

## TEHNICI DE PROGRAMARE

# RESTAURANT MANAGEMENT SYSTEM TEMA 4

## DOCUMENTAȚIE

Student: Nemes Emilia

Grupa: 30228

Profesor: Pop Cristina

## Cuprins

1.	Obiectivul temei .....	3
1.1	Obiectivul principal.....	3
1.2	Obiectivele secundare.....	3
2.	Analiza problemei.....	4
2.1	Cazuri de utilizare.....	4
3.	Proiectare.....	5
3.1	Diagramă UML.....	5
3.1.1	Diagramă de pachete.....	5
3.1.2	Diagrame de clase.....	5
3.2	Proiectare pachete.....	7
3.3	Structuri de date.....	7
3.4	Interfața utilizator.....	8
4.	Implementare.....	9
4.1	Pachetul <i>main</i> .....	9
4.2	Pachetul <i>presentationLayer</i> .....	10
4.5	Pachetul <i>dataLayer</i> .....	10
4.6	Pachetul <i>businessLayer</i> .....	11
5.	Rezultate.....	13
6.	Concluzii.....	13
7.	Bibliografie.....	13

## 1. Obiectivul temei

### 1.1 Obiectivul principal

Proiectați o aplicație numită Restaurant Management System. Aplicația trebuie să aibă trei tipuri de utilizatori: administrator, chelner (waiter) și bucătar-șef (chef). Administratorul poate să insereze, să șteargă și să modifice produse existente din meniu. Chelnerul poate să creeze o comandă nouă pentru o masă, să-i adauge elemente din meniu, și să genereze nota de plată pentru o comandă. Bucătarul-șef este notificat de fiecare dată când ceva produs este comandat.

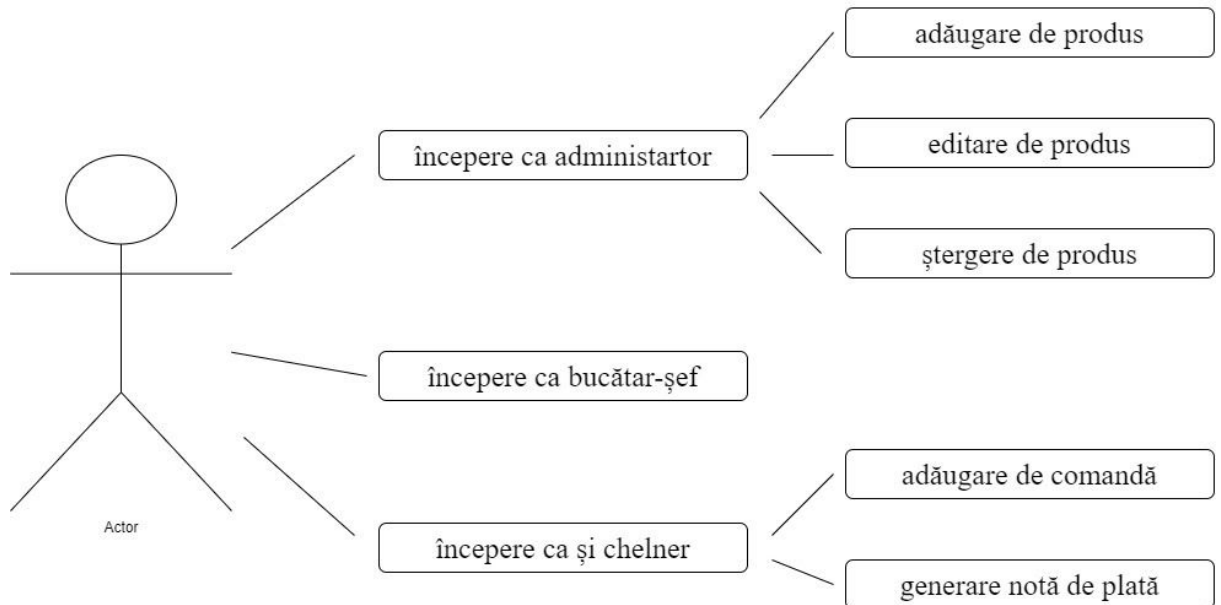
Se consideră că aplicația are un singur utilizator de tip *administrator*, de tip *waiter* și de tip *chef*.

### 1.2 Obiectivele secundare

Nr.	Obiectivul secundar	Descriere	Capitolul
1.	Dezvoltarea unui caz de utilizare și a scenariilor posibile	Cazul de utilizare descrie secvența de interacțiune între utilizator și aplicație, aici se descrie modul de utilizare a aplicației și a cazurilor posibile de funcționare a acesteia	2
2.	Decizii de proiectare	Decizii asupra modului de proiectare cu scopul de a implementăm proiectul	3
3.	Alegerea structurilor de date	Structurile de date pe care le vom folosi în cadrul proiectului	3
4.	Proiectarea claselor	Enumerarea claselor pe care le vom implementa și descrierea rolurilor pentru fiecare clasă în parte	3
5.	Interfața utilizator	Descrierea interfeței utilizator, și a funcționării acesteia	3
6.	Implementarea soluției	Cum am implementat soluția, detalii despre cod	4

## 2. Analiza problemei

### 2.1 Cazuri de utilizare



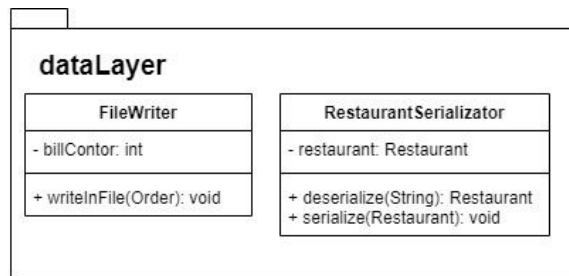
Proiectul realizat este o aplicație pentru sistemul de management al unui restaurant.

Actorul, adică utilizatorul acestei aplicații poate fi de trei tipuri: administrator, chelner sau bucătar-șef. Administratorul poate să adauge meniului produse noi, poate să editeze produse deja existente sau poate să ștergă produse din meniu. Chelnerul poate să adaugă comenzi noi, sau poate să genereze nota de plată pentru comanda selectată. Bucătarul-șef poate să vadă ce fel de produse trebuie să pregătească.

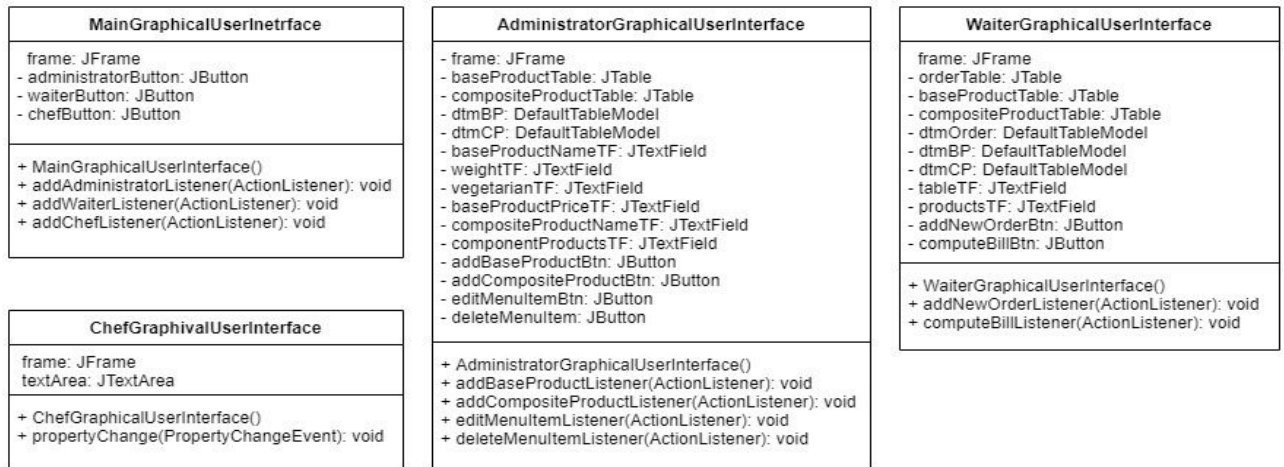
Descrierea cazurilor de utilizare:

- în momentul lansării programului apare interfața grafică principală, în care se poate alege tipul de utilizator
- dacă se alege “Administrator”, atunci utilizatorul poate să introducă produse noi, ori de tipul *Base Product*, ori de tipul *Composite Product*; utilizatorul trebuie să introducă proprietățile produsului în câmpurile din interfața grafică, iar la apăsarea butonului *Add Base Product* produsul inserat se introduce în meniu
- administratorul mai poate să editeze produsele deja existente: utilizatorul trebuie să apasă pe produsul (rândul produsului) pe care vrea să-l editeze; proprietățile produsului vor apărea în câmpurile de text de lângă tabele, unde pot fi modificate; când vrea să salveze modificările trebuie să apasă pe butonul *Add Base Product* sau *Add Composite Product*, în funcție de tipul produsului editat
- dacă administratorul vrea să ștergă un produs selectează din tabelă rândul acestuia, și se apasă pe butonul *Delete Menu Item*
- dacă se alege “Waiter” se poate introduce un order nou, prin completarea de la tastatură a câmpurilor *Table* și *Products*; *Table* va conține numărul mesei care face comanda, iar în *Products* se introduc numele produselor comandate (cum apar în tabelul alăturat)

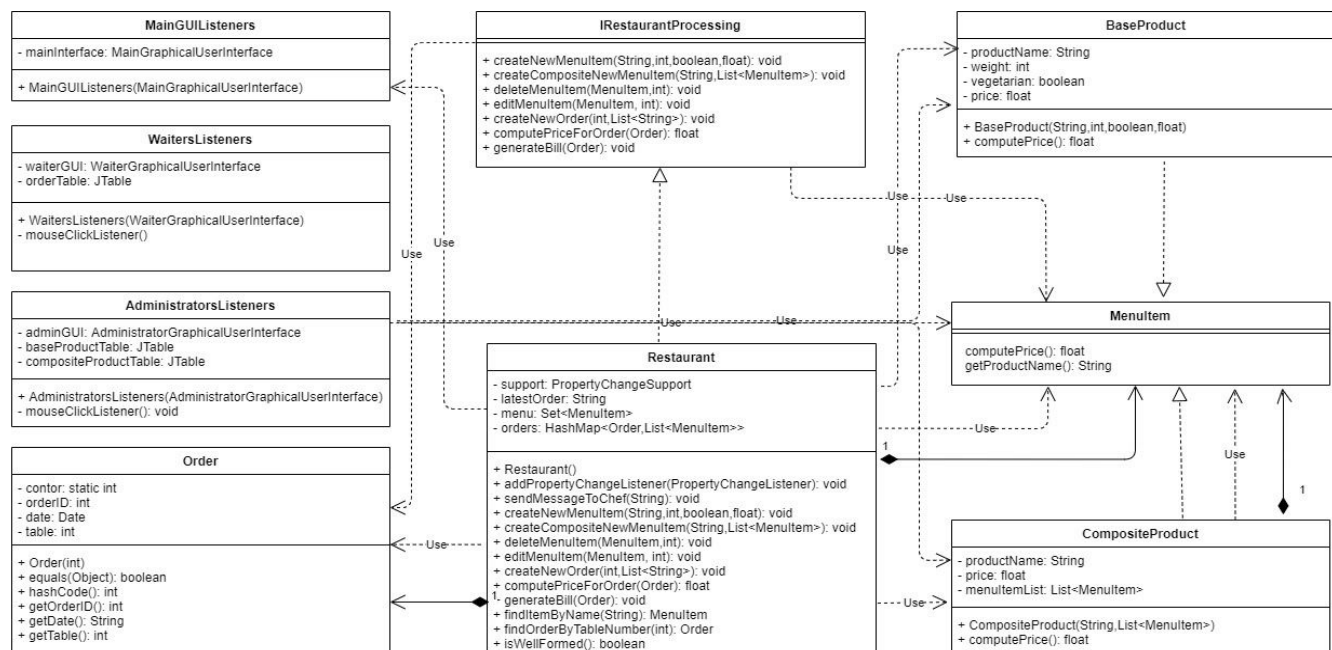
- 5



## presentationLayer



## businessLayer



## 3.2 Proiectare pachete

Diagrama pachetelor și a claselor, împreună cu dependențele dintre ele, este prezentată în figura de la secțiunea 3.1 Diagramă UML. Aceasta arată legăturile dintre pachetele, care sunt de tip de compoziție și de folosire (use).

După cum se vede din diagrame în realizarea proiectului am implementat patru pachete, fiecare având un scop bine definit. Pachetul *main* conține clasa *Main*, din care se începe rularea aplicației, deoarece conține metoda *main(String[])*. Pachetul *presentationLayer* conține clasele care realizează interfețele grafice cu utilizatorul, anume: *MainGraphicalUserInterface*, *AdministratorGraphicalUserInterface*, *WaiterGraphicalUserInterface* și *ChefGraphicalUserInterface*. Pachetul *businessLayer* conține clasele de listener pentru interfața principală, interfața administratorului și interfața chelnerului, anume: *MainGUIListeners*, *AdministratorsListeners* și *WaitersListeners*, care conțin la rândul lor clasele(inner-classes) pentru listener-urile butoanelor din interfețele lor. Pe lângă acestea pachetul mai conține clasele care modelează aplicația, precum: *Restaurant*, *Order*, *BaseProduct*, *CompositeProduct*, precum și interfețele: *IRestaurantProcessing* și *MenuItem*. În cazul implementării lui *MenuItem*, *BaseProduct* și *CompositeProduct* se folosește Composite Design Pattern-ul. Pachetul *dataLayer* conține clasele care realizează lucrul cu fișiere (citire din fișiere, scriere în fișiere), anume clasele *RestaurantSerializator* și *FileWriter*.

## 3.3 Structuri de date

În cazul atributelor folosesc diferite tipuri de date: tipul primitiv *int*, pentru attribute care stochează valori întregi (precum ID, numărul mesei, gramaj, contor); tipul primitiv *float*, pentru atributul care stochează valoare reală (în cazul prețului); tipul *String*, pentru attribute care stochează șiruri de caractere(precum nume), tipul *Date* pentru stocarea datei în format “dd-MM-YYYY hh:mm:ss”.

Totodată folosesc și structuri de date mai complexe precum *List*, *Set* și *HashMap*. Variabile de tip *List* folosesc în cazul stocării produselor care compun un produs compus (*CompositeProduct*). Lista conține elemente de tip *MenuItem*, deoarece un *CompositeProduct* poate să conțină atât produse de bază(*BaseProduct*), cât și produse compuse. Variabile de tip *Set* folosesc pentru stocarea produselor din meniu. Am ales acest de tip de date, deoarece un produs poate să apară numai o singură dată în meniu, iar *Set* nu permite elemente duplicate. (În cazul stocării produselor care compun un produs compus nu era necesară folosirea tipului *Set*, deoarece un produs compus poate să conțină mai multe bucăți din aceeași produs). Variabile de tip *HashMap* folosesc în cazul stocării comenzilor. Am ales acest tip de date deoarece o comandă conține atât elementele comandate, cât și anumite date, precum: masa, care a făcut comanda, data când s-a dat comanda. Pentru stocarea elementelor folosesc o listă de produse, iar restul informațiilor stochez într-o variabilă de tip *Order*. *HashMap*-ul are ca și cheie order-ul, iar ca și valoare are lista de produse. Astfel se realizează asocierea obiectelor de tip *Order* cu elementele de tip *MenuItem*, care sunt de fapt produsele comandate în cazul order-ului respectiv.

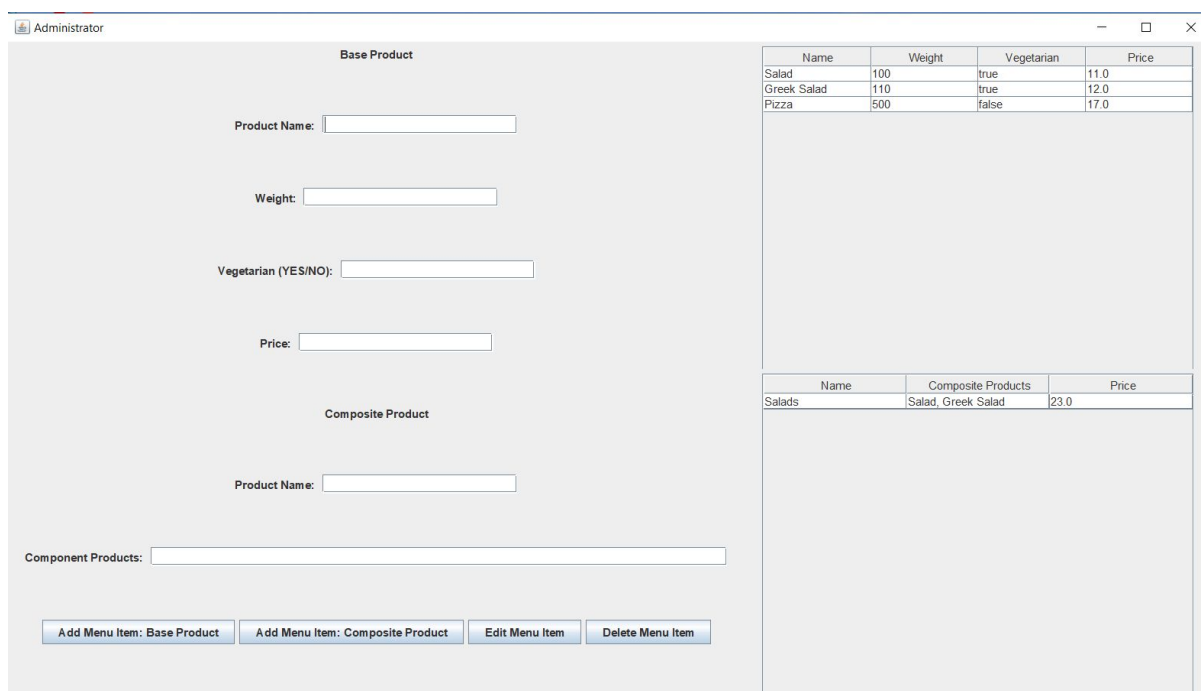
Pe lângă acestea mai folosesc structuri de date necesare pentru lucrul cu fișiere, precum *FileWriter*, *BufferedWriter* în cazul scrierii în fișiere de tip *txt*, respectiv *FileInputStream*, *ObjectInputStream*, *FileOutputStream*, *ObjectOutputStream* în cazul în care se scrie și se citește în/din fișiere folosind *serialization*.

### 3.4 Interfața utilizator

Interfața principală arată în felul următor:



Interfața administratorului arată în felul următor:



The screenshot shows a window titled 'Administrator' with a light gray background. The window is divided into two main sections. The left section contains a form for adding menu items, with fields for 'Product Name', 'Weight', 'Vegetarian (YES/NO)', and 'Price'. Below these fields are two sections: 'Base Product' and 'Composite Product', each with a 'Product Name' field. At the bottom of the form are four buttons: 'Add Menu Item: Base Product', 'Add Menu Item: Composite Product', 'Edit Menu Item', and 'Delete Menu Item'. The right section contains two tables. The top table has columns 'Name', 'Weight', 'Vegetarian', and 'Price'. The bottom table has columns 'Name', 'Composite Products', and 'Price'.

Name	Weight	Vegetarian	Price
Salad	100	true	11.0
Greek Salad	110	true	12.0
Pizza	500	false	17.0


Name	Composite Products	Price
Salads	Salad, Greek Salad	23.0

Interfața bucătarului-șef arată în felul următor:





Interfața chelnerului arată în felul următor:

 Waiter

Order

Products:

Table:

Add Order

Compute Bill

Table	Data	Ordered Products
1	07-05-2020 at 02:23:04	Salad

Name	Weight	Vegetarian	Price
Salad	100	true	11.0
Greek Salad	110	true	12.0
Pizza	500	false	17.0

Name	Composite Products	Price
Salads	Salad, Greek Salad	23.0

Modul de utilizare ale interfețelor grafice este prezentată în secțiunea **2.1.Cazuri de utilizare**.

## 4.Implementare

### 4.1. Pachetul *main*

Pachetul *main* conține o singură clasă numită *Main*. Clasa aceasta implementează metoda *main(String[])*, din care se instanțiază un obiect de tipul clasei *Restaurant*, și se inițializează cu datele citite prin deserializare din fișierul *restaurant.ser*. Pe lângă acesta, se mai declară în metoda *main* câte o instanță statică de tip *MainGraphicalUserInterface*, *AdministratorGraphicalUserInterface*, *WaiterGraphicalUserInterface* și *ChefGraphicalUserInterface* și *MainGUIListener*. Variabilei *restaurant* i se mai adaugă un Listener de tip *PropertyChangeListener()*, care dă notificare interfeței *ChefGraphicalUserInterface*, de fiecare dată când se inserează o comandă nouă.

#### 4.2. Pachetul *presentationLayer*

Pachetul *presentationLayer* conține patru clase, pe nume: *MainGraphicalUserInterface*, *AdministratorGraphicalUserInterface*, *WaiterGraphicalUserInterface* și *ChefGraphicalUserInterface*. Clasa *MainGraphicalUserInterface* conține patru atribute, unul de tip *JFrame*, care este fereastra principală, și încă trei atribute de tip *JButton*, câte un buton pentru deschiderea ferestrei pentru fiecare utilizator: *Administrator*, *Waiter* și *Chef*. Clasa conține un constructor, care realizează interfața în sine, și trei metode care creează listener-uri pentru cele trei butoane. Constructorul mai conține un *WindowListener*, care permite salvarea datelor din clasa *Restaurant* (utilizând *serialization*) la închiderea ferestrei. Aplicația se oprește doar când se închide fereastra principală.

Clasa *AdministratorGraphicalUserInterface* conține un număr mare de atribute, necesare pentru realizarea unei ferestre, care conține label-e, field-uri, butoane și tabele. Astfel se declară atribute de tip *JFrame*, *JTable*, *DefaultTableModel*, *JTextField* și *JButton*. Aceste atribute se mapează în 13 panel-uri, care i se adaugă frame-ului, adică ferestrei. Una dintre tabele conține produsele de bază prezente în meniu, iar cealaltă produsele compuse. Câmpurile de text sunt corespunzătoare câmpurilor produselor de cele două tipuri pentru a permite introducerea item-urilor noi, și a editării item-urilor deja existente. Clasa conține totodată metode de tip *getter* și *setter* pentru aproape toate atributele, pentru a putea obține și seta valorile lor și din alte clase.

Clasa *WaiterGraphicalUserInterface* este similară cu clasa *AdministratorGraphicalUserInterface*, cu deosebirea că conține mai puține câmpuri de text, și trei tabele în loc de două. Cele două tabele cu produsele de bază și produsele compuse se regăsesc și aici, și mai apare încă o tabelă cu comenzile, care conține pentru fiecare comandă numărul mesei, data comandării și numele produselor comandate.

Clasa *ChefGraphicalUserInterface* conține două atribute: unul de tip *JFrame*, care va conține fereastra pentru utilizatorul *chef*, și unul de tip *JTextArea* care va conține mesajele trimise bucătarului-șef, pentru a-i comunica ce produse trebuie să pregătească pentru noua comandă. Clasa conține un constructor, care plasează elementele necesare pe fereastră, metoda *getFrame()* pentru accesarea ferestrei, și suprascrierea metodei *propertyChange()*, în care adaugă noul mesaj apărut la cele existente deja. Suprascrierea acestei metode este necesară, deoarece clasa implementează interfața *PropertyChangeListener*, fiind o clasă implementată pe baza Design Pattern-ului *Observer*. *ChefGraphicalUserInterface* este clasa care observă (observer) clasa *Restaurant* (clasa observat/observable), și în cazul în care apare o nouă comandă în clasa *Restaurant*, aceasta trimite o notificare clasei observer. Interfața aceasta, precum toate interfețele de utilizator sunt create în clasa *Main*, deci la lansarea programului, însă fiecare devine vizibilă doar când se apasă butonul corespunzător ei.

#### 4.3. Pachetul *dataLayer*

Pachetul *dataLayer* conține două clase: *RestaurantSerializator* și *FileWriter*, care se ocupă cu citirea respectiv scrierea din/în fișiere. Clasa *RestaurantSerializator* conține două metode: una pentru deserializare, anume: *deserialize(String)*, care citește datele din fișierul primit ca argument (care este de fapt un obiect de tip *Restaurant*) și le returnează, în timp ce cealaltă metodă este pentru serializare, anume *serialize()*, care scrie într-un fișier denumit "restaurant.ser" obiectul de tip *Restaurant*, primit ca parametru. Scrierea și citirea se efectuează cu ajutorul obiectelor de tip *FileInputStream*, *ObjectInputStream*, *FileOutputStream* și *ObjectOutputStream*.

Clasa *FileWriter* conține o singură metodă, numită *writeInFile(Order)*, care generează un fișier de tip *txt*, drept notă de plată, care conține informațiile order-ului primit ca parametru. Generează fișiere cu un

nume care conține și al cătelea notă de plată este bill-ul respectiv, pentru a putea fi salvate bill-urile fără suprascriere.

#### 4.4. Pachetul *businessLayer*

Pachetul *businessLayer* conține nouă clase/interfețe, care se ocupă de logica aplicației: *MainGUIListener*, *AdministratorsListeners*, *WaitersListeners*, *Order*, *MenuItem*, *BaseProduct*, *CompositeProduct*, *IRestaurantProcessing* și *Restaurant*.

Clasa *MainGUIListener* are un constructor, în care atribuie *MainGraphicalUserInterface*-ului primit ca parametru câte un listener de administrator, chelner și bucătar-șef, corespunzătoare celor trei butoane din interfața principală. Implementarea claselor de listener asociate fiecărei metode se face în cadrul acestei clase, prin inner-classes, anume: *AdministratorListener*, *WaiterListener* și *ChefListener*. Aceste trei clase implementează interfața *ActionListener*, și suprascriu metoda *actionPerformed*, în cadrul căreia setez vizibilitatea ferestrelor corespunzător lor la *true*, și primele două clase mai inițializez cu interfața făcută în clasa *Main* clasele de listener corespunzător lor, care sunt: *AdministratorsListeners* și *WaitersListeners*, implementate tot în acest pachet.

Clasa *AdministratorsListeners* conține un constructor în care asociază butoanelor *Add Base Product* și *Add Composite Product* câte un listener, ale căror implementare se face în cadrul acestei clase sub formă de inner-classes, și tot în constructor se apelează metoda *mouseClickListener()*, în care se verifică dacă s-a făcut un click pe un anumit rând al tabelelor din interfețe grafice, și în caz afirmativ asociază un listener și butoanelor *Edit Menu Item* și *Delete Menu Item* câte un listener, ale căror implementare se face în mod asemănător în cadrul clasei. Clasa *BaseProductListener* preia datele din câmpurile interfeței grafice al administratorului și pe baza lor face un obiect nou de tip *BaseProduct*, apelează metoda *createNewItem(String,int,boolean,float)* a clasei *Restaurant*, care inserează noul produs în meniu, și în tabelele interfețelor. Clasa *CompositeProductListener* se comportă similar cu clasa *BaseProductListener*, numai lucrează cu obiecte de tip *CompositeProduct*. Clasa *MenuItemEditor* apelează metoda *editMenuItem(MenuItem,int)* din clasa *Restaurant*, care implementează editarea rândului trimis ca parametru. Clasa *MenuItemDeleteListener* caută produsul care are numele primit, cu metoda *findItemByName(String)* implementată în clasa *Restaurant*, și îl șterge (din meniu și tabele).

Clasa *WaiterListener* conține un constructor în care asociază butonului *Add Order* un listener, al cărui implementare se face în cadrul acestei clase sub formă de inner-class, și tot în constructor se apelează metoda *mouseClickListener()*, în care se verifică dacă s-a făcut un click pe un anumit rând al tabelului de comenzi din interfața grafică a waiter-ului, și în caz afirmativ asociază și butonului *Generate Bill* un listener, al cărui implementare se face în mod asemănător în cadrul clasei. Clasa *OrderAdderListener* preia datele din câmpurile interfeței grafice al chelnerului anume numărul mesei care a făcut comanda, și un *String* cu numele produselor comandate. Face un obiect de tip *List<String>* din numele produselor, și apelează metoda *createNewOrder(int,List<String>)* a clasei *Restaurant*, care inserează noul order în *HashMap*-ul *orders*, și în tabelul interfeței chelnerului. Clasa *BillListener* apelează metoda *generateBill(Order)* din clasa *Restaurant*, care generează nota de plată corespunzătoare order-ului trimis ca parametru.

Clasa *Order* are patru attribute: *contor* de tip *int static*, *orderID* și *table* de tip *int* respectiv *date* de tip *Date*. Constructorul *Order*-ului conține ca parametru doar un *int* pentru numărul mesei, iar celelalte attribute, precum *date* și *orderID* se setează implicit, *date* la data curentă și *orderID* cu ajutorul atributului *contor*, care este unică pe clasă și se incrementează de fiecare dată când se face un obiect nou de tip *Order*. Clasa mai are cinci metode: *equals(Object)*, care suprascrie metoda *equals(Object)*, și returnează *true* doar dacă cele trei attribute (în afară de *contor*) ale obiectelor sunt egale. Clasa suprascrie și metoda *hashCode()* care returnează un hashcode bazat pe cele trei attribute. Acestea sunt

necesare din cauza că clasa *Order* implementează interfața *Serializable* (clasa *Restaurant*, pentru care se face serializarea conține și obiecte de tip *Order*). Celelalte metode sunt gettere pentru cele trei atribute: *getOrderID()*, *getDate()* și *getTable()*.

Interfața *MenuItem* conține două metode *computePrice()* și *getProductName()*. Ea este implementată de către clasele *BaseProduct* și *CompositeProduct*.

Clasa *BaseProduct* conține atributele *productName* de tip *String*, *weight* de tip *int*, *vegetarian* de tip *boolean* și *price* de tip *float*. Constructorul clasei setează toate cele patru atribute la valorile furnizate ca parametri. Implementează cele două metode ale interfeței *MenuItem*, împreună cu getter-ile celorlalte atribute.

Clasa *CompositeProduct* conține atributele *productName* de tip *String*, *price* de tip *float* și *menuItemList* de tip *List<MenuItem>*. Constructorul ei setează doar atributele *productName* și *menuItemList* pe baza parametrilor, *price*-ul este setat prin apelarea metodei *computePrice()* din clasă, care calculează suma produselor care sunt componentele composite product-ului. Clasa mai conține metode de getter pentru cele trei atribute: *getProductName()*, *getPrice()* și *getMenuItemList()*.

Interfața *IRestaurantProcessing* conține metodele *createNewMenuItem(String,int,boolean,price)*, *createCompositeNewMenuItem(String, List<MenuItem>)*, *deleteMenuItem(MenuItem, int)*, *editMenuItem(MenuItem, int)*, *createNewOrder(table, List<String>)*, *computePriceForOrder(Order)*, *generateBill(Order)*.

Clasa *Restaurant* implementează interfața *Serializable*, ea fiind clasa a cărei date se salvează prin *serialization*, și și interfața *IRestaurantProcessing*, care conține metodele ce trebuie implementate, pentru ca clasele de interfață să comunice cu clasa *Restaurant*. Atributele clasei sunt: *support* de tip *PropertyChangeSupport*, care permite observarea clasei (clasa *Restaurant* este clasă de tip *Observable* pentru *Observer*-ul *ChefGraphicalUserInterface*), *latestOrder* de tip *String*, care conține mesajul despre comanda cea mai recentă, și care trebuie trimisă clasei *ChefGraphicalUserInterface* prin notificare, *menu* de tip *Set<MenuItem>*, care conține produsele unice care compun meniul restaurantului și *orders* de tip *HashMap<Order, List<MenuItem>>*, care conține asocierile între obiectele de tip *Order* și o listă de produse (deoarece un obiect de tip *Order* nu conține produsele comandate). Constructorul clasei inițializează atributele *menu* și *support*. Clasa conține diverse metode: *addPropertyChangeListener(PropertyChangeListener)*, care asociază un listener cu *support*, *sendMessageToChef(String)*, care notifică clasa *ChefGraphicalUserInterface*, și îi trimite un mesaj, *findItemByName(String)*, care caută în meniu produsul cu numele primit ca parametru, *findOrderByTableNumber(int)*, care caută în *orders* order-ul cu numărul mesei primite ca parametru, *getOrders()*, care construiește un *String* bidimensional care conține informațiile despre toate comenzile existente, pentru a le afișa în tabelul din interfață grafică, *getBaseProducts()* și *getCompositeProducts()*, care construiesc câte un *String* bidimensional care conține informațiile despre toate produsele de bază respectiv produsele compuse existente, pentru a le afișa în tabelele din interfață grafică. Metoda *isWellFormed()* realizează calcularea invariantului al clasei, prin verificarea dacă menu item-urile unei comenzi au hashcode diferit de order-ul cu care se asociază.

Clasa conține totodată și suprascriderile metodelor din *IRestaurantProcessing*: *createNewMenuItem(String, int, boolean, price)*, care face un obiect de tip *BaseProduct* pe baza parametrilor primite și îl adaugă în meniu, respectiv în tabelele interfețelor de administrator și chelner. Metoda *createCompositeNewMenuItem(String, List<MenuItem>)* face un obiect de tip *CompositeProduct* pe baza parametrilor primite și îl adaugă în meniu, respectiv în tabelele interfețelor de administrator și chelner, Metoda *deleteMenuItem(MenuItem, int)* șterge produsul primit ca

parametru mai întâi din meniu, după acesta și din tabelele interfețelor de administrator și chelner pe baza *row*-ului primit tot ca parametru. Metoda *editMenuItem(MenuItem, int)* aduce datele produsului selectat în câmpurile de text de lângă tabele, și șterge elementul din meniu și din listă. După editare produsul trebuie readăugat în tabelă și meniu prin apăsarea butonului *Add Base Product* sau *Add Composite Product*, în funcție de tipul produsului editat. Metoda *createNewOrder(int, List<String>)* face un obiect de tip *Order*, pe baza primului argument, care este numărul mesei. Pe baza celui de-al doilea argument caută în meniu produsele cu numele specificate, și face o listă cu ele. Cu obiectul *Order* făcut și cu lista construită se face un obiect nou de tip *HashMap(Order, List<MenuItem>)*, care se inserează în *orders*. Datele noului order se afișează și în tabelul de comenzi al interfeței chelnerului. Metoda *computePriceForOrder(Order)* parcurge lista de produse asociată order-ului primit ca parametru și calculează suma prețurilor produselor. Metoda *generateBill(Order)* apelează metoda *writeToFile(Order)* a clasei *FileWriter*, care realizează scrierea în fișier a datelor order-ului trimis ca parametru.

## 5.Rezultate

Rezultatul tuturor operațiilor ce pot fi efectuate sunt afișate automat pe interfețele grafice după apăsarea butoanelor corespunzătoare. Mai puțin în cazul în care se apasă butonul de *Generate Bill* de pe interfața chelnerului, pentru că în cazul acesta se generează un fișier de tip *txt*, care este nota de plată corespunzătoare order-ului selectat.

## 6.Concluzii

Aplicația implementată este una care realizează sistemul de management al unui restaurant. Se pot introduce produse noi în meniu, care pot fi produse de bază sau produse compuse. În același timp produsele pot fi și editate sau șterse. Se mai pot introduce comenzi, pentru care se pot genera și note de plată. Toate datele din meniu și comenzi pot fi vizualizate sub formă de tabel. Se mai poate vizualiza într-o fereastră și notificarea bucătarului-șef la introducerea comenzilor noi.

Aplicația poate fi dezvoltată prin introducerea a noi câmpuri în structura produselor și a comenzilor. Se poate adăuga un proces de logare, pentru diferitele tipuri de utilizator. Se poate dezvolta posibilitatea de a avea mai mulți utilizatori de același tip deodată.

## 7.Bibliografie

<https://www.baeldung.com/java-serialization>  
<https://www.geeksforgeeks.org/serialization-in-java/>  
<http://javarevisited.blogspot.ro/2011/02/how-hashmap-works-in-java.html>  
<http://docs.oracle.com/javase/8/docs/technotes/guides/language/assert.html>  
<https://docs.oracle.com/javase/7/docs/technotes/tools/windows/javadoc.html#tag>