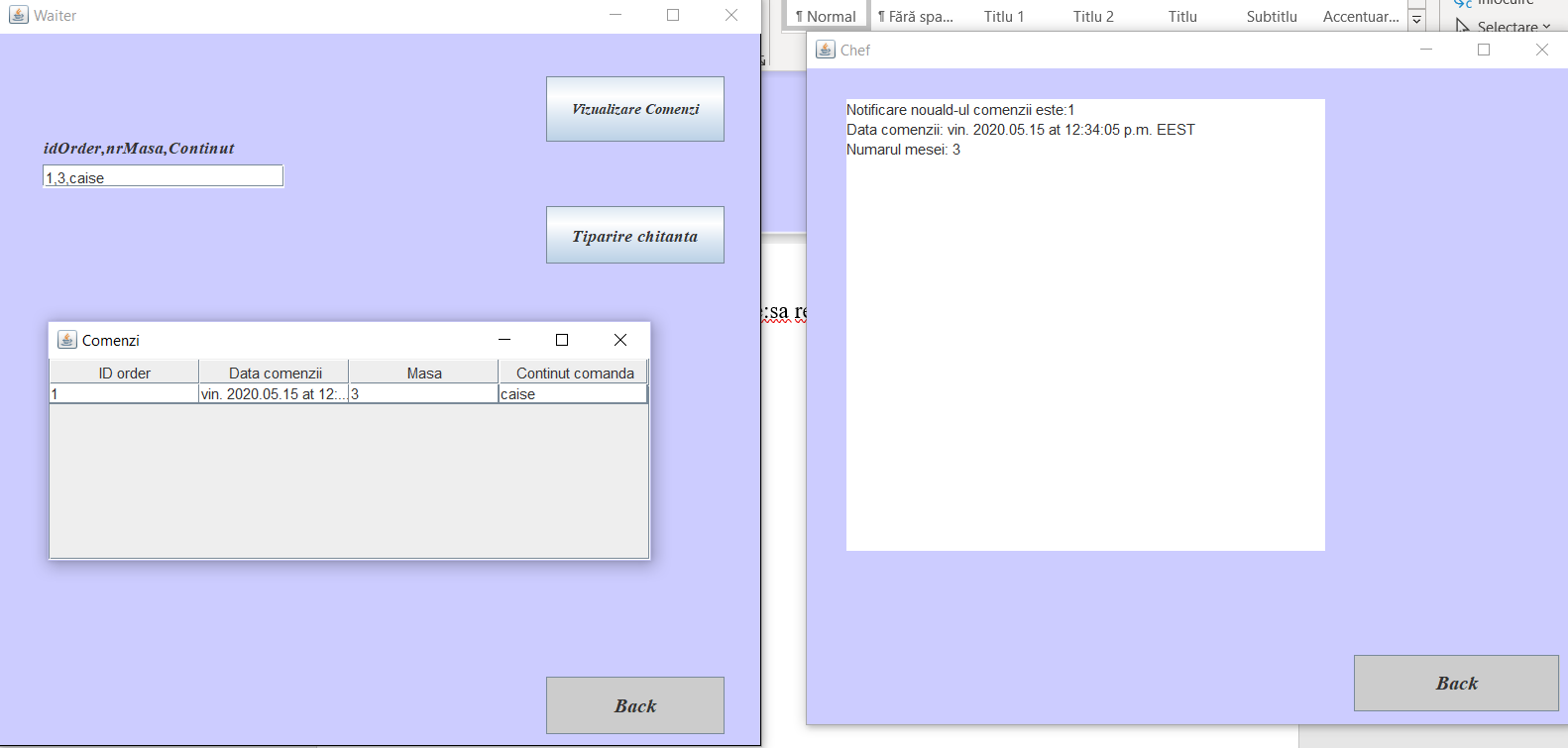
Aceasta clasa reprezinta interfata de chelner. Similar cu admin,dar cu functionalitati diferite.



Ospatarul are la dispozitie optiunile:sa realizeze o comanda,sa vizualizeze comanda si sa tipareasca chitanta.

**5. Rezultate**

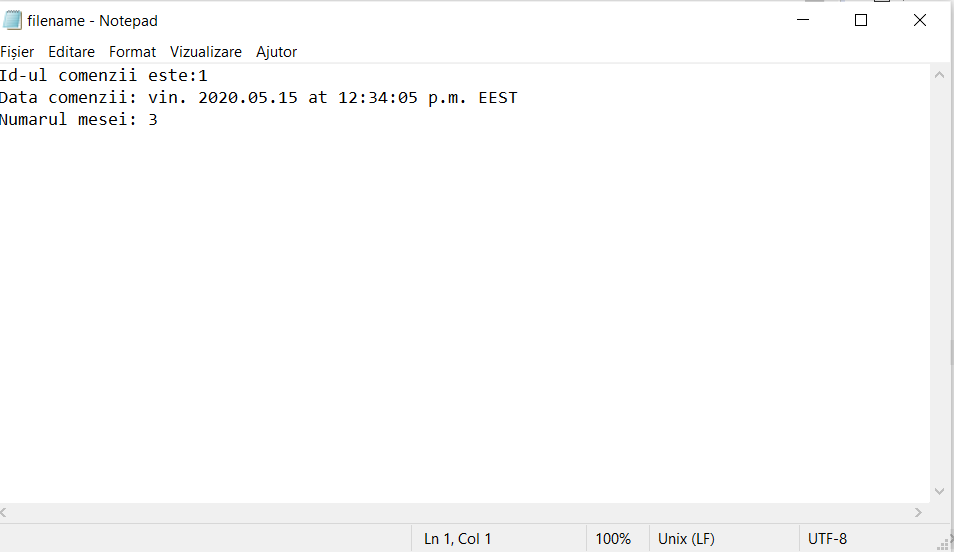
Exemplu realizare comanda

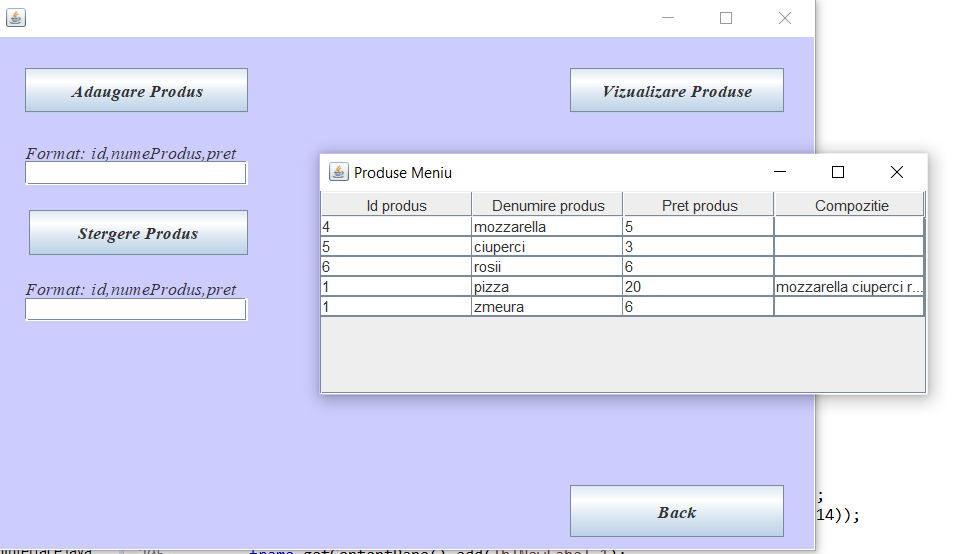


Chef-ul primeste o notificare cu noua comanda.

Se afiseaza datele noii comenzi.

Se tipareste chitanta.





Vizualizarea produselor din meniu

**6.Concluzii**

Aceasta tema ma ajutat sa imi imbunatatesc lucrul cu interfetele grafice in Java.

**7.Bibliografie**

<https://www.geeksforgeeks.org/serialization-in-java/>

<https://www.youtube.com/watch?v=--PuaGLHATs&t=364s>

<http://programare-in-java.blogspot.com/2016/04/serializarea-si-deserializarea.html>

<https://www.w3schools.com/java/java_hashmap.asp>

<http://coned.utcluj.ro/~salomie/PT_Lic/4_Lab/Assignment_4/Assignment_4.pdf>