Tema 4

Restaurant management

system

Stefanovici Miruna Andreea

Grupa 30229

**Continut**

**1.***Obiectivul temei*

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

**3.***Proiectare (decizii de proiectare, diagrame UML, structuri de date, proiectare clase, interfete, relatii, packages, algoritmi, interfata utilizator)*

**4.***Implementare*

**5.***Rezultate*

**6***.Concluzii*

**7.***Bibliografie*

**-1- Obiectivul temei**

Obiectivul principal al proiectului este simularea unui “restaurant”, in cadrul caruia actioneaza un “chelner”, un “ bucatar” si un “administrator al restaurantului”. Fiecare dintre acestia poate realiza anumite actiuni, reprezentative pentru atributiile din viata reala.

Pentru ca scopul principal sa poata fi atins, trebuie urmarita o serie de pasi:

-analizarea potentialelor scenarii de folosire a aplicatiei de catre un utilizator, a interactiunii dintre user si program;

-alegerea modului de proiectare, a structurilor de date folosite in cadrul acesteia;

-alegerea modului de implementare a algoritmilor;

-descrierea claselor prin evidentierea rolului fiecarui camp si fiecarei metode;

-analizarea rezultatelor in urma implementarii, pe testarea unor exemple ce creeaza mai multe scenarii de abordare.

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

Scenariul de utilizarea se desfasoara pe mai multe etape:

- se presupune ca in meniu exista deja un numar de produse ( adaugate la prima rulare );

- fiecare fereastra corespunzatoare unuia dintre cei 3 reprezentati (bucatar, chelner, administrator) va fi afisata pe ecran;

- fiecare fereastra contine “actiunile” pe care le poate realiza fiecare reprezentant;

- la aceasta etapa, ramane la alegerea utilizatorului de a accesa unele dintre aceste actiuni, de exemplu adaugarea unu produs in meniu ( atributie a administratorului ) sau plasarea unei comenzi( atributie a chelnerului ) ( vor fi explicate in detaliu in sectiunea de “Implementare”);

- la o urmatoare rulare se observa ca produsele adaugate sau editate anterior, apar in meniu, iar cele ce se sterg anterior, nu mai apar ulterior;

- actiunile se pot repeta, simuland cu succes atributiile de baza ale angajatilor principali dintr-un restaurant.

**-3-Proiectare (decizii de proiectare, diagrame UML, structuri de date, proiectare clase, interfete, relatii, packages, algoritmi, interfata utilizator)**

In ceea ce priveste proiectarea aplicatiei, conceptele din spate ce se evidentiaza sunt “Modelul MVC”, “Serializarea” si folosirea colectiei HashMap.

**I**deea Modelului MVC este de a separa logica de implementare a interfetei grafice in:

* Model: algoritmii ce trebuie implementati si in urma carora se ajunge la obiectivul propus;
* View: “vedere”, vizualizare a afisajului grafic al interfetei ( buton, camp de text, label, panel, etc) ce poate interactiona cu modelul;
* Controller: mecanismul de a raspunde la cererile utilizatorului, interactionand cu vederea.

In spatele ideii de “Model” al sablonului MVC, sta logica si implementarea metodelor reprezentative pentru actiunile chelnerului, administratorului si bucatarului. Modelul, in cadrul acestui proiect, este clasa “Restaurant” ce va implementa interfata “IRestaurantProcessing”. Aceste metode sunt impartite ca atributii alte chelnerului si atributii ale administratorului, bucatarul neavand niciun fel de atributie directa ( acesta doar va fi “anuntat” cand este plasata o comanda ce contine un produs compus ). Astfel, chelnerul si administratorul vor dispune de cele 3 componente ale modelului MVC, iar bucatarul va dispune doar de “View” la care se adauga o altfel de metoda de setare a unui text intr-un JTextField .

**U**tilizarea colectiei HashMap are la baza o anumita logica. HashMap-ul este una dintre implementarile pentru interfata Map, obiect ce mapeaza chei la anumite valori, neputand exista intr-o astfel de structura chei duplicate. Fiecare cheie va fi mapata la o singura valoare. Acest obiect poate vi vazut ca o modelare a conceptului de functie: primeste o entitate ca parametru(cheia) si intoarce o alta entitate si anume valoarea. Metodele caracteristice sunt getKey ( returneaza cheia), getValue( returneaza valoarea) si setValue( permite stabilirea unei valori associate cu o anumita cheie). In cadrul proiectului, acesta colectie HashMap va fi utilizata pentru a stoca toate comenzile plasate de catre “chelner”. Cheia va fi generata pe baza clasei Order si va avea asociat un ArrayList<MenuItems> ca valoare.

***S****erializarea* consta in transformarea unui anumit obiect intr-o secventa de mai multi octeti, ca ulterior din aceasta sa poata fi reconstruit obiectului initial.

Procesul de citire a unui obiect ce a fost serializat, pentru a-l reconstiui poarta numele de *deserializare*. Referintele care se ocupa de reconstruirea starii unui anumit obiect formeaza o retea de obiecte : ObjectInputStream, ObjectOutputStream, DataInputStream, DataOutputStream.

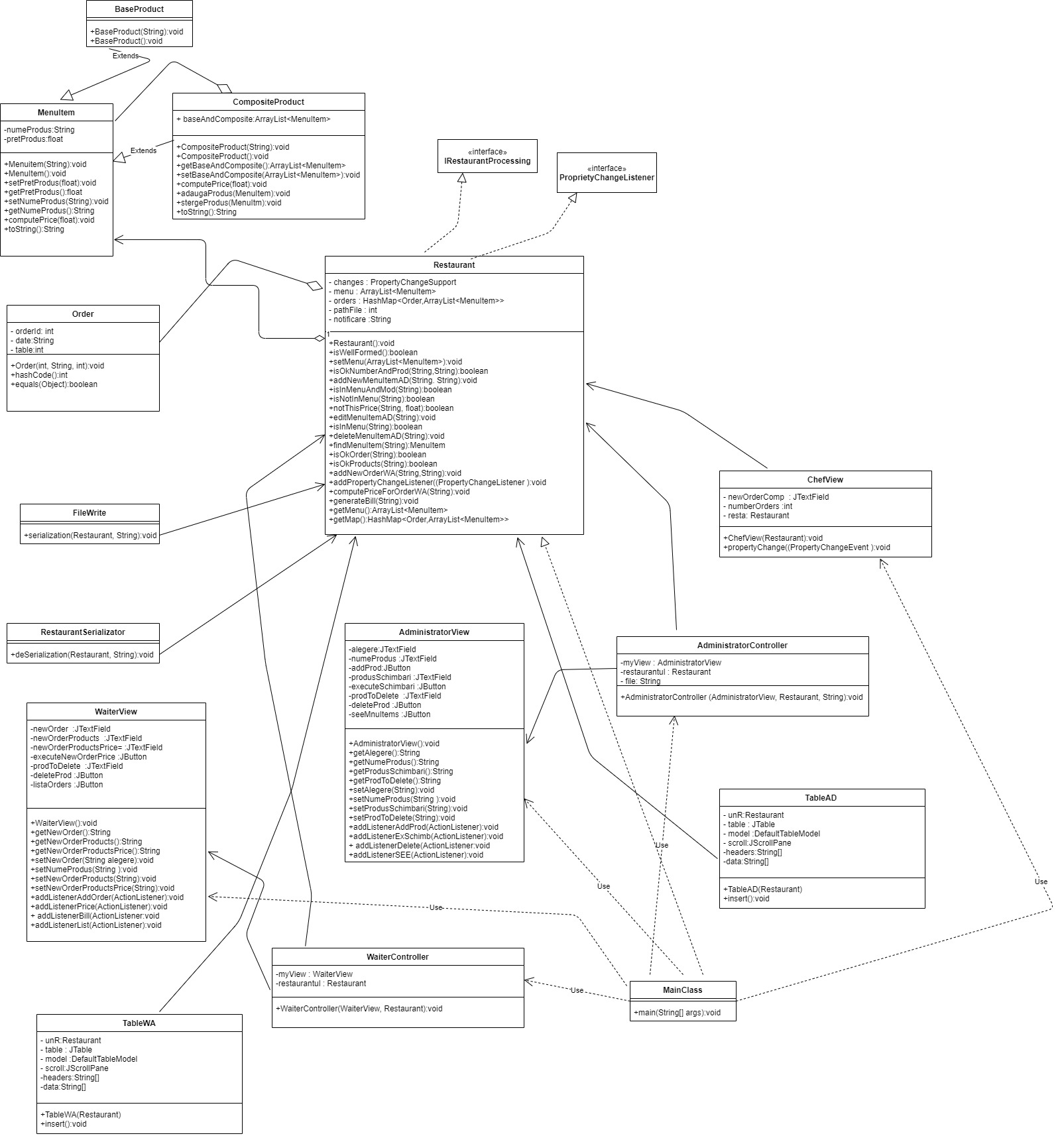
Este un mecanism simplist de a salva si reconstituit datele. In cadrul proiectului, serializarea se foloseste pentru salvarea “produselor” din meniul restaurantului. Astfel, fiecare modificare adusa meniului va fi disponibila si la urmatoarea rulare, dupa fiecare metoda de are ca obiectiv o anumita modificare, fiind realizata serializarea meniului ( obiectului ArrayList<MenuItem>).

Clasele sunt organizate in pachete ( cu exceptia clasei MainClass) in functie de utilitatea comuna pe care o aduc in ansamblu proiectului:

* pachetul BusinessLayer : *MenuItem, BaseProduct, CompositeProduct, Order, IRestaurantProcessing, Restaurant*;
* pachetul DataLayer : *FileWriter, RestaurantSerializator*;
* pachetul PresentationLayer : *ChefView, AdministratorView,*
* *AdministratorController, TableAD, WaiterView, WaiterController, TableWA*;
* clasa *MainClass*.

Fiecare clasa va fi explicate in detaliu in sectiunea “Implementare”.

Diagrama UML:



**-4-Implementare**

Pachetul BusinessLayer

*Clasa MenuItem*

Are ca variabile de clasa String-ul numeProdus, ce prerezinta numele produsului si float-ul pretProdus ce reprezinta pretul produsului. Pe langa obisnuitele settere,gettere ( ce returneaza valoarea variabilelor de clasa sau le seteaza ca avand o anumita valorea, si constructori ( unul ce seteaza numele unui produs atunci cand esti creat si unul gol), clasa contine si functia toString() ce are rolul de a afisa un mesaj clar, ce enumera valorile variabilelor de clasa. Aceasta clasa implementeaza ineterfata Serializable deoarece in cadrul serializarii meniului unui restaurant, se foloseste un ArrayList de elemente MenuItems, serializare ce nu ar fi posibila fara implementarea acestei interfete.

*Clasa BaseProduct*

Aceasta clasa va mosteni clasa MenuItem, reprezentand un tip de produs de meniu si anume, unul simplu. Cand spun simplu, ma refer la faptul ca acesta va dispune doar de un nume si un pret setat initial. Va putea face parte din compozitia unui produs compus. Clasa dispune doar de constuctori : unul gol ( atunci cand se doreste crearea unui BaseProduct fara valori setate) si unul care, mostenind clasa parinte MenuItem, va seta numele. Prin mostenire, va dispune de toate metodele si variabilele de clasa din MenuItem.

*Clasa CompositeProduct*

Aceasta clasa va mosteni clasa MenuItem, reprezentand un tip de produs de meniu si anume, unul compus. Cand spun compus, ma refer ca acesta pe langa un nume propriu, va dispune de o lista de produse de tip MenuItem. Acest lucru inseamna ca in lista sa se pot afla atat obiecte de tip BaseProduct cat si obiecte de tip CompositeProduct, respectand stilul Composite Design Pattern. Acest model descrie un grup de mai multe obiecte ce sunt tratate ca o singura instanta a aceluiasi tip de obiect. Pe langa setter-ul si getter-ul listei de produse, se regasesc functia de adaugare a unui produs in propria lista, adaugaProdus(MenuItem Produs), de stergere a unui produs, stergeProdus(MenuItem Produs). Functia de computePrice() ce are ca semnificatie calcularea pretului unui produs MenuItem, este suprascrisa aici, deoarece pentru un produs compus, pretul total nu va mai fi unul setat, ci reprezentat de suma tuturor preturilor produselor ce alcatuiesc lista lui. Functia toString() este de asemenea suprascrisa, deoarece apare in plus ArrayList-ul de MenuItem, fapt ce presupune o afisare mai detaliata a continutului. Prin mostenire, va dispune de toate metodele si variabilele de clasa din MenuItem.

*Clasa Order*

Aceasta clasa este reprezentata a cheii pe baza careia se vor adauga comenzi in HashMap. Fiecare Order va avea asociata o lista de produse cerute de catre unu client. Are ca variabile de clasa int-ul orderId, ce reprezinta ID-ul unei comenzi (am ales sa le introduce valorile in ordinea in care sunt facute : de ex. , pentru primul order id-ul va fi 1, pentru al doilea, va fi 2, etc), String-ul date ce va simboliza data in care va fi facut Order-ul si un int table care se refera la masa de la care va fi preluata o comanda. Toate aceste date au scop simbolic ca semnificatie, dar pe baza valorilor acestora se va genera o cheie prin apelarea functiei hashCode(), ce are la baza un algoritm de hashing : Objects.hash (orderId, date, table). HashMap()-ul are la baza metoda de chaining ce rezolva problema coliziunilor ( momentul in care doua obiecte diferite au aceeasi valore returnata de hashCode() ). Astfel, cand doua obiecte se afla in coliziune, acestea vor fi puse intr-o lista inlantuita si astfel hashTable-ul va fi alcatuit dintr-un vector de liste inlantuite, obiectele cu aceeasi cheie fiind puse in lista inlantuita de pe indexul ( cheia ) comun ( bucket ). Metoda equals() trebuie suprascrisa, deoarece functia originala considera ca egalitatea consta in egalitatea identitatilor unor obiecte. Astfel, desi doua obiecte au aceleasi valori pentru variabilele de clasa, ele vor fi considerate ca fiind doua instante diferite. Functia va fi suprascrisa asa fel incat sa fie luata in considerare si valoarea campurilor obiectelor.

*Clasa IRestaurantProcessing*

Aceasta clasa este interfata implementata de clasa Restaurant. Aici sunt stabilite metodele representative celor doi reprezentanti ai restaurantului ce au anumite atributii : chelnerul si administratorul. Administratorul poate adauga un nou produs in meniu (addNewMenuItemAD(String alegere, String nameProduct)), poate sterge un produs din meni(deleteMenuItemAD(String numeProdCautat)) sau il poate edita (editMenuItemAD(String produsDeEditat) - prin editare am stabilit ca se intelege schimbarea numelui sau/si a pretului). Chelnerul poate plasa o anumita comanda(addNewOrderWA(String Order, String Products)), poate calcula pretul comenzii de la momentul plasarii acesteia(computePriceForOrderWA) si tot la momentul plasarii comenzii respective, poate genera o factura in format txt (generateBillWA(String Products). Fiecare functie va primi unul sau mai multe String-uri, asupra carora se va face parsarea pentru a identifica datele necesare crearii obiectelor cerute de logica metodei. Tot in aceasta interfata are loc definirea prin intermediul JavaDoc a preconditiilor si postconditiilor.Preconditiile au rolul de a stabili niste reguli de validare pentru datele de intrare, iar postconditiile, de a verifica daca fiecare metoda returneaza ceea ce ar trebui in mod corect sa returneze in functie de logica metodei.Preconditiile sunt testate inainte de inceperea efectiva a codului din metoda respective, iar postconditiile sunt testate dupa terminarea codului, pentru a avea rezultatul final al functiei, putand astfel fi comparat cu un rezultat asteptat. In interfata, aceste reguli sunt definite prin amplasarea unor tag-uri custom, @pre si @post inaintea fiecarei reguli stabilie (tag-uri definite in linia de comanda JavaDoc). Testarea respectari acestor reguli se realizeaza in clasa Restaurant prin assert.

*Clasa Restaurant*

In aceasta clasa se afla esenta proiectului. Ca variabile de clasa, contine meniul de tip ArrayList<MenuItem>, lista cu comenzi orders de tip HashMap<Order,ArrayList<MenuItem>> ( cum am explicat si la Clasa Order, obiectul Order reprezinta cheia la care vor fi stocate listele de produse din cadrul unei comenzi), int-ul pathFile ce va reprezenta numarul atasat la numele facturilor(se va incrementa mereu pentru a le diferenta la creare), String-ul notificare ce isi va schimba valoarea din “0” in “1” atunci cand comanda plasata contine un produs compus, anuntandu-l astfel pe bucatar si obiectul changes de tip PropertyChangeSupport ce poate gestiona o lista de ascultatori, transmitand evenimente ( PropertyChangeEvents) asupra lor. Constructorul instantiaza meniul si lista de comenzi.

Metoda **addNewMenuItemAD(String alegere, String nameProduct)** are rolul de a adauga produse in meniu. In functie de alegere, va fi adaugat un produs de baza(alegere egala cu “1”) sau un produs compus (alegere egala cu “2”). In functie de alegere, String-ul nameProduct va putea avea doua forme : pentru un produs de baza, acesta va fi alcatuit din numele produsul, un spatiu si pretul produsului, iar pentru un produs compus, acesta va fi alcatuit din numele produsului, un prim spatiu si in continuare, numele produselor ce alcatuiesc produsul compus( nume scrise cu cate un spatiu intre ele). In cazul produsul de baza, obiectul este creat si adaugat in meniu, dar in cazul celui compus, se verifica daca produsele componente se afla deja in meniu. E de ajuns ca unul dintre produsele din aceasta lista sa nu se gaseasca in meniu ca produsul compus sa nu fie creat. Ca preconditie se apeleaza cu assert metoda isOkNumberAndProd(alegere, nameProduct), asigurandu-se astfel ca alegere poate fi egal doar cu “1” sau “2”, iar in cadrul lui nameProduct, in cazul ca este produs de baza, numele produsul sa fie alcatuit doar din litere si pretul doar din cifre( se admite existenta caracterului ‘.’, pretul fiind de tip float), iar in cazul ca este produs compus, fiecare nume de produs din lista sa fie alcatuit din litere ( la fel si numele noului produs compus creat). Ca postconditie, se verifica daca size-ul meniului de dupa adaugare este egal cu size-ul meniului de dinainte de adaugare plus 1. Astfel ne asiguram ca un produs a fost adaugat.

Metoda **editMenuItemAD(String produsSchimbari)** are rolul de a edita anumite produse. Prin editare am stabilit ca se intelege modificarea numelul sau/si a pretului.La un produs de baza se poate modifica si pretul si numele, pe cand la un produs compus doar numele(deoarece pretul e alcatuit din suma preturilor produselor componente).Metoda primeste ca prametru un String care poate lua urmatoarele forme:

* numeProdusDeEditat numeNou 0 (atunci cand se doreste doar modificarea numelui);
* numeProdusDeEditat X pretNou (atunci cand se doreste doar modificarea pretului);
* numeProdusDeEditat numeNou pretNou (atunci cand se doreste editarea ambelor valori).

In functie de forma acestui String, variabila locala alegere poate fi ( conform ordinii de mai sus) 1, 2 sau 3. Se parcurge meniul, iar in functie de alegere se fac setarile respective cand se gaseste produsul de editat. Se mai parcurge meniul inca o data pentru a se asigura ca in urma modificarii pretului anumitor produse, sa se recalculeze pretul tuturor produselor conform noilor modificari. Ca preconditie se apeleaza cu assert metoda isInMenuAndMod(produsSchimbari), asigurandu-se faptul ca produsul de editat se afla in meniu si forma String-ului trimis ca parametru metodei, respecta una dintre cele 3 forme. Ca postconditie, se apeleaza cu assert isNotInMenu(preNume) || notThisPrice(preNume, prePrice), prima functie avand rolul de a verifica daca in urma modificarii numelui, produsul nu se mai afla in meniu cu numele vechi, editarea fiind astfel realizata cu success sau, in caz de modificare de pret, sa nu se mai gaseasca produsul respective cu pretul de dinainte de modificare. Pentru al treilea caz, in care ambele valori sunt schimbate, cele doua functii vor returna fiecare true in caz de success.

Metoda **deleteMenuItemAD(String numeProdCautat)** are rolul de a sterge un produs din meniu. Primeste ca parametru numele produsului de sters.Inainte de a-l sterge pe el, trebuie verificat daca in meniu exista si alte produse ce content elemental respective. Prin asociere, vor fi sterse si acestea. Pentru a face acest lucru, trebuie initial sa cautam produsul in meniu. De asta de ocupa functia findMenuItem(String numeProdCautat) ce returneaza produsul cu numele cautat. Se initiaza ulterior Iteratorul cauta care va parcurge meniul. Cand ajunge la un produs compus, se verifica daca in lista lui de produse se afla produsul cautat. Daca se afla, numele lui va fi adaugat la o lista de String-uri deSters. Ulterior lista de String-uri va fi parcursa, iar pentru fiecare element ce trebuie sters, se va reapela functia recursiv.Ca preconditie se apeleaza cu assert isInMenu(numeProdCautat), asigurandu-se ca in meniu se afla produsul ce trebuie sters, iar ca postconditie se verifica daca size-ul actual este mai mic decat size-ul de dinainte de stergere, asigurandu-se astfel ca cel putin un produs a fost sters.

Metoda **addNewOrderWA(String Order, String Products)** are rolul de a adauga in HasMap-ul orders o noua comanda. Primeste ca parametri un String Order ce va fi reprezentat de elementele necesare crearii unui Order(un id, o data si table) si un String Products ce va fi reprezentata prin insiruirea, cu spatii intre ele, a numelor de produse comandate. Ca un order sa fie plasat trebuie sa contina cel putin un element existent in meniu. In caz contrat, order-ul nu va fi adaugat in lista. Cand se face cautarea unui anumit produs in meniu, in caz ca acesta este gasit se verifica daca e instanta a lui CompositeProduct, iar daca este, bucatarul va trebui anuntat de acest lucru prin apelarea metodei firePropertyChange ce face un update la ascultatorii ce au fost inregistrati sa observe schimbari ale unei anumite proprietati.Variabila notificare se va face “1” cand este gasit un produs compus, se va anunta schimbarea, dupa care va reveni la valoarea initiala pentru urmatoarele verificari de produse. In caz ca se poate adauga in orders, se va adauga folosind metoda put cu parametrii order si orderedProducts.Ca preconditie, se apeleaza cu assert metodele isOkOrder(Order) && isOkProducts(Products) && !menu.isEmpty() care au rolul ( pe rand) : de a asigura ca pentru un order id-ul si table-ul sunt numere, de a asigura ca lista de produse din comanda e alcatuita din nume de produse ce contin doar litere si de a se asigura ca meniul nu este gol ( caci nu putem face o comanda, daca in meniu nu se afla niciun produs). Ca postconditie se verifica daca in caz ca s-a adaugat un order, size-ul a crescut cu 1 sau in caz ca nu s-a adaugat orderul, daca size-ul a ramas acelasi.

Metoda **computePriceForOrderWA(String Products)** are rolul de a calcula pretul pentru comanda curenta ( adica va avea acces la acelasi String la care are acces si functia de plasare a unei comenzi). Se parcurge meniul, se adauga produsele comandate intr-o lista(asta daca se gasesc in meniu, respectand aceeasi regula ca si metoda de addNewOrder), iar ulterior se parcurge lista respective si se aduna la variabila price, pretul fiecarui produs.Ca preconditie se verifica daca lista de produse e valida si daca meniul nu este gol. Ca postconditie se asigura ca pretul sa fie mai mare decat 0 ( asta in cazul in care order-ul are cel putin un produs existent in meniu) sau daca este 0 ( asta caz in care order-ul nu are niciun produs existent in meniu).

Metoda **generateBillWA(String Products)** face acelasi lucru ca si computePrice, cu exceptia faptului ca in caz ca pretul este diferit de 0 ( adica s-a adaugat comanda), se genereaza o factura : un fisier de tip txt care va avea numele alcatuit din String-ul “Bill” la care se va concatena numberBill( varianta de String a lui pathFile, ce se mareste ulterior pentru ca urmatoarea factura sa aiba nume diferit). Ca preconditie se verifica daca lista de produse e valida si daca meniul nu este gol.Ca postconditie se asigura ca pretul scris pe factura este acelasi ca cel generat de calcularea pretului in functia anterioara.

Pe langa getter-ele pentru meniu si orders, mai exista si functia de addPropertyChangeListener(PropertyChangeListener l) ce are ca rol adaugarea la obiectul changes un ascultator.

Metoda wellFormed() va verifica daca meniul este null sau nu si daca fiecarui order ii este asociat un ArrayList<MenuItems> nenul. Aceasta metoda va reprezenta invariantul clasei si va fi apelata in interiorul fiecarei metode la inceput si la final. Aceasta trebuie sa isi pastreze valorea true in orice moment, indifferent de modificari.

Pachetul PresentationLayer

*Clasele AdministratorView, WaiterView*

Aceste clase au rolul de a stabili pentru fiecare fereastra reprezentativa pentru administrator, respectivebucatar, vizualizarea afisajului grafic al interfetei ( buton, camp de text, label, panel, etc) ce poate interactiona cu modelul. Se declara campurile de tip JTextFile si de tip JTextButton si se amplaseaza in pagina intr-o ordine stabilita ( amplasare realizata in constructorul clasei). Ca metode, regasim gettere si settere pentru campurile de text si functiile de adaugare de ascultatori pentru butoane la realizarea unui anumit eveniment.

*Clasele AdministratorController, WaiterController*

Aceste clase au rolul de a stabili ce actiuni se vor petrece in urma apasarii butoanelor din View-urile corespunzatoare fiecareia, actiuni reprezentate de clasele cu metodele de tip actionPerformed(ActionEvent ev). Astfel, se declara String-uri in care vor fi salvate datele pe care le introducem de la tastatura, care ulterior vor fi trimise la functiile din restaurant.Fiecare Controller are ca parametrii ai constructorului modelul Restaurant, view-ul corespunzator, iar in plus, AdministratorController are parametrul de tip String numit file, necesar in cadrul serializarii ( se va explica in detaliu in cadrul informatiilor despre serializare).

*Clasa ChefView*

Acesta este un View diferit de celelalte doua. Fereastra pentru bucatar va contine doar o casuta de text in care va fi afisat(la fiecare anuntare de adaugare intr-un order a unui produs compus) textul “Am primit un nou order ce contine un produs compus!” la care se concateneaza valorea variabilei de clasa numberOrders, ce se mareste la fiecare notificare de acest fel. Astfel, daca adaug un order ce contine un produs compus se va afisa “Am primit un nou order ce contine un produs compus!1". Daca mai adaug un order ce contine doua produse compuse se va afisa “Am primit un nou order ce contine un produs compus!3". In constructor, este adaugata clasa ChefView ca ascultator pentru restaurant.

*Clasele TableAD, TableWA*

Aceste clase contin codul reprezentativ pentru crearea unui tabel JTable. Acestuia i se declara ca variabila de clasa un obiect model de tip DefaultTableModel ce va constitui modelul de afisare al tabelului ce va fi ulterior setat pentru tabelul JTable declarat tot ca variabila de clasa. Functia insert este apelata pentru a insera date in table. In TableAD se insereaza intr-un tabel(in fereastra Administratorului) datele din meniu prin apelarea functiei modelului addRow ( ce lucreaza cu datele stabilite anterior, de scriere a numelor produselor intr-un ArrayList de String-uri si de asemenea aceeasi procedura si pentru preturi). Pentru a se vizualiza tabelul se va apasa un buton care are ca actionEvent crearea unui obiect de tip TableAD ce la fiecare moment de timp va afisa datele existente in meniu. Acelasi lucru se repeta si pentru TableWA care doar va afisa in cadrul ferestrei Waiter-ului un tabel cu order-urile realizate.

Pachetul DataLayer

*Clasele FileWriter si RestaurantSerializator*

In clasa FileWriter este implementata metoda statica de serializare a unui obiect. Este trimis ca parametru obiectul de tip Restaurant si un String file, ce va reprezenta numele fisierului ce se vac rea in urma serializarii, cu extensia ser. Se creaza fisierul de output FileOutputStream ce va fi trimis ca parametru la crearea obiectului de tip ObjectOutputStream. Acest obiect va apela functia writeObject ce are rolul de a transforma datele transmise ca parametru in octeti. Ca parametru voi trimie obiectul pe care doresc sa-l serializez si anume meniul restaurantului.

In clasa RestaurantSerializator este implementata metoda statica de deserializare a unui anumit obiect. Se merge pe acelasi principiu, dar pe o logica inversa. Acum se doreste reconstituirea obiectului ce a fost serializat. Se creeaza obiectul FileInputStream (ce va primi in constructor ca parametru fisierul restaurant.ser) ce va trimis ca parametru pentru obiectul de tip ObjectInputStream. Acest obiect va apela functia readObject ce va reprezenta intocmai meniul serializat mai devreme. Este setat ca meniu la obiectul de tip Restaurant trimis ca parametru.

Functia de deserializare nu se apeleaza la prima rulare deoarece initial nu exista creat deja fisierul restaurant.ser. Dar de la a doua rulare se va apela. Astfel ce a fost adaugat in meniu la rularea anterioara, se va regasi in meniu si la rularea curenta. Se creeaza astfel iluzia de pastrare a meniului, deoarece si in realitate putem intra intr-un restaurant, putem face comenzi si sa revenim a doua zi, iar a doua zi meniul din restaurantul respective este inca accesibil.

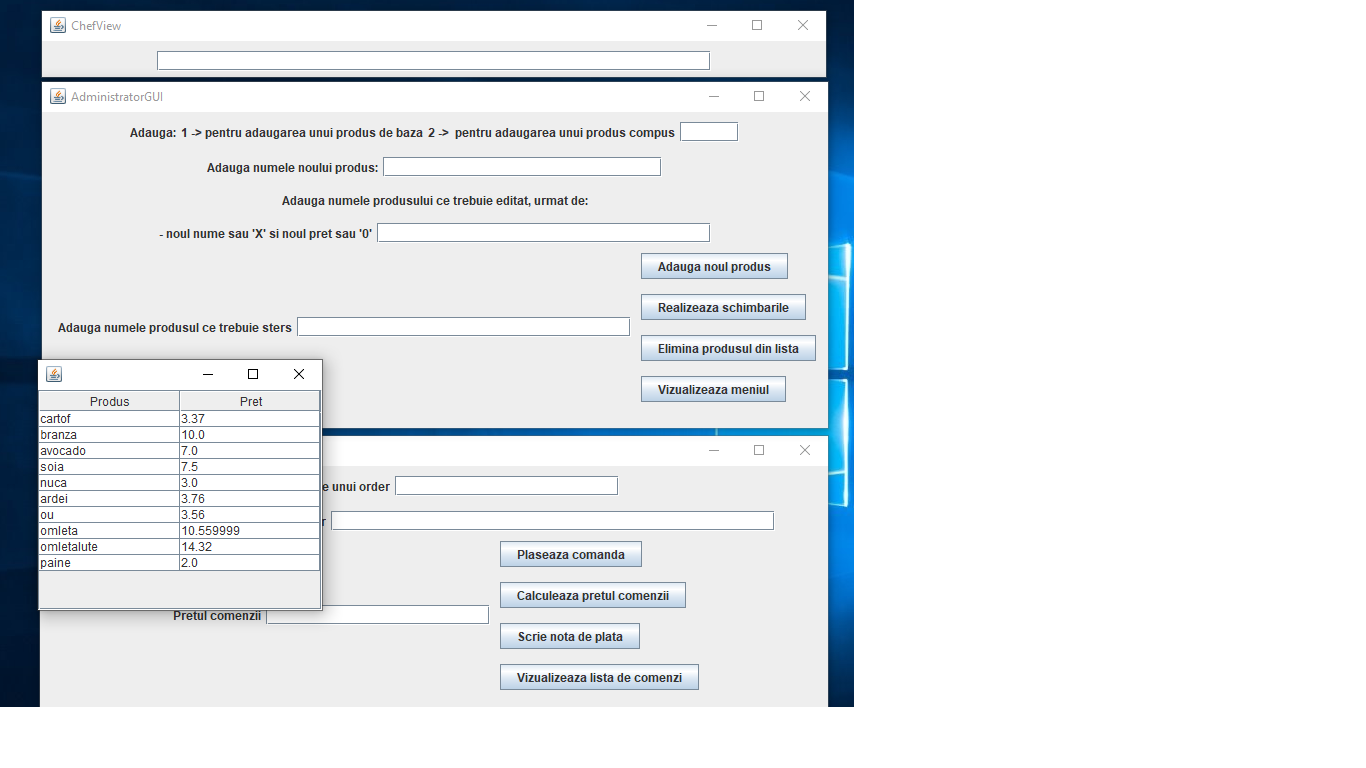
Functia de serializare se apeleaza in clasa AdministratorController dupa apelarea fiecarei metode din restaurant ( de adaugare meniu, editare si stergere). Astfel se pastreaza fluxul modificarilor, cumva se suprascrie la fiecare modificare fisierul restaurant.ser.

In clasa MainClass este creat restaurantul, este apelata functia de deserializare( de la a doua rulare) si sunt create clasele ce alcatuiesc cele 3 interfete grafice.

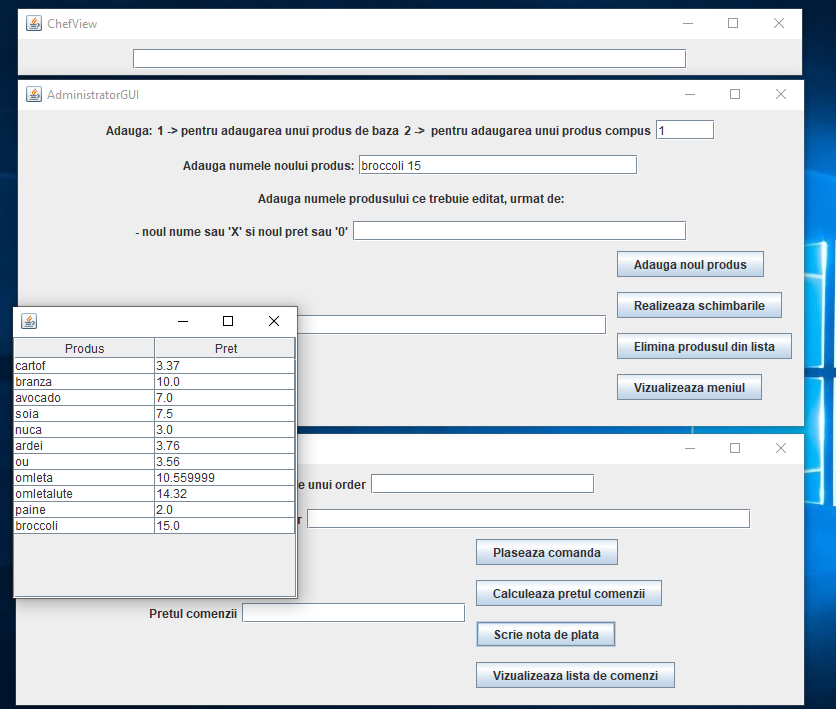
**-5- Rezultate**

Rezultatele sunt vizibile in interfata grafica si depend de datele adaugate de utilizator si de deciziile luate de acesta. Voi atasa niste imagini cu se intampla in functie de diferite comenzi.

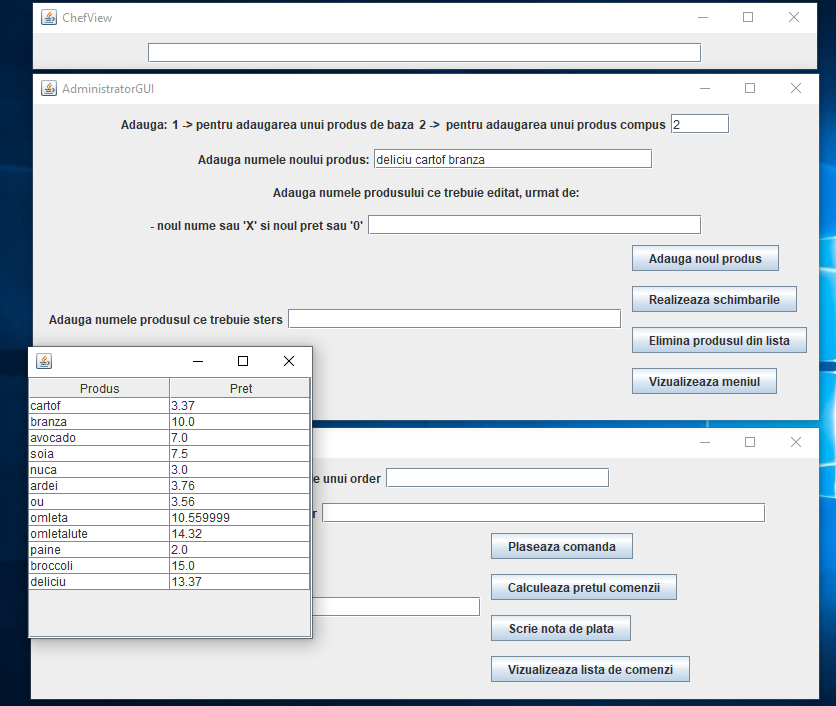
Initial verificam meniul, sa vedem produsele existente.



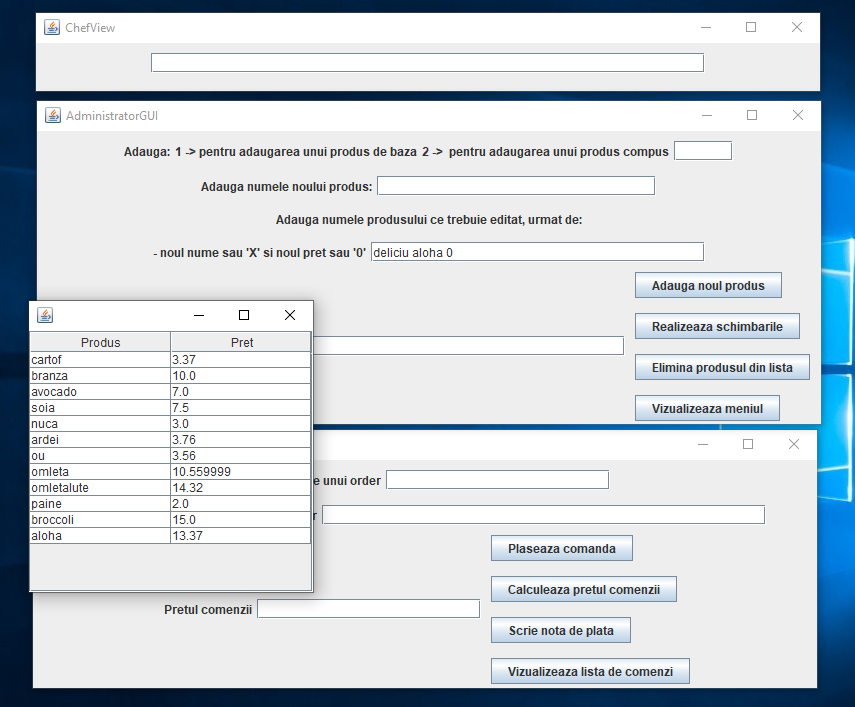
Adaugam un produs de baza broccoli, cu pretul 15.



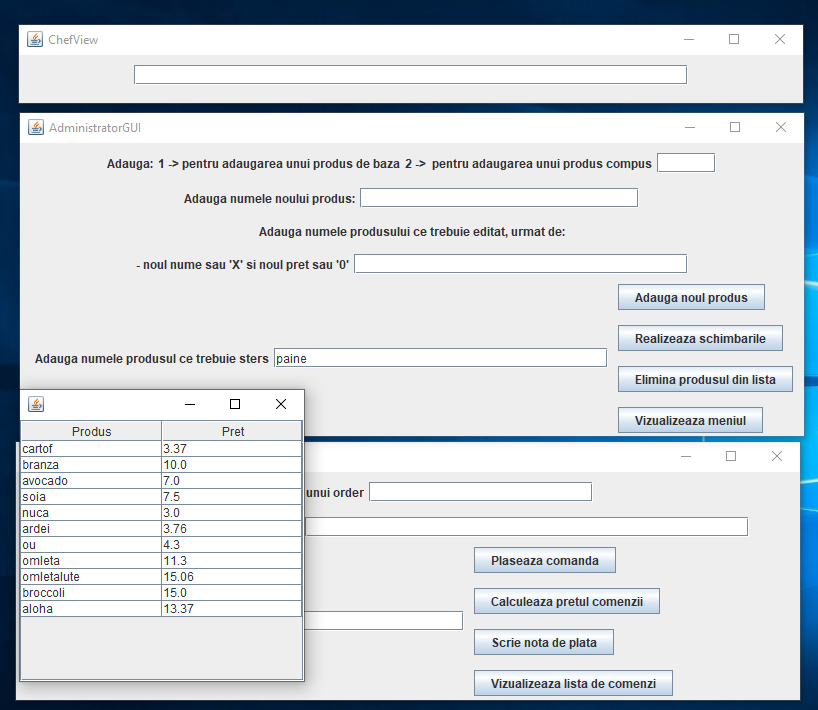
Adaugam un produs compus cu numele deliciu, alcatuit din cartof si branza.



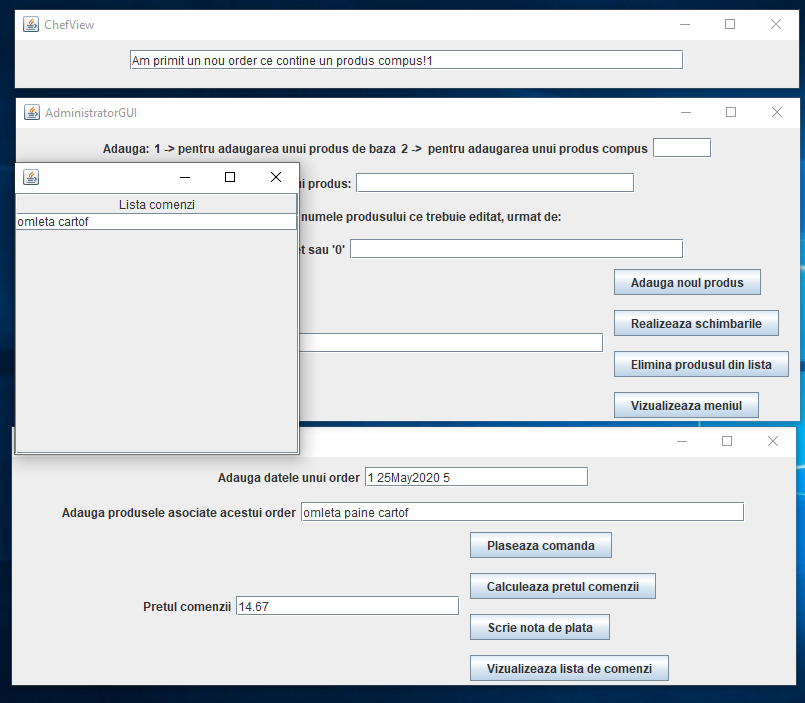
Modificam numele produsul deliciu in aloha.



Stergem produsul de baza paine.



Realizam un order cu produsele omleta, paine si cartof. Se observa ca order-ul a fost adaugat, fiind vizualizat in tabelul, iar bucatarul a fost anuntat ca s-a comandat un produs compus(omleta).



Produsele initale au fost adaugate la prima rulare. Ulterior, pentru verificarea anumitor functionalitati, se pot crea produse compuse noi pentru a testa corectitudinea metodei de adaugare a unui produs. De exemplu se poate adauga produsul prod1 si produsul prod2, dupa care produsul compus prod12 ce contine aceste doua produse. Sau se poate adauga doar produsul prod1 si produsul prod12 care sa aiba in lista de produse mentionata un produs prodNo care nu se afla in meniu. Se va observa ca acesta nu va fi adaugat in meniu.

La fel si cu stergerea. In produsele adaugate initial, produsul omleta si omletaIute sunt produse compuse. Produsul omlet contine produsul de baza ou, iar produsul omletaIute contine produsul compus omleta. Daca dorim sa sterge, produsul ou se va observa ca din meniu se vor sterge atat ou, cat si omleta si omletaIute.

**-6- Concluzii**

In concluzie, acest proiect mi-a adus informatii noi in legatura cu diferitele metode de abordare ale ideilor cand dorim sa le transpunem in Java. Faptul ca cerinta a constat in simularea unui proces existent in viata reala, m-a ajutat sa imi organizez mai bine codul, sa gandesc mai clar implementarea.

Proiectul acesta poate fi dezvoltat, unele functionalitati putand fi imbunatatie cum ar fi adaugarea unei functii de verificare a existentei unei anumite comenzi in lista orders pentru apelarea acesteia in cadrul functiei de computePrice. Cumva sa se poata realiza calculul unui order ales din lista de order-uri realizate.

**-7- Bibliografie**

<https://stackoverflow.com/questions/4987476/java-propertychangelistener>

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

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