MINISTRY OF EDUCATION AND RESEARCH

TECHNICAL UNIVERSITY
OF CLUJ-NAPOCA, ROMANIA

TEHNICI DE PROGRAMARE

TEMA_3

RESTAURANT MANAGEMENT SYSTEM

Author: Tóth Szilveszter

Gr.: 30224

1) Obiectivul temei

La aceasta tema am avut de implementat o aplicatie prin care se pot gestiona comenzile unui restaurant, si totodata, prin intermediul acestei paltforme se pot aplica modificari asupra meniuului de catre administratorul firmei. Interfata grafica este foarte folositoare din acest punct de vedere, facand gestionarea aplicatiei mult mai usoare.

2) Analiza problemei

Avand in vedere ca aplicatiea trebuie sa fie cat mai compacta si de user friendly (care tot odata impune si faptul ca trebuie sa ruleze instant), vom apela la o solutie orientate pe obiect (pe cat se poate) in care trebuie sa luam in vedere acea structura Data / Business / Presentation layer, care seamana intr-un fel cu arhitectura de tip MVC. Acesta presupune sa inpartim toate implementarea partea de view / presentation, care presupune crearea si gestionarea tuturor frame-urile (deci practic pentru tot GUI), si componenta de business, care face lucrul efectiv pe date, printre care si generarea chitantelor.

Urmatorul lucru pe care trebuie sa o rezolvam este popularea meniului, care in implementarea asta particulara se face dintr-un fisier extern, care va citi de fiecare data elementele din meniu impreuna ca pretul lor, mai apoi le va insera intr-o lista de tip MenuItem. Va trebui sa avem trei tipuri de angajati: administratorul, chelnerul si cheful. Administratorul va avea control deplin asupra meniului, mai precis sa adauge un element, sa strearga, sa modifice. Chelnerul este cel care o sa realizeze comenzile pe care le vom stoca intr-un hash map. Chef-ul nu va avea rol explicit de data asta, el fiind doar un observator, care va fi notificat prin intermediul chelnerului daca s-a depus o comanda si trebuie preparata (pentru asta vom folosi design pattern-ul observer). Adminul poate afisa meniul in intregime si chelnerul toate comenzile depuse de la inceputul programului, iar notificarea pentru bucatar va aparea intr-o ferestra pop-up.

3) Proiectare si implementare

Pentru a putea realiza toate acele taskuri pe care ne-am propus in paragrafele anterioare vom incepe prin implementarea interfetei grafice. Aici trebuie sa avem mare grija sa nu incurcam panel-urile. In mare parte folosim obiecte de tip JButton, JFrame, JTextField (pentru a putea salva comenzile cerute de catre client). Crearea acestor obiecte se realizeaza cu ajutorul java.swing, si codul sursa (mai precis un fragment din aceasta ar arata in urmatorul fel:

```
JFrame frame = new JFrame ("Restaurant");

frame.setSize (650, 160);
frame.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);
frame.setLayout (new BorderLayout());

JPanel panel = new JPanel();
panel.setLayout (new GridLayout(2,2));
JButton btn1 = new JButton ("Admin");
JButton btn2 = new JButton ("Waiter");
panel.add(btn1);
panel.add(btn2);
frame.add (panel,BorderLayout.NORTH);
```

Aceasta bucata de cod creaza un frame simplu cu doua butoane: cu eticheta Admin si Waiter. In cees ce urmeaza va trebuie sa le dam un scop acestor butoane, deci trebuie folosit actionListenere, care se afla intr-o clasa separat (in cazul nostru clasa ButtonController). Aceasta clasa va face legatura intre clasa view si clasele Admin + Waiter, in care se vor realiza toate operatiile necesare pentru a duce la bun sfarsit task-ul. Un actionListener arata asa:

```
btnAdmin1.addActionListener (e -> {
        String productName = txt1.getText();
        String productPrice = txt2.getText();
        admin.addProduct (productName,productPrice);
});
```

In particular acesta este responsabil pentru a prelua textul din panoul adminului, si a-l trimite la clasa Admin in care se afla metoda de inserare a unui produs nou, dupa care aceste informatii (din care se va crea un obiect de tipul MenuItem) vor fi stocate intr-o lista. Metoda pentru adaugarea unui nou element:

```
public void addProduct (String name, String price) {
        if(price.matches("[0-9]+") == false)
        {
                System.out.println ( "Price must be digits! Check input data!" );
        }
        else
        {
                if (checkExist(name) == true) {
                        System.out.println( "Item already exists!" );
                }
                else
                {
                        MenuItem item = new MenuItem(name, "0", price);
                        menu.add(item);
                        System.out.println( "Item succesfully added!" );
                }

        }
}
```

In primul rand, informatiile trebuie sa fie testate, pentru a evita erori in cele ce urmeaza. Pentru aceste testari se vor folosi functii auxuliare scrise in aceasta clasa. In cazul nostru, in primul rand pretul introdus trebuie testat daca contine doar cifre , pentru ca acestea vor fi transformate in intregi la un moment dat, de aceea trebuie evitate caracterele care pot produce erori la conversii. In cazul in care sunt si alte caractere inafara de cifre, si va afisa un mesaj de eroare si inserarea nu va mai avea loc. Urmatorul lucru care trebuie verificat este daca produsul este deja in meniu, pentru a evita inserarea de doua ori. Pentru aceasta vom folosi o functie auxiliara care va parcurge meniul si va verifica pentru fiecare produs daca numele coincide sau nu. Ca si in cazul anterior, daca se gaseste produsul, se va afisa un mesaj corespunzator si iese din metoda. Insa daca niciuna din cazurile anterioare nu se indeplineste, vom crea un obiect de tip MenuItem, care dispune de trei atribute: nume, cantitate si pret (atributul cantite va fi de folos doar in cazul in care se va face comanda pentru acest produs, dar pana atunci, vom instantia campul cu 0). Dupa ce este creat, se adauga la lista si se va afisa un mesaj de succes.

Celelalte doua operatii (stergerea si actualizarea) lucreaza pe acelasi principiu.

Pentru a putea realiza cu succes o comanda, trebuie sa cream a clasa Order, in care sa specificam ce atribute ar trebui sa aiba. In cazul de fata, acesta dispune de un id pentru masa la care trebuie livrata si o lista cu produse care contine mancarurile dorite de catre client. Identificatorul comenzii este creat cu ajutorul functie hashCode(), care va fi scrisa direct in hashmap ca si o cheie pentru tupla. In rest aceasta clasa dispune doar de niste getter -e si setter -e.

Urmatoare clasa care este esentiala pentru ca programul sa functioneze in intregime este clasa Waiter, adica a chelnerului. In primul rand trebuie specificat ca dispune de un observer (bucatarul) care va fi alertat dupa ce se plaseaza o comanda. Acesta clasa are doua campuri: lista (hashmap-ul) pentru comenzi si o referinta la administratorul restaurantului (acesta este necesar pentru ca meniul este asociat admin-ului, iar chelnerul va avea nevoie de meniu pentru a putea crea nota de plata). La acest punct trebuie mentionat ca aplicatia foloseste LinkedHashMap in loc de HashMap pentru a putea stoca comenzile oarecum in ordine cronologica. Daca foloseam doar simplu HashMap, comenzile ar fi fost intr-o ordine aleatoare. In aceasta clasa probabil cea mai complexa si cea mai importanta methoda este cea de executeOrder, care are ca si parametru un sir de caractere in care se afla informatiile introduse de catre chelner in fereastra pop-up pentru introducerea unei noi comenzi. Pentru a evita erorile, acesta este testat sa se vada daca sunt date introduse, sau este lasat gol. Daca se constata ca nu sunt informatiile necesare pentru a putea plasa comanda, sa va afisa un mesaj de eroare si se va intrerupe plasarea comenzii. In caz contrar, stringul trebuie spart pentru a putea salva doar informatiile necesare cum ar fi numarul mesei si comenzile. Restul de informatii ajutatoare nu mai au scop avand in vedere ca au fost scrise doar pentru a specifica ce informatii unde trebuie introduse. Pentru aceasta spargere vom apela la functia split din Regex, care ne va returna intr-un array fiecare linie noua. Dupa ce s-au aflat produsele si cantitatea acestora de care sunt nevoie in comanda, se trec printr-un ,,filtru,, care verifica daca un produs se afla in meniu sau nu. Daca nu raman produse in urma filtrului in comanda, se va afisa un mesaj corespunzator in care se va specifica ca au fost cerute doar produse care nu se afla in meniu. Codul care realizeaza aceste operatii este:

```
ArrayList<MenuItem> restaurantMenu = admin.getMenu();
	String checkText = "Table num: \nEnter order in newline below: (product x quantity)\n ";
	if (orderText.equals(checkText))
	{
		System.out.println ("You have to enter something in order to place the order! ");
	}
	else
	{
		String[] arr = orderText.split("\n", -2);
		String tableNum = arr[0].substring(11);
		ArrayList<MenuItem> orderList = new ArrayList<MenuItem> ();
		MenuItem x;
		int db=0;
		for (int i = 2; i < arr.length; i++) {
			String[] arr2=arr[i].split(" x ",-2);

			if (checkExist (arr2[0], restaurantMenu))
			{
				db++;
				String price = getPrice(arr2[0],restaurantMenu);
				x = new MenuItem(arr2[0], arr2[1], price);
				orderList.add(x);
			}
			else
			{
				System.out.println(arr2[0] + " is currently not in menu!" );
			}
		}
```

```
if (db == 0)
{
        System.out.println( "There are no other items to be served!" );
}
```

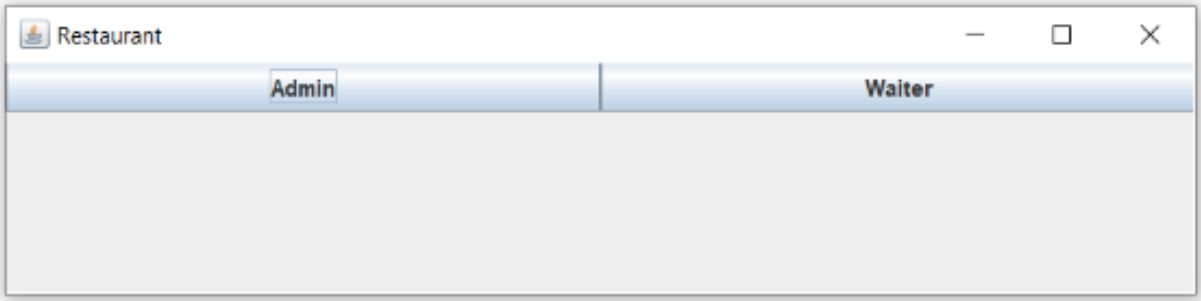
Daca nu se mai gaseste nicio greseala de input, se va plasa comanda care va fi adaugata la hashmap-ul in care se afla toate comenzile. Trebuie mentionat ca in acest pas va fi notificat bucatarul ca este nevoie sa gateasca.

O alta metoda care este relativ importanta este cea in care se genereaza nota de plata. Dupa fiecare comanda plasata cu succes se apeleaza aceasta functie in care practic se creeaza un fisier nou cu un ID unic in care se vor scrie informatiile: identificatorul comenzii, masa de la care s-a plasat comanda, iar apoi sunt insirate produsele care s-au cerut impreuna cu cantitatea si pretul acestora. Intre timp se calculeaza totalul de plata, care presupune doar inmultirea cantitatii cu pretul, mai apoi aceste produse fiind insumate. Codul pentru aceasta:

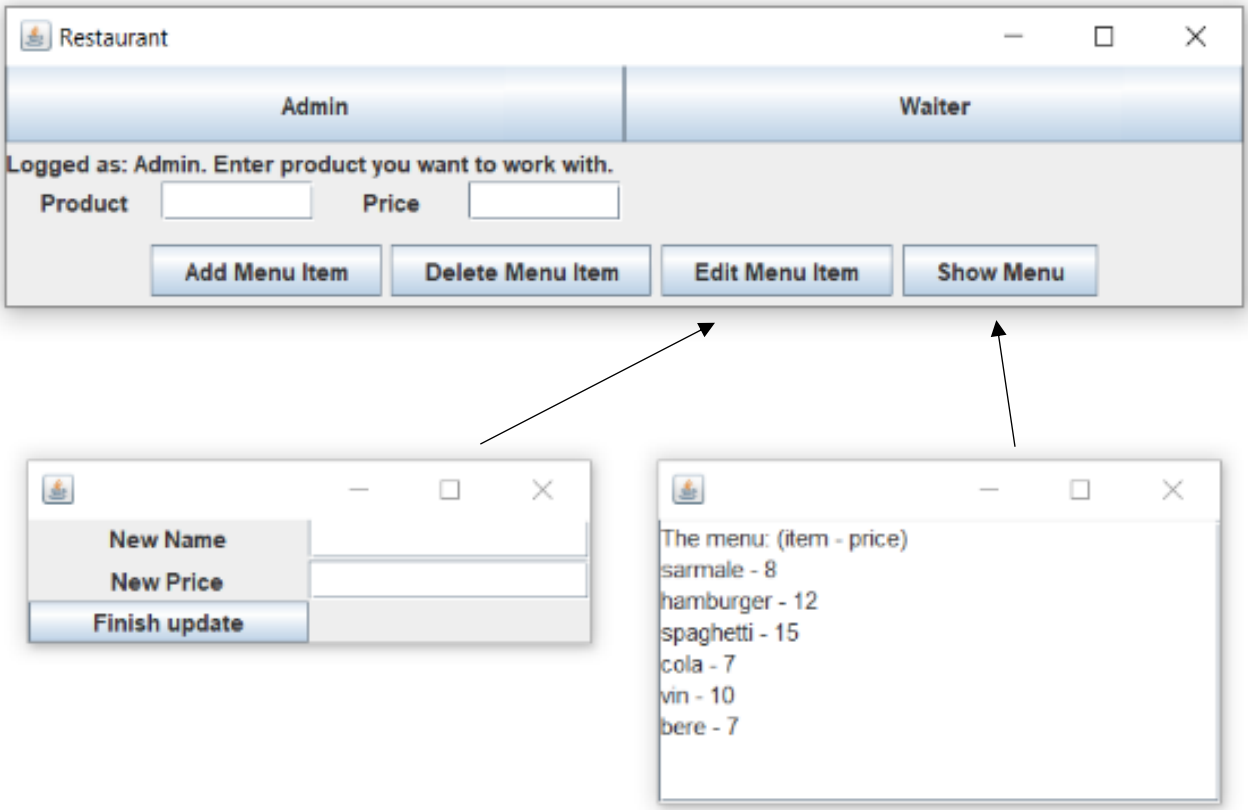
```
String name="Bill"+Integer.toString(billID)+".txt";
FileWriter output=new FileWriter (name);
String answer="";
int totalPay=0;
answer += "Order ID: ";
answer += Integer.toString(id);
answer += "\n";
answer += "Table num: ";
answer += order.getIdTable();
answer += "\n";
for (MenuItem m: order.getMenu())
{

answer += "Name: " + m.getName() + "; Quantity: " + m.getOrderQuantity() + "; Price: " + m.getPrice() + "\n";
totalPay += (int)(Integer.parseInt(m.getOrderQuantity()) * Integer.parseInt(m.getPrice()));

}

answer += "With a total payment of: ";
answer += Integer.toString(totalPay);
answer +="\n";
output.write(answer);
output.close();
billID++;
```

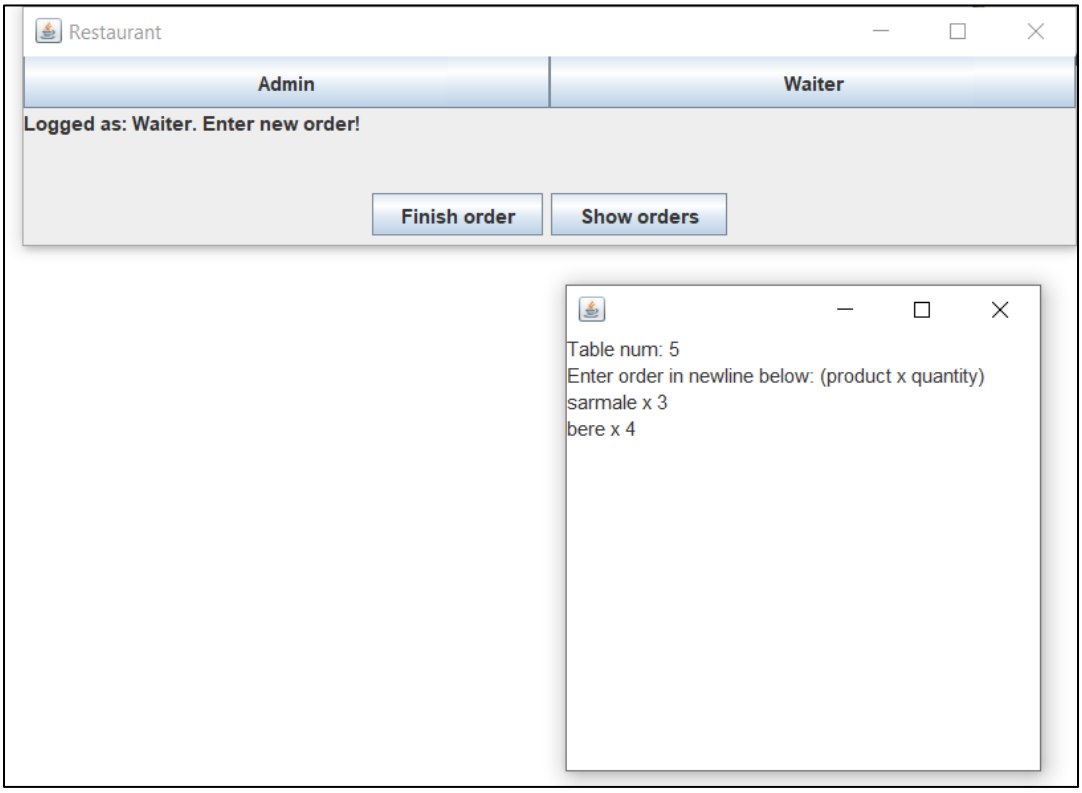
4) Rezultate

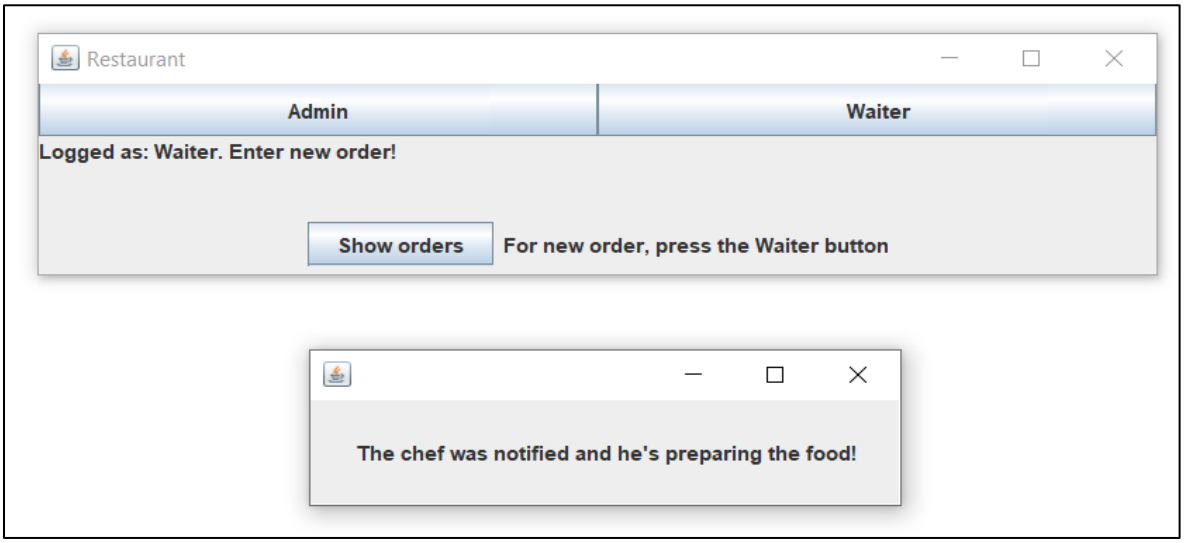
Corectitudinea si rezultatele se pot verifica ruland programul, primul lucru care se poate oberva este o ferestra in care trebuie ales sa ne logam ca si administrator sau chelner:



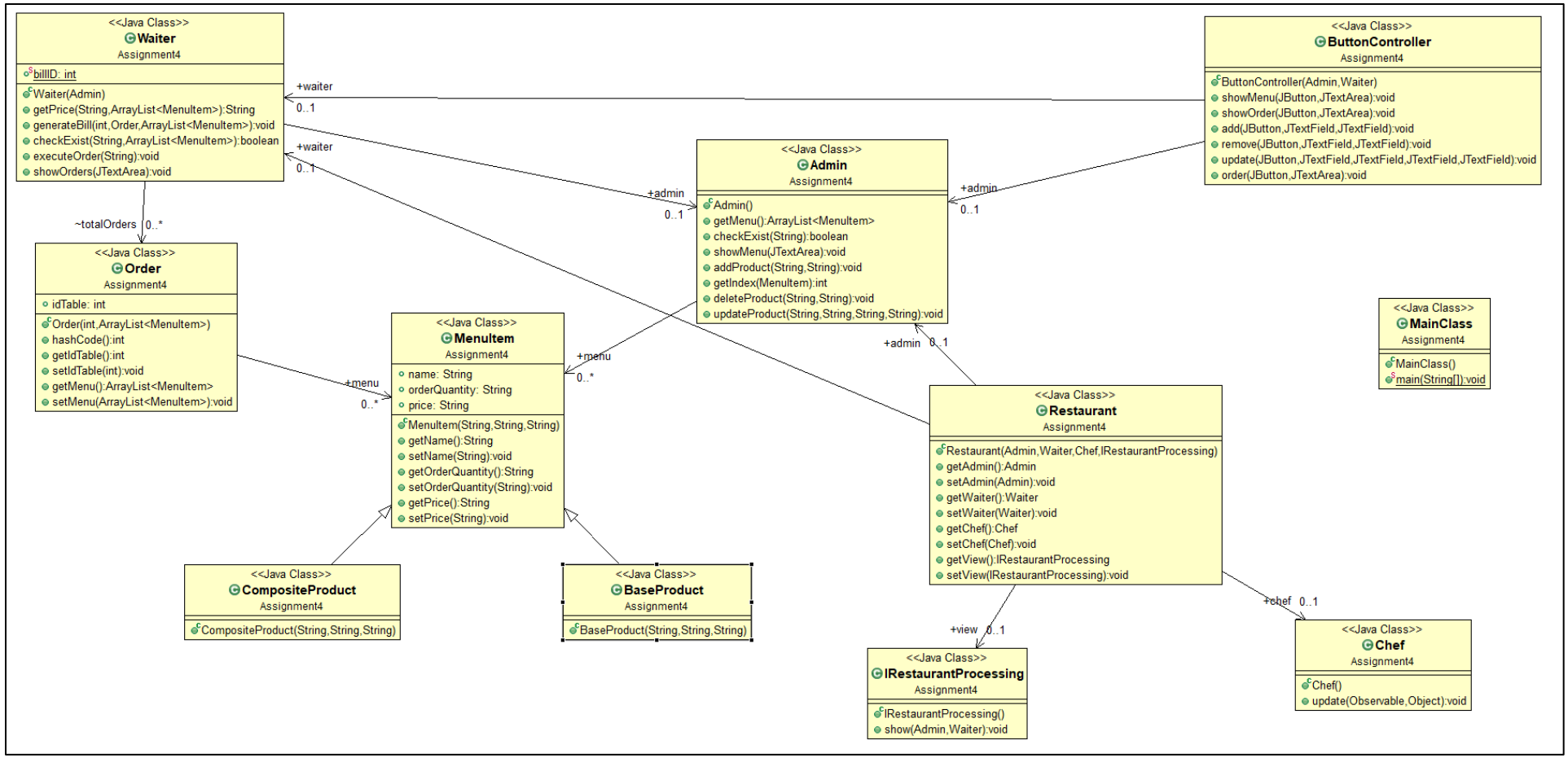
Depinzand pe ce se alege, vom avea doua posibilitati:




Restaurant
Admin
Waiter
Logged as: Waiter. Enter new order!
Finish order
Show orders
Table num: 5
Enter order in newline below: (product x quantity)
sarmale x 3
bere x 4


Restaurant
Admin
Waiter
Logged as: Waiter. Enter new order!
Show orders
For new order, press the Waiter button
The chef was notified and he's preparing the food!

Iar diagrama claselor per total arata in urmatorul fel:



6) Concluzii

In concluzie, aceasta tema (ca si cele anterioare) au avut un inpact bun asupra cunostintelor in sensul ca am reusit sa pun in practica noi cunostinte (de exemplu serializarea, design pattern-ul observer etc.) M-au ajutat foarte mult lucrurile pe care le stiam din temele anterioare, deci cu aceasta tema am reusit sa-mi reinprospatez informatiile legate de java.

Daca este sa vorbim despre eventuale dezvoltari in viitor, as putea sa numesc cateva instantaneu, de ex la generarea notei de plata sa apara data si ora curenta, ca sa fie cat mai aproape de realitate. Si totodata daca vorbim despre data, s-ar putea pune ca si un camp la Order. O alta dezvoltare posibila la partea de generare a notei este ca in cazul in care de la aceeasi masa se cer comenzi de mai multe ori, in final, toate comenzile sa fie pe aceeasi bon (pentru cu in cazul de fata ar fi pe doua chitante diferite).

La partea de input s-ar putea realiza o modificare care elimina spatiile in plus (sau introduce unde este nevoie de ele) care duce la faptul ca inputul nu va fi atat de restrictiv in ceea ce priveste spatiile in plus sau in minus.

Dezvoltarile cele mai importante in opinia ar fi ca MenuItem sa fie implementat ca la carte. Adica ma refer la faptul in varianta actuala exista doar un singur tip de produs, dar ar fi frumos sa fie implementat cum se zice in cerinta, cu baseProduct si compositeProduct. Pentru aceasta insa este necesara intelegerea acestor aspecte, care pentru mine nu au fost prea clare.

Si in ultimul rand, o modificare care ar aduce toate aplicatia mai aproape de realitate, ar fi eventual sa se introduca un stoc, in care sunt specificate din ce produs cate se afla in acest moment in bucataria / magazie, si in urma unei comenzi acestea sa fie decrementate corespunzator. Sau mai mult, sa se introduca si o operatie de ,,refill,, in care se aprovizioneaza restaurantul cu noi produse pentru a satisface nevoile clientilor pe cat se poate mai mult.

7) Bibliografie

http://www.tutorialspoint.com/java/java_serialization.html

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

http://coned.utcluj.ro/~salomie/PT_Lic/4_Lab/Assignment_4/Assignment_4_Indications.pdf

https://stackoverflow.com/questions/8849063/adding-a-scrollable-jtextarea-java/41238607