



# Assignment 4

## *Restaurant management system*

Ilovan Bianca-Maria  
Grupa: 302210

# **Cuprins:**

**1.Obiectivul temei**

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

**2.1.Analiza problemei**

**2.2.Modelare**

**2.3.Cazuri de utilizare**

**2.4.Scenarii**

**3.Proiectare**

**3.1.Decizii de proiectare**

**3.2.Diagrama UML**

**3.3.Diagrama de Pachete**

**3.2.Structuri de date, algoritmi**

**4.Implementare**

**5.Rezultate**

**6.Concluzii**

**7.Bibliografie**

# 1.Obiectivul temei:

**Obiectivul principal** al temei era realizarea unei aplicatii ce reprezenta practic sistemul de organizare al unui restaurant cu 3 tipuri de useri: administrator, chelner si bucatar. Diferentierea dintre ei se facea mai exact prin rolul pe care fiecare il are pentru buna desfasurare a unui restaurant. Administratorul poate adauga, sterge sau modifica anumite produse din meniu. Chelnerul poate lua comenzi, adaugand “elemente” din meniu, cat si este responsabil de facturile pentru clienti dupa ce acestia au comandat. Bucatarul primeste o notificare de fiecare data cand trebuie sa prepare ceva in urma unei comenzi.

Acest obiectiv principal implica si alte **obiective secundare**, cum sunt: lucrul cu Composite Design Pattern pentru a defini clasele din MenuItem, Observer Design Pattern pentru ca chef-ul sa primeasca o notificare cand are ceva de gatit, lucrul cu Jtable pentru afisarea elementelor introduse in interfata grafica, dar si familiarizarea cu assertions, pre si post conditions

## 2.Analiza problemei, modelare, scenarii, use case-uri

### 2.1.Analiza problemei

Dupa cum rezulta din analiza cerintei assignment-ului, aplicatia s-ar concretiza prin implementarea unei interfete grafice pentru 2 dintre cei 3 useri si anume: administrator si waiter. Atat administratorul cat si chelnerul au atributii specifice. Astfel:

Administratorul este responsabil cu:

- adaugarea unor produse in meniu
- stergerea unor produse din meniu
- modificarea unor produse din meniu, avand posibilitatea sa poate vizualiza la orice moment de timp ce a modificat, adaugat sau sters.

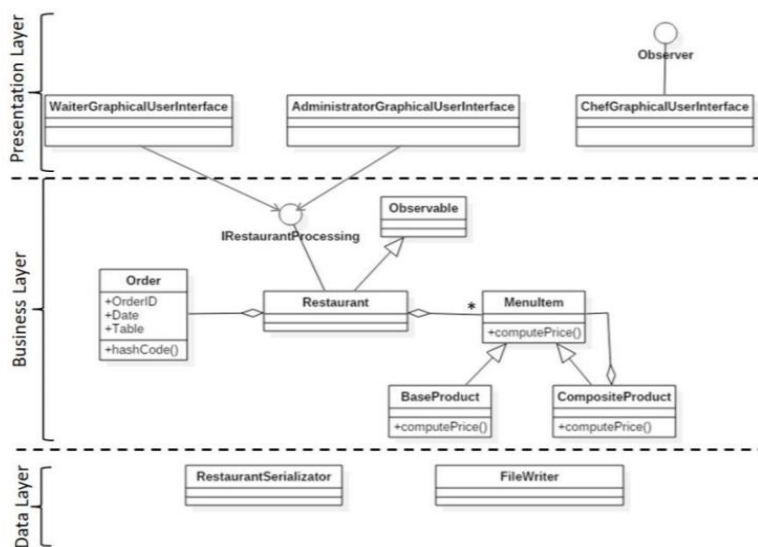
Chelnerul este cel care:

- se ocupa cu preluarea unor unoi comenzi
- adauga elemente din meniu in lista de comenzi
- are responsabilitatea de a crea o nota de plata pentru o comanda preluata

Desi nu are o interfata grafica, bucatarul ar trebui sa primeasca o notificare in momentul in care exista o comanda, fiind instiintat ca are ceva de preparat.

## 2.2.Solutie aleasa pentru modelarea problemei

Avand in vedere cerintele, am considerat potrivit ca aplicatia mea **Restaurant management system** sa respecte diagrama data in cerinta assignmentului, cu mentiunea ca am ales sa imi fac cate o clasa separata pentru fiecare “operatie” care poate fi realizata fie de catre administrator, fie de catre chelner, tocmai pentru o mai buna organizare a aplicatiei. Probabil o varianta mai “eleganta” ar fi fost sa am o pagina principala din care sa poata fi selectat userul dorit, insa am ales ca atat interfata administratorului, cat si cea a chelnerului sa se deschida simultan in urma rularii proiectului.

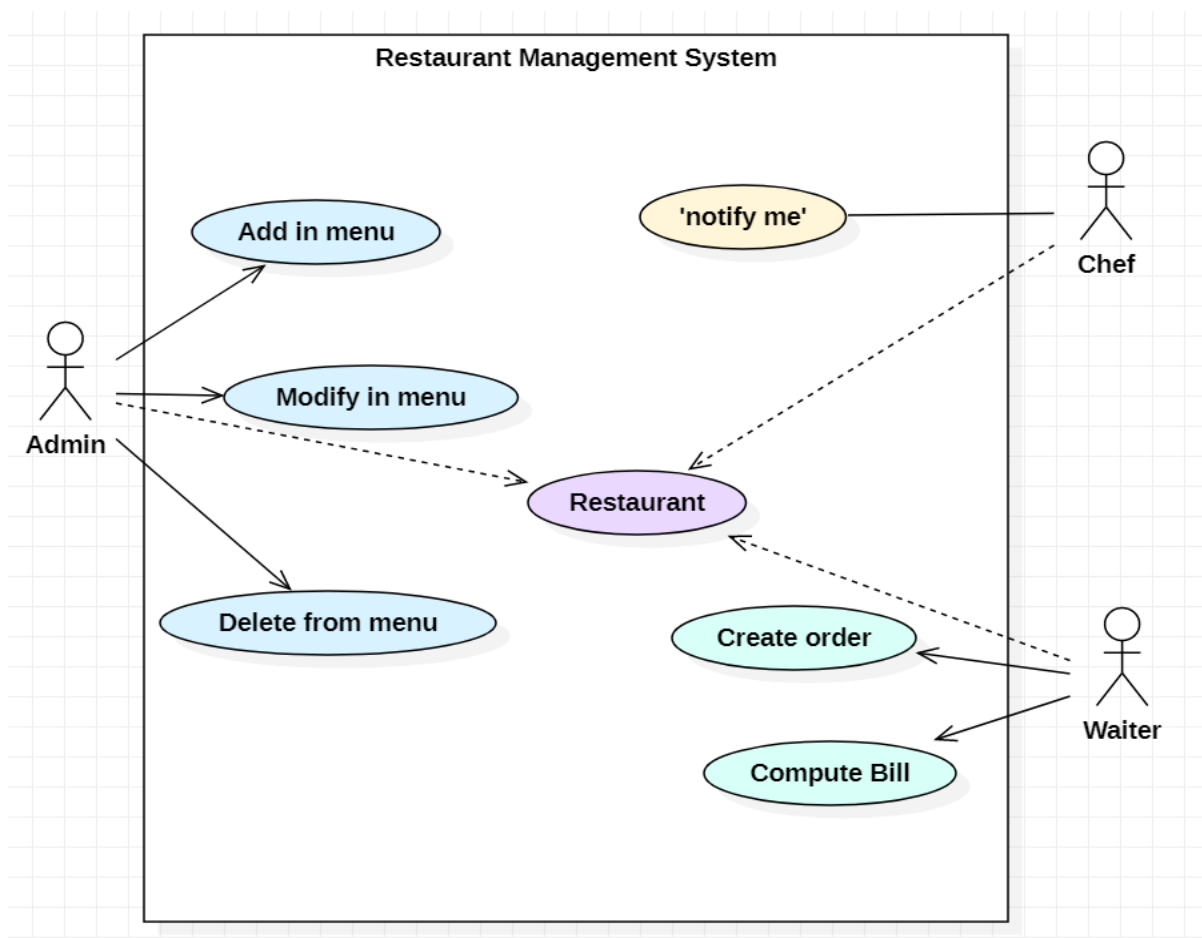


O alta decizie a fost sa nu folosesc serializarea, pentru ca nu am avut timpul necesar, astfel clasa de `RestaurantSerializator` lipseste din aplicatia mea.

## 2.3.Use case-uri

Functionalitatea aplicatiei e descrisa de diferite cazuri de utilizare ale acesteia, in cazul restaurantului fiind vorba despre rolurile avute de fiecare user in parte, amintite mai sus, astfel incat sa poate fi indeplinite cerintele clientilor. Aceasta aplicatie poate fi rulata atat in Eclipse cat si cu ajutorul urmatoarei comenzi:

```
java -jar <the_absolute_path.jar>
```



User-ul trebuie sa ruleze practic aplicatia, iar mai apoi, in urma celor 2 combo box-uri deschise, sa aleaga ce operatii efectueaza, iar mai apoi sa introduca datele necesare. Pentru a rula din linia de comanda, se foloseste comanda precizata mai sus. In functie de alegerea userului, pe care utilizatorul are dreptul sa-l aleaga, aplicatia incearca sa dea raspunsul dorit si totul sa mearga conform "scenariului".

## 2.4.Scenarii

**Use case:** se rezuleaza aplicatia fie din main class din eclipse, fie ruland fisierul jar

**Primary actors:** administrator, waiter

**Scenariu de succes:**

Pentru a utiliza aplicatia sunt necesare urmatoarele:

1. Rularea programului, moment in care se vor deschide 2 combo box-uri
2. Alegerea combo box-ului pentru Administrator, intrucat acesta este cel care trebuie sa adauge produse in meniu
3. Adaugarea in meniu a mai multor Base Products sau Composite Products, apasand butonul "Add in menu"
4. Vizualizarea produselor introduse la pasul anterior, prin apasarea butonului "View all" pentru a vedea ca acestea s-au introdus cu succes.
5. Optional, stergerea sau modificarea unui produs introdus anterior, apasand butoanele "Delete from menu", respectiv "Change in menu"
6. Odata ce meniul a fost updatat, vom alege combo box-ul potrivit pentru Waiter, intrucat acesta este cel care poate face o comanda
7. Prin apasarea butonului "Create new order", chelnerul va putea crea o comanda pe baza produselor existente in meniu.
8. Chelnerul va introduce detaliile necesare specificate: numele produsului, pretul, un id al comenzii si numarul mesei.
9. Ulterior, comanda va putea fi vizualizata prin apasarea butonului "View All"
10. Pentru crearea unei note de plata, se va apasa butonul "Compute Bill", loc in care se va introduce pretul ce trebuie achitat.

## 3.Proiectare

### 3.1. Decizii de proiectare

O prima decizie de proiectare a fost structurarea aplicatiei in 4 pachete, dupa modelul diagramei date: **businessLayer, dataLayer, presentation, PT2020.Assignment4**.

Desi am urmat modelul diagramei de clase date, am ales sa fac fara serializare, deci clasa RestaurantSerializator lipseste. Am ales sa imi creez cate o clasa pentru fiecare "operatie" ce poate fi executata fie de catre administrator, fie de catre waiter. Practic, am ales ca interfetele pentru fiecare dintre cei 2 useri sa contina cate un buton pentru fiecare operatie ce poate fi executata. In functie de butonul ales, datele vor trebui introduse in text field-uri denumite sugestiv, date care ulterior vor fi introduse si intr-un Jtable.

## 3.2. Diagrama UML

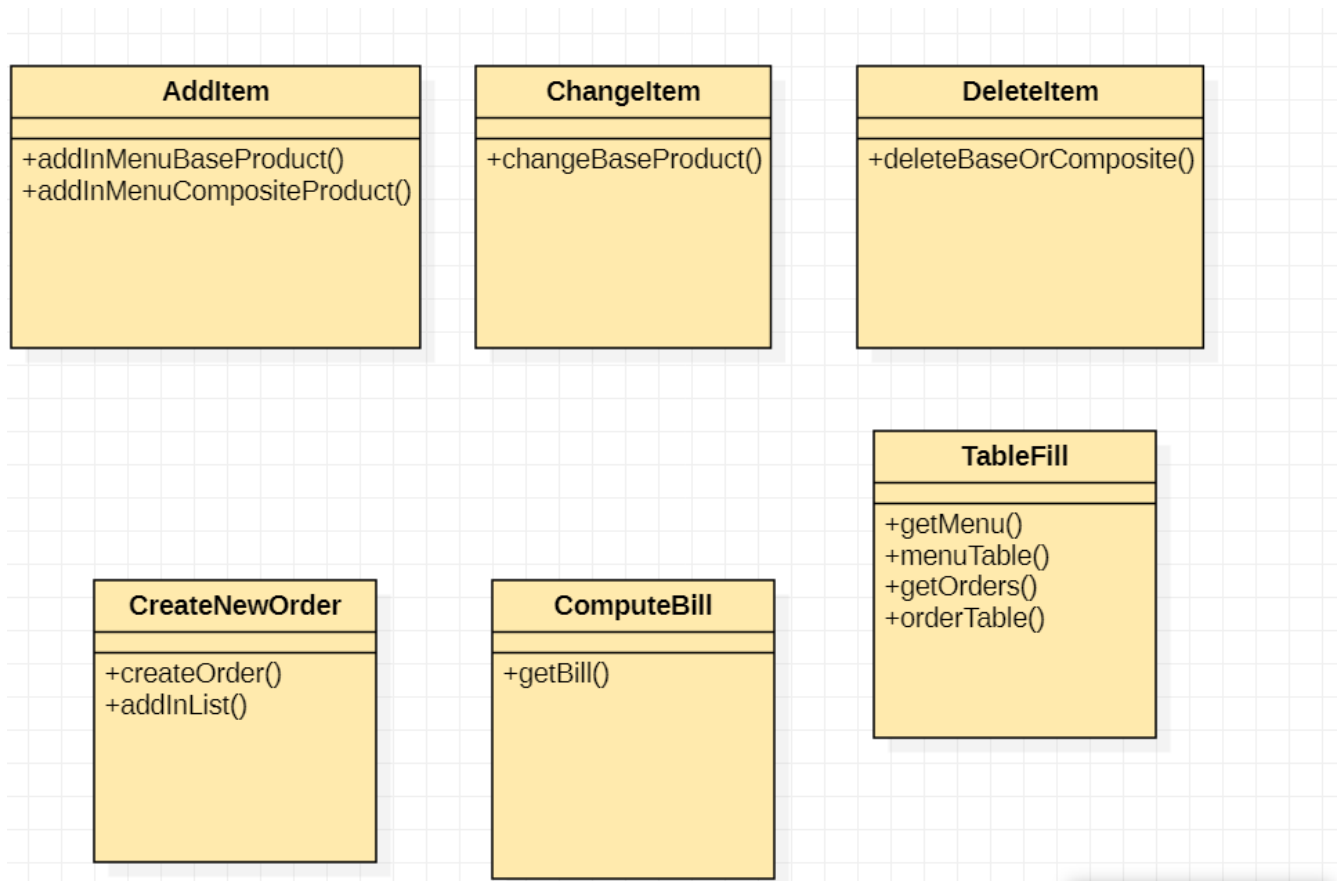
Pachetul **businessLayer** contine clasele: *BaseProduct*, *CompositeProduct*, clase ce extind clasa abstracta *MenuItem*, *IRestaurantProcessing*, *Restaurant*, *Order* si *Observable*.

Pachetul **dataLayer** contine clasa *FileWrite*.

Pachetul presentation contine clasele: - *AdministratorGUI*, *AddItem*, *ChanegItem*, *DeleteItem*  
- *WaiterGUI*, *CreateNewOrder*, *ComputeBill*  
- *ChefGUI*, *Observer*  
- *TableFill*

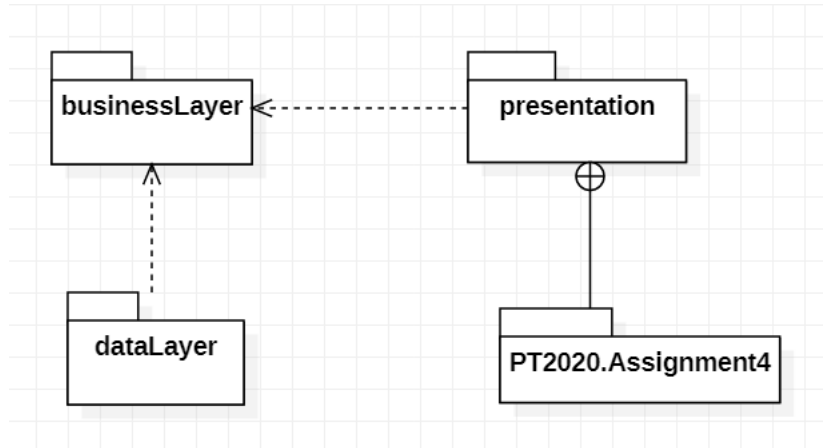
Pachetul **PT2020.Assignment4** contine clasa *App* care este practic clasa main a aplicatiei.

Pe langa diagrama data pe care am respectat-o in mare parte, am urmatoarele clase:



### 3.3. Diagrama de pachete

Cele 4 pachete despre care am facut precizari mai sus, se pot ilustra astfel:



### 3.4. Structuri de date, algoritmi

Ca **structuri de date**, am folosit **ArrayList-uri** de diferite dimensiuni pentru a stoca spre exemplu diverse menu items, comenzi ale clientilor, base products sau composite products. Exemplu de folosire:

```
-ArrayList<Order> clOrders=new ArrayList<Order>();  
-ArrayList<MenuItem> menuItems=new ArrayList<MenuItem>();  
-ArrayList<BaseProduct> bp=new ArrayList<BaseProduct>();  
-ArrayList<CompositeProduct> cp=new ArrayList<CompositeProduct>();
```

De asemenea, o alta structura de date folosita este **HashMap**, reusind sa stochez astfel elementele dupa cheia si valoarea acestora. Un exemplu de folosire:

```
-HashMap<Order,ArrayList<MenuItem>>clientsOrders=new HashMap<Order,ArrayList<MenuItem>>()
```

Cu ajutorul acestor structuri de date, am reusit atat sa inserez cat si sa sterg dintr-un ArrayList, iar in momentul in care se schimba un produs cu alt produs pentru a putea fi inserat din nou pe aceeaasi pozitie trebuia retinuta si pozitia produsului de dinainte.



## 4. Implementare

### 4.1. Clase

Pachetul **businessLayer**:

- Clasa **MenuItem** este o clasa abstracta in care intalnim metodele de **computePrice()** si **getNameP()**, metode care vor fi implementate de cele 2 clase: **BaseProduct** si **CompositeProduct**.

- Clasa **BaseProduct** exinde clasa **MenuItem** si descrie practic un produs de baza, produs pentru care am ales ca attribute: numele acestuia si pretul. Ca metode in aceasta clasa intalnim metodele de **get** si **set**, pentru nume si pret, cat si o metoda **toString()** pentru afisarea acestora.

- Clasa **CompositeProduct** exinde clasa **MenuItem**, descriind un produs compus din mai multe produse de baza. Astfel, ca attribute pentru aceasta clasa am ales: un nume si un **ArrayList<MenuItem>**. Ca metode, in aceasta clasa intalnim metodele de **get**, **set** si **toString()**. Pe langa acestea, apar metodele de **computePrice()**, pentru calcularea pretului produsului compus ca suma produselor de baza. De asemenea, apare metoda de **findAllBaseProducts()** pentru afisarea produselor simple din compozitia produsului.

- Interfata **Observable** contine metodele **addObserver(T o)**, **notifyObserver()**, **deleteObserver(T o)**, metode care adauga, notifica sau sterg un observator. Metodele vor fi implementate in clasa **Restaurant**.

- Clasa **Restaurant** este una din clasele de baza ale aplicatiei intrucat in ea regasim metode ce au legatura cu partea de logica a aplicatiei, metode care vor fi ulterior folosite mai departe. Aceasta implementeaza interfata **Observable**, implementand astfel si metodele definite acolo. Aici avem diferite colectii de elemente. Aici apare metoda **isWellFormed()**, metoda booleana, care va returna true in cazul in care listele de comenzi ale clientilor si de menu item-uri nu sunt nule.

Alte metode importante, fiecare dintre acestea continand *assert isWellFormed()*

- metoda **addSomethingInMenu(MenuItem menuItem)** adauga in meniu produse, fie ca vorbim despre base products, fie composite products.

- metoda **deleteSomethingFromMenu(ArrayList<MenuItem> whatYouWantToRemove)** sterge dintr-un ArrayList de menu item-uri produsul dorit de admin.

- metoda **changeSomethingInMenu(int k, Array<MenuItem> menuItems, MenuItem changeBase)** schimba dintr-un ArrayList de menu item-uri un produs cu un alt menu item, fiind necesara transmiterea pozitiei noului element ce vrem sa-l introducem, cu pozitia elementului ce dorim a fi inlocuit.

- metoda **addOrder(Order order)** verifica in prima faza ca order sa nu fie null. Rolul acestei metode este de a adauga o comanda, pe care ulterior o vom vizualiza in Jtable.

- Clasa **Order** are ca attribute id-ul comenzii, o lista de comenzi ale clientilor, data in care s-a facut comanda si masa pentru care se face comanda practic. Metoda specifica acestei clase este **findOrderPrice()**, metoda care returneaza pretul total la comenzii.

- Interfata **IRestaurantProcessing** contine metodele:

- pentru administrator: **newMenuItem()**, **changeInMenu()**, **deleteMenuItem()**, fiecare avand pre si post conditiile specifice.

- pentru waiter **newOrder()**, **newBill()**, fiecare avand pre si post conditiile specifice.

Pentru pachetul **dataLayer**:

- Clasa **FileWrite** contine metode **writer(String file)**, metoda care se ocupa cu crearea unui fisier, in cazul nostru, factura pentru client care va contine pretul total ce trebuie achitat.

Pentru pachetul **presentation**:

- Clasa **AdministratorGUI** este practic “interfata” administratorului care contine 4 butoane: Add in menu, Delete from menu, Change in menu si View All. Fiecare din aceste 4 butoane are cate o clasa separata. Importanta in aceasta clasa este metoda **showMenu()**, metoda datorita careia vom putea vizualiza in tabel ce produse au fost introduse in urma apasarii butonului Add in menu. Astfel, metodele principale ale acestei clase sunt **newMenuItem()**, **changeMenuItem()**, **deleteMenuItem()** si **viewAll()**.

- Clasa **AddItem** implementeaza metodele de adaugare a unor base products si composite products in meniu. Pentru asta avem functiile **addInMenuBaseProduct()**, pentru adaugarea unui produs de baza in meniu si **addInMenuCompositeProduct()**, pentru adaugarea unui produs compus din base products in meniu.

- Clasa **ChangeItem** implementeaza metoda **changeBaseProduct()**, pentru schimbarea unui produs de baza introdus, cu unul nou, pe aceeasi pozitie, folosindu-ma de functia **changeSomethingInMenu()** din clasa Restaurant.

- Clasa **DeleteItem** contine metoda **deleteBaseOrComposite()**, pentru stergerea unui base product sau a unui composite product.

- Clasa **WaiterGUI** este “interfata” pentru chelner, continand 3 butoane: Create new order, Compute Bill si View All. Fiecare din aceste 3 butoane are cate o clasa separata.

- Clasa **CreateNewOrder** contine metoda **createOrder(ArrayList<MenuItem> arraylist)**, metoda care creeaza o comanda pentru client. Produsele pe care le avem la dispozitie sunt cele introduse de administrator in meniu. Am cate o functie de adaugare de base product si adaugare de composite product, functii pe care ulterior le voi folosi in metoda **ArrayList<MenuItem> addInList()**

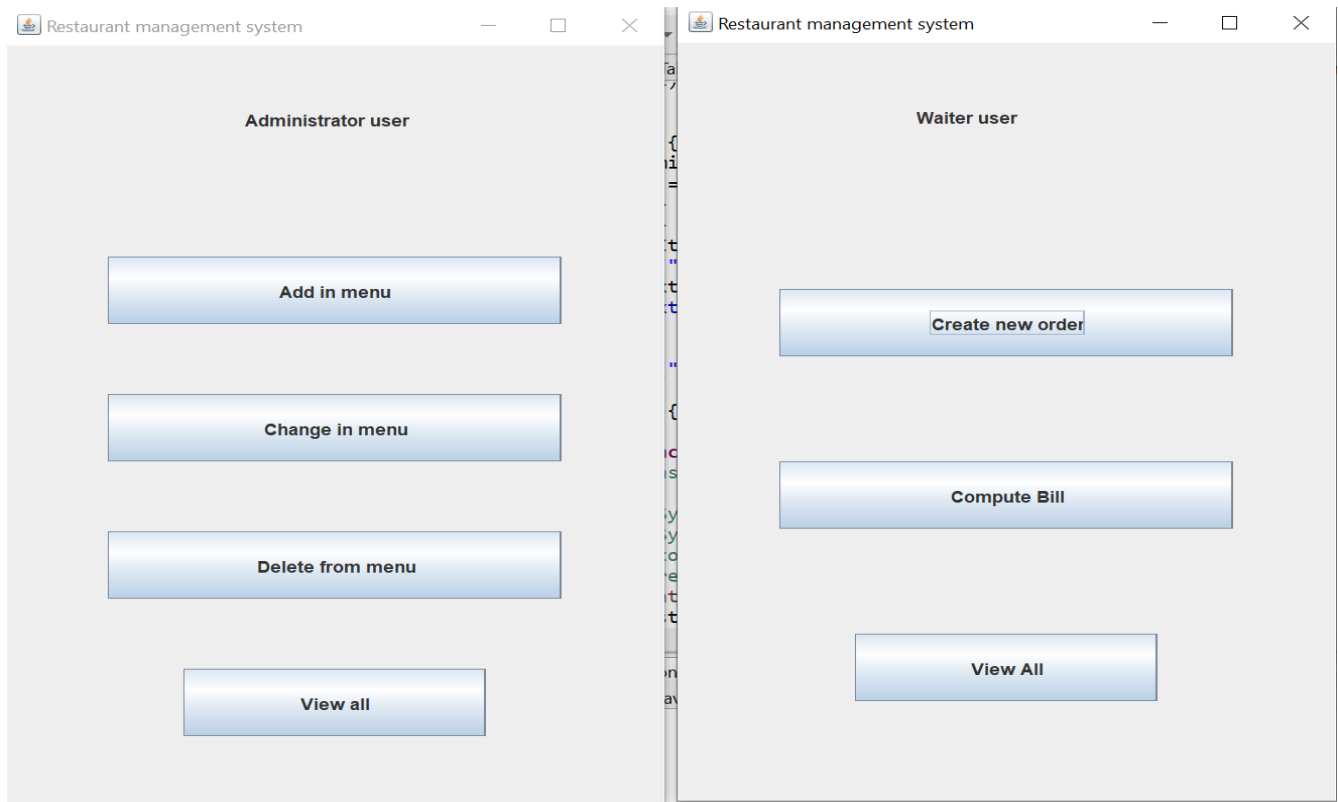
- Clasa **ComputeBill** contine metoda **getBill()** in urma careia, in cazul in care chelnerul va vrea sa faca nota de plata, va putea sa introduca pretul comenzii avute, ca mai pe urma sa se genereze fisierul bil.txt cu pretul respectiv.

- Clasa **TableFill** ma ajuta la crearea tabelelor in care ulterior vor fi introduse fie produsele din meniu, fie comenzile clientilor. Pentru asta am functiile **getMenu(ArrayList<MenuItem>)**, **menuTable(String[][] menu)**, respectiv **getOrders(ArrayList<Order>)**, **orderTable(String[][] order)**.

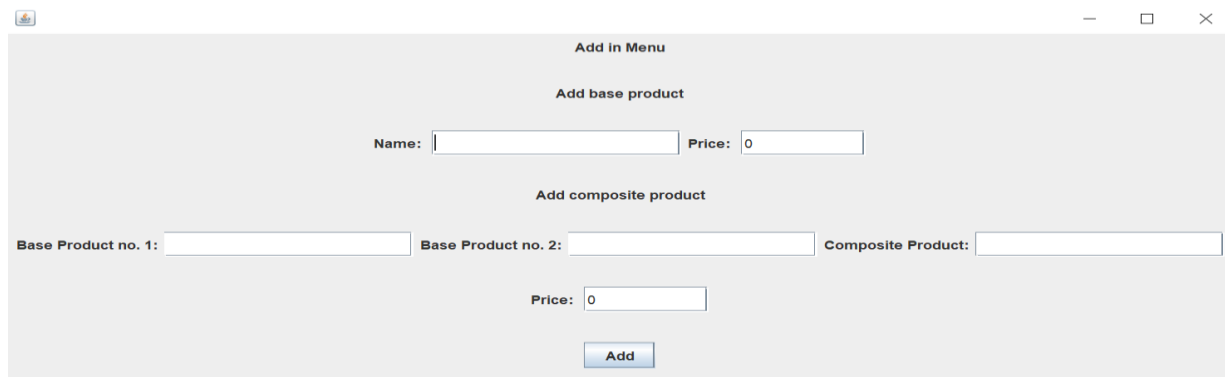
- Clasa **ChefGUI** implementeaza metoda de update, care va afisa un mesaj in momentul in care exista o comanda, bucatarul trebuind instiintat.

## 5. Rezultate

În urma rularii programului se vor deschide 2 combo box-uri: unul pentru administrator, altul pentru waiter.



Administratorul este cel care trebuie să adauge în meniu produsele. Astfel, în momentul apăsării butonului “Add in menu” acesta poate adauga atât un base product, cât și un composite product format din 2 base-uri.



**Add In Menu**

**Add base product**

Name:  Price:

**Add composite product**

Base Product no. 1:  Base Product no. 2:  Composite Product:

Price:

Dupa acest model se vor introduce menuItem-urile, fiecare in text field-ul sau corespunzator.

Astfel, dacă vom introduce câteva produse, în momentul în care vom apăsa butonul “View All” ar trebui să vedem tot ce am adăugat la pasul anterior.

[illegible]

Se poate observa ca in dreptul fiecarui produs este specificat daca este Base Product sau Composite Product. Detaliile care apar in tabel sunt cele introduse de catre administrator la parte de adaugare in meniu. Tabelul fiind un fel de meniu al restaurantului.

In cazul in care vrem sa modificam base product-ul “rosie” cu “avocado” spre exemplu, apasand butonul “Change in Menu” si introducand pretul produsului ce doreste a fi inlocuit si pretul noului produs in text field-ul “new price” dupa acest model:

Change in Menu

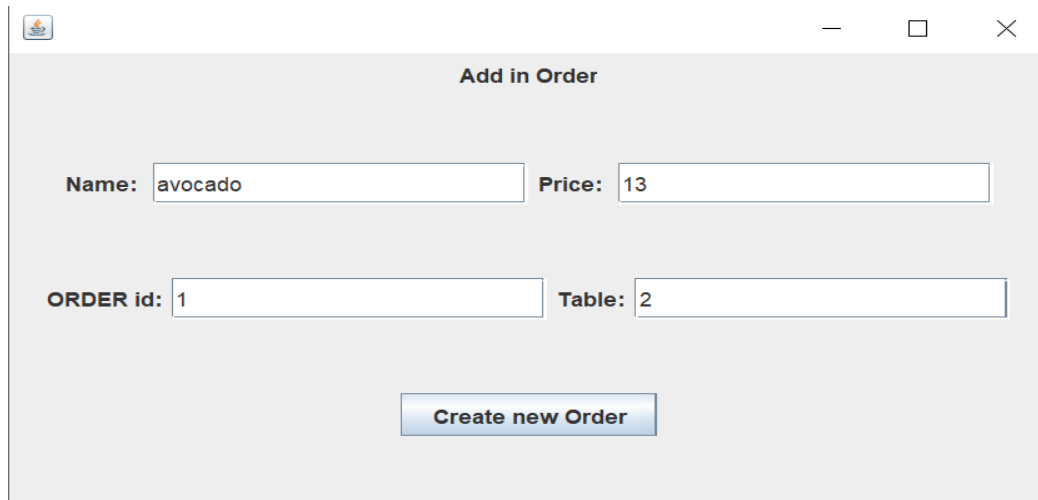
Name:  New name:

Price:  New price:

În urma apăsării butonului “View all” se va putea vedea lista de produse modificată:

[illegible]

Mai departe, odata stabilit meniul, chelnerul este cel care va putea crea o noua comanda in functie de dorintele clientului. In cazul in care clientul vrea sa comande avocado, in urma apasarii butonului “Create new order” se va putea crea o noua comanda cu produsul dorit. Astfel, chelnerul trebuie sa introduca produsul dorit, pretul acestuia, un id al comenzii si masa de la care se ia comanda:



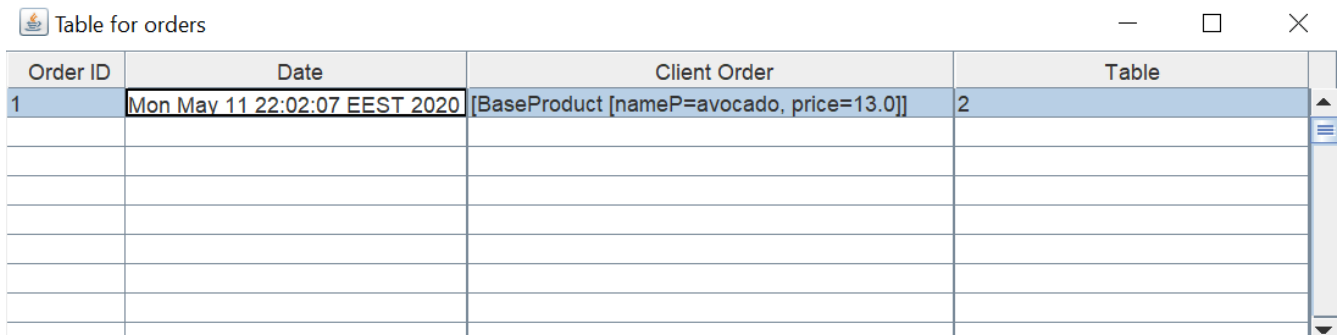
**Add in Order**

**Name:**  **Price:**

**ORDER id:**  **Table:**

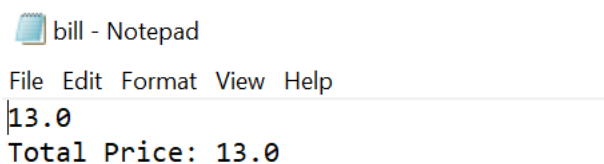
**Create new Order**

Acesta va putea vedea lista comenzilor apasand butonul “View all”.



Order ID	Date	Client Order	Table
1	Mon May 11 22:02:07 EEST 2020	[BaseProduct [nameP=avocado, price=13.0]]	2

Daca se doreste “emiterea” unei facturi, chelnerul va apasa butonul de “Compute Bill”, va introduce pretul comnezii si un fisier bill.txt se va crea, factura ce contine pretul de platit.



bill - Notepad

File Edit Format View Help

13.0

Total Price: 13.0

## 6. Concluzii

Datorita acestei teme am intrat in contact mai mult cu tot ce tine de interfata unei aplicatii, intrucat nu prea am avut cerinte legate de interfata pana acum. De asemenea, un alt lucru importat cu care nu am mai intrat in contact pana acum e tot ce tine de Jtable-uri.

De asemenea, am invatat cate ceva despre HashMap-uri si am inteles mult mai multe decat o facusem semestrul trecut la OOP.

Lucrul cu pre si post conditii a fost un lucru cu care nu m-am mai inalnit pana acum si cred ca poate fi considerat destul de util.

## 7. Bibliografie

<https://stackoverflow.com/>

[https://www.tutorialspoint.com/design\\_pattern/observer\\_pattern.htm](https://www.tutorialspoint.com/design_pattern/observer_pattern.htm)

<https://github.com/>

[https://www.w3schools.com/java/java\\_hashmap.asp](https://www.w3schools.com/java/java_hashmap.asp)

<https://www.geeksforgeeks.org/java-util-hashmap-in-java-with-examples/>

<https://docs.oracle.com/javase/tutorial/uiswing/components/table.html>

<https://docs.oracle.com/cd/E19683-01/806-7930/assert-13/index.html>