**Documentație**

la disciplina

**Tehnici de programare**

**Tema 4 – Aplicație de catering**

**Popa Cristian Mihai, grupa 30224**

**An academic: 2021-2022**

Cuprins

[1. Obiectivul temei: 3](#_Toc104508850)

[2. Analiza problemei: 3](#_Toc104508851)

[3. Proiectare: 4](#_Toc104508852)

[4. Implementare: 7](#_Toc104508853)

[5. Concluzie: 10](#_Toc104508854)

[6. Bibliografie: 11](#_Toc104508855)

# Obiectivul temei:

* **Obiectiv principal:**

Obiectivul acestei teme consta in creare unui program care să simuleze o aplicație des întâlnita in ziua de astăzi, sub diferite forme. Aplicația constă în simularea unui sistem ce gestionează comenzile unui restaurant, fiind o aplicație de catering ce permite realizarea comenzilor de catre clienti.

Aplicația are 3 interfețe grafice pentru: administrator, client si angajat. Administratorul vede tot meniul care este pus la dispozaitia clientului, poate sa facă update la anumite produse, sa le stearga sau sa creze alte produse compuse. De asemenea poate vedeae diferite statistici despre clienti, produse si comenzi. Clientul poate comanda orice mancare doreste, sa caute anumite produse pe baza unor criterii si sa faca o comanda. Angajatul poate vedea toate comenzile plasate in vederea prelucarii comenzii.

* **Obiectiv secundare**:
  + Revizualizarea unor paradigme de programare orientata pe obiect: Interfețe, tehnici de reflecție, serializare, expresii lambda, stream-uri etc.;
  + Dezvoltarea unor clase de obiect adecvate: crearea unor clase potrivite pentru o persoana(in cazul nostru, client), produs și comandă, ce va conține informații generale despre client și despre produsele alese.
  + Dezvoltarea unor algoritmi eficienți si ușor de înțeles pentru prelucrarea in paralel a mai multor cereri;
  + Implementarea operațiilor si structuri de date specifice pentru a ne folosi de tehnicile de refelcție astfel încât programul sa funcționeze conform cerinței;
  + Implementarea unei interfețe grafice ușor de utilizat si ușor de înțeles;
  + Implementarea soluției pentru a avea un o aplicație ce simulează o aplicație din viața realăș

. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

# Analiza problemei:

Trăim într-o lume tehnologizată, putem sa obținem tot ceea ce ne dorim doar cu un singur click, iar după o perioadă de timp va ajunge la noi în fața ușii. Aplicația are ca scop simularea unei aplicații folosite pentru a plasa comenzi ce conțin produse alimentare. Aplicația va putea înregistra noi comenzi, noi produse compuse, noi clienti, iar clienții vor putea să plaseze diferite comenzi. Pentru a vedea cum funcționează acest sistem, utilizatorul are la dispoziție o interfață prietenoasă și ușor de utilizat ce este formată din 4 ferestre, cu nume sugestive: Login, Administrator, Client , Angajat:

* Login: interfata ce se apare la pornirea programului, interfata comuna de login pentru cele 3 tipuri de utilizator. De aici se pot inregistra si noi clienti. Inainte de login, trebuie sa se selecteze ce tip de user este cel care se loghheaza.
* Client: aici clientul poate comanda mâncare, având de ales dintr-o varietate mare de produse. Poate filtra produsele in funcție de: rating, calorii, grăsime, sodiu, proteine sau preț.
* Administrator: are drept de a modifica meniul prin: creare de produse noi sau compuse din cele existente, stergerea unui produs sau update. De asemenea poate obtine diferite statisitci cu privire la clienti(cine a comandat de x ori), produse(ce produs s-a comandat de x ori, sau in ce zi de cate ori), comenzi(ce clienti s-au facut intre anumite ore).
* Angajat: poate sa vizualizeze o comanda, cu informațiile aferente: preț, ce produse conține etc. in vederea preparării comenzii.

Pentru a funcționa corect, programul presupune introducerea datelor sub o anumita forma, ușor de înțeles. Câmpurile sunt sugestive, cu ce date trebuie introduse.

Programul verifica fiecare input al utilizatorului, deoarece pot apărea greșeli(de exemplu: utilizatorul introduce litere sau alte semne nepermise) . In cazul unei erori, utilizatorul va fi notificat.

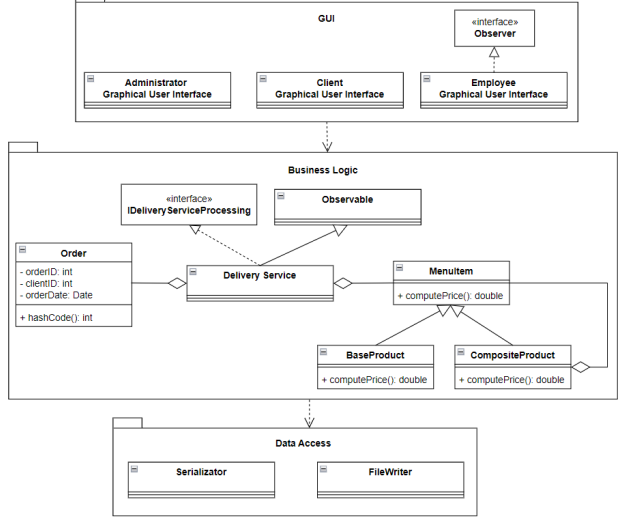
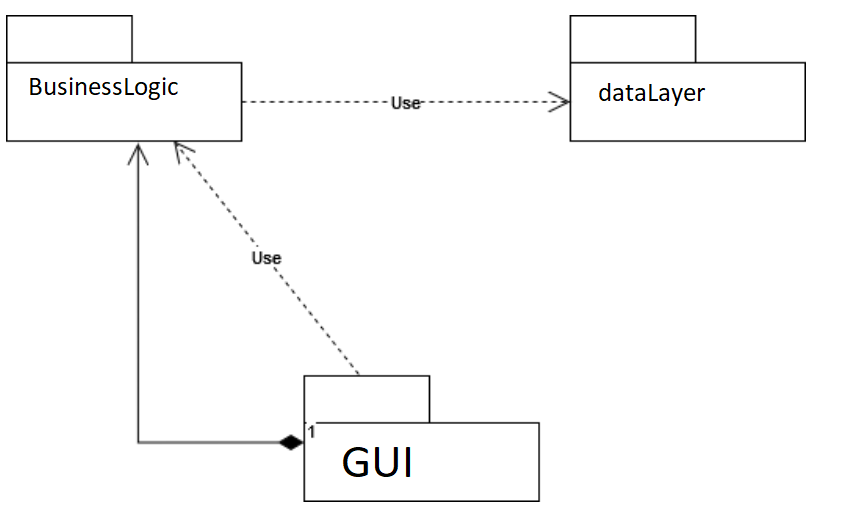
După introducerea datelor, în funcție de ceea ce dorește să introducă, poate purcede mai în a vizualiza și alte operații, cum ar fi ștergere sau update sau chiar crearea unei comenzi.. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

# Proiectare:

* **Alegerea unor clase de obiecte de baza adecvate:**
  + O comanda, descrisa de clasa Order va fi formata din unul sau mai multe produse și un singur client. Atributele ce le poseda aceasta clasa sunt: idComanda, idProdus si data si ora la care a fost realizata comanda.
  + Clientul este descris de clasa Client, ce poseda următoarele atribute: id, username și o paraola
  + BaseProdusct reprezinta produsul de baza, un fel de mancare(ex. cartofi prajiti cu piept de pui). Contine urmatoarele atribute: id, rating, calorii, grăsime, sodiu, proteine, preț si de cate ori a fost comandata.
  + Clasa CompositeProduct reprezinta felul de mancare compus din mai multe feluri de mancare de baza. Contine o lista de BaseProduct precum si un atribut de cate ori a fost comandat felul compus.
  + O interfata IDeliveryServiceProcessing ce se ocupa de prelucarea datelor, sta la baza aplicatiei. De aici se fac toate operatiile pentru toate cele 3 tipuri de utilziator si se gasesc si filtrele pentru statisticile administratorului.
  + Interfata MenuItem implenteaza functii pentru a prelucra campurile de id, rating, calorii, grăsime, sodiu, proteine, preț si de cate ori a fost comandata, si pentru BaseProduct si pentru CompositeProduct.
  + Aceste clase stau la baza programului, iar ceea ce conține fiecare este foarte intuitiv si ușor de înțeles. Clasele de obiecte au fost alese special astfel incat sa se poata serializa. Clasele de produse de mai sus implemteaza interfata Serializable
  + Serializarea consta in converitrea unui obiect intr-o serie de octeti, pentru manipularea mai usoara a datelor, iar ulterior sa fie refacut(deserializat) la loc. Astfel, putem manipula datele la nivel local, pentru un management mai bun al resurselor.
* **Împărțirea programului pe clase:**

Programul este împărțit 3 pachete mari:

* Pachetul GUI (Grafical User Interface) ce cuprinde 8 clase si o interfata, Controller si View pentru toate cele 4 ferestre mentionate mai sus. Clasa View este responsabila pentru crearea interfeței grafice, iar clasa Controller ne ajuta sa interacționăm cu Text-filed-urile si butoanele de pe Form;
* Pachetul Serialization conține o clasă ce este responsabilă de serializarea datelor cu care lucram, precum si salvarea lor la nivel local.
* Pachetul BussinesLogic conține clasele de produse, comanda si de prelucarea a comenzilor date de utilizator.
* In plus, mai avem o clasa numita „MainClass” ce nu aparține niciunei clase care deschide interfața si inițializează controller-ul.
* Împărțirea lor pe clase:
* O imagine care conține text

  Descriere generată automat
* Impartirea lor pe clase:
* 
* Impartirea lor pe pachete:
* 

. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

* **Algoritmi folosiți:**

Cum prelucrează programul datele din câmpurile date ca si input? Nu este foarte mare filozofie, întrucât în toate câmpurile se încearcă convertirea datelor introduse. In acest caz, utilizatorul va fi notificat de aplicație, repetând introduce datele din nou, dacă greșește. După ce datele sunt validate și prelucrate, se vor trimite datele spre serializare. Practic, simulam un fel de baza de date locala. Totusi, pentru prelucarea datelor. Avem nevoie de stream-uri iar datele vor fi extrase cu ajutorul expresiilor lambda(expresii de cod ce primesc parametrii si retureneaza o valoare, nu este necesar sa aiba nume, parameterii->expresie).

# Implementare:

In cele ce urmează, se vor prezenta modul in care au fost implementate clasele de obiecte, din cele 3 pachete. Nu vor fi incluse setter si getter, deoarece se subînțelege ca au fost folosite.

* **Pachetul Gui:**
  + Clasa View(interfata Login):

package GUI;

public class View extends JFrame{....}

Clasa View conține si inițializează toate componentele vizuale menționate mai sus: 2 butoane, 2 TextField-uri, 2 laber-uri si un comboBox pentru selectarea tipului de utilizator. Sunt plasate un panouri invizibile care la rândul lor aranjate intr-un mod specific in fereastra principala(putem considera ca sunt toate afisate sub forma de o coloana). Câteva inițializări si metode din clasa View:

* public String getFiled2Text(){  
   return text2.getText().toString();  
  }  
    
  public String getCombBoxSelectedItem(){  
   if(comb1.getSelectedIndex()==0)  
   return "";  
   else  
   return comb1.getSelectedItem().toString();  
  }
  + Clasa Controller: metoda de verifiacare la login pt. client
* ArrayList<Client> listaClienti = dsp.getClienti();  
  boolean ok=false;  
  for(Client c : listaClienti){  
   if(c.getUser().equals(user) && c.getPass().equals(pass)){  
   JFrame frame = new ClientView("Client", dsp, c);  
   frame.setDefaultCloseOperation(JFrame.*EXIT\_ON\_CLOSE*);  
   frame.setLocationRelativeTo(null);  
   frame.setResizable(false);  
   frame.setVisible(true);  
   view.setLabelStatus("Client logat cu succes! id: " + c.getClientId());  
   return;  
   }
  + Clasa AdminView, metoda de returnare a datelor din tabel:
* public ArrayList<MenuItem> getTable1Data(){  
   ArrayList<MenuItem> meniuUpdated = new ArrayList<>();  
    
   for(int i=0; i<adminController.getMenuSize();i++) {  
   meniuUpdated.add(new BaseProduct(table1.getValueAt(i,0).toString(),  
   Double.*parseDouble*(table1.getValueAt(i,1).toString()),  
   Integer.*parseInt*(table1.getValueAt(i,2).toString()),  
   Integer.*parseInt*(table1.getValueAt(i,3).toString()),  
   Integer.*parseInt*(table1.getValueAt(i,4).toString()),  
   Integer.*parseInt*(table1.getValueAt(i,5).toString()),  
   Integer.*parseInt*(table1.getValueAt(i,6).toString())));  
   }  
   return meniuUpdated;  
  }
  + Clasa AdminControler:
* case "ModificaProdus":  
  {  
   System.*out*.println("ModificaProdus");  
   ArrayList<MenuItem> updatedProducts=adminView.getTable1Data();  
   ArrayList<MenuItem> oldMenu = dsp.getMeniu();  
   int cont=0;  
   for(MenuItem mi : oldMenu){  
   MenuItem miiix = updatedProducts.get(cont);  
   if(!mi.equals(miiix))  
   {  
   dsp.updateProduct(mi.getTitle(), updatedProducts.get(cont));  
   System.*out*.println(mi.toString());  
   }  
   cont++;  
   }  
   adminView.initTables();  
  }break;

Analog pot fi descrise si celelate clase de View si Controller pentru client si angajat. De asemenea avem si o interfata ObservableGUI, interfata simpla ce contine doar o funcție. Este folosita pentru notificarea angajaților cand primesc o comanda noua. Functia este descrisa in EmployController.

public interface ObservableGUI {  
 public void update();  
}

* **Pachetul DataAcces:**
  + Clasa Serialization:

Aceasta clasa se ocupa cu salvarea si receptarea serializarii din fisierul local. Din fisierul local ,,dsp.ser” vor fi citite date la inceperea aplciatiei iar la sfrasit se vor salva toate modificarile aduse facute din aplicatiei pe parcurs(la clienti, produse sau comenzi). Mai jos este metoda de extragere a datelor din fisierul local.

public static DeliveryServiceProcessing retrieveDsp(String file) {  
 DeliveryServiceProcessing ret = null;  
 try {  
 FileInputStream fin = new FileInputStream(file);  
 ObjectInputStream obin = new ObjectInputStream(fin);  
 ret = (DeliveryServiceProcessing) obin.readObject();  
 obin.close();  
 fin.close();  
 }  
 catch (Exception e) { e.printStackTrace();}  
 return ret;  
}

* **Pachetul BussinesLogic:**

Acest pachet contine clasele ce stau la baza aplciatiei: BaseProduct, CompositePorduct, Client, Order, Observable, DeliveryServiceProcessing si doua interfete. Aici se afla clasele care vor fi serialzate. Mai jos vom expune clasele mai interesante precum si fragmente de cod mai importante.

* + Clasa DeliveryServiceProcessing:

In aceasta clasa sunt stocate date esentiale, cum ar fi meniul, lista de comenzi etc. Aici sunt implementate si metodele de importare meniu, ștergere produs, update sau creare client, creare comanda. Totodată, avem filtrele folosite pentru sortarea produselor, comenzilor sau a clienților. Filtrele se folosescc de stream-uri si expresii labda pentru a obtine rezultatul dorit.

Functie ce creaza o comanda noua:

public Order createOrder(int clientId, ArrayList<MenuItem> listaProduse, ObservableGUI obs) {  
 assert obs != null;  
 assert listaProduse != null;  
 Order newOrder = new Order(clientId);  
 comanda.put(newOrder, listaProduse);  
 notifyObserver(obs);  
  
 for (MenuItem mi : listaProduse){  
 for(MenuItem mix : meniu){  
 if(mi.getTitle() == mix.getTitle())  
 mix.setOrdersNumbers(mix.getOrdersNumbers()+1);  
 }  
 }  
 // mi.setOrdersNumbers(mi.getOrdersNumbers()+1);  
  
 assert newOrder != null;  
 return newOrder;  
}

Metoda filtru pentru a filtra produsele dupa criteriile mentionate mai sus, din interfata clientului:

public ArrayList<MenuItem> findByCriteria  
 ( String title,  
 Double lrating,Double hrating,  
 int lcalories, int hcaloreis,  
 int lprotein,int hprotein,  
 int lfat,int hfat,  
 int lsodium,int hsodium,  
 int lprice,int hprice  
 )  
{  
 ArrayList<MenuItem> ret = (ArrayList<MenuItem>) meniu.stream().filter(  
 s -> s.getRating() >= lrating && s.getRating() <= hrating  
 && s.getCalories() >= lcalories && s.getCalories() <= hcaloreis  
 && s.getProtein() >= lprotein && s.getProtein() <= hprotein  
 && s.getFat() >= lfat && s.getFat() <= hfat  
 && s.getSodium() >= lsodium && s.getSodium() <= hsodium  
 && s.getPrice() >= lprice && s.getPrice() <= hprice  
 && s.getTitle().trim().contains(title.trim())  
 ).collect(Collectors.*toList*());  
 return ret;  
}

Functie pentru a gasi o comanda in functie de ziua in care a fost creata:

public ArrayList<MenuItem> findOrderByDay(int day)  
{  
 ArrayList<Order> allOrders = (ArrayList<Order>) comanda.keySet().stream().filter(s -> s.getDate().getDayOfMonth() == day).collect(Collectors.*toList*());  
 HashSet<MenuItem> ret = new HashSet<>();  
 for (Order o : allOrders)  
 ret.addAll(comanda.get(o));  
 return new ArrayList<>(ret);  
}

# Concluzie:

Aplicația dezvoltata respecta paradigmele programării orientate pe obiect și a manipulării stream-urilor si a expresiilor lambsa, si este un bun reminder al cunoștințelor învățate sem. trecut precum si acest semestru. Sistemul creat gestionează corespunzător o aplicație ce poate suporta mai multe operații cu privire la manipularea comenzilor efectuate de la clienti. Se aseamenă cu aplicațiile ce se regăsesc în viața reală(aplicații de food delivery de pe telefon de exemplu). Aceasta tema ne dovedeste adevarata putere de manipularea ce o putem avea asupra datelor daca lucram cu serializare.

Deși este simpla aplicația si își deservește scopul, totuși este loc de îmbunătățiri si upgrade-uri. Aplicația poate avea pe viitor o metodă de creare a produselor de baza, din interfata adminului. De asemenea, se pot programa statistici mai complexe, cum ar fi vanzarile lunare, angajatul cu cele mai multe comenzi realizate. Se poate implementa si un istoric al comenzilor care va aparea in interfata utilizatorului.

Per total, aplicatia indeplineste cerintele mentionate mai sus, deservindu-și scopul de a simula o aplicatie din viata reala.

# Bibliografie:

<https://docs.oracle.com/en/>

<https://www.baeldung.com/java-write-to-file>

https://www.baeldung.com/java-jdbc

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

<https://www.jetbrains.com/help/idea/working-with-code-documentation.html>

<https://www.baeldung.com/java-pdf-creation>

<https://jenkov.com/tutorials/java-reflection/index.html>

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

<https://www.tutorialspoint.com/java/java_serialization.htm>

https://docs.oracle.com/javase/8/docs/technotes/guides/language/assert.html

http://tutorials.jenkov.com/java-concurrency/creating-and-starting-threads.html