**DOCUMENTATIE TEMA 4**

**RESTAURANT MANAGEMENT SYSTEM**

Student: Ionescu Raluca Ionela

Grupa: 302210

**1.Obiective**

**1.1 Obiectivul principal al proiectului**

Obiectivul principal al proiectului este acela de a implementa o aplicatie , Restaurant Management System, ce ar trebui sa aiba 3 tipuri de utilizator: administrator, waiter (chelner) si chef (bucatar) . Administratorul poate sa adauge, sa stearga si sa modifice produse din meniul unui restaurant. Waiter-ul poate crea o comanda noua prin adaugarea de produse existente in meniu, si poate sa creeze o chitanta pentru comanda respectiva. Bucatarul este notificat de fiecare data cand o coamnda este adaugata de chelner.

Pentru a simplifica aplicatie vom presupune ca exista un singur administrator, un singur chelner, si un singur bucatar.

**1.2 Obiective secundare**

|  |  |  |
| --- | --- | --- |
| **Obiectiv secundar** | **Descriere** | **Capitol** |
| Dezvoltarea de use case-uri si scenarii | “În sistemele si ingineria software, un ‘use case’ este o listă de acțiuni sau pași de eveniment care definesc de obicei interacțiunile dintre un rol și un sistem pentru a atinge un obiectiv. Actorul poate fi un sistem uman sau alt sistem extern.” | 2 |
| Alegerea structurilor de date | Structurile de date folosite pentru realizarea proiectului | 3 |
| Proiectare clase, diagrama UML | Proiectarea claselor si a pachetelor , precum si prezentarea diagramei UML | 4 |
| Rezultate | Se vor prezenta scenariile pentru testatrea efectruarii operatiilor de catre un administrator, bucatar sau chelner | 5 |
| Concluzii | Se vor prezenta concluziile cat si posibilele dezvoltari ulterioare | 6 |

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

**2.1 Analiza problemei**

Ne propunem ca aplicatia sa creeze o interfata din care se poate alege modul de administrator, waiter sau chef.

In cazul in care se alege „administrator” urmatoarele optiuni sunt valabile: create new menu item, edit product from menu, delete product from menu si view menu.

In cazul in care se alege „waiter” se pot efectua urmatoarele: add item in menu, add order, view orders, compute bill.

Pentru „chef” interfata va informa user-ul daca bucatarul asteapta sa primeasca o comanda, sau prepara comanda curenta.

**2.2 Scenarii / Use case-uri**

**Use case:** se ruleaza fisierul .jar file

**Primary actor**: administrator

**Cazul de success:**

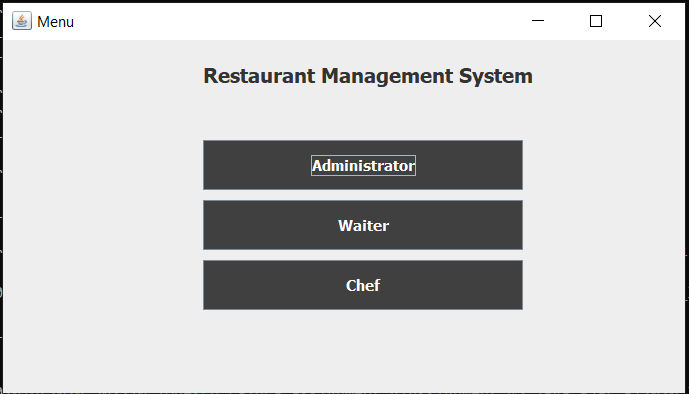
Pentru ca aplicatia sa ruleze, utilizatorul trebuie sa dea ca parametru „restaurant.ser”, pentru ca datele din restaurant introduse anterior sa se poata incarca si folosi.

Utilizarea aplicatiei presupune:

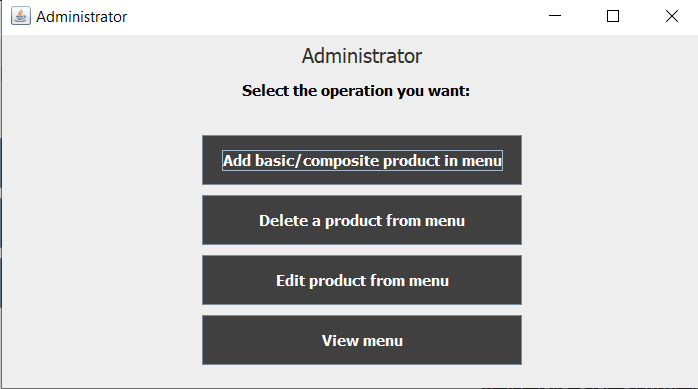
1. Se ruleaza aplicatia cu urmatoarea comanda: ( in cazul in care nu este deja rulata)

Java -jar PT2020\_302210\_Ionescu\_Raluca\_Assignment\_4.jar restaurant.ser

1. O interfata precum cea de jos o sa apara

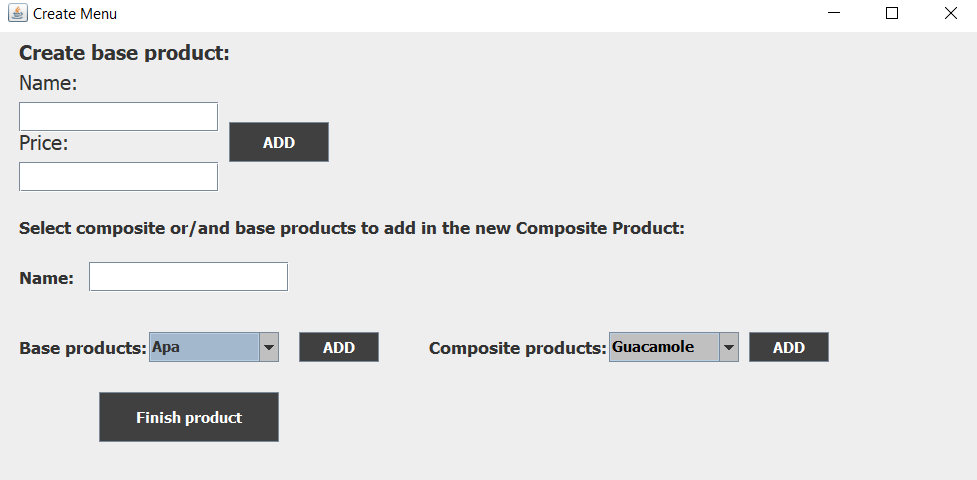


1. **Butonul Administrator este apasat. In consecinta o noua fereastra apare:**



Cele 4 operatii ce se pot efectua sunt :

1. Add basic/composite product in menu:

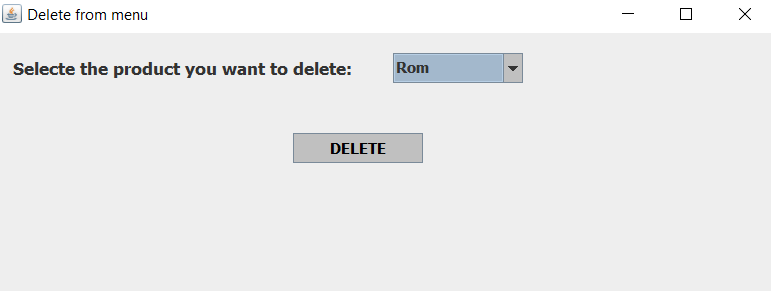


Dupa cum se poate observa, un restaurant poate avea doua tipuri de produse, base si composite product .

Pentru base product, se introduce numele si pretul acestuia, dupa care se apasa pe butonul ADD.

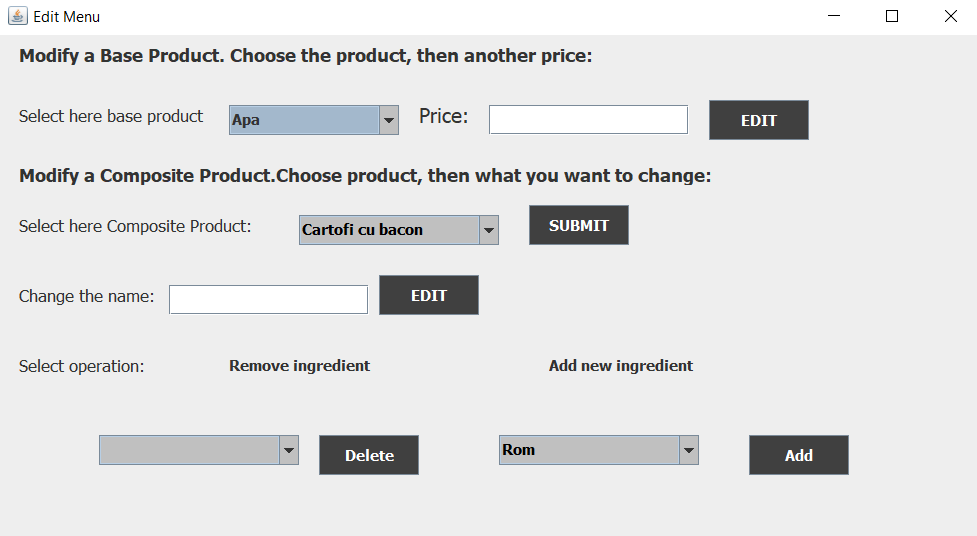
Pentru composite product, se va introduce numele, dupa care se va selecta ce base sau composite products va contine. Pentru fiecare se va apasa pe ADD, astfel se pot adauga oricate produse. Lista de produse este actualizata in timp real, astfel incat daca se introduce un element, acesta apare imediat in lista corespunzatoare si se poate folosi in continuare. Cand suntem multumiti cu produsele ce vor alcatui composite product-ul se va apasa pe Finish product, si acesta va fi adaugat in meniul restaurantului.

1. Delete product from menu



Utilizarea este una simpla si intuitiva. Se va selecta din lista produsul ce se vrea sters, dupa care se apasa pe butonul DELETE.

3. Edit product from menu

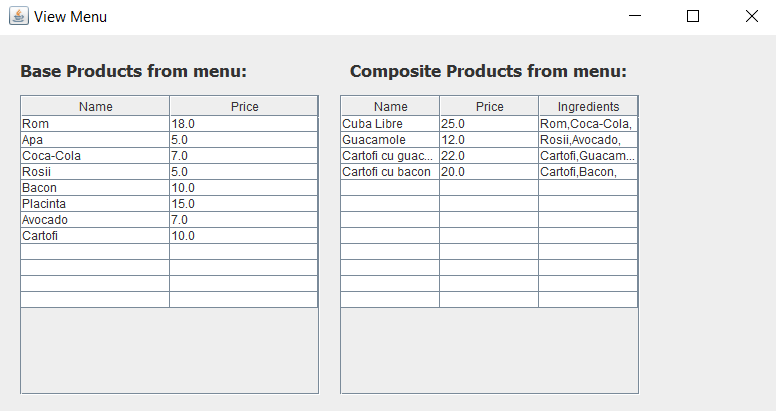


Modificarea se poate face, din nou, atat pentru un base product, cat si pentru unul composite.

Pentru base, se modifica pretul, asa ca, dupa ce se alege produsul dorit si se introduce pretul nou, se apasa pe butonul EDIT.

Pentru composite avem urmatoarele posibilitati ( dupa ce se alege produsul se apasa pe SUBMIT, dupa care se poate edita) : editarea numelui, aduagarea unui nou produs in compozitia lui sau stergerea unuia. Se selecteaza elementele dorite si se apasa pe butoanele corespunzatoare.

4.View menu



In aceasta fereastra o sa apara meniul restaurantului , impartit in Base Products si Composite Products. Se pot vedea numele, pretul cat si ce contin (daca este cazul).

**Scenarii de esec:**

* Se introduc preturi negative
* Se introduce in composite product un produs ce nu exista in menu ( desi listele cu elemente disponibile sunt actualizate mere)

**Use case:** se ruleaza fisierul .jar file

**Primary actor**: waiter

**Cazul de success:**

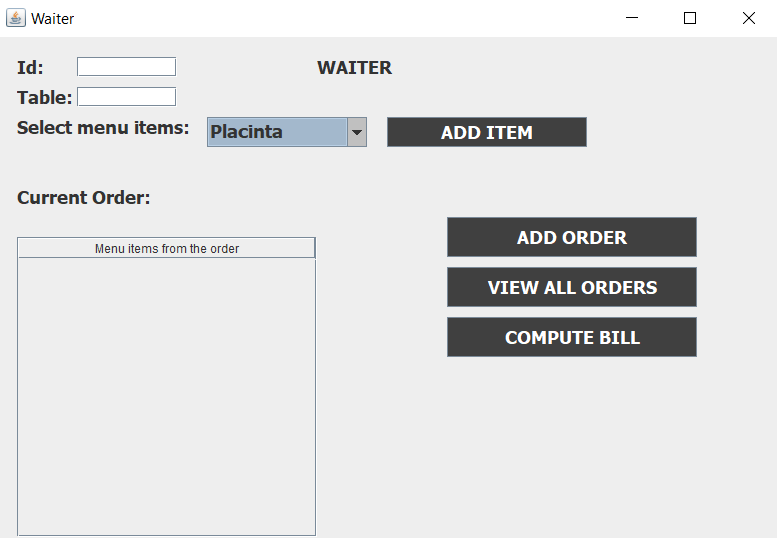
Utilizarea aplicatiei presupune:

1. Se ruleaza aplicatia cu urmatoarea comanda: ( in cazul in care nu este deja rulata)

Java -jar PT2020\_302210\_Ionescu\_Raluca\_Assignment\_4.jar restaurant.ser

1. O interfata precum cea de sus o sa apara

**3.Butonul Waiter este apasat. Urmatoarea fereastra este deshisa:**



Comanda este caracterizata de id, si numarul unei mese, precum si lista de produse ce le contine. Pentru acestea, se selecteaza din meniu ce se doreste , si se apasa pe ADD ITEM. Imediat se poate observa in tabelul „Menu items from the order” ce contine comanda. Dupa ce id-ul si numarul mesei sunt introduse, se poate apasa pe ADD ORDER , iar comanda a fost efectuata. Daca se doreste si emiterea unei chitante se va apasa pe COMPUTE BILL. Dupa ce se adauga comanda toate campurile sunt golite, pentru a se putea adauga mai usor o alta comanda. De asemena se pot vedea toate comenzile prin apasarea VIEW ALL ORDERS.

**Scenarii de esec:**

* Id-ul este un numar negativ, sau nu este numar. Un mesaj corespunzator apare
* Table este un numar negativ, sau nu este numar. Apare un mesaj de eroare.
* Se da compute bill fara sa existe vreo comanda

**Use case:** se ruleaza fisierul .jar file

**Primary actor**: chef

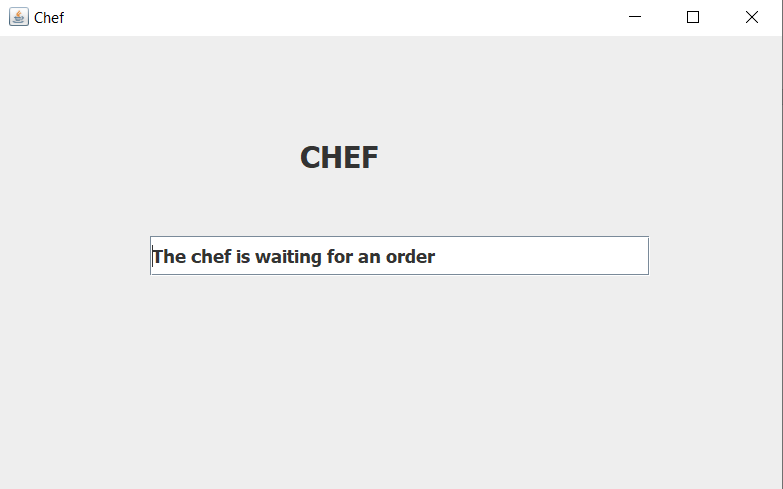
**Cazul de success:**

Utilizarea aplicatiei presupune:

1. Se ruleaza aplicatia cu urmatoarea comanda: ( daca nu este deja rulata)

Java -jar PT2020\_302210\_Ionescu\_Raluca\_Assignment\_4.jar restaurant.ser

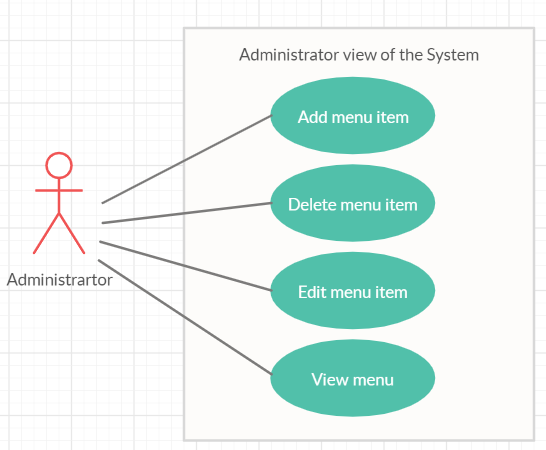
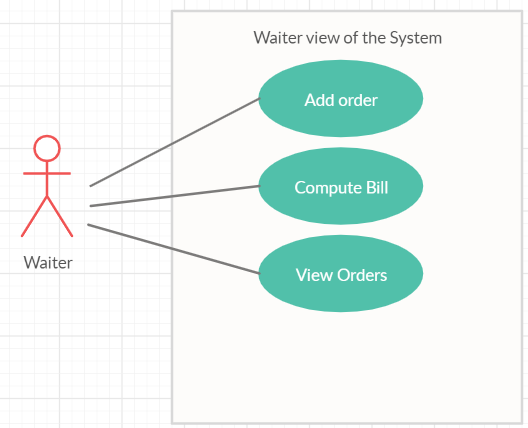
1. O interfata precum cea de sus o sa apara
2. **Butonul Chef este apasat. Se va deschide urmatoarea fereastra:**

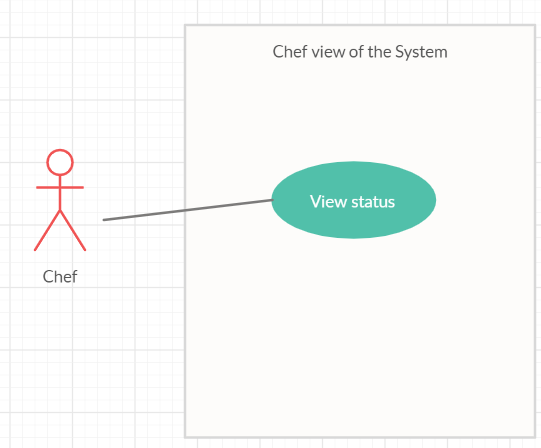


Aici se poate vedea daca bucatarul are sau nu o comanda de efectuat.

De asemena, se pot deschide aceste ferestere in orice ordine, si nu separat .

**Use case diagram:**



**3. Proiectare**

**3.1. Structuri de date folosite**

Structurile de date folosite pentru realizarea aplicatiei sunt urmatoarele:

* **List (Array List) –** folosit pentru listele de base products si composite products
* **Map<Order, Collection<MenuItem>**> ( HashMap) – folosita pentru memorarea comenziilor , fiecare comanda avand o lista de produse din care este formata.
* **Set –** utilizata pentru meniul restaurantului, pentru a fi stocate doare produse unice.

**3.2 Proiectare clase. Diagrama UML**

* **Design pattern-uri folosite**

Am folosit **Observer Design Pattern** pentru a rezolva problema notificarii bucatarului atunci cand trebuie sa gateasca ceva. Acest pattern se bazeaza pe o dependenta one-to-may intre obiecte, insa fara ca acestea sa fie strans legate. Astfel vom avea doua obiecte: Subject si Observer, iar atunci cand starea subiectului (in cazul nostru acesta este restaurantul ) se schimba, toate obiectele observer (bucatarul) vor fi notificate si se vor actualiza automat.

Un alt design pattern utilizat este **Composite Pattern,** care va trata un grup de obiecte la fel ca o singura instanta a aceluiasi tip de obiect. In aplicatia noastra putem observa ca avem doua tipuri de produse: baseProduct, compositeProduct, insa acestea sunt tratate ca si un MenuItem . Intentia acestui model de proiectare este de a „compune” obiectele in structuri de arbori pentru a reprezenta ierarhii partiale ( „recursive composition”). Astfel un composite product poate contine atat produse base, cat si composite ( care la randul lui poate contine cele 2 tipuri de produse, si tot asa). De aceea a aparut nevoia de a le structura dupa acest design pattern.

* **Pachete folosite**

Structurarea pe pachete a fost realizata conform diagramei de clasa din specificatiile

temei:

* **Data** : contine FileWriter si RestaurantSerialization, clase care implementeaza metode pentru crearea unui fisier .txt pentru nota de plata a unei comenzi, precum serializarea si deserializarea starii aplicatiei.
* **Bussines**: contine structurile de date necesare acestei aplicatii. In acest pachet se afla :

-Interfata IrestaurantPrcedding, care este implementata de clasa Restaurant, si contine functionalitatile unui administrator si chelner.

- Clasele MenuItem ce este o clasa abstracta cu subclasele : BaseProduct si CompositeProduct

**- Presentation:** include interfetele grafice pentru administrator, chef si waiter. Fiecare clasa principala a acestor useri contine alte clase, pentru fiecare ferestra noua ce se deschide.

**- PT2020.Assignment\_4 :** in care se afla metoda main a aplicatiei.

**Diagrama UML:**

A close up of a map

Description automatically generated

A screenshot of a cell phone

Description automatically generated

**4.Implementarea celor mai importante clase**

**4.1. MenuItem**

Aceasta clasa contine un nume, si un pret, variabile ce vor fi mostenite de BaseProduct si CompositeProduct. Fiind o clasa abstracta, va avea metoda abstracta: computePrice();

**4.2 BaseProduct si CompositeProduct**

Clasa BaseProduct extinde MenuItem si reprezinta „frunza” in colectia reprezentata de menu. CompositeProduct mosteneste numele si pretul de la MenuItem, si rescrie metoda computePrice, insumand pretul tuturor elementelor din care e alcatuit, fie obiecte de tip base, fie composite

**4.3 Restaurant**

Va implementa interfata IRestaurantProcessing si contine un Set (implementat folosind HashSet) menu ce va contine elemente de tipul MenuItem, precum si un Map itemsPerOrder ( folosind HashMap) care contine comenzile ce sunt asociate cu listele de produse din care sunt compuse.

Principalele metode implementate in aceasta clasa sunt:

* **public BaseProduct createMenuItem(String name, float price) {}:** Metoda creaza si adauga un BaseProduct in meniul restaurantului.
* **public void createCompositeMenuItem(String name, List<MenuItem> composite) –**

Metoda creaza si adauga in menu un Composite Product. Se verifica daca produsele pe care le contine se gasesc sau nu in menu.

* **public void removeMenuItem(MenuItem item) {} –** Metoda sterge un menu item din menu. De asemenea, se verifica daca este continut in alte produse prin apelarea a doua metode : containsCompositeProduct ( in cazul in care obiectul de sters este composite) si containsBaseProduct ( daca obiectul de sters este base ) . Cautarea se face recursiv
* **public void editMenuItem(MenuItem toEdit, float price) {} –** Metoda modifica un base product , prin setarea altui pret. Se verfica daca alte produse composite il contin, iar atunci se actualizeaza si pretul acestora.
* **public void editMenuItem(MenuItem toEdit, String name, List<MenuItem> list) –**

Metoda modifica un composite product , prin schimbarea numelui si a listei din care este compus.

* **public Order createOrder(int OrderId, Date date, int Table, List<MenuItem> menuItems)** - Metoda creaza o comanda si o adauga in itemsPerOrder. De asemenea in aceasta metoda se notifica si bucatarul prin metoda **notiifyOberver()**
* **computePrice(Order o){} -** se calculeaza pretul total al unei comenzi
* **generateBill(Order o) –** se apeleaza functia de generare a unui fisier.txt in care sunt scrise datele comenzii date ca parametru

**4.4 AdminGUI**

-In aceasta clasa se creaza o interfata grafica in care sunt butoanele corespunzatoare operatiilor ce pot fi efectuate de catre un administrator. La apasarea fiecarui buton o instanta a clasei corespunzatoare este creata si se va deschide o fereastra. Clasele sunt:

* AdminCreateMenu care contine metodele necesare crearii unei interfete dar si efectuarii operatiilor de adaugare a produselor in menu.
* AdminDeleteMenu – creaza interfata de stergere a unui element , si implementeaza operatiile necesare
* AdminEditMenu – creeaza interfata de editare a unui produs, continand metodele efectuate la apasarea diverselor butoane.
* AdminViewMenu – creeaza o interafata ce contine 2 tabele: unul cu produsele de baza din meniu, celalalt cu produsele compuse

Informatiile se vor lua din JtextField-urile corespunzatoare si se vor apela metodele din Restaurant.Acest lucru este valabil pentru toate clasele de mai sus.

**4.5 WaiterGUI**

- In aceasta clasa interfata grafica pentru chelner este creata. Se implementeaza metodele de adaugare a comenzii , de adaugare a unui item intr-o lista ce va fi daca ca parametru la crearea comenzii, de vizualizare a comenzilor, cat si cea de eliberare a notei de plata

**4.6 ChefGUI**

**-** Aici se construieste interfata grafica pentru bucatar, care implementeaza interfata Observer, si suprascrie metoda update(Order o). In acesta vom seta un mesaj corespunzator comenzii daca ca parametru;

**5. Utilizarea proiectului si rezultate**

Programul se utilizeaza in modul urmator:

Trebuie utilizat fisierul .jar care va fi folosit cu urmatoarea comanda in command prompt, dupa ce ne asiguram ca suntem in directorul in care este si fisierul jar :

java -jar PT2020\_302210\_Ionescu\_Raluca\_Assignment\_4.jar restaurant.ser

Dupa rulare interfata grafica ar trebui sa apara, si se va putea alege tipul de user pentru a efectua operatiile dorite.

**6. Concluzii**

In urma realizarii acestui proiect am reusit sa am o imagine mai clara asupra design pattern-ului Observer si Composite. De asemenea am reusit sa inteleg mai bine serializarea si importanta ei .

Am reusit sa implementez o interfata grafica mai complexa decat ceea ce am experimentat pana in momentul de fata.

In ceea ce priveste o posibila dezvoltare ulterioara, o imbunatatire ar fi introducerea unor operatii noi pentru bucatar, cum ar fi vizualizarea tuturor comenzilor efectuate si pe cele pe lista de asteptare.

**7. Bibliografie**

**-** [**https://sourcemaking.com/design\_patterns/composite**/](https://sourcemaking.com/design_patterns/composite/)

- <https://stackoverflow.com/questions/2677564/how-to-create-custom-javadoc-tags/>

- <https://blog.frankel.ch/programming-by-contract-jvm/>

- <http://javarevisited.blogspot.ro/2011/02/how-hashmap-works-in-java.html/>