**Urmatoarele capitole sintetizeaza continutul documentatiei :**

**Capitolul 1.Obiectivul temei (principal)**

Obiectivul principal al lucrarii este implementarea unei aplicatii de procesare a activitatilor dintr-un restaurant desfasurate de un administrator , un chelner si un bucatar folosind paradigma de programare POO , design patterns discutate in capitolele ce urmeaza , design by contract ,serializare si colectii din framework-ul java.

**1.1.Obiectivele secundare sunt adresate in urmatoarele capitole :**

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

Cadrul de cerinte functionale e realizat de adaugarea, editarea si stegerea unui element din meniu de catre administrator , crearea unei comenzi si generarea unei factori de catre ospatar si vizualizarea produselor compuse pe care bucatarul trebuie sa le prepare . Se vor prezenta descrieri de use-case impreuna cu lista pasilor de executie a fiecarui use-case.

**Cap.3 : Proiectarea diagramelor UML , a algoritmilor si intefetelor utilizator**

Se vor prezenta decizii legate de proiectarea claselor , structurilor de date, interfetelor utilizate in descrierea problemei , precum si a relatiile dintre acestea , organizarea in pachete precum si diagramele de clase si pachete utile in modelarea OOP pentru prelucrarea elementelor de meniu si comenzi.

**Cap.4 : Implementarea propriu-zisa**

Prezentarea campurilor , constructorilor , metodelor importante care realizeaza corespondenta OOP-simularea activitatii de restaurant. Se detaliaza de ce s-a ales aceasta implementare .Tot aici se va prezenta interfata cu utilizatorul si cateva exemple de utilizare.

**Cap5. : Concluzii**

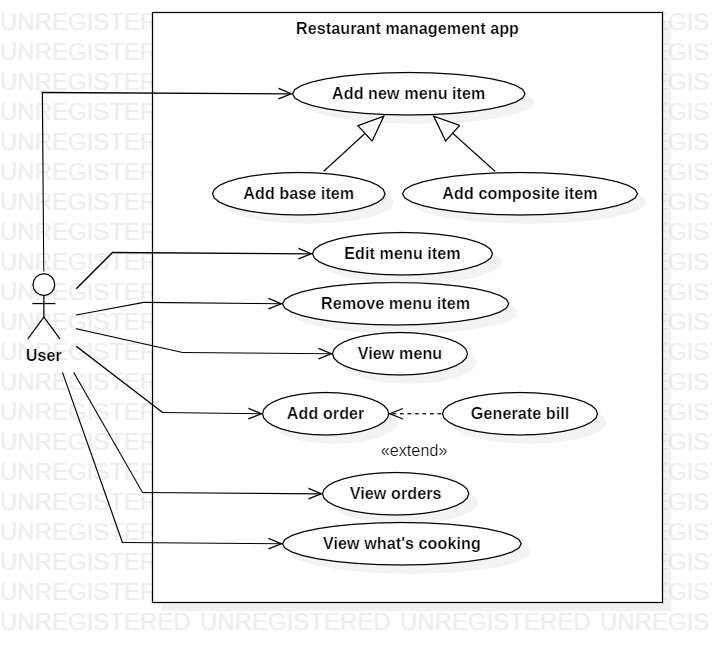
In acest capitol , se discuta concluzii , cunostiinte dobandite , precum si dezvoltari posibile ulterioare.

**Cap6. : Bibliografie**

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

Se prezinta in continuare functionalitatile oferite de aplicatie sub forma unei diagrame ce prezinta principalele cazuri de utilizare.

Diagrama contine 6 cazuri principale de utilizare si toate rezuma operatiile pe care un utilizator (bucatar , ospatar, administrator) le poate efectua cu ajutorul aplicatiei. Cazurile principale sunt adaugarea , eliminarea ,vizualizarea detaliata si editarea unui element simplu sau compus din meniu ( operatii specifice unui administrator , se va vedea in capitolele urmatoare ca aceste operatii sunt grupate intr-o interfata care specifica faptul ca e destinata utilizarii de catre administratori de restaurante) , adaugarea unei comenzi , generarea unei facturi (operatii specifice unui ospatar) si vizualizarea comenzilor ce contin produse compuse din meniu (destinate unui bucatar) .



Se observa ca pe langa cele 6 cazuri principale mai intalnim 3 cazuri secundare ( 2 dintre ele : “Add base item “ si “Add composite item” sunt specializari ale cazului principal de adaugare de element nou in meniu, iar al treilea : “Generate bill” apare doar cand utilizatorul doreste acest lucru ; nu intotdeauna un ospatar genereaza factura imediat dupa preluarea comenzii ).

Se prezinta cazul adaugarii si editarii unui element de meniu compus , cel de stergere fiind sugestiv iar cel de adaugare de comanda fiind similar cu cel de adaugare de element de meniu ,iar cazul generarii de factura apare doar la apasarea unui buton fara alte actiuni .

**• Use Case: Add composite item**

**• Primary Actor: Utilizatorul aplicatiei**

**• Main Success Scenario:**

1. Utilizatorul selecteaza elemente de meniu ( pot fi atat simple cat si compuse)
2. Apasa butonul “add composite” care deschide o fereastra cu campuri de completat
3. Introduce datele noului element compus de meniu (nume,cantitate)
4. Datele sunt verificate (nume = String, cantitate = numar intreg)
5. Daca pasii 1-4 s-au efectuat cu succes ,utilizatorul apasa butonul de “add” si elementul de meniu creat se adauga in tabel

**• Alternative Sequences:**

**Case 1:**

1. Utilizatorul nu selecteaza niciun element din meniu si apasa butonul de adaugare de element compus
2. Se afiseaza un mesaj corespunzator (“No row was selected”)
3. Utilizatorul e ghidat spre tabelul cu elementele de meniu

**Case 2:**

1. Utilizatorul nu introduce date valide
2. Se afiseaza un mesaj corespunzator (“invalid field”)
3. Utilizatorul e ghidat spre tabelul cu elementele de meniu

**Case 3:**

1. Utilizatorul introduce un element care exista deja in meniu
2. Se afiseaza un mesaj corespunzator (“item already exists”)
3. Utilizatorul e ghidat spre tabelul cu elementele de meniu

**• Use Case: Edit menu item**

**• Primary Actor: Utilizatorul aplicatiei**

**• Main Success Scenario:**

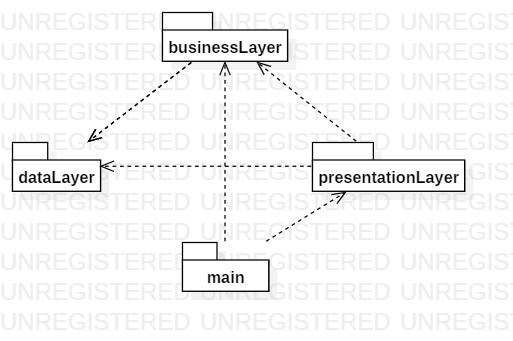
1. Utilizatorul selecteaza 1 element din meniu
2. Apasa butonul “edit” care deschide o fereastra cu campuri completate deja de valorile elementului de editat din tabel
3. Introduce noile valori pentru elementul selectat
4. Datele sunt verificate
5. Daca pasii 1-4 s-au efectuat cu succes ,utilizatorul apasa butonul de “edit” si elementul se actualizeaza in tabel

**• Alternative Sequences:**

1. Utilizatorul nu selecteaza niciun element sau selecteaza mai mult de 1 element pentru editare
2. Se afiseaza un mesaj corespunzator “ select only 1 row”
3. Utilizatorul e ghidat spre tabelul interfetei

**Capitolul 3. Proiectare (decizii de proiectare, diagrame UML, structuri de date, proiectare clase, interfete, relatii, packages, algoritmi, interfata utilizator)**

Proiectul respecta urmatoarea arhiectura de proiectare fiind impartit in 4 pachete : businessLayer, dataLayer, main, presentationLayer pentru a facilita modularizarea si eventuala extindere a proiectului . Acest lucru e tradus in urmatoarea diagrama de pachete:



**Pachetul** **“main”** contine o singura clasa numita MainClass care serveste pornirii aplicatiei prin deschiderea intefetelor grafice prezente in “presentationLayer” (imprima vizual implementarea metodelor din “businessLayer” care foloseste date si metode din “dataLayer”).

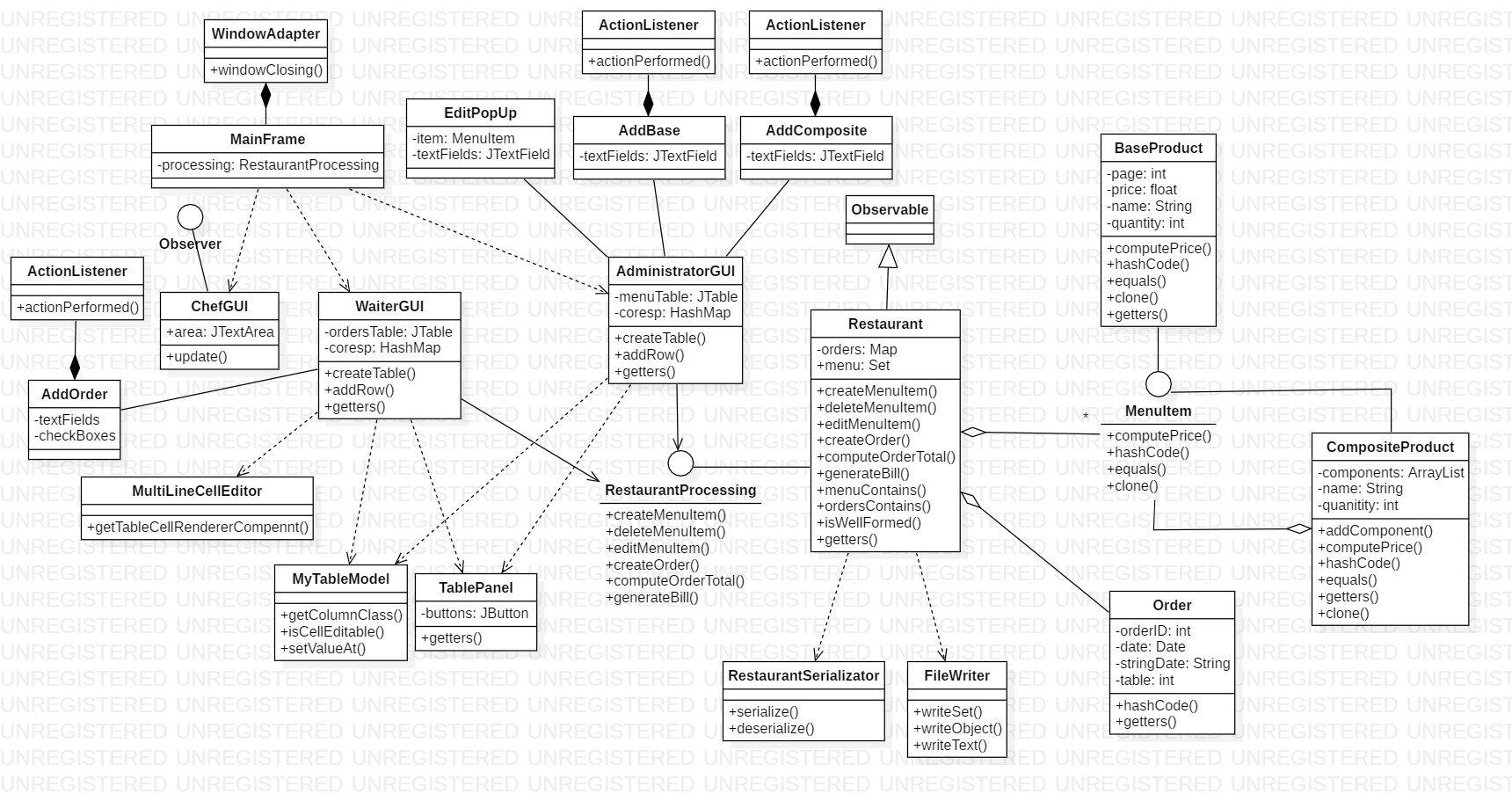
**Pachetul “businessLayer”** contine clase care modeleaza entitatile cu care se fac operatii : MenuItem, BaseProduct, CompositeProduct , Order . Tot aici intalnim interfata RestaurantProcessing care contine metode asociate cu cazurile de utilizare prezentate mai sus si o clasa Restauarant care implementeaza aceste metode si tine o corespondenta cu elementele din meniu si comenzile efectuate.

**Pachetul “dataLayer”** contine 2 clase importante : FileWriter si RestaurantSerializator care contin metode statice ceea ce le face reutilizabile si independente . FileWriter este utilizat pentru lucrul cu fisiere si RestauratSerializator contine o metoda de deserializare a unui fisier care incarca meniul aplicatiei cu obiecte serializate acolo de la ultima actualizare si o metoda de serializare pentru actualizarea fisierului.’

**Pachetul “presentationLayer”** contine 11 clase care descriu interfata graficasi pot fi grupate in functie de utilizatorul curent al aplicatiei in felul urmator :

* Clasele : AddBase, AddComposite, AdministratorGUI, EditPopUp descriu actiunile specifice unui administrator
* Clasele : AddOrder, WaiterGUI descriu cele specifice unui ospatar
* Clasa ChefGUI descrie comenzile pe care bucatarul le primeste pentru gatit
* Clasele :TablePanel , MyTableModel ,MultiLineCellEditor sunt folosite de aproape fiecare interfata individuala si acestea permit adaugarea mai multor elemente ( checkboxes in cazul acestei aplicatii) intr-o celula de tabel
* Clasa MainFrame : este prima interfata cu care se intalneste utilizatorul si contine 3 butoane care pot conduce utilizatorul spre una din celelalte 3 interfete : pentru Administrator , Waiter sau Chef

**Toate aceste pachete si clasele continute prezinta relatiile descrise de urmatoarea diagrama de clase:**



**Cap.4 : Implementarea propriu-zisa**

Urmeaza detalii legate de implementarea celor mai importante clase , deoarece cateva prezinta similaritati . Se ia pentru explicatie urmatoarele cazuri de functionare : adaugarea unui element compus si generarea unei comenzi cu acesta , se explica fiecare pas prin care trece un utilizator pentru a ajunge la rezultatul dorit ,facandu-se legetura cu portiunile de cod care realizeaza acesti pasi.

**Pasul 1.**

Utilizatorul deschide aplicatia . Acest lucru se realizeaza cu metoda main care creaza un obiect de tip Restaurant pe care se pot efectua operatii si fixeaza prima interfata a aplicatiei ( de selectie a utilizatorului :”frame”) .Totodata aici se adauga ascultator restaurantului interfata grafica a bucatarului . Se observa astfel utilizarea design pattern-ului Observer :

Restaurant restaurant=**new Restaurant();**

**MainFrame frame=new MainFrame(restaurant);**

**restaurant.addObserver(frame.getGui());**

La crearea unui obiect de tip Restaurant va avea loc deserializarea (popularea cu datele ultimei actualizari) a meniului din obiectul creat . Acest lucru se traduce in constructorul clasei Restaurant si a metodei statice deserialize() din RestaurantSerializator :

Set<Object> set=new HashSet<Object>();  
FileInputStream file= null;  
ObjectInputStream objectReader=null;  
try {  
 file = new FileInputStream(*fileName*);  
 objectReader = new ObjectInputStream(file);  
 set=(Set<Object>)objectReader.readObject();  
} catch (FileNotFoundException e) {  
 try {  
 FileOutputStream tmp=new FileOutputStream(*fileName*);  
 tmp.close();  
 } catch (IOException ex) {  
 ex.printStackTrace();  
 }  
} catch (IOException e) {  
 e.printStackTrace();  
} catch (ClassNotFoundException e) {  
 e.printStackTrace();  
}finally {  
 if(file!=null && objectReader!=null) {  
 try {  
 file.close();  
 objectReader.close();  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
}  
return set;

**Pasul 2.**

Intrucat am stabilit mai sus ca vom efectua o adaugare de element compus de meniu , trebuie selectata interfata cu administratorul care va fi deschisa in modul de mai jos din MainFrame si va prezenta urmatorul tabel create cu metoda createTable din AdministratorGUI :

Deschiderea din MainFrame :

public **MainFrame(final RestaurantProcessing processing)**

...

new AdministratorGUI(processing);

Crearea tabelului pentru vizualizarea elementelor de meniu :

String[] columnNames = new String[]{"Name","Composed by","Quantity", "Price" , "Check" };

DefaultTableModel tableModel = new MyTableModel(columnNames);

JTable table = new Jtable(tableModel);

for (Iterator<MenuItem> it=((Restaurant)processing).getMenu().iterator(); it.hasNext();)

{

MenuItem item=it.next(); coresp.put(tableModel.getRowCount(),item); addRow(tableModel,item);

}

table.setRowHeight(30);

table.setDefaultEditor(Object.class, null);

return table;

**Pasul 3.**

Pentru a adauga un element de meniu compus trebuie apasat butonul de “Add composite” care creeaza un obiect de tip AddComposite ( extinde Jframe ) ce reprezinta interfata de completare cu valori pentru elementul nou de adaugat :

new AddComposite(frame);

Aici se pot selecta alte elemente de meniu din cele existente pentru a compune noul item :

Acest lucru este evidentiat aici :

String name=jtf1.getText();int quantity=Integer.*parseInt*(jtf2.getText());CompositeProduct item=new CompositeProduct(name,quantity);for(int i=0;i<size;i++){ if((Boolean)frame.getMenuTable().getValueAt(i,4)){ item.addComponent(frame.getCoresp().get(i)); }}boolean can=frame.getProcessing().createMenuItem(item);if(can) { frame.addRow((DefaultTableModel) frame.getMenuTable().getModel(), item); frame.getCoresp().put(frame.getMenuTable().getRowCount(), item);}

Se creeaza un element nou de tip CompositeProduct (item) , se parcurg elementele selectate din meniu , se apeleaza metda addComponent pe item si se verfica in utlima instanta daca elementul nou de adaugat exista sau nu in meniu ( acest lucru se poate verifica usor deoarece meniul este implementat ca un Set care nu permite existenta duplicatelor in structura sa :

Set<MenuItem> menu;

...

if(!menuContains(item)) { menu.add(item);

...

Dupa adaugare structura tabelului se modifca (se adauga practic o noua linie ) cu ajutorul metodei addRow din AdministratorGUI :

ArrayList<Object> oneRowData = new ArrayList<Object>();oneRowData.add(0,item.getName());String[] composed=item.getComposedBy();StringBuilder stringBuilder=new StringBuilder();for(int j=0;j<composed.length;j++){ stringBuilder.append(composed[j]); stringBuilder.append(",");}Object[] oneRowDataArray = new Object[oneRowData.size()];oneRowDataArray=oneRowData.toArray(oneRowDataArray);model.addRow(oneRowDataArray);

**Pasul 4.**

In acest pas se poate efectua o comanda cu noul element compus (implementarea adusa va notifica bucatarul: ChefGUI mai exact doar in cazul in care comanda contine cel putin un element de meniu selectat compus de catre un ospatar(WaiterGUI) :

public void createOrder(Order o,ArrayList<MenuItem> list) { orders.put(o,list); for(int i=0;i<list.size();i++){ if(list.get(i) instanceof CompositeProduct){ this.setChanged(); this.notifyObservers(list.get(i)); } }

La notificarea bucatarului se apeleaza metoda update din ChefGUI care va adauga la finalul unui JtextArea un text transmis ca parametru astfel :

public void update(Observable o, Object arg) {  
 if(arg instanceof CompositeProduct){  
 *count*++;  
 area.append(*count*+". "+"To cook : ");  
 area.append(((CompositeProduct) arg).getName()+"\n");  
 }  
 }

**Influenta documentarii javadoc**

Implementarea propusa impune respectarea unor conditii la efectuarea operatiilor pe restaurant , conditii specificate si in interfata RestaurantProcessing care face trimite la respectarea unui design by contract prin adnotari , algebra booleana si invarianti de clasa :

Invariantul clasei va impune respectarea urmatoarelor conditii :

* meniul si orders trebuie instantiate altfel assert-ul va arunca o exceptie
* daca meniul contine elemente nule se arunca acelasi tip de exceptie
* daca in comenzi exista perechi <Comanda,Elemente de meniu> si unul din membrii perechii este null , se arunca aceeasi exceptie

protected boolean isWellFormed(){  
 if(menu==null || orders==null)  
 return false;  
 for(Iterator<MenuItem> it=menu.iterator();it.hasNext();){  
 if(it.next()==null){  
 return false;  
 }  
 }  
 for(Map.Entry<Order,ArrayList<MenuItem>> entry:orders.entrySet()){  
 if(entry.getValue()==null || entry.getKey()==null){  
 return false;  
 }  
 }  
 return true;

**Capitolul 6. Concluzii**

Subiectul proiectului este complex , si prezinta multe posibilitati de implementare insa varianta aleasa a adus urmatoarele cunostiinte dobandite : realizarea de diagrame de pachete, diagrame use case si de clase , folosirea design paterrn-urilor Composite si Observer , intelegerea modului in care sunt serializate obiectele, crearea de pre, post conditii, invarianti de clase, intelegerea modului de lucru cu frameworkul Java Collections si cel mai important : realizarea corespunzatoare a unui proiect Java de la proiectare pana la implementare ,testare si crearea unei documentatii.

Proiectul poate fi dezvoltat in multe directii datorita structurarii pe pachete insa propunerea adusa ar fi modificarea interfetei si implementarii astfel incat sa suporte mai multi ospatari si bucatari sau chiar crearea unei aplicatii care sa gestioneze mai multe aplicatii de management de restaurante intr-un lant de restaurante in care aceasta aplicatie sa fie suboordonata.

**Capitolul 7:Bibliografie**

* JtableCellRenderer si MyTableModel :

**https://stackoverflow.com/questions/9955595/how-to-display-multiple-lines-in-a-jtable-cell**

* Metodele de serializare au fost inspirate din:

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