**DOCUMENTATIE TEMA 4**

**RESTAURANT MANAGEMENT SYSTEM**

**Tudor Colceriu**

**Grupa 30227**

**Profesor Laborator: Dan Mitrea**

# Cerinte Functionale

# Propuneti si implementati un sistem pentru gestionarea si managementul unui restaurant. Sistemul implementat trebuie sa aiba 3 tipuri diferite de utilizatori: administrator, chelner (waiter) si bucatar (chef). Proiectul realizat trebuie sa respecte anumite conditii de implementare (moduri de aranjare a claselor + diagrama UML).

# Obiective

## Obiectiv Principal:

Asa cum am precizat anterior, principalul scop al acestui proiect este de a realiza un system de management al restaurantului. Pentru acest lucru, cel mai usor mod de exemplificare a functionalitatilor prezentate este de a implementa o interfata grafica. Cele 3 tipuri de utilizatori (admin, waiter, chef) prezinta fiecare diferite atribute si functionalitati. Administratorul poate sa aduge, stearga si sa modifice elemente din meniul restaurantului. Chelnerul poate sa creeze o noua comanda (Order) , sa adauge elemente la o comanda curenta (acest mod va fi detaliat in pct.4 si 5, la implementarea si proiectarea aplicatiei) si sa genereze o chitanta pentru fiecare masa. Bucatarul este folosit nu pentru a genera ori a modifica elemente active din proiect, ci pentru a fi notificat de fiecare data cand trebuie sa “gateasca” (i.e. cand un chelner adauga o noua comanda).

## Obiective Secundare:

Implementarea claselor si metodelor dupa modele pre-specificate si bine definite, pentru o complexitate mai mare a proeictului, insa cu un plus din punct de vedere al flexibilitatii si eficientei sale (conform cerintei prezentate, dar si cu diferite adaugari si completari necesare pentru o buna functionare a aplicatiei prezentate).

# Analiza problemei

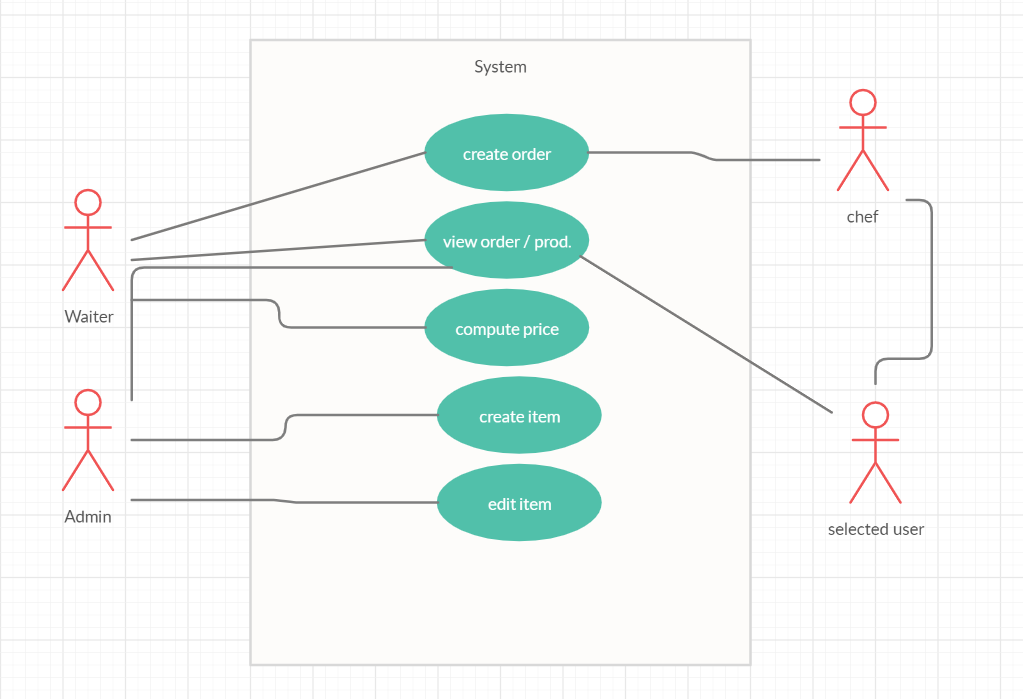
## Use case-uri si scenarii:

Utilizarea programului prezentat consta strict in interactiunea dintre utilizator si interfata grafica (GUI). Pentru acest lucru, proiectul trebuie rulat din terminal cu o comanda specifica (se ruleaza fisierul executabil .jar):

java **-jar** PT2020\_30227\_Colceriu\_Tudor\_Assignment\_4.jar <nume\_fisier.ser>

In momentul introducerii acestei comenzi, programul se poate astepta si la parametrii. Ca si parametru se poate transmite numele fisierului .ser, ce va contine obiectul initial de tip Restaurant (detaliat la pct.5), fapt ce va preincarca meniul cu anumite date. Daca nu se va introduce argumentul, un fisier .ser va fi incarcat (el trebuie sa fie deja existent!! – programul cauta fisierul specific in folderul curent de lucru : „restaurant.ser”).

Dupa rularea comenzii, interfata grafica va fi vizibila pentru utilizator, complet functionala. Fiind simpla si concisa, folosirea ei este intuitiva pentru utilizatori. Interfata grafica va fi detaliata si explicata in punctele ce urmeaza (la sectiune 5, cea de implementare a proiectului).



# Proiectare (etape de proiectare):

## Structuri de date:

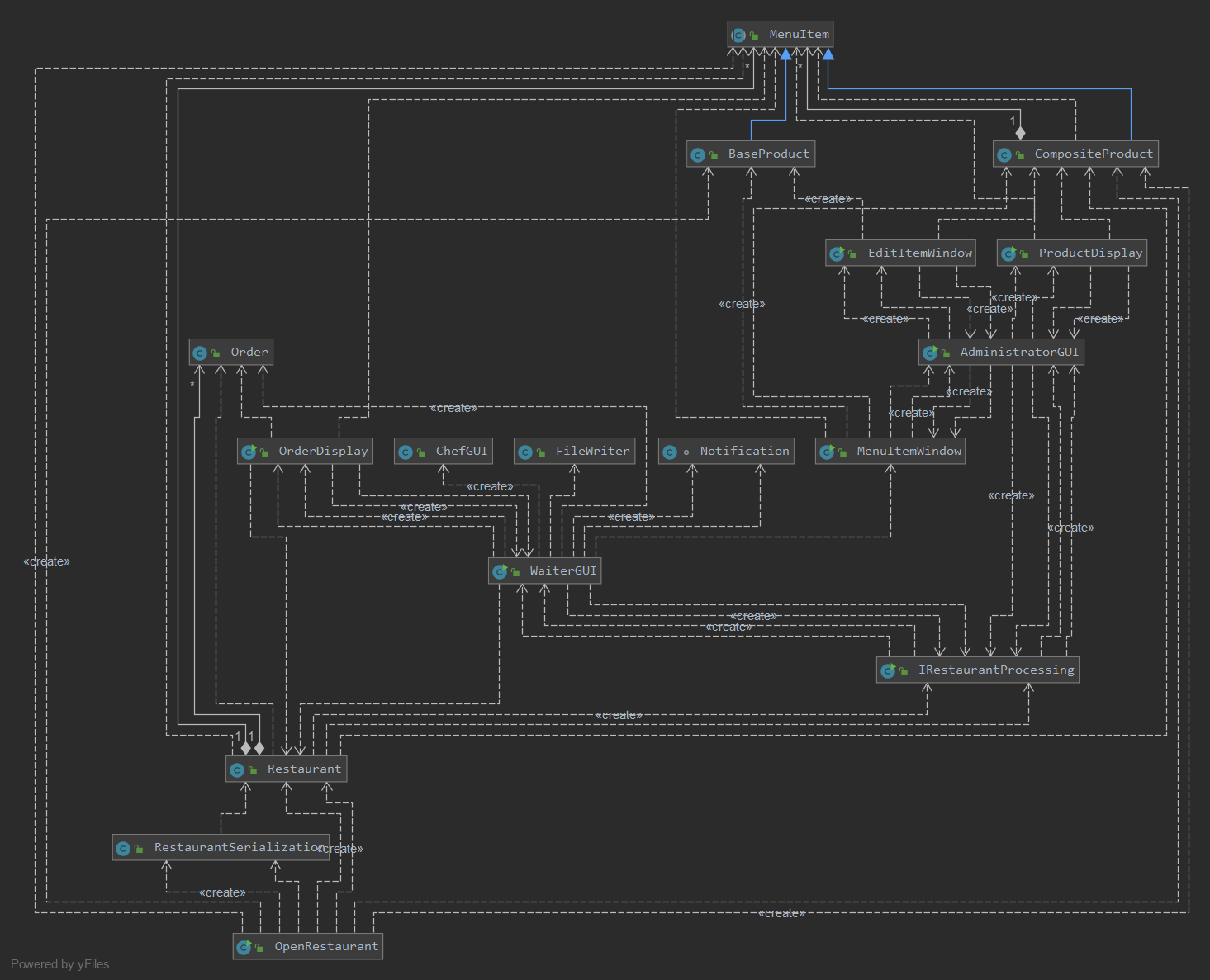
Pentru o buna functionare a proiectului au fost folosite structuri de tip colectii (liste si tabele de dispersie), fapt ce ajuta atat la stocarea datelor, cat si la transmiterea acestora intre pachetele si clasele utilizate.

Pentru o buna intelegere a sistemului, ne vom uita la cum este implementat meniul unui restaurant. De regula, acesta contine toate articolele pe care restaurantul le pune la dispozitia clientilor pentru a fi cumparate, impreuna cu pretul acestora. MenuItem reprezinta operatiile necesare implementarii acestor functionalitati, iar BaseProduct reprezinta un produs din meniu. Utilizand o colectie (ArrayList), CompositeProduct reprezinta meniul propriu-zis pe care restaurantul il are, in care se pot sterge, adauga, sau chiar modifica obiectele (fiecare obiect are propriile caracteristici, care pot sau nu sa fie schimbate).

Pentru a realiza cat mai bine „functionarea” fluenta si eficienta a restaurantului, in clasa Restaurant se utilizeaza un tabel de dispersie (HashMap). In aceasta structura se retin obiecte de tip Order (comenzile din restaurant), cat si articolele comandate de clienti din meniu, pentru fiecare comanda (ArrayList<MenuItem>). Functionarea in detaliu (modul de dispersie a obiectelor si accesarea lor) este prezentat in punctul 5, la partea de implementare a proiectului.

## Diagrama de clase (UML):

In cerintele functionale ale proietului, se cere ca diagrama de clase sa respecte o anumita structura, ea fiind impartita pe etaje (layer-uri), in functie de functionaltatile fiecarui etaj (in mare, un etaj are asociat un pachet, cu acelasi nume). Asadar, s-a incercat respectarea cat mai in detaliu a structurii cerute.



# Implementare:

Pentru acest proiect a fost realizata o impartire in pachete, fiecare fiind vital pentru buna executie a programului.

1. *Pachetul DataLayer:*

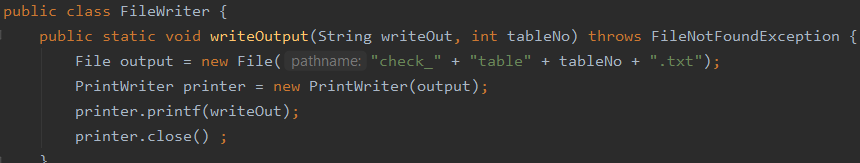
Contine clasele ce se vor ocupa de interactiunea programului, respectiv al utilizatorului (prin comenzi si interfata grafica) cu fisiere. Contine 2 clase, RestaurantSerialization si FileWriter.

**Clasa FileWriter** este cea responsabila de generarea chitantei pentru o masa. In momentul in care chelnerul apasa pe butonul dedicat pentru aceasta operatie, numarul mesei si mesajul sunt trimise catre functia writeOutput, ce se ocupa de generarea fisierului. In urma acestui lucru, un fisier text va fi creat in folderul curent de lucru, cu numele check\_table\_<nr\_masa>.txt, in functie de masa pentru care s-a apelat generarea chitantei. Mesajul care se afiseaza contine informatii despre: tipurile de mancaruri ce au fost comandate la masa respectiva, cat si pretul total al notei de plata. Mesajul este de forma:

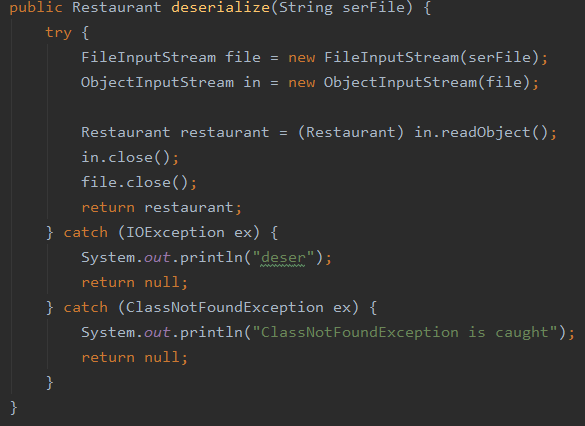
Table: <tableno> containing: <ordered items list>

TOTAL: <total price>RON

Billing date <date selected by the waiter for bill>

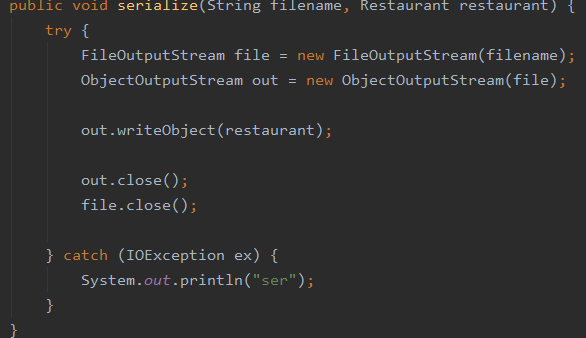


**Clasa RestaurantSerialization** contine metoda de stocare a informatiilor pentru un restaurant. Daca in comanda din consola exista un argument (argumentul dat este numele fisierului serializat), atunci se va apela metoda pentru deserializare din aceasta clasa.



Aceasta metoda primeste ca si argument numele fisierului (cel dat in argumentul initial, in linia de comanda). Apoi, functia realizeaza deschiderea fisierului (aflat in folderul de lucru , sau transmis printr-o cale explicita) si creaza un obiect, instanta a clasei Restaurant (explicata in pachetul de BusinessLayer – 5.2), care va contine toate informatiile din obiectul serializat anterior.

Pe acelasi principiu, daca in linia de comanda nu se specifica un argument, programul va instantia un obiect de tipul Restaurant, pe care il va initializa si serializa. Valorile initiale pentru restaurant sunt gasite in clasa OpenRestaurant (5.4). Fisierul creat in urma acestei operatii se va numi implicit „restaurant.ser”.

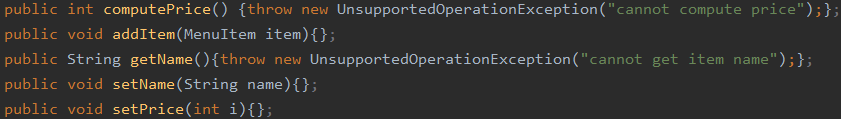


Asadar, daca se doreste ca restaurantul sa aiba valorile implicite, se poate executa serializare, apoi deserializare si restaurantul are valorile functiei din ExecuteTasks.

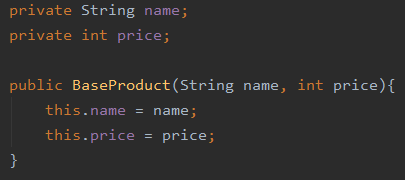
1. *Pachetul BusinessLayer:*

Conform cerintei si specificatilor tehnice ale proiectului (diagrama UML prezentata), acest pachet ce contine clasele referitoare la tipurile de obiecte din meniu, a fost implementata utilizant metoda de design Composite Design Pattern.

**Clasa MenuItem** este utilizata si interpretata aici din mai multe puncte de vedere. In primul rand, constituie modul in care toate obiectele dintr-un meniu sunt create, ea fiind clasa parinte pentru orice obiect din meniu (relatia de mostenire din cerinta functionala – 1 este acoperita). Mai mult de atat, aceasta clasa contine metodele principale si importante pentru fiecare articol din meniu. Metoda specificata in cerinta (minima) este cea de calculare a pretului unui articol: computePrice(). Aceasta este implementata in restul claselor din pachet ce mostenesc MenuItem, prin suprascriere (override).



**Clasa BaseProduct** reprezinta in esenta baza, asa cum ii spune de altfel si numele, oricarui produs existent in meniul restaurantului. Asadar, un produs din restaurant este de tipul BaseProduct si are urmatoarele atribute: un nume si un pret (asa cum este si in situatia reala la un meniu de restaurant). Acestia sunt de altfel si parametrii ce trebuie specificati cand dorim sa instantiem un obiect pentru aceasta clasa. Mai mult de atat, clasa BaseProduct mosteneste clasa MenuItem, ea implementand metodele necesare din aceasta.

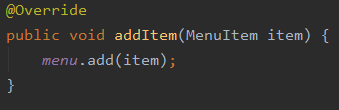


**Clasa CompositeProduct** reprezinta in mare intregul meniu, fizic, pe care il are restaurantul. Folosind o structura de tipul ArrayList, intr-un CompositeProduct se retine o colectie de obiecte MenuItem (clasa pe care de altfel o mosteneste) :

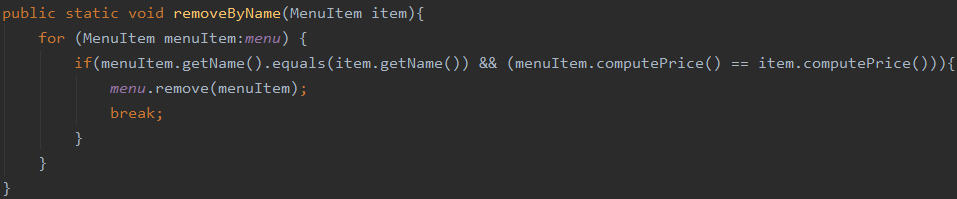


Acesta este meniul in sine, pentru care sunt implementate diferite operatii:

* Adaugarea unui produs in meniu

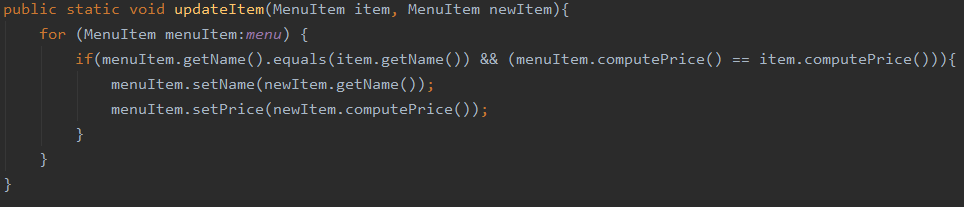


* Stergerea unui produs (trimis ca argument pentru functie) din meniu



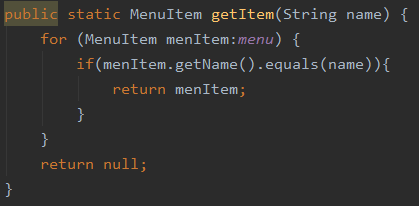
Aceasta functie itereaza prin toate obiectele din meniu, pana gaseste acel obiect care are nu doar numele, dar si pretul, la aceeasi valoare cu cea a obiectului transmis ca parametru. Acest mod de implementare a fost ales pentru a evita situatia in care un articol al meniului, sa zicem „salata de cruditati” ar putea sa apara cu pretul de xRON si yRON din diverse motive (reduceri de pret pe moment / introducere eronata din partea administratorului).

* Modificarea unui produs din meniu



Aceasta operatie este realizata in urma dreptului pe care il are un administrator, de a edita anumite produse, aflate in deja in meniu. Asemanator cu functia de mai sus, acesta itereaza pana la gasirea produsului potrivit (**item**), iar apoi il modifica cu valorile transmise prin parametrul **newItem**.

* Returnarea unui obiect dintr-un string

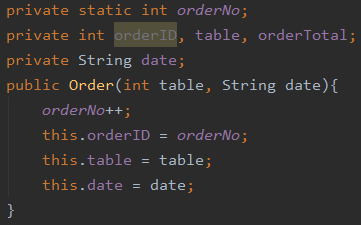


Metoda cauta in meniu articolul (prima aparitie) ce are numele **name** (string trimis ca si parametru).

**Clasa Order** realizeaza implementarea pentru o comanda din restaurant. Fiecare comanda in parte este executata doar de ospatari (waiter) si retine cateva detalii importante ce sunt folosite pe tot parcursul implementarii ulterioare: masa pentru care a fost facuta comanda si pretul total al comenzii respective. Pretul din aceasta clasa se calculeaza pentru **fiecare comanda in parte**, nu pentru masa totala!

Numarul comenzii (id-ul) este auto – incrementat de fiecare data cand o comanda este efectuata, deoarece un chelner nu are nici un motiv pentru a schimba sau a edita ordinul unei comenzi.

In plus, se retine si data comenzii (pentru fiecare comanda in parte !!), ce este introdusa de chelner. Fiecare chelner trebuie sa introduca ca si data a comenzii, data zilei curente.



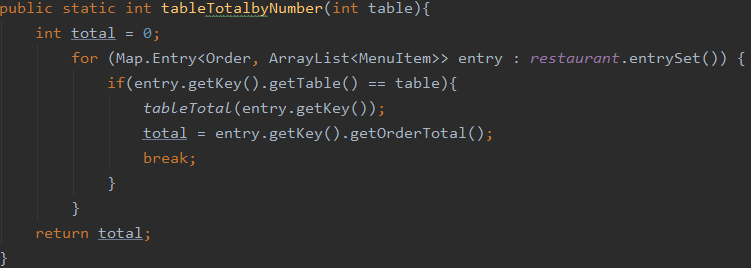
Cel mai important aspect al acestui proiect este reprezentat de catre functia de dispersie pentru comenzi. Aceasta este implementata in clasa Restaurant si realizeaza distributia comenzilor in functie de criteriul selectat in functie. Pentru acest proiect, criteriul dupa care sunt impartite comenzile este reprezentat de catre numarul mesei pentru care se decide sa se execute comanda. Pentru a realiza acest lucru, metoda hashCode( ) este suprascrisa.

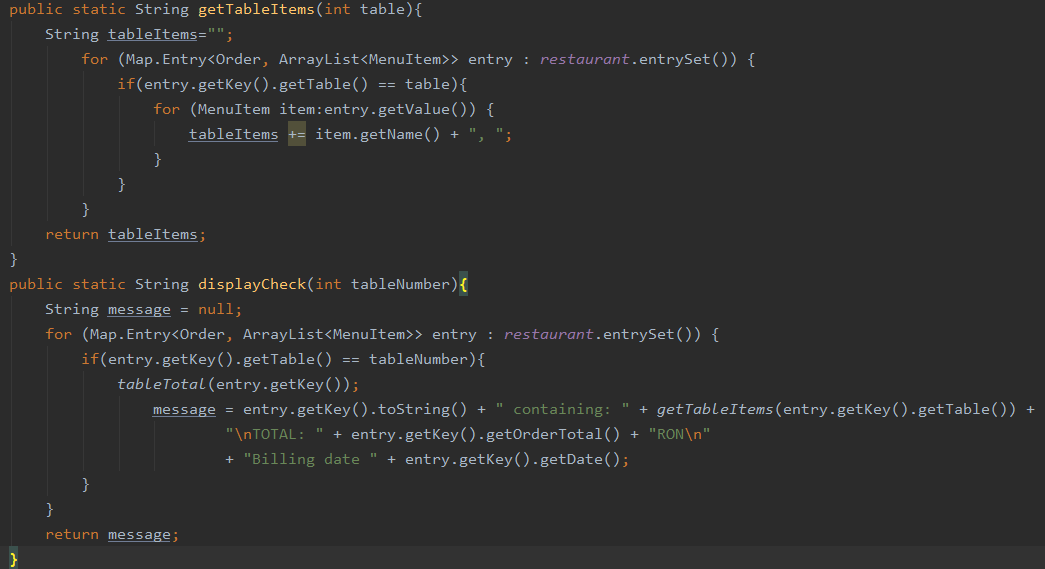


**Clasa Restaurant** realizeaza, in final, legatura dintre toate metodele si clasele implementate in acest pachet. Asa cum am precizat anterior, pentru a retine fiecare comanda (Order) la ce masa a fost transmisa, cat si ce contine (MenuItem), a fost implementat un table de dispersie, ce se foloseste de cheie ca fiind un obiect Order, cu criteriul de dispersie numarul mesei.



Am ales sa il fac static, deoarece nu se doreste implementarea mai multor restaurante in acelasi timp, asadar indiferent unde ne aflam in proiect (in ce etapa), restaurantul referinta este acelasi.

Pentru un restaurant se pot: adauga articole (atat dupa nume *addItembyName*, cati si printr-un obiect transmis *addItem*), vizualiza articole (*getTableItems –* returneaza un String ce contine toate articolele comandate la masa cu numarul transmis ca parametru), calcula totaul unei mese (dupa numarul mesei sau dupa un obiect Order), si in final se pot afisa elementele necesare pentru masa transmisa ca argument (*displayCheck* - folosit pentru creare de chitanta).

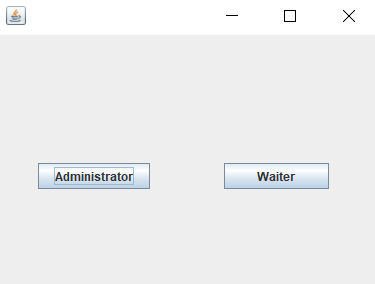


1. *Pachetul PresentationLayer:*

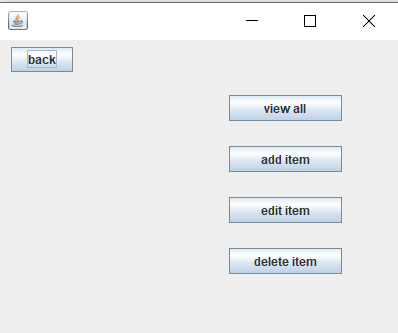
Pachetul PresentationLayer contine toate calsele care se ocupa de interfata grafica. Primul lucru ce este vizibil cand se ruleaza proiectul este fereastra clasei IRestaurantProcessing, ce contine modul de interactiune cu proiectul: administrator sau waiter, ce va limita optiunile ulterioare, in functie de alegere.

Fiecare fereastra contine un buton „back” ce va duce utilizatorul la fereastra **precedenta**.

Fereastra prezinta 2 butone, pentru selectarea tipului de user.



Fiecare buton este etichetat cu tipul de user pe care il va accesa.

**Fereastra de administrator: AdministratorGUI**

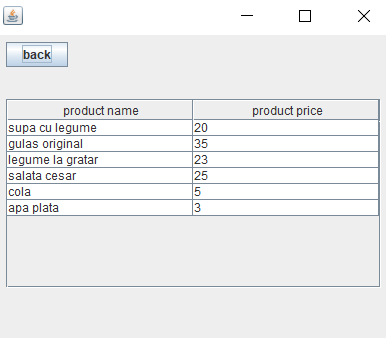
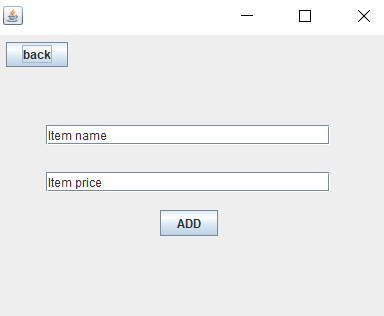
Aceasta fereastra contine toate operatiile pe care un administrator le poate executa. Daca se apasa „view all”, un tabel cu toate elementele din meniu va fi creat. La apasarea butoanelor add / delete utilizatorul va fi dus intr-o noua fereastra unde trebuie introduse detaliile pentru noul element, respectiv pentru elementul ce se va dori sa fie sters.

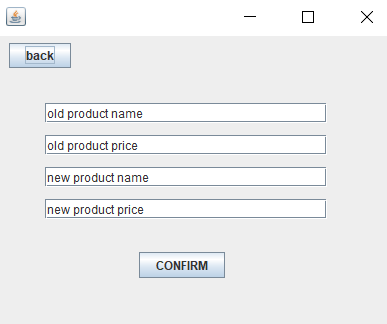
Fereastra pentru editare contine informatiile curente (articolul ce se doreste sa fie editat) si cele noi, introduse de utilizator, ce vor actualiza apoi prin apasarea noului buton articolul introdus.

Fereastra de editare

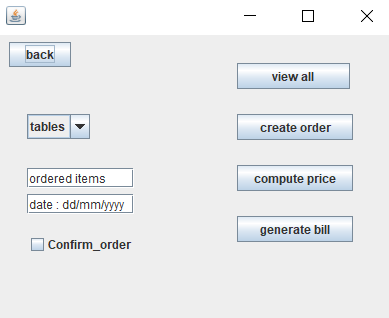
Fereastra de adaugare (ca si stergere)

Fereastra de vizualizare





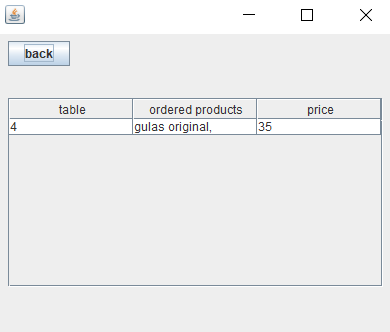
**Fereastra de chelner: WaiterGUI**

Analog cu fereastra de la administrator, aici se afla toate operatiile pe care le poate executa un chelner.

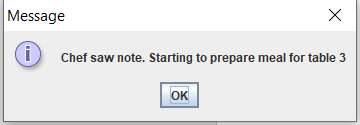
!! pentru a executa o operatie pe comanda (generare chitanta / adaugare comanda / afisare pret masa), din meniul pentru mese (implicit sunt 10 mese in restaurant), se va selecta masa.

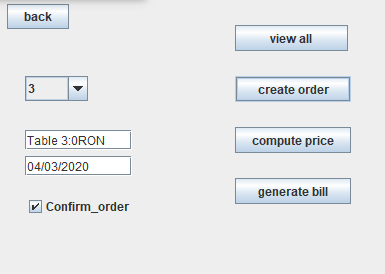
!! daca se doreste adaugarea unei noi comenzi, se va face in zonele text prin introducerea informatiilor necesare (asa cum scrie pe ele – JtextField) iar apoi casuta Confirm\_order **trebuie apasata**, altfel nu se executa comanda

Ca si la **AdministratorGUI**, view all arata pentru fiecare masa comenzile si mancarea comandata, cat si pretul mesei



„generate bill” genereaza chitanta pentru o masa (conform 5.1 – **FileWriter**)

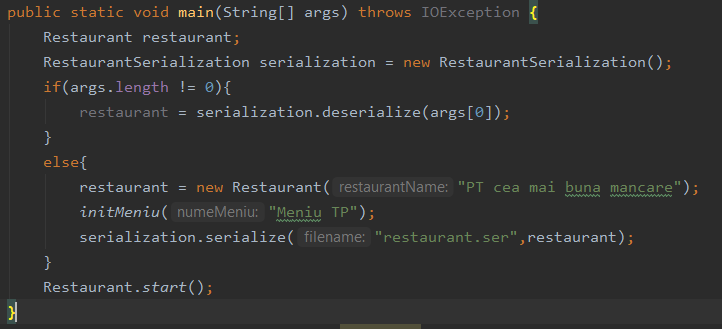
„create order” genereaza o noua comanda si trimite o notificare catre bucatar (utilizarea Observer pattern), fapt semnalat de un mesaj – **ChefGUI** + **Notify**

„compute price” actualizeaza prima zona de text cu pretul pentru masa selectata

!! Pentru ca o comanda sa fie executata cu un anumit set de articole din meniu, ele trebuie sa fie existente in meniul restaurantului.

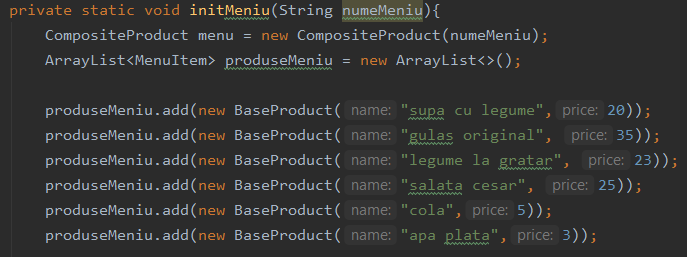
1. *Pachetul ExecuteTasks:*

Contine cea mai importanta metoda, cea de **OpenRestaurant**, care este metoda main pentru proiect. Aceasta se ocupa de transmiterea argumentului din linia de comanda, in functie de ce este scris, si de initializarea meniului in cazul in care se va dori acest lucru. Acest lucru este verificat in conditia metodei main:



Metoda start din clasa Restaurant (apelata de fiecare data dupa introducerea comenzii in terminal), va porni interfata grafica, setand vizibila prima pagina (**IRestaurantProcessing**), cea principala de altfel.

In aceasta clasa (**OpenRestaurant**) se pot observa si valorile initiale din meniul restaurantului, din metoda initMeniu(). Acestea sunt valorile pentru care (fara sa adaugam alte elemente in meniu) se pot crea comenzi.



# Concluzii si dezvoltare ulterioara:

Acest proiect a imbinat practic toate cunostintele dobandite in restul temelor (interfata grafica, organizare pe pachete etc.) si, in opinia mea, m-a ajutat foarte mult pentru o mai buna stapanire a proiectarii unei aplicatii Java. Factorul decisiv din acest lucru a fost introducerea „restrictiei” prin tipurile de organizare a proiectului (CompositeDesignPattern, Observable etc.), dar mai ales prin oferirea unei diagrame de clase ce trebuia ulterior respectata, un lucru pe care nu l-am facut pana acum.

# Bibliografie:

* Pdf – urile puse la dispozitie cu cerinta temei
* Tutorialspoint.com (serializare)
* Geeksforgeeks.com (serializare, pattern-uri de design)
* Stackoverflow.com pentru eventuale erori mici aparute pe parcursul implementarii