MINISTERUL EDUCAȚIEI ȘI CERCETĂRII ȘTIINȚIFICE

UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

Facultatea de Automatică și Calculatoare

Specializarea Calculatoare

Tema 4: Sistem de manageriere a unui restaurant

-documentație-

Strujan Florentina

Gr. 302210

An 2, semestrul 2

Cuprins:

1. Obiectivul temei

Obiectivul temei de laborator a fost proiectarea și implementarea unei aplicații ce se ocupă cu managerierea comenzilor unui restaurant, cu 3 interfete grafice usor de folosit, pentru fiecare posibil utilizator (ospatar, administrator si chef), cu butoane si campui coresunzatoare. Generarea chitantei pentru o anumita comanda se va realiza in format .txt.

Aplicatia trebuie sa aiba 3 tipuri de utilizatori. Admnistratorul va putea adauga, sterge sau modifica un produs din meniu, ospatarul va putea crea o comanda cu produsele deja existente in meniu si emite o chitanta aferenta comenzii, iar chef-ul va afisa un mesaj de fiecare data cand se va face o comanda.De fiecare data cand va fi efectuata vreo operatie, va fi afisat comntinutul listei de meniuri sau a continutului comenzilor.

Obiectivele secundare sunt:

-folosirea programării orientate pe obiect (definirea de clase, metode, folosirea încapsulării etc.)

-folosirea limbajului de programare Java

-folosirea metodelor de maxim 30 linii și a claselor de maxim 300 linii

-folosirea conventiilor de denumire in Java

-folosirea javadoc pentru a documenta clasele incluzand etichetele si descrierile asociate conditiilor pre,post in inv si a genera fisiere JavaDoc corespunzatoare

-implementare diagramei de clase specificate

-alegerea structurilor de date potrivite pentru stocarea comenzilor si a meniurilor

-crearea unei ferestre pentru fiecare tip de utilizator cu operatiile corespunzatoare

-folosirea Jtable pentru afisarea tuturor comenzilor si a meniurilor

-un fișier PT2020_Group_LastName_FirstName_Assignment_4.jar creat și configurat pentru a putea fi rulat conform cerințelor

Optional:

- folosirea Composite Design Pattern pentru clasele MenuItem, BaseProduct, CompositeProduct

-crearea unei chitante pentru fiecare comanda ca fisier .txt

-folosirea Design by Contract: preconditii si postconditii in interfata IrestaurantProcessing. Implementare lor in clasa Restaurant folosind instructiunile assert. Definirea unui invarnat pentru clasa Restaurant

-crearea unei ferestre pentru chef folosind Observer Design Pattern pentru notificare crearii unei comenzi

-salvarea informatiei din Restaurant intr-un fisier folosind serializarea

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

Pentru implementarea aplicației cerute, vom analiza cerințele date, iar ulterior vom analiza tipul de programare „Programare Orientata pe Obiecte”, deoarece conceptele acestui tip de programare stau la baza implementarii acestei aplicatii. Bazându-ne pe aceste concepte, putem incepe dezvoltarea aplicatiei.

O astfel de aplicatie poate fi realizata prin mai multe metode, diferenta facandu-se la metoda de introducere a datelor, metoda de prelucrare a datelor si afisarea acestora.

Initial am folosit 8 campuri de introducere a meniurilor , limitand posibilitatile aplicatiei la adaugarea unui CompositeProduct de maxim 4 BaseProduct uri. Inaintand in realizarea aplicatiei, am realizat eficienta folosirii unui comboBox pentru stocarea ficarui meniu adaugat, cu posibilitatea alegerii cate unuia din acestea pentru a forma un altul, in cazul utilizatorului Waiter .Pentru adaugarea a mai multe meniuri existente la un nou meniu, se vor selecta pe rand, apasandu-se dupa fiecare pe butonul Create menu. Pentru a face diferenta intre un BaseProduct si un CompositeProduct, am folosit campul Menu Price astfel: daca este diferit de 0, meniul adaugat este unul basic cu pretul dat, iar daca e 0, e un meniu compus din meniurile selectate din comboBox. Stergerea e simpla,la fel ca editarea, fiind necesara alegerea meniului de sters/editat, alegerea unui nou nume pentru meniu in cazul editarii si cu apasarea butonului corespunzator.

In cazul utilizatorului Administrator, meniurile stocate in comboBox se for actualiza in functie de operatiile facute de Waiter, doar acestea putandu-se folosi pentru generarea unei comenzi, a carui id se va da in campul coreaspunzator . Pentru generarea unei chitante, este necesara introducerea id ului comenzii deja existente. La adaugarea unei comenzi, Chef ul va fi notificat si va afisa un mesaj.

Metoda aleasă de mine este simplă, eficientă, însă are și neajunsuri.

Un caz mai fericit ar fi fost alegerea simultana a mai multor meniuri de folosit in crearea unui nou meniu sau a unei comenzi.

Aceasta aplicatie se poate folosi cu usurinta intr-un restaurant pentru facilitarea fiecarei actiuni a fiecarui angajat. Folosind o astfel de aplicatie Chef ul isi va putea lua pauze cand nu sunt produse de preparat iar ospatarul va sti mereu in timp util ce meniuri sunt disponibile pentru realizarea unei comenzi si va genera cu usinta o chitanta doar dupa id ul acesteia. Pentru administrator va fi usoara tinerea evidentei meniurilor, adaugand, stergand si modificand.

3. Proiectare

3.1Decizii de proiectare

Pe plan intern,pentru a implementa aplicatia, am folosit mai multe modele pentru simplificarea operatiilor, cum ar fi: Composite Design Pattern pentru definirea claselor MenuItem,BaseProduct si CompositeProduct si Observer Design Pattern pentru notificarea chef ului de existenta unei noi comenzi. Am folosit o interfata ce contine toate operatiile principale realizare de utilizatori, interfata fiind implementata de clasa Restaurant, unde se vor detalia operatiile.Am folosit Design by Contract pentru implementarea clasei Restaurant si am definit structuri de tipul List pentru meniuri si de tipul HashMap petru comenzi. Am folosit clase diferite pentru serializare, deserializare si generarea chitantei pentru evitarea aglomerararii claselor de le folosesc.

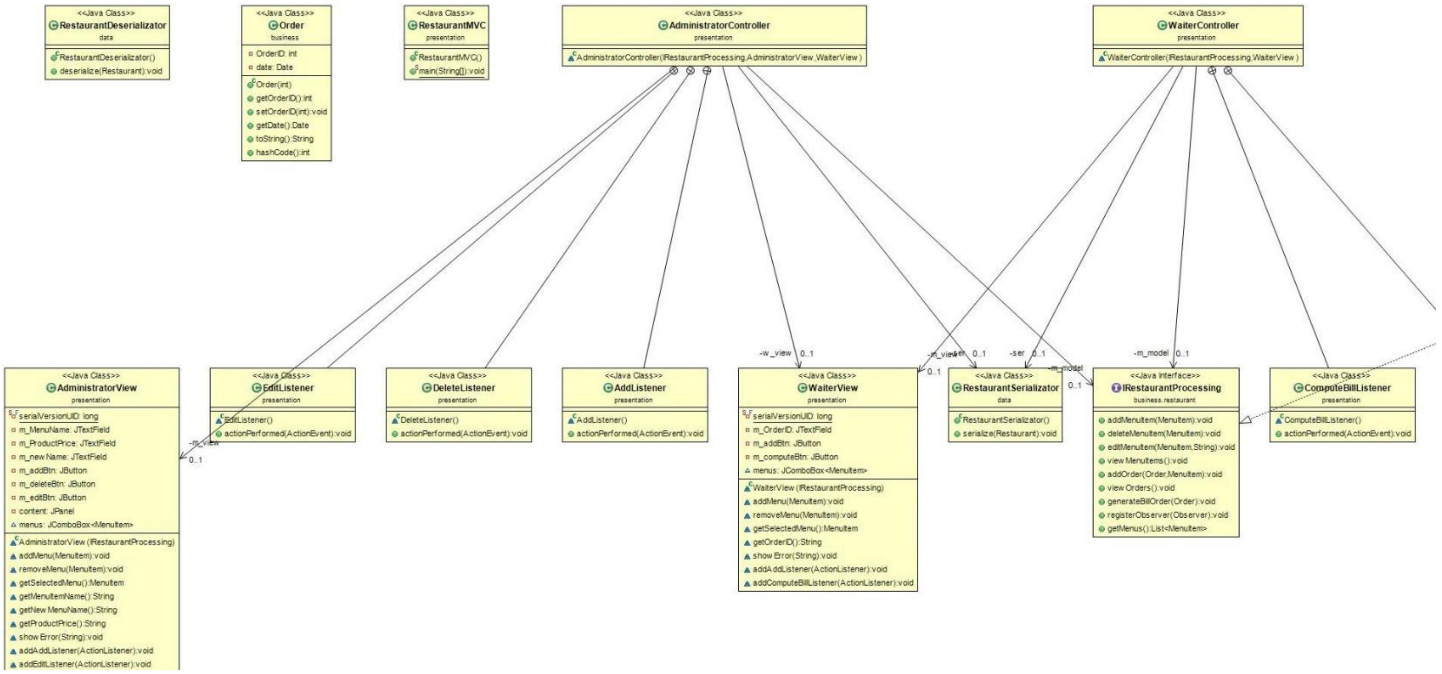
Pe plan extern, vom afișa în fisiere de tip .txt chitantele rezultate de efectuarea unor comenzi cu id-ul lor, data comenzii si pretul total. La rularea programului se vor deschide 2 ferestre, urmand sa se deschida si o a 3 a, cea a chef ului de cate ori se adauga o comanda. Am folosit pentru acestea strictul necesar de campuri si butoane

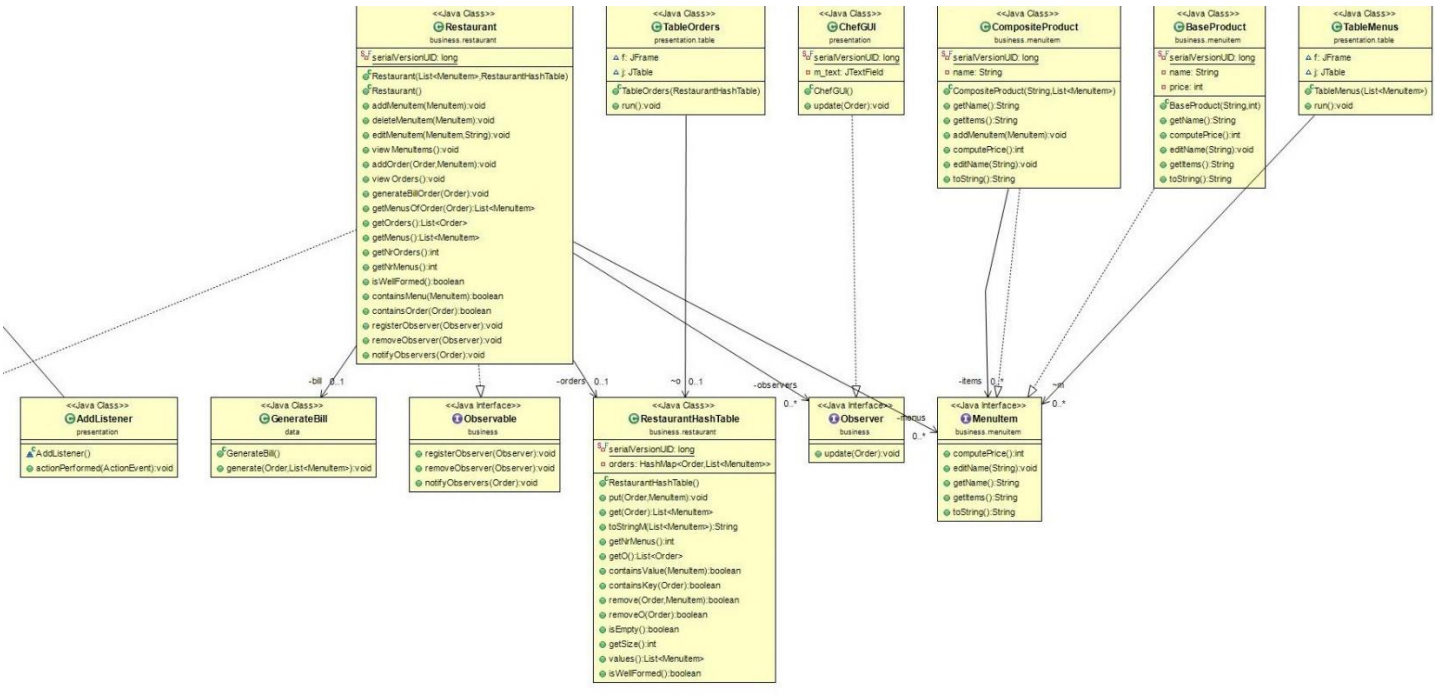
3.2 Diagrama UML

UML este notaţia internaţională standard pentru analiza şi proiectarea orientată pe obiecte. Diagramele UML de clase sunt folosite in modelarea orientata obiect pentru a descrie structura statica a sistemului, modului in care este el structurat. Ofera o notatie grafica pentru reprezentarea: claselor - entitati ce au caracteristici comune relatiilor - relatiile dintre doua sau mai multe clase .

Unified Modeling Language sau UML pe scurt este un limbaj standard pentru descrierea de modele si specificatii pentru software. UML a fost la bază dezvoltat pentru reprezentarea complexității programelor orientate pe obiect, al căror fundament este structurarea programelor pe clase, și instanțele acestora (numite și obiecte). Cu toate acestea, datorită eficienței și clarității în reprezentarea unor elemente abstracte, UML este utilizat dincolo de domeniul IT. Așa se face că există aplicații ale UML-ului pentru management de proiecte, pentru business Process Design etc. Reprezentarea UML a claselor pentru Client.

Mai departe voi imparti in 2 diagrama pentru vizualizare mai usoara.


RestaurantDeserializator
Order
RestaurantMVC
AdministratorController
WaiterController
AdministratorView
EditListener
DeleteListener
AddListener
WaiterView
RestaurantSerializator
IRestaurantProcessing
ComputeBillListener


Restaurant
TableOrders
ChefGUI
CompositeProduct
BaseProduct
TableMenus
AddListener
GenerateBill
Observable
RestaurantHashTable
Observer
MenuItem

3.3 Proiectare clase

In cadrul acestei aplicatii avem 16 de clase .

Prima, cea din pachetul business e clasa ce contine notiunile abstracte despre o comanda.

Urmatoarele doua, din subpachetul business.menuItem contin notiunile abstracte despre cele doua tipuri de meniu. Acestea implementeaza interfata MenuItem .

Clasele din subpachetul business.restaurant contin notiunile abstracte despre un restaurant cu meniuri si comenzi, clasa RestaurantHashTable organizand aceste notiuni intr-o structura de date de timp HashMap.

Cele 3 clase din pachetul data sunt folosite pentru scrierea in fisiere .txt si .ser si citirea din fisiere .ser.

Cele 6 clase din pachetul presentation fac legatura directa cu utilizatorul , definind interfetele grafice si toate operatiile prin care se iau datele si se afiseaza.

Clasele din subpachetul presentation.table contin metodele de generare si populare a celor doua JTable uri pentru comenzi, respectiv meniuri.

3.4 Pachete

In cadrul acestei aplicatii avem 3 pachete si 3 subpachete .

Primul pachet, business, contine clasele ce realizeaza logica aplicatiei.

Subpachetul business.menuItem contine logica din spatele prelucrarii meniurilor folosind Composite Design Pattern.

Subpachetul business.restaurant contine logica din spatele prelucrarii restaurantului cu toate meniurile si comenzile detinute. Aici este folosita colectia JCF bazata pe o structura de date de tip HashMap<Order,List<MenuItem>>.

Pachetul data contine clasele folosite pentru scrierea in fisiere .txt si .ser si citirea din fisiere .ser.

Pachetul presentation face legatura directa cu utilizatorul , definind interfetele grafice si toate operatiile prin care se iau datele si se afiseaza.

Subpachetul presentation.table contine metodele de generare si populare a celor doua JTable uri pentru comenzi, respectiv meniuri.

3.5 Structuri de date

Ca structuri de date am folosit HashMap<Order,List<MenuItem>> continand toate comenzile cu meniurile corespunzaoare. Metoda hshCode() a fost suprascrisa in clasa Order, obiectele de tip Order fiind chei (index) obiectelor de tip List<MenuItem>(valoare). HashMap ul stocheaza perechi de tipul cheie/valoare.

3.6 Interfete

Interfața este folosită pentru a descrie un contract între clase: o clasă care implementează o interfață va implementa metodele definite în interfață. Astfel orice cod care folosește o anumită interfață știe ce metode pot fi apelate pentru aceasta.

Pentru proiect am folosit 4 interfete. Interfata MenuItem contine metodele principale pe care le poate efectua un meniu, fiind implememntata de clasele CompositeProduct si BaseProduct. Aceasta definire foloseste Composite Design Pattern.

Interfetele Observable si Observer au fost definite conform Observer Design Pattern si sunt implementate de clasele Restaurant si ChefGUI, astfel ca la adaugarea unei noi comenzi in cadrul unui restaurant(observable ul) , chef ul(observer ul) va fi notificat.

Interfata IRestaurantProcessing contine principalele operatii care vor fi executate de utilizatorii Administrator si Waiter si va fi implementata de clasa Restaurant. Un obiect de tip IResraurantProcessing va fi folosit pentru interfetele grafice.

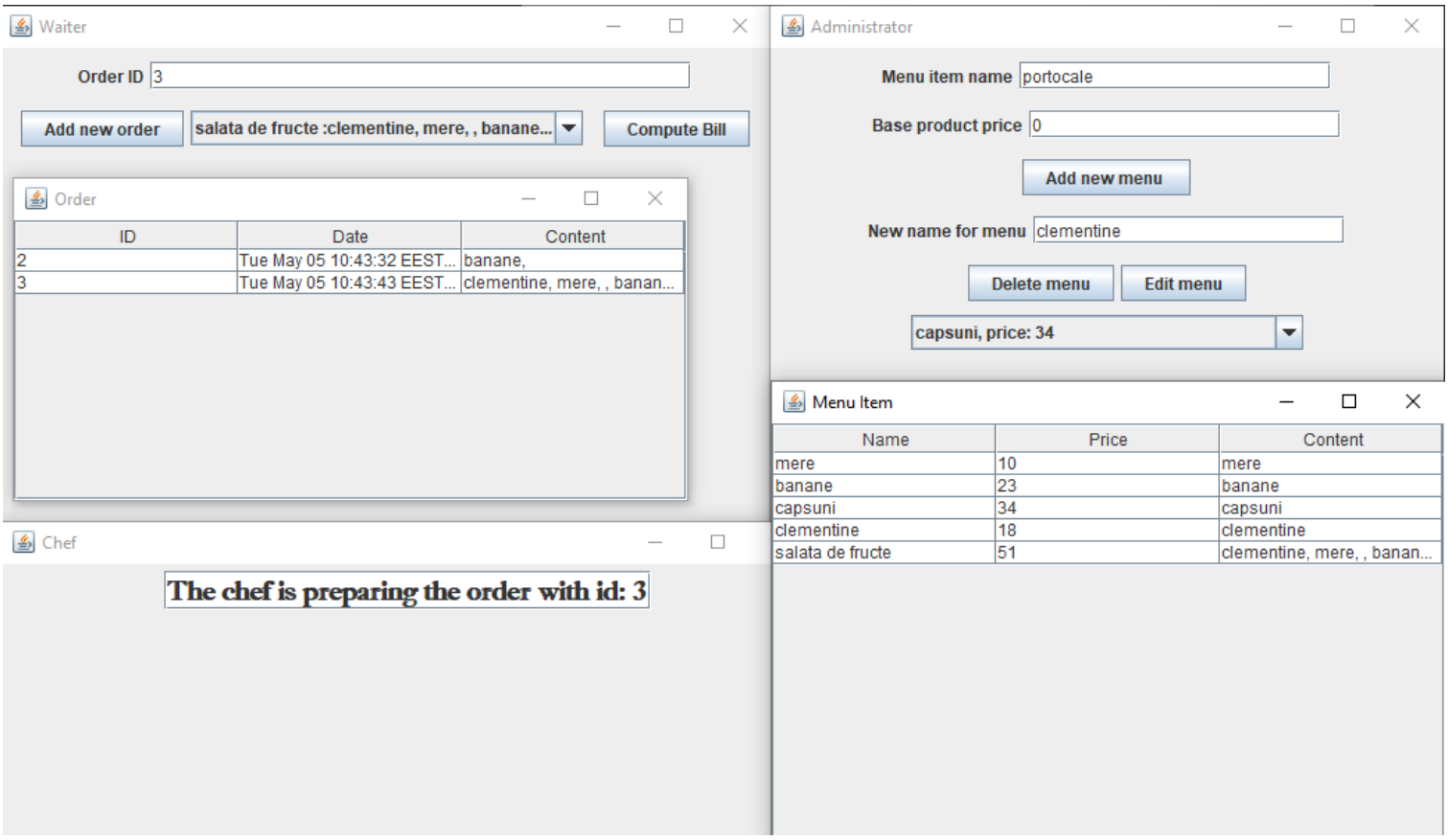
3.7 Interfata utilizator

Interfața grafică sau Graphical User Interface este o interfață cu utilizatorul bazată pe un sistem de afișaj ce utilizează elemente grafice. Interfața grafică este numit sistemul de afișaj grafic-vizual pe un ecran, situat funcțional între utilizator și dispozitive electronice. Folosim interfata grafica User-Friendly cu scopul de a facilita folosirea aplicatiei de utilizatorii corespunzatori.

Pentru aplicatia mea am folosit 3 interfete grafice, fiecare cu un Frame cu titlu specific care dupa utlizare, apasand butonul X din dreapta sus se pot inchide (Exit_on_close).

Interfata Chef contine doar un JTextField cu textul afisat in cazul in care se emite o comanda.

Interfetele Administrator si Waiter contin atat JTextField uri, cat si Jbutton uri si JComboBox uri. In JTextField uri se vor introduce datele corespunzatoare definirii obiectelor, precum numele meniului, pretul acestuia, un nou nume pentru modificare si id ul unei comenzi. La apasarea butoanelor de va efectua una din operatiile de creare, stergere, editare meniu sau crearea unei comenzi si generarea unei chitante. In JComboBox uri se vor introduce pe rand noi meniuri, pe masura ce acestea vor fi create de Administrator, iar prin selectarea unuia din acestea, se vor putea folosi pentru crearea unui meniu compus sau a unei comenzi. La apasarea fiecarui buton se va deschide o fereastra pe baza de JTable ce va contine informatiile din obiectul respectiv.



4.Implementare

Clasa Order contine notiunile abstracte ale unei comenzi, cu metodele basic de get, set si toString. In plus, avem metoda suprascrisa hashCode() intrucat obiecte de tip order sunt folosite ca si chei pentru tabela de dispersie continand comenzile cu meniurile aferente.

```
	@Override
public int hashCode() {
	return OrderID;
}
```

Clasele BaseProduct si **CompositeProduct** contin notiunile abstracte ale unui meniu si implementeaza interfata MenuItem, continand metodele definite in aceasta. Metodele sunt cele basic, in plus avand metoda de editare a numelui comenzii:

```
@Override
public void editName(String newName) {
	this.name = newName;
}
```

Clasa RestaurantHashTable defineste continutul unui restaurant intr-o structura de date de tip HashMap, cu metodele necesare prelucrarii si accesarii datelor din aceasta. Cheile vor fi obiecte de tip Order, iar valorile colectii de tipul List de MenuItem uri.Aici avem si metoda isWellFormed pentru a verifica corectitudinea unui HashMap.

```
public boolean isWellFormed() {
	for (Order ord : orders.keySet()) {
		if (orders.get(ord) == null)
			return false;
	}
	return true;
}
```

Metoda de adaugare a unui meniu la lista de meniuri a unei comenzi

```
public void put(Order order, MenuItem item) {
	int ok = 0;
	for (Order o : getO())
		if (order.getOrderID() == o.getOrderID()) {
			orders.get(o).add(item);
			ok = 1;
		}
	if (ok == 0) { // creeaza o noua lista de menu item de adaugat in hash map
		List<MenuItem> list = new ArrayList<MenuItem>();
		list.add(item);
		orders.put(order, list);
	}
}
```

Metoda de stergere a unui meniu din tabela

```
public boolean remove(Order order, MenuItem item) {
	if (!containsKey(order) || !containsValue(item))
		return false;
	List<MenuItem> itemsForOrd = get(order);
	itemsForOrd.remove(item);
	orders.put(order, itemsForOrd);
	return true;
}
```

Clasa Restaurant este cea mai ampla, implementand interfata IRestaurantProcessing cu toate metodele acesteia. Clasa contine o lista de meniuri folosite pentru crearea unei comenzi, un HashMap de comenzi si o lista de observeri pentru a notifica chef ul cand se creeaza o comanda. Clasa este implementata folosind Design by Contract implicand preconditii, postconditii, invarianti si assert.

Metoda de adaugare a unei comenzi cu notificarea observerilor

```
@Override
public void addOrder(Order order, MenuItem menu) {
	// precondition
```

```
            assert order != null && menu != null && containsMenu(menu);
            int prevNrOrders = orders.getSize();
            int prevNrMenus = orders.getNrMenus();
            orders.put(order, menu);
            notifyObservers(order);
            // postcondition
            assert getNrOrders() == prevNrOrders + 1 && orders.getNrMenus() == prevNrMenus + 1;
            // invariant
            assert isWellFormed();
      }
```

Metoda de afisare a unei comenzi

```
      @Override
      public void viewOrders() {
            TableOrders table = new TableOrders(orders);
            // preconditie
            assert table != null;
            int nrOrders = orders.getSize();
            table.run();
            // postcondition
            assert nrOrders == getNrOrders();
            // invariant
            assert isWellFormed();
      }
```

Metoda de verificare a continerii unui meniu de restaurant

```
      public boolean containsMenu(MenuItem menu) {
            // precondition
            assert menu != null;
            List<MenuItem> m = getMenus();
            boolean success = menus.contains(menu);
            // postconditions
            assert m.equals(getMenus());
            // invariant
            assert isWellFormed();
            return success;
      }
```

Clasa GenerateBill ce genereaza o chitanta in format txt cu metoda generate().

```
      public void generate(Order ord, List<MenuItem> list) {
            int total = 0;
            FileWriter f = null;
            try {
                  f = new
FileWriter("D:\\PT2020\\pt2020_302210_strujan_florentina_assignment_4\\BillClientId"
                              + ord.getOrderID() + ".txt", true);
                  f.write("Order id: " + ord.getOrderID() + "\n");
                  f.write("Date: " + ord.getDate() + "\n");
                  for (MenuItem m : list)
                        total += m.computePrice();
```

```
                f.write("Total price: " + total + "\n");
                f.close();
            } catch (IOException e) {
                System.out.println("Eroare la scriere");
            }
        }
```

Clasele RestaurantSerializator si **RestaurantDeserializator** cu metode de serializare si deserializare a unui Restaurant.

```
        public void serialize(Restaurant r) {
            try {
                FileOutputStream fileOut = new FileOutputStream("restaurant.ser");
                ObjectOutputStream out = new ObjectOutputStream(fileOut);
                out.writeObject(r);
                out.close();
                fileOut.close();
                System.out.printf("Serialized data is saved in restaurant.ser");
            } catch (IOException i) {
                i.printStackTrace();
            }
        }
        public void deserialize(Restaurant r) {
            try {
                FileInputStream fileIn = new FileInputStream("restaurant.ser");
                ObjectInputStream in = new ObjectInputStream(fileIn);
                r = (Restaurant) in.readObject();
                in.close();
                fileIn.close();
            } catch (IOException i) {
                i.printStackTrace();
                return;
            } catch (ClassNotFoundException c) {
                System.out.println("TotalItems class not found");
                c.printStackTrace();
                return;
            }
        }
```

Clasele TableMenus si **TableOrders** ce genereaza tabele de timpul JTable pentru stocarea si afisarea datelor obiectului corespunzator cu metoda de run pentru a le face visibile.

```
        public TableMenus(List<MenuItem> m) {
            this.m = m;

            f = new JFrame();
            String[] columnNames = { "Name", "Price", "Content" };
            int s = m.size();
            Object[][] data = new Object[s][3];
            for (int i = 0; i < s; i++)
```

```
                data[i] = new Object[] { m.get(i).getName(), m.get(i).computePrice(),
m.get(i).getItems() };

            j = new JTable(data, columnNames);
            j.setBounds(50, 50, 500, 500);

            JScrollPane sp = new JScrollPane(j);
            f.setTitle("Menu Item");
            f.add(sp);
            f.setSize(500, 600);
      }
      public void run() {
            f.setVisible(true);
      }
```

Clasele AdministratorView,ChefGUI si **WaiterView** ce definesc aspentul interfetei cu butoanele, campurile si comboBox urile necesare si metodele de definira a listener ilor pentru butoane si de obtinere a datelor introduse in JTextField uri .

```
      WaiterView(IRestaurantProcessing model) {
            JPanel content = new JPanel();
            JPanel a = new JPanel();
            a.setLayout(new FlowLayout());
            a.add(new JLabel("Order ID"));
            a.add(m_OrderID);

            JPanel p = new JPanel();
            p.add(m_addBtn);
            p.add(menus);

            JPanel b = new JPanel();
            b.add(m_computeBtn);

            content.add(a);
            content.add(p);
            content.add(b);
            this.setContentPane(content);
            this.pack();

            this.setTitle("Waiter");
            this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
            this.setSize(600, 600);
      }
```

Clasele AdministratoController,GhefGUI si **WaiterController** fac legatura intre interfata si operatiile definite intern, ce trebuie sa fie efectuate pe datele introduse.

```
      class AddListener implements ActionListener {
            @Override
            public void actionPerformed(ActionEvent e) {
                  String menuName = "";
```

```
                int productPrice;
                try {
                        menuName = m_view.getMenuItemName();
                        productPrice = Integer.parseInt(m_view.getProductPrice());
                        if (productPrice != 0) {
                                MenuItem m = new BaseProduct(menuName, productPrice);
                                m_view.addMenu(m);
                                w_view.addMenu(m);
                                m_model.addMenuItem(m);
                                m_model.viewMenuItems();
                        } else {
                                List<MenuItem> list = new ArrayList<MenuItem>();
                                list.add(m_view.getSelectedMenu());
                                MenuItem menu = new CompositeProduct(menuName, list);
                                m_view.addMenu(menu);
                                w_view.addMenu(menu);
                                m_model.addMenuItem(menu);
                                m_model.viewMenuItems();
                        }

                } catch (NumberFormatException nfex) {
                        m_view.showError("Bad input");
                }
        }
    }
```

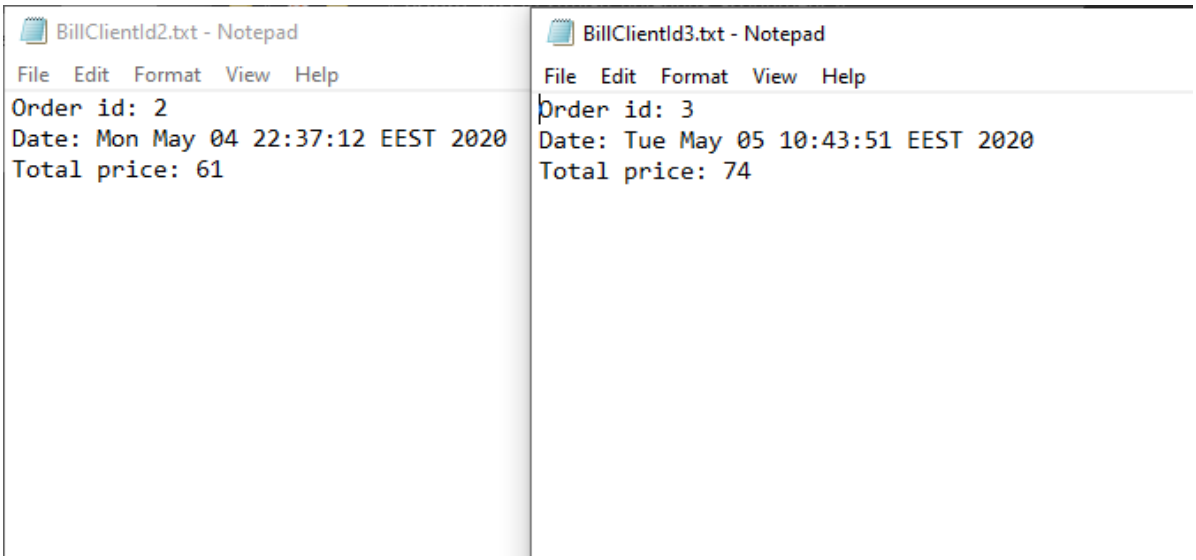
Clasa Restaurant MVC combina toate componentele si asigura rularea eficienta a aplicatiei.

```
    public class RestaurantMVC {
    @SuppressWarnings("unused")
    public static void main(String[] args) {
            RestaurantDeserializator deser = new RestaurantDeserializator();
            List<MenuItem> menus = new ArrayList<MenuItem>();
            RestaurantHashTable orders = new RestaurantHashTable();

            IRestaurantProcessing model = new Restaurant(menus, orders);
            deser.deserialize((Restaurant) model);
            WaiterView viewW = new WaiterView(model);
            WaiterController controllerW = new WaiterController(model, viewW);

            AdministratorView viewA = new AdministratorView(model);
            AdministratorController controllerA = new AdministratorController(model, viewA,
viewW);

            viewW.setVisible(true);
            viewA.setVisible(true);
    }
}
```

5.Rezultate

Am obtinut o implementare a cerintei proiectului , punand in practica o aplicatie a managementului unui restaurant folosind interfete grafice, generand in timp real chitantele coreaspunzatoare comenzilor .

Daca datele sunt introduse corespunzator si butoanele apasate conform cerintelor, aplicatia va fi utila pentu utilizatori.



Javadoc este un generator de documentație creat de Sun Microsystems pentru limbajul Java pentru generarea documentației API în format HTML din codul sursă Java. Formatul HTML este utilizat pentru a adăuga comoditatea de a putea hiperliga documente asociate. Am folosit Javadoc pentru documentare proiectului meu, adaugand comentarii interfetei IRestaurantProcessing, definind etichetele pre, post si inv.

All Methods | Instance Methods | Abstract Methods

Modifier and Type	Method	Description
void	addMenuItem(business.menuitem.MenuItem item)	Adauga un nou menu la lista restaurantului
void	addOrder(business.Order order, business.menuitem.MenuItem menuItem)	Adauga o noua comanda la restaurant
void	deleteMenuItem(business.menuitem.MenuItem item)	Stergeo un meniu din lista restaurantului si comenzile aferente
void	editMenuItem(business.menuitem.MenuItem item, java.lang.String name)	Modifica un meniu al restaurantului
void	generateBillOrder(business.Order order)	Genereaza o chitanta pentru o camnda
java.util.List<business.menuitem.MenuItem>	getMenus()	
void	registerObserver(business.Observer chef)	
void	viewMenuItems()	Afiseaza toate meniurile restaurantului
void	viewOrders()	Afiseaza toate comenzile unui restaurant

Method Details

addMenuItem

void addMenuItem(business.menuitem.MenuItem item)

Adauga un nou menu la lista restaurantului

Parameters:
item - Meniul de adaugat

Precondition:
item!=null

Postcondition:
getNrMenus() == getNrMenus() @pre + 1;

Invariant
isWellFormed()

6.Concluzii

Sunt de părere că în urma realizării acestei teme am reușit să-mi reamintesc și să aprofundez materia de semestrul trecut, să-mi îmbunătățesc tehnicile de programare în acest limbaj, am deprins o mai buna aptitudine in a lucra cu structurile de date si a organiza obiectele, in a lucra cu interfere, precum si generarea si rularea unui jar si lucrul cu interfetele grafice, generarea documentatiei JavaDoc

In acelasi timp, am invatat sa definesc etichete pentru JavaDoc, sa folosesc Design by Contarct si pattern urile Composite Design si Observer Design, sa lucrez cu JTable si cum sa folosesc serializarea, desi nu am resit sa o duc la final din cauza unor probleme si cum sa folosesc structurile de date mai avansate.

Ca o posibila dezvoltare ulterioara, aplicatiei ii mai pot fi aduse unele imbunatatiri, ca de exemplu:
- selectarea mai multor meniuri pentru un meniu compus sau o comanda
-afisarea JTable urilor in in fereastra corespunzatoare utilizatorului
-finalizarea serializarii

Nu am reusit sa rulez fisierul .jar folosind cmd, insa aplicatia se deschide la dublu click pe fisierul .jar.

7.Bibliografie

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

https://www.geeksforgeeks.org/composite-design-pattern/

https://www.tutorialspoint.com/design_pattern/observer_pattern.htm

https://www.geeksforgeeks.org/arraylist-contains-java/

https://www.javatpoint.com/java-string-to-date

https://www.journaldev.com/1739/observer-design-pattern-in-java

https://www.codejava.net/java-se/swing/a-simple-jtable-example-for-display

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

http://javarevisited.blogspot.ro/2012/01/what-is-assertion-in-java-java.html

http://stackoverflow.com/questions/11415160/how-to-enable-the-java-keywordassert-in-eclipse-program-wise

https://intellij-support.jetbrains.com/hc/en-us/community/posts/207014815-How-toenable-asser