DOCUMENTATIE

**TEMA 4**

NUME STUDENT: CÎNDEA ALEXANDRU

GRUPA: 30228

**CUPRINS**

1. Obiectivul temei………………………………………………………….….3
2. Analiza problemei, modelare, scenarii, cazuri de utilizare………………….3
3. Proiectare…………………………………………………………………….3
4. Implementare………………………………………………………………...4
5. Concluzii……………………………………………………………………..14
6. Bibliografie…………………………………………………………………..14

1.Obiectivul temei:

Implementarea unui sistem de menegiere a unui serviciu de catering. Acesta presupune crearea unei interfete interactive in stansa legatura cu un fisier tip .csv si unul .txt care sa ne recunoasca si salveze utilizatorii aplicatiei , aceasta functionand pe baza unui cont de client / administrator.

Obiectivele secundare sunt:

1. Analizarea problemei pentru a cunoaste cazurile si scenariile care pot aparea.
2. Proiectarea de diagrame UML pentru a putea structura datele date in clase si interfete.
3. Implementarea reprezinta scrierea codului efectiv si gandirea acestuia intr-un mod cat mai eficient cu putinta.

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

Se cere implementarea unei interfete de logare care sa directioneze utilizatorul spre portalul destinat acestuia. Pentru utiliztorul de tip administrator trebuiesc implementate functia de importare a datelor salvate la inceputul programului in meniul virtual , functia de adaugare a unui nou produs , functia de stergere a unui produs existent , functia de modificare a unui produs , functia de creare a unui produs nou compus din mai multe produse deja existente si functia de creare a rapoartelor. Pentru utilizatorul de tip client trebuiesc implementate functia de cautare dupa anumite criterii , funtia de adaugare a unui produs la o comanda si functia de a crea o comanda. Comanda creata de client trebuie totodata sa fie transmisa mai departe unui angajat pentru a putea fi facuta si livrata.

3.Proiectare:

4.Implenemtarea proiectului:

Acest proiect inglobeaza foarte multe clase care sunt aranjate pe pachete speciale fiecare cu o functionalitate aparte. In continuare voi prezenta fiecare pachet:

* Pachetul businessLogic , pachet in care sunt declarate obiectele efective care sunt folosite in rezolvarea cerintelor. In acest pachet regasim clasa MenuItem care este o clasa abstracta extinsa in clasele BaseProduct si CompositeProduct.

private String title;  
private int rating;  
private int calories;  
private int protein;  
private int fat;  
private int sodium;  
private double price;  
  
public MenuItem*(*String title, int rating, int calories, int protein, int fat, int sodium, double price*) {* this.title = title;  
 this.rating = rating;  
 this.calories = calories;  
 this.protein = protein;  
 this.fat = fat;  
 this.sodium = sodium;  
 this.price = price;  
*}*

Clasa Order care reprezinta obiectele de tip order in care se salveaza ora si data comenzii si o lista de produse adaugate de catre un utilizator client.

int orderID;  
String clientUser;  
LocalDate date;  
LocalTime time;  
  
public Order*(*int orderID, String clientUser, LocalDate date,LocalTime time*) {* this.orderID = orderID;  
 this.clientUser = clientUser;  
 this.date=date;  
 this.time=time;  
*}*

Clasa Clienti care deserveste salvarii utilizatorilor de tip client si ajuta la functionalitatea logarii. String user;  
String password;  
  
public Clienti*(*String user, String password*) {* this.user = user;  
 this.password = password;  
*}*

* Pachetul presentation este probabil cel mai important pachet al acestui proiect. Acesta include doua pachete secundare si anume Pachetul view in care se regasesc clasele destinate interfetelor de vizualizare si anume:

Angajat

Client

LogInView

Angajat

AddView

ModifyView

Si pachetul controller care contine o singura clasa ControllerAll care se ocupa de functionalitatea tuturor interfetelor si care defineste functionalitatea fiecarui buton in parte. Aici se produc mare parte din operatiile inportante si necesare acestui proiect.

class logInListener implements ActionListener*{* @Override  
 public void actionPerformed*(*ActionEvent e*){* String user=logInView.getUserField*()*;  
 String password=logInView.getPasswordField*()*;  
  
 if*(*user.equals*(*"admin"*)* && password.equals*(*"admin"*)){* logInView.setVisible*(*false*)*;  
 administratorView.setVisible*(*true*)*;  
 *}*else*{* int ok=0;  
 try *{* File file =new File*(*"E:\\PT2022\_30228\_Cindea\_Alexandru\_4\\clienti.txt"*)*;  
 Scanner myReader=new Scanner*(*file*)*;  
 while*(*myReader.hasNext*()){* String userC=myReader.next*()*;  
 String paswordC=myReader.next*()*;  
 Clienti client=new Clienti*(*userC,paswordC*)*;  
 clientView.getComboBox*()*.removeAllItems*()*;  
 if*(*user.equals*(*client.getUser*())* && password.equals*(*client.getPassword*())){* ok=1;  
 for*(*MenuItem i:menuItems*){* clientView.getComboBox*()*.addItem*(*i.getTitle*())*;  
 *}* logInView.setVisible*(*false*)*;  
 clientView.setVisible*(*true*)*;  
 clientS=client;  
 break;  
 *}  
 }* if*(*ok==0*){* logInView.showMessage*(*"Username sau parola gresita!!"*)*;  
 *}* myReader.close*()*;  
 *}* catch *(*FileNotFoundException ex*) {* ex.printStackTrace*()*;  
 *}  
 }  
  
 }  
}*class newUserListener implements ActionListener*{* @Override  
 public void actionPerformed*(*ActionEvent e*){* String user=logInView.getUserField*()*;  
 String password=logInView.getPasswordField*()*;  
 FileWriter fw=null;  
 BufferedWriter bw=null;  
 PrintWriter pw=null;  
 try *{* fw = new FileWriter*(*"clienti.txt", true*)*;  
 bw= new BufferedWriter*(*fw*)*;  
 pw=new PrintWriter*(*bw*)*;  
 if*(*user.length*()*!=0 && password.length*()*!=0*) {* pw.println*(*user + " " + password*)*;  
 *}*else*{* logInView.showMessage*(*"Datele nu au fost introduse!"*)*;  
 *}* pw.flush*()*;  
 fw.close*()*;  
 bw.close*()*;  
 pw.close*()*;  
  
 *}*catch *(*Exception e1*){}  
 }  
}*class importListener implements ActionListener*{* @Override  
 public void actionPerformed*(*ActionEvent e*){* try *{* String line="";  
 String splitBy=",";  
 BufferedReader br=new BufferedReader*(*new FileReader*(*"E:\\PT2022\_30228\_Cindea\_Alexandru\_4\\products.csv"*))*;  
 br.readLine*()*;  
 while*((*line=br.readLine*())*!=null*) {* String*[]* product = line.split*(*splitBy*)*;  
 String name = product*[*0*]*;  
 int rating = Integer.*parseInt(*product*[*1*])*;  
 int calories = Integer.*parseInt(*product*[*2*])*;  
 int protein = Integer.*parseInt(*product*[*3*])*;  
 int fat = Integer.*parseInt(*product*[*4*])*;  
 int sodium = Integer.*parseInt(*product*[*5*])*;  
 double price = Double.*parseDouble(*product*[*6*])*;  
  
 BaseProduct baseProduct = new BaseProduct*(*name, rating, calories, protein, fat, sodium, price*)*;  
 int ok=0;  
 for*(*MenuItem i:menuItems*){* if*(*baseProduct.equals*(*i*)){* ok=1;  
 *}  
 }* if*(*ok==0*) {* menuItems.add*(*baseProduct*)*;  
 *}  
 }  
 }* catch *(*IOException ioException*) {* ioException.printStackTrace*()*;  
 *}* administratorView.showMessage*(*"Produsele au fost importate cu succes!!"*)*;  
 *}  
}*class addOpenListener implements ActionListener*{* @Override  
 public void actionPerformed*(*ActionEvent e*){* administratorView.setVisible*(*false*)*;  
 addView.setVisible*(*true*)*;  
 *}  
}*class deleteOpenListener implements ActionListener*{* @Override  
 public void actionPerformed*(*ActionEvent e*){* String s=null;  
 deleteView.getDeleteBox*()*.removeAllItems*()*;  
 for*(*MenuItem i:menuItems*){* s=i.getTitle*()*+" , "+i.getRating*()*+" , "+i.getProtein*()*+" , "+i.getFat*()*+" , "+i.getSodium*()*+" , "+i.getPrice*()*;  
 deleteView.getDeleteBox*()*.addItem*(*s*)*;  
 *}* deleteView.setVisible*(*true*)*;  
 *}  
}*class modifyOpenListener implements ActionListener*{* @Override  
 public void actionPerformed*(*ActionEvent e*){* administratorView.setVisible*(*false*)*;  
 modifyView.setVisible*(*true*)*;  
 *}  
}*class createOpenListener implements ActionListener*{* @Override  
 public void actionPerformed*(*ActionEvent e*){* createComposedView.getProd1comboBox*()*.removeAllItems*()*;  
 createComposedView.getProd2comboBox*()*.removeAllItems*()*;  
 createComposedView.getProd3comboBox*()*.removeAllItems*()*;  
 createComposedView.getProd3comboBox*()*.addItem*(*null*)*;  
 createComposedView.getProd2comboBox*()*.addItem*(*null*)*;  
 createComposedView.getProd1comboBox*()*.addItem*(*null*)*;  
 for*(*MenuItem i:menuItems*){* try *{* createComposedView.getProd1comboBox*()*.addItem*(*i.getTitle*())*;  
 createComposedView.getProd2comboBox*()*.addItem*(*i.getTitle*())*;  
 createComposedView.getProd3comboBox*()*.addItem*(*i.getTitle*())*;  
 *}*catch *(*Exception ex*){}  
 }* administratorView.setVisible*(*false*)*;  
 createComposedView.setVisible*(*true*)*;  
 *}  
}*class raportOpenListener implements ActionListener*{* @Override  
 public void actionPerformed*(*ActionEvent e*){* raportsView.setVisible*(*true*)*;  
 *}  
}*class backListener1 implements ActionListener *{* @Override  
 public void actionPerformed*(*ActionEvent e*){* logInView.setVisible*(*true*)*;  
 administratorView.setVisible*(*false*)*;  
 logInView.refresh*()*;  
 *}  
}*class UpdateListener implements ActionListener*{* @Override  
 public void actionPerformed*(*ActionEvent e*){* int rating=0;  
 if*(*clientView.getRatingField*()*.length*()*!=0*) {* rating = Integer.*parseInt(*clientView.getRatingField*())*;  
 *}* int calories=0;  
 if*(*clientView.getCaloriiField*()*.length*()*!=0*) {* calories = Integer.*parseInt(*clientView.getCaloriiField*())*;  
 *}* int protein=0;  
 if*(*clientView.getProteineField*()*.length*()*!=0*) {* protein = Integer.*parseInt(*clientView.getProteineField*())*;  
 *}* int fat=0;  
 if*(*clientView.getFatField*()*.length*()*!=0*) {* fat = Integer.*parseInt(*clientView.getFatField*())*;  
 *}* int sodium=0;  
 if*(*clientView.getSodiumField*()*.length*()*!=0*) {* sodium = Integer.*parseInt(*clientView.getSodiumField*())*;  
 *}* double price=0;  
 if*(*clientView.getPretField*()*.length*()*!=0*) {* price = Double.*parseDouble(*clientView.getPretField*())*;  
 *}* if*(*rating==0 && calories==0 && protein==0 && fat==0 && sodium==0 && price==0*){* clientView.getComboBox*()*.removeAllItems*()*;  
 for*(*MenuItem i:menuItems*){* clientView.getComboBox*()*.addItem*(*i.getTitle*())*;  
 *}  
 }*else*{* List*<*MenuItem*>* temp=menuItems;  
  
 if*(*rating!=0*){* int finalRating = rating;  
 temp= temp.stream*()*.filter*(*x->x.getRating*()*== finalRating*)*.collect*(*Collectors.*toList())*;  
 *}* if*(*calories!=0*){* int finalCalories = calories;  
 temp= temp.stream*()*.filter*(*x->x.getCalories*()*== finalCalories*)*.collect*(*Collectors.*toList())*;  
 *}* if*(*protein!=0*){* int finalProtein = protein;  
 temp=temp.stream*()*.filter*(*x->x.getProtein*()*== finalProtein*)*.collect*(*Collectors.*toList())*;  
 *}* if*(*fat!=0*){* int finalFat = fat;  
 temp=temp.stream*()*.filter*(*x->x.getFat*()*== finalFat*)*.collect*(*Collectors.*toList())*;  
 *}* if*(*sodium!=0*){* int finalSodium = sodium;  
 temp=temp.stream*()*.filter*(*x->x.getSodium*()*== finalSodium*)*.collect*(*Collectors.*toList())*;  
 *}* if*(*price!=0*){* double finalPrice = price;  
 temp=temp.stream*()*.filter*(*x->x.getPrice*()*== finalPrice*)*.collect*(*Collectors.*toList())*;  
 *}* clientView.getComboBox*()*.removeAllItems*()*;  
 for*(*MenuItem i:temp*){* clientView.getComboBox*()*.addItem*(*i.getTitle*())*;  
 *}* temp=menuItems;  
 rating=0;calories=0;protein=0;fat=0;sodium=0;price=0;  
 *}  
 }  
}*class AddToCartListener implements ActionListener*{* @Override  
 public void actionPerformed*(*ActionEvent e*){* Object el=clientView.getComboBox*()*.getSelectedItem*()*;  
 for*(*MenuItem i:menuItems*){* if*(*el.equals*(*i.getTitle*())){* cart.add*(*i*)*;  
 clientView.showMessage*(*"Produsul "+el+" a fost adaugat in cos!"*)*;  
 *}  
 }  
 }  
}*class FinalizareListener implements ActionListener*{* @Override  
 public void actionPerformed*(*ActionEvent e*){* orderID++;  
 int minMon=1;  
 int maxMon=5;  
 int minDay=1;  
 int maxDay=30;  
 int mont=*(*int*)*Math.*floor(*Math.*random()*\**(*maxMon-minMon+1*)*+minMon*)*;  
 if*(*mont==1 || mont==3 || mont==5 || mont==7 || mont==8 || mont==10 || mont==12*){* maxDay=31;  
 *}* if*(*mont==2*){* maxDay=28;  
 *}* int day=*(*int*)*Math.*floor(*Math.*random()*\**(*maxDay-minDay+1*)*+minDay*)*;  
 LocalDate date=LocalDate.*of(*2022,mont,day*)*;  
 Random rand=new Random*()*;  
 int hour=rand.nextInt*(*24*)*;  
 int min=rand.nextInt*(*60*)*;  
 LocalTime time=LocalTime.*of(*hour,min*)*;  
 Order order=new Order*(*orderID,clientS.getUser*()*,date,time*)*;  
 finalOrder.put*(*order,cart*)*;  
 clientView.showMessage*(*"Comanda Finalizata!!"*)*;  
 FileWritter.*scrieIn(*cart,order*)*;  
 angajatView.setVisible*(*true*)*;  
 String s="DE PREPARAT: "+order.getTime*()*.getHour*()*+" "+order.getDate*()*.getDayOfMonth*()*+"\n";  
 for*(*MenuItem i:cart*){* s=s+i.getTitle*()*+"\n";  
 *}* angajatView.setTextArea*(*s*)*;  
 cart=new ArrayList*<>()*;  
  
 *}  
}*class backListener2 implements ActionListener *{* @Override  
 public void actionPerformed*(*ActionEvent e*){* cart=new ArrayList*<>()*;  
 clientView.showMessage*(*"Cosul a fost golit automat!"*)*;  
 logInView.setVisible*(*true*)*;  
 clientView.dispose*()*;  
 logInView.refresh*()*;  
 clientView.refresh*()*;  
  
 *}  
}*class CreateComposedListener implements ActionListener*{* @Override  
 public void actionPerformed*(*ActionEvent e*){* String nume= createComposedView.getNumeField*()*;  
 Object prod1=createComposedView.getProd1comboBox*()*.getSelectedItem*()*;  
 Object prod2=createComposedView.getProd2comboBox*()*.getSelectedItem*()*;  
 Object prod3=createComposedView.getProd3comboBox*()*.getSelectedItem*()*;  
 int rating=0;  
 int calories=0;  
 int protein=0;  
 int fat=0;  
 int sodium=0;  
 double price=0;  
  
 if*(*prod1==prod2 || prod1==prod3 || prod2==prod3*){* createComposedView.showMessage*(*"Alege produse diferite!!!"*)*;  
 *}*else*{* if *(*prod1 != null && prod2 != null && prod3 != null*) {* for *(*MenuItem i : menuItems*) {* if *(*prod1.equals*(*i.getTitle*())) {* rating = rating + i.getRating*()*;  
 calories=calories+i.getCalories*()*;  
 protein=protein+i.getProtein*()*;  
 fat=fat+i.getFat*()*;  
 sodium=sodium+i.getSodium*()*;  
 price=price+i.getPrice*()*;  
 *}* if *(*prod2.equals*(*i.getTitle*())) {* rating = rating + i.getRating*()*;  
 calories=calories+i.getCalories*()*;  
 protein=protein+i.getProtein*()*;  
 fat=fat+i.getFat*()*;  
 sodium=sodium+i.getSodium*()*;  
 price=price+i.getPrice*()*;  
 *}* if *(*prod3.equals*(*i.getTitle*())) {* rating = rating + i.getRating*()*;  
 calories=calories+i.getCalories*()*;  
 protein=protein+i.getProtein*()*;  
 fat=fat+i.getFat*()*;  
 sodium=sodium+i.getSodium*()*;  
 price=price+i.getPrice*()*;  
 *}  
 }  
 }*else*{* if*(*prod1!=null && prod2!=null*){* for *(*MenuItem i : menuItems*) {* if *(*prod1.equals*(*i.getTitle*())) {* rating = rating + i.getRating*()*;  
 calories=calories+i.getCalories*()*;  
 protein=protein+i.getProtein*()*;  
 fat=fat+i.getFat*()*;  
 sodium=sodium+i.getSodium*()*;  
 price=price+i.getPrice*()*;  
 *}* if *(*prod2.equals*(*i.getTitle*())) {* rating = rating + i.getRating*()*;  
 calories=calories+i.getCalories*()*;  
 protein=protein+i.getProtein*()*;  
 fat=fat+i.getFat*()*;  
 sodium=sodium+i.getSodium*()*;  
 price=price+i.getPrice*()*;  
 *}  
 }  
 }*else*{* if*(*prod1!=null && prod3!=null*){* for *(*MenuItem i : menuItems*) {* if *(*prod1.equals*(*i.getTitle*())) {* rating = rating + i.getRating*()*;  
 calories=calories+i.getCalories*()*;  
 protein=protein+i.getProtein*()*;  
 fat=fat+i.getFat*()*;  
 sodium=sodium+i.getSodium*()*;  
 price=price+i.getPrice*()*;  
 *}* if *(*prod3.equals*(*i.getTitle*())) {* rating = rating + i.getRating*()*;  
 calories=calories+i.getCalories*()*;  
 protein=protein+i.getProtein*()*;  
 fat=fat+i.getFat*()*;  
 sodium=sodium+i.getSodium*()*;  
 price=price+i.getPrice*()*;  
 *}  
 }  
 }*else*{* if*(*prod2!=null && prod3!=null*){* for *(*MenuItem i : menuItems*) {* if *(*prod2.equals*(*i.getTitle*())) {* rating = rating + i.getRating*()*;  
 calories=calories+i.getCalories*()*;  
 protein=protein+i.getProtein*()*;  
 fat=fat+i.getFat*()*;  
 sodium=sodium+i.getSodium*()*;  
 price=price+i.getPrice*()*;  
 *}* if *(*prod3.equals*(*i.getTitle*())) {* rating = rating + i.getRating*()*;  
 calories=calories+i.getCalories*()*;  
 protein=protein+i.getProtein*()*;  
 fat=fat+i.getFat*()*;  
 sodium=sodium+i.getSodium*()*;  
 price=price+i.getPrice*()*;  
 *}  
 }  
 }*else*{* createComposedView.showMessage*(*"Trebuie selectate cel putin 2 produse!!!"*)*;  
 *}  
 }  
 }  
 }* if*(*nume.length*()*==0*){* createComposedView.showMessage*(*"Te rog nu uita sa introduci nume!!"*)*;  
 *}*else *{* CompositeProduct compositeProduct = new CompositeProduct*(*nume, rating, calories, protein, fat, sodium, price*)*;  
 menuItems.add*(*compositeProduct*)*;  
 Serializator serializator=new Serializator*(*menuItems*)*;  
 serializator.realizareSerializare*()*;  
 createComposedView.showMessage*(*"Produsul compus " + nume + " creat si adaugat cu succes!!"*)*;  
 *}  
 }  
  
 }  
}*class backListener3 implements ActionListener *{* @Override  
 public void actionPerformed*(*ActionEvent e*){* administratorView.setVisible*(*true*)*;  
 createComposedView.setVisible*(*false*)*;  
 createComposedView.refresh*()*;  
 *}  
}*class DeleteListener implements ActionListener*{* @Override  
 public void actionPerformed*(*ActionEvent e*){* Object x=deleteView.getDeleteBox*()*.getSelectedItem*()*;  
 String str=*(*String*)*x;  
 List*<*String*>* elemente= Arrays.*asList(*str.split*(*" , "*))*;  
 for*(*MenuItem i:menuItems*){* if*(*elemente.get*(*0*)*.equals*(*i.getTitle*())* && Integer.*parseInt(*elemente.get*(*1*))*==i.getRating*()){* menuItems.remove*(*i*)*;  
 deleteView.showMessage*(*"Produsul "+elemente.get*(*0*)*+" a fost sters!!"*)*;  
 break;  
 *}  
 }* deleteView.getDeleteBox*()*.removeAllItems*()*;  
 for*(*MenuItem i:menuItems*){* String s=i.getTitle*()*+" , "+i.getRating*()*+" , "+i.getProtein*()*+" , "+i.getFat*()*+" , "+i.getSodium*()*+" , "+i.getPrice*()*;  
 deleteView.getDeleteBox*()*.addItem*(*s*)*;  
 *}* Serializator serializator=new Serializator*(*menuItems*)*;  
 serializator.realizareSerializare*()*;  
 *}  
}*class backListener4 implements ActionListener *{* @Override  
 public void actionPerformed*(*ActionEvent e*){* administratorView.setVisible*(*true*)*;  
 deleteView.setVisible*(*false*)*;  
 *}  
}*class ModifyListener implements ActionListener*{* @Override  
 public void actionPerformed*(*ActionEvent e*){* String nume=modifyView.getNameField*()*;  
 int ok=0;  
 for*(*MenuItem i:menuItems*){* if*(*i.getTitle*()*.equals*(*nume*)){* i.setRating*(*Integer.*parseInt(*modifyView.getRatingField*()))*;  
 i.setCalories*(*Integer.*parseInt(*modifyView.getClaoriiField*()))*;  
 i.setProtein*(*Integer.*parseInt(*modifyView.getProteineField*()))*;  
 i.setFat*(*Integer.*parseInt(*modifyView.getGrasimeField*()))*;  
 i.setSodium*(*Integer.*parseInt(*modifyView.getSodiuField*()))*;  
 i.setPrice*(*Double.*parseDouble(*modifyView.getPretField*()))*;  
 ok=1;  
 modifyView.showMessage*(*"Produsul "+nume+" a fost modificat cu succes!!"*)*;  
 Serializator serializator=new Serializator*(*menuItems*)*;  
 serializator.realizareSerializare*()*;  
 break;  
 *}  
 }* if*(*ok==0*){* modifyView.showMessage*(*"Produsul "+nume+" nu a fost gasit!"*)*;  
 *}  
 }  
}*class backListener5 implements ActionListener *{* @Override  
 public void actionPerformed*(*ActionEvent e*) {* administratorView.setVisible*(*true*)*;  
 modifyView.setVisible*(*false*)*;  
 modifyView.refresh*()*;  
 *}  
}*class addToListener implements ActionListener*{* @Override  
 public void actionPerformed*(*ActionEvent e*){* try *{* String nume = addView.getNameField*()*;  
 int ok = 0;  
 for *(*MenuItem i : menuItems*) {* if *(*nume.equals*(*i.getTitle*())) {* ok = 1;  
 addView.showMessage*(*"Produsul exista deja!!"*)*;  
 break;  
 *}  
 }* if *(*ok == 0*) {* if*(*nume.length*()*!=0*) {* int rating = Integer.*parseInt(*addView.getRatingField*())*;  
 int calories = Integer.*parseInt(*addView.getClaoriiField*())*;  
 int protein = Integer.*parseInt(*addView.getProteineField*())*;  
 int fat = Integer.*parseInt(*addView.getGrasimeField*())*;  
 int sodium = Integer.*parseInt(*addView.getSodiuField*())*;  
 double price = Double.*parseDouble(*addView.getPretField*())*;  
  
 BaseProduct product = new BaseProduct*(*nume, rating, calories, protein, fat, sodium, price*)*;  
 menuItems.add*(*product*)*;  
 Serializator serializator=new Serializator*(*menuItems*)*;  
 serializator.realizareSerializare*()*;  
 addView.showMessage*(*"Produsul a fost adaugat cu succes!!"*)*;  
 *}  
 }  
 }*catch *(*Exception ex*){}  
 }  
}*class backListener6 implements ActionListener *{* @Override  
 public void actionPerformed*(*ActionEvent e*){* administratorView.setVisible*(*true*)*;  
 addView.setVisible*(*false*)*;  
 addView.refreh*()*;  
 *}  
}*class generateRaport1 implements ActionListener*{* @Override  
 public void actionPerformed*(*ActionEvent e*){* File myFile=new File*(*"raport1.txt"*)*;  
 try *{* myFile.createNewFile*()*;  
  
 *}* catch *(*IOException ex*) {}* try *{* FileWriter myWriter=new FileWriter*(*"raport1.txt"*)*;  
 Map*<*Order, List*<*MenuItem*>>* filtred=finalOrder.entrySet*()*.stream*()*.  
 filter*(*x->x.getKey*()*.getTime*()*.getHour*()*>4*)*.collect*(*Collectors.*toMap(*p->p.getKey*()*,p->p.getValue*()))*;  
 filtred=filtred.entrySet*()*.stream*()*.  
 filter*(*x->x.getKey*()*.getTime*()*.getHour*()*<18*)*.collect*(*Collectors.*toMap(*p->p.getKey*()*,p->p.getValue*()))*;  
 filtred.forEach*((*order, menuItems1*)* -> *{* try *{* myWriter.write*(*order.getOrderID*()*+" Client: "+order.getClientUser*()*+"\n"+"Produse: "*)*;  
 for*(*MenuItem i:menuItems1*){* myWriter.write*(*i.getTitle*()*+" "*)*;  
 *}* myWriter.write*(*"\n"*)*;  
 *}* catch *(*IOException ex*) {}  
 })*;  
 myWriter.close*()*;  
 raportsView.showMessage*(*"Raport finalizat!!"*)*;  
 *}* catch *(*IOException ex*) {}  
  
 }  
}*class generateRaport2 implements ActionListener*{* @Override  
 public void actionPerformed*(*ActionEvent e*){* int times=Integer.*parseInt(*raportsView.getDayField*())*;  
 File myFile=new File*(*"raport2.txt"*)*;  
 try *{* myFile.createNewFile*()*;  
  
 *}* catch *(*IOException ex*) {}* try *{* FileWriter myWriter=new FileWriter*(*"raport2.txt"*)*;  
  
 for*(*MenuItem i:menuItems*){* AtomicInteger count= new AtomicInteger*(*0*)*;  
 Map*<*Order, List*<*MenuItem*>>* filtred=finalOrder.entrySet*()*.stream*()*.filter*(*x->x.getValue*()*.  
 contains*(*i*))*.collect*(*Collectors.*toMap(*p->p.getKey*()*,p->p.getValue*()))*;  
 filtred.forEach*((*order, menuItems1*)*->*{* for*(*MenuItem z:menuItems1*){* if*(*z.getTitle*()*==i.getTitle*()) {* count.getAndIncrement*()*;  
 *}  
 }  
 })*;  
 if*(*count.get*()*>times*){* myWriter.write*(*i.getTitle*()*+" "+i.getPrice*()*+"\n"*)*;  
 *}  
 }* myWriter.close*()*;  
 raportsView.showMessage*(*"Raport finalizat!!"*)*;  
 *}* catch *(*IOException ex*) {}  
  
 }  
}*class generateRaport3 implements ActionListener*{* @Override  
 public void actionPerformed*(*ActionEvent e*){* int amount=Integer.*parseInt(*raportsView.getDayField*())*;  
  
 File myFile=new File*(*"raport3.txt"*)*;  
 try *{* myFile.createNewFile*()*;  
  
 *}* catch *(*IOException ex*) {}* try *{* FileWriter myWriter=new FileWriter*(*"raport3.txt"*)*;  
  
 myWriter.close*()*;  
 raportsView.showMessage*(*"Raport finalizat!!"*)*;  
 *}* catch *(*IOException ex*) {}  
 }  
}*class generateRaport4 implements ActionListener*{* @Override  
 public void actionPerformed*(*ActionEvent e*){* int day=Integer.*parseInt(*raportsView.getDayField*())*;  
 File myFile=new File*(*"raport4.txt"*)*;  
 try *{* myFile.createNewFile*()*;  
  
 *}* catch *(*IOException ex*) {}* try *{* AtomicInteger count= new AtomicInteger*(*0*)*;  
 FileWriter myWriter=new FileWriter*(*"raport4.txt"*)*;  
 Map*<*Order, List*<*MenuItem*>>* filtred=finalOrder.entrySet*()*.stream*()*.filter*(*x->x.getKey*()*.getDate*()*.getDayOfMonth*()*==day*)*.  
 collect*(*Collectors.*toMap(*p->p.getKey*()*,p->p.getValue*()))*;  
 for*(*MenuItem i:menuItems*) {* filtred.forEach*((*order, menuItems1*)* -> *{* for *(*MenuItem z : menuItems1*) {* if *(*z.getTitle*()* == i.getTitle*()) {* count.getAndIncrement*()*;  
 *}  
 }  
 })*;  
 if*(*count.get*()*!=0*) {* myWriter.write*(*i.getTitle*()* + " " + count.get*()* + "\n"*)*;  
 count.set*(*0*)*;  
 *}  
 }* myWriter.close*()*;  
 raportsView.showMessage*(*"Raport finalizat!!"*)*;  
 *}* catch *(*IOException ex*) {}  
  
 }  
}*

* Pachetul dataAccess in care se afla doua clase care se ocupa de stabilitatea programului. Prima clasa si anume FileWritter este o clasa in care se realizeaza generarea de facturi in fisiere de tip .txt pentru fiecare comanda plasata.

public static void scrieIn(List<MenuItem> menuItems, Order order){

try {

double totalPrice=0;

i++;

FileWriter fw=new FileWriter("bill"+i+".txt");

fw.write("FACTURA "+"\n");

fw.write("Comanda cu ID-ul: "+order.getOrderID()+" care contine produsele:\n");

for(MenuItem i:menuItems){

fw.write("Produsul "+i.getTitle()+" cu rating "+i.getRating()+" la pretul de "+i.getPrice()+"\n");

totalPrice=totalPrice+i.getPrice();

}

fw.write("Totalul comenzii: "+totalPrice);

fw.close();

} catch (IOException e) {}

}

Cea de-a doua clasa si anume Serializtor , clasa care se ocupa implicit de stabilitatea programului si de serializarea datelor. Aceasta clasa se ocupa cu scrierea in fisier de tip .csv a noilor date obtinute in urma operatiilor efectuate , pentru a putea fi folosite la urmatoarea rulare a programului.

public void realizareSerializare*() {* dataLines.add*(*new String*[] {*"title","rating","calories","protein","fat","sodium","price"*})*;  
 for*(*MenuItem i:menuItems*){* dataLines.add*(*new String*[]{*i.getTitle*()*, String.*valueOf(*i.getRating*())*, String.*valueOf(*i.getCalories*())*,  
 String.*valueOf(*i.getProtein*())*, String.*valueOf(*i.getFat*())*, String.*valueOf(*i.getSodium*())*, String.*valueOf(*i.getPrice*())})*;  
 *}* try*{* FileWriter writer=new FileWriter*(*"E:\\PT2022\_30228\_Cindea\_Alexandru\_4\\products.csv"*)*;  
 for*(*String*[]* j:dataLines*){* for*(*String z:j*){* writer.append*(*z*)*;  
 writer.append*(*","*)*;  
 *}* writer.append*(*"\n"*)*;  
 *}* writer.flush*()*;  
 writer.close*()*;  
 *}*catch*(*Exception e*){}  
}*

5.Concluzii:

Aceasta tema te pune in postura de a crea o aplicatie existenta si folositoare din lumea real. Prin folosirea maparii de tip HashMap am invatat cum doua sau mai multe obiecte pot fi trimise si salvate simultan cu ajutorul expresiilor lambda si a procesarii stream care sunt foarte folositoare si usureaza munca programatorului foarte mult. Ca si implementari viitoare se poate incerca utilizarea observatorului pentru notificarea angajatului in timp real la fiecare modificare in HashMap-ul de comenzi pentru a nu mai fi nevoit sa trimiti singur datele mai departe.

6.Bibliografie:

<https://dsrl.eu/courses/pt/materials/A4_Support_Presentation.pdf>

https://stackoverflow.com/