MINISTERUL EDUCAȚIEI ȘI CERCETĂRII ȘTIINȚIFICE

UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

FACULTATEA DE AUTOMATICA SI CALCULATOARE

DEPARTAMENTUL CALCULATOARE

DISCIPLINA TEHNICI DE PROGRAMARE

Documentatie Tema4

- Restaurant Management System -

Popa Alexandra Maria

Grupa 30224

CTI-ro

Cuprins:

1.Obiectivul Temei:

Enuntul Temei : Propunerea, proiectarea si implementarea unei diagrame de clase care descrie functionarea unei sistem de management al unui restaurant. Sistemul poate sa permita logarea pentru trei tipuri de utilizatori: administrator, chelner si bucatar. Administratorul poate adauga, șterge și modifica produsele existente din meniul. Chelnerul poate crea o comanda noua pentru o anumita masa, poate adauga elemente din meniu și poate calcula facture pentru o comanda. Bucatarul este informat de fiecare data cand trebuie sa gateasca alimente comandate prin intermediul unui chelner.Aplicatia trebuie sa utilizeze tehnicile de programare Design By Contract, polimorfismul, Design Pattern-urile: Observer si Composite, implementare HashMap si serialzare.

Obiective :

Obiectiv principal : Obiectivul acestei teme este proiectarea si implementarea unei aplicatii de gestionare a comenzilor dintr-un restaurant avand in vedere utilzarea anumitor tehnici. Aplicatia trebuie sa utilizeze tehnicile de programare Design By Contract, ce constau in adaugarea unor pre si post conditii aferente fiecarei metode ce descrie o anumita functionalitate a aplicatiei, cat si adaugarea unor asertiuni corespunzatoare acestor conditii.Totodata, aplicatia trebuie ca utilizeze polimorfismul, cat si structura de tip HashMap pentru stocare datelor din interiorul aplicatiei. Se va folosi implementarea design pattern-urilor Observer si Composite pentru simplifcarea comunicarii intre clasele, cat si tehnica de serializare pentru salvarea informatiilor din sistem.

Obiective secundare :

- organizarea pe clase si pachete-> utilizarea diagramelor de clase si pachete
- dezvoltarea algoritmilor folosind POO -> algoritmi de preluarea si procesarea a informatiilor introduse in interfata grafica
- utilizarea structurilor de date -> descrierea structurilor de date utilizate in implementarea aplicatiei

- formularea de scenarii -> modalitati de introducere, editate si stergere produselor din meniu, adugare de de comenzi si generarea facturilor pentru acestea
- implementarea si testarea solutiilor obtinute -> testare prin vizualizarea datelor obtinute in urma unor opeartii efectuate de carte utilizator; aceste informatii sunt afisate intr-un table in interfata grafica sau printr-un mesaj , in cazul introducerii unor date necorespunzatoare.

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

Problema enuntata necesita cunostinte legaate de lucrul cu design pattern-urile *Composite*, pentru modelarea claselor *MenuItem*, *BaseProduct* si *compositeProduct* si *Observer* pentru notificarea bucatarului de fiecare data cand chelerul a intodus o comanda noua. De asemenea, se cere utilizarea tehnicii de salvare prin serializare a informatiilor din clasa *Restaurant* intr-un fisier.Aceste informatii vor fi incarcate la pornirea aplicatiei si in acest fel se realizeaza stocarea datelor utilizate in program. De asemenea este necasara cunoasterea programarii orientate pe obiect pentru crearea si manipularea obiectelor cat si realizarea unor operatii de prelucrare a datelor . Totodata, este nevoie de cunoasterea modalitatilor de crearea a interfetei grafice si de afisarea a tabelelor ce contin informatiile existente in aplicatie, cat si preluarea datelor de intrare prin intermediul acestei interfete .

Din analiza problemei rezulta faptul ca avem nevoie de un set de date de intrare si un set de date de iesire. Datele de intrare sunt specifice pentru fiecare tip utilizator in parte : chelner sau administrator, cat si in functie de operatia care se doreste a fi realizata de carte acestia. Astfel, chelnerul poate aduga noi comenzi in functie de produsele existente in meniu prin introducerea informatiilor necesare, si poate genera factura ce include produsele comandate si costul total al unei anumite comenzi. De asemenea, chelnerul poate vizualiza intr-un tabel toate informatiile cu privire la comenziile preluate. Adaugarea unei noi comenzi necesita

introducerea in interfata garfica a ID-ului unic al comenzii, a datei cand a fost inregistata, in format dd/MM/yyyy, cat si numarul mesei de unde a fost preluata comanda. Administratorul poate introduce noi produse in meniu, poate sa sterga produse din meniu, sau poate edita denumirea si pretul acestor prouse, totodata putand vizualiza intr-un tabel toate produsele existente in meniul restaurantului.Pentru adaugare unui nou produs in meniu, administratorul trebuie sa specifice ce fel de produs doreste sa intoduca: compus sau de baza. In cazul unui produs compus, este necesara in primul rand introducerea informatiilor corespunzatoare produsului si mai apoi introducerea pe rand a ID-ului fiecarui produs care il compue. Editarea se face prin introducerea ID-ului produsului respectiv impreuna cu noul pret sau noua denumire. Bucatarul este notificat la fiecare comanda noua si poate doar vizualiza comenziile primate prin intermediul unui tabel. Acest tabel este actualizat de fiecare data cand chenerul introduce o noua comanda.

In cazul introducerii unor informatii ce nu se afla in concordanta cu cerintele aplicatiei, se va afisa un mesaj in consola, ce notifica faptul ca datele sunt invalide sau inexistente sau existente deja.

Metoda de introducere a datelor a fost special aleasa pentru ca utilizatorul sa poata vizualiza cu usurinta datele introduse de acesta si sa le poata modifica in cazul in care este necesar acest lucru. In plus, tabelete folosite fac posibil accesul reapid si simplu la informatiile din aplicatie .

Dupa introducerea datelor initiale de catre utilizator,acestea sunt preluate si modelate in functie de operatia ceruta. Datele sunt preluate, iar prin intermediul unor metode specifice se realizeaza adaugarea, stergerea si editarea de produse, se introduc comenzi si se generaeza facturi. Rezultatele obtinute vor fi puse la dispozitia utilizatorului sub forma unor tabele ce cotin toate informatiile necesare despre produsele sau comenzile din restaurant la un moment dat. Afisarea trebuie sa fie clara pentru utilizator, scopul fiind ca produsul final sa fi cat mai practic si usor de folosit.

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

Proiectarea aplicatiei de management a unui sistem de tip restaurant a necesitat proiectarea a 2 pachete de clase specifice : BusinessLayer si PresentationLayer, plus pachetul ce contine clasa principala, din care ruleaza aplicatia-*App. BusinessLogicClasse*s cuprinde clasele care alcatuiesc logica aplicatiei :*Restaurant, RestaurantProcessing, BaseProduct, CompositeProduct, MenuItem, Order si Validator; PresentationLayer* cuprinde clase ce realizeaza interfata grafica si totodata comunicarea utilizatorului cu modelul si cu logica din spatele aplicatiei. Interfata grafica se bazeaza pe sablonul MVC ce presupune separarea proiectului in *Model, View si Controller*, in functie de nevoi, reliefand astfel principiile POO.In continuare voi prezenta proiectarea in mare a claselor din pachetele ce formeaza aplicatia, metodele acestora fiind descrise mai pe larg in capitolul 4.

BusinessLayer :

MenuItem -> aceasta clasa are 3 variabile instanta ce reprezinta atributele aferente unui produs : *idItem* - unic, de tipul *int* , *name* - *String*, reprezentand denumirea produsului si *price – int*, reprezentand pretul. Clasa contine metode de tip setter si getter : *getIdItem( ), getName( ), getPrice( ), setIdItem( ), setName( ), setPrice( ),* cat si o metoda de *computePrice( ),* totodata implementand metoda *toString( );*

BaseProduct -> aceasta clasa extinde clasa *MenuItem*.

CompositeProduct -> extinde clasa *MenuItem* si implementeaza metoda *computePrice();* clasa are ca variabila instanta o lista de obiecte de tip *MenuItem*, cat si metode de set si get pentru acesata lista.

Order -> clasa contine 3 variabile instanta corespunzatoare atributelor unei comenzi : *orderId* si *table* de tipul *int*, reprezentand ID-ul unic al comenzii si masa de la care s-a preluat comanda si *date* de tipul *Date*, fiind data in care s-a preluat comanda. Clasa are metode de get si set pentru fecare din

variabilele instanta, totodata implementand metoda *hashCode( ) , equals( ) si toString( ).*

RestaurantProcessing -> interfata ce defineste semnatura metodelor ce urmeaza a fi implemetate in clasa *Restaurant* care implementeaza acesata interfata : *addNewMenuItem(MenuItem mi), deleteMenuItem(int id), editNameOfMenuItem(int id,String newName), editPriceOfMenuItem(int id,int newPrice), addNewOrder(Order o,List<Integer> mi), computePriceForOrder(int id), generateBillForOrder(Order o)*. Totodata, in aceasta clasa se adauga pre si post conditii aferente fiecarei metode.

Restaurant -> aceasta detine 3 variabile instanta : o structura de tip *HashMap* care are ca si cheie o variabila de tip *Order* si ca valoare o variabila de tip *List* ce contine obiecte de tip *MenuItem*, numita *restaurant* si care stocheaza toate comenziile existenta la un momentat in aplicatia restaurantului; o variabila *menu* de tip *List* ce centine obiecte de tip *MenuItem* pentru stocarea tuturor produselor ce compun meniul restaurantului si o variablila *observers* de tipul *List* ce contine obiecte de tip *Observer* , utilizata pentru implementarea design pattern-ului *Observer*. De asemenea, clasa extinde clasa Observable si implementeaza clasele *RestaurantProcessing* si *Serializable*. Clasa *Restaurant*, implementeaza metodele din interfata *RestaurantProcessing* si anume: *addNewMenuItem(MenuItem mi), deleteMenuItem(int id), editNameOfMenuItem(int id,String newName), editPriceOfMenuItem(int id,int newPrice), addNewOrder(Order o,List<Integer> mi), computePriceForOrder(int id), generateBillForOrder(Order o*). Pe langa aceste metode clasa Restaurant defineste si implementeaza si alte metode precum : *isWellFormed( ), viewTabelMenu( ), viewTabelRestaurant( ), serialization(Restaurant r), deserialization( ), attach(Observer observer), notifyAllObservers( ), getObservers( )*. Fiecare metoda implementeaza conditii de tipul assert care asigura faptul ca pre si post conditiile ce au fost difinite din interfata sunt respectate de catre fiecare metoda in parte.

Validator -> clasa ce are ca variabila instanta un obiect de tip *Restaurant-rest* si o metoda metoda de validare pentru fiecare metoda din interfara *RestaurantProcessing*, implementata in *Restaurant*:

validareAddNewMenuItem(MenuItem mi), validareDeleteMenuItem(int id), validareEditNameOfMenuItem(int id,String newName), validareEditPriceOfMenuItem(int id,int newPrice), validareAddNewOrder(Order o,List<Integer> mi), validareComputePriceForOrder(int id), validareGenerateBillForOrder(Order o) .

PresentationLayer :

Pentru fiecare tip de utilizator: chelner, administrator sau bucatr, exista cate o fereastra de introducere si vizualizare de catre utilizator a datelor specifice. Ca urmare vom avea nevoie pentru fiecare fereastra de o clasa de tip *View* si una de tip *Controller*.

ViewPrincipal -> corespunde ferestrei principale in care sunt prezentate optiunile pentru utilizator, in functie de pozitia pe care o ocupa: chelner, administrator sau bucatar. Contine o variabila instanta -*optiuni* -de tip *JComboBox<String>* , un buton de tip *JButoom* care face trecerea spre fereastra corespunzatoare alegerii utilizatorului si o variabila instanta de tip *JLabel*. Totodata clasa contine metode de get si set cat si metode ce adauga ascultatori (*action listener*) pentru *comboBox* si buton.

ViewChelner -> foloseste 6 variabile instanta de tip *JLablel* care indica semnificatia campurilor text : *optiuniChelnerL, dateComandaL, idComandaL, dataComandaL, masaComandaL, idProdusL*; 4 de tip *JTextField*, editabile, pentru introducerea datelor necesare operatiilor specifice : *idComandaTF, dataComandaTF, masaComandaTF, idProdusTF*; 4 variabile de tip *JButton*, la apasarea carora se executa operatia dorita : *addComandaB, vizualizareComenziB , generareFacturaB, addProdusB*; o variabila instanta de tip *JTable*, una de tip *JScrollPane* si una de tip *DefaultTableModel* pentru afisarea produselor din meniu sau a comenziilor . Clasa contine de asemenea metode de set si get, metode ce adauga ascultatori pe butoane, precum si o metoda ce seteaza tabelul afisat in interfata *setTable(JTable tabel).*

Implementarea clasei *ViewAdministrator* este asemanatoare cu implementarea clasei *ViewClient*, diferenta constand in numarul,

denumirea sau tipul anumitor campurilor ce preiau datele de intrare. Clasa *ViewBucatar* contine doar o variabila instanta de tip *JLablel* , una de tip *JTable*, una de tip *JScrollPane* si una de tip *DefaultTableModel* pentru afisarea comenzilor venite de la chelner.

ControllerPrincipal -> reprezinta partea de control al aplicatiei, ce decide ce pasi urmeaza sa faca modelul. Clasa contine o variabila instanta- *viewPrincipal*, de tipul *ViewPrincipal* , ce realizeaza legatura cu clasa *ViewPrincipal.* Constructorul clasei contine un ascultator pentru butonul *Continua* , care este adaugat la *viewPrincipal*. Clasa de control contine doua clase interne *ComboBoxListener* si *ViewPrincipalListener* ce implementeaza *ActionListener* si contin metoda *actionPerformed(ActionEvent e)*.

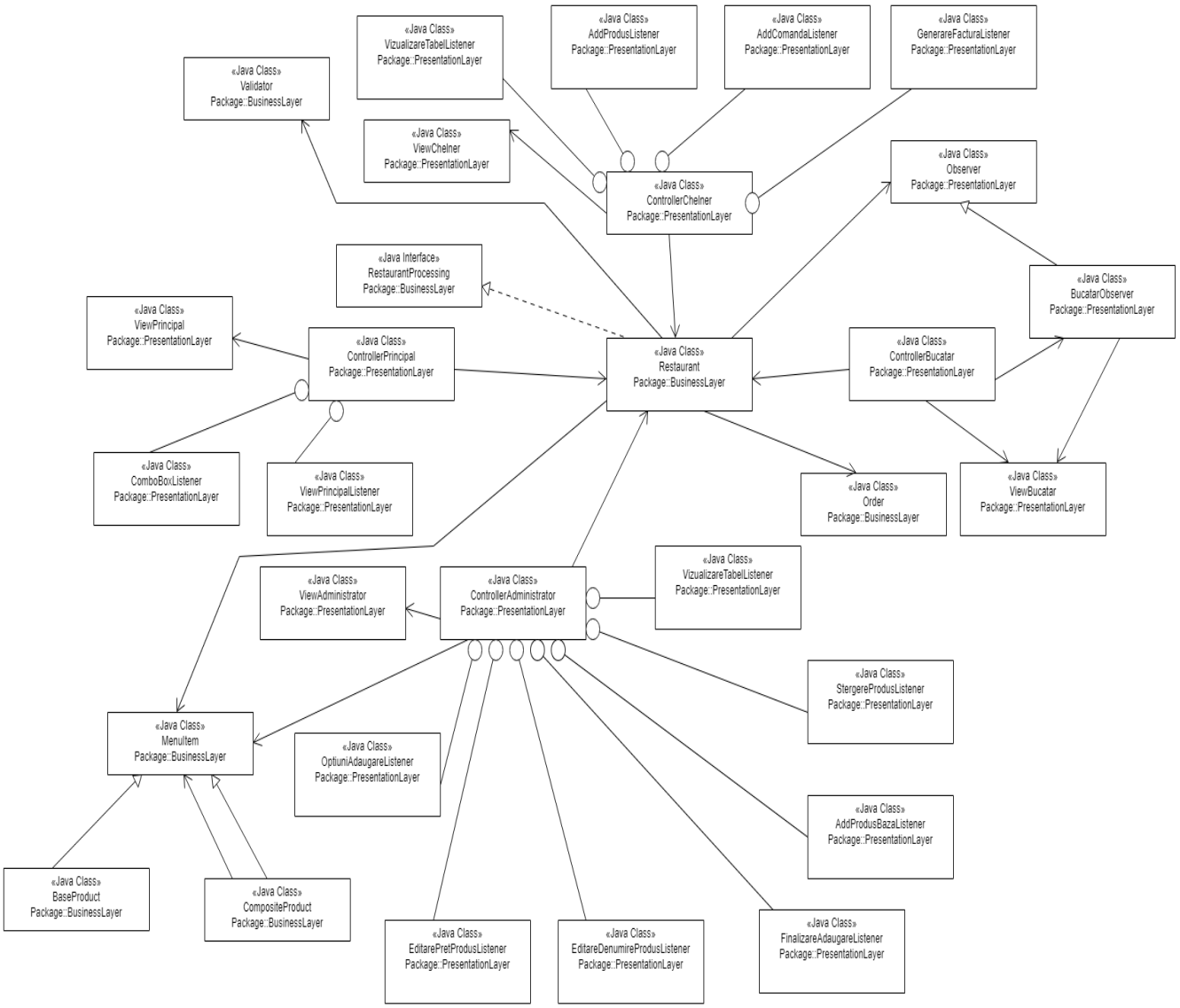
ControllerChelner-> contine o variabila instanta- *viewChelner* de tipul *ViewChelner*, ce realizeaza legatura cu clasa *ViewChelner.* De asemenea, constructorul clasei contine cate un ascultator pentru fiecare buron , care este adaugat la *viewChelner* , un obiect de tip Restaurant si o lista de obiecte de tip Integer. Clasa contine la randul ei patru clase interne *AddComandaListener( ), AddProdusListener( ), GenerareFacturaListener( ) si VizualizareTabelListener( )* ce implementeaza *ActionListener* si contin metoda *actionPerformed(ActionEvent e)*.

Clasele *ControllerAdministrator* si *ControllerBucatar* contin la randul lor variabile instanta de tipul view-ului corespunzator. Clasa adauga *ControllerAdministrator* ascultatori pe butoane pentru ca mai apoi sa implementeze logica de preluare si afisarea datelor prin intermediul celor 7 clase interne corespunzatoare fiecaria. Clasa *ControllerBucatar,* in schimb, nu contine clase interne si nici metode, doar o variabila de tip BucatarObserver.

Clasele *BucatarObserver* si *Observer* sunt folosite in implementare design pattern-ului *Observer*, si prin intermediul acestor metode se semnaleaza in fereastra coresounzatoare bucatarului de fiecare data cand chelerula adauga o noua comanda .

App -> contine programul principal care initializeaza interfata si leaga componentele din pachetele aplicatiei impreuna.

In continuare este prezentata diagrama UML de clase a proiectului:



Ca structura de date, a fost folosita structura de tip *HashMap*, utilizata pentru stocarea tuturor comenziilor dintr-un obiect de tip *Restaurant*.

4. Implementare :

In acest capitol se vor descrie deciziile de implementare ale metodelor fiecarei clase in parte, impreuna cu algoritmii ce sunt folositi in realizarea acestora .

CompositeProduct :

computePrice( ) -> calculeaza pretul unui obiect de tip *CompositeProduct* in functie de preturile produselor care il compun

Restaurant :

addNewMenuItem(MenuItem mi) -> metoda de adugare a unui nou produs in meniul restaurantului

deleteMenuItem(int id) -> metoda de stergere a unui produs din meniul restaurantului

editNameOfMenuItem(int id,String newName) -> metoda prin intermediul careia se editeaza denumirea unui anumit produs din meniu

editPriceOfMenuItem(int id,int newPrice) -> metoda prin intermediul careia se editeaza pretul unui anumit produs din meniu; pretul produselor care sunt compuse din produsul a carui pret a fost editat isi actualize pretul

addNewOrder(Order o,List<Integer> mi) ->metoda ce adugare a unei noi comenzi pe baza produselor existente in meniul restaurantului

computePriceForOrder(int id) -> metoda ce returneaza pretul unei comenzi anume

generateBillForOrder(Order o) -> metoda ce genereaza un fisier .txt ce contine factura corespunzatoare comenzii cerute, cu produsele ce o compun si pretul total

isWellFormed( ) -> metoda ce reprezinta invariantul clasei, fiind apelata la inceputul si sfarsitul fiecariei metode pentru a asigura faptul ca nu exista comenzi cu chei nule sau valori nule, si faptul meniul nu este nul

viewTabelMenu( ) ->metoda ce returneaza o matrice de obiecte, necesara pentru afisarea tabelului cu toate produsele din meniu in interfata grafica

viewTabelRestaurant( ) -> metoda ce returneaza o matrice de obiecte, necesara pentru afisarea tabelului cu toate comenzile de la un moment dat in interfata grafica

serialization(Restaurant r) -> metoda ce serializeaza obiectele de tip Restaurant si pe care le salveaza sub forma de bytes intr-un fisier cu extensie .ser de unde pot fi accesate mai apoi prin deserializare

deserialization( ) -> metoda deserializeaza obiectele dintr-un fisier cu extensia *.ser*

attach(Observer observer) -> metoda ce adauga obiecte in lista de obiecte de tip *Observer*

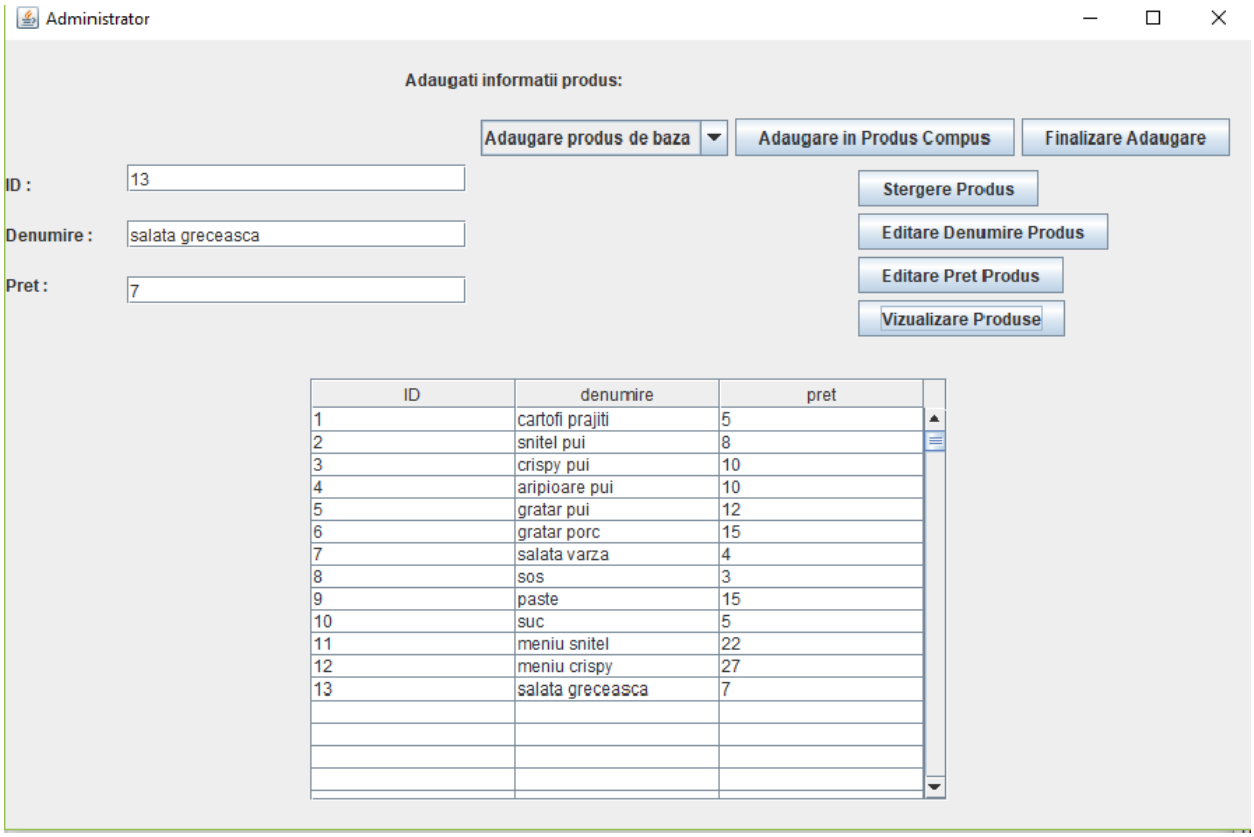
notifyAllObservers( ) -> metoda ce notifica toti observarii cu privire la faptul ca s-a produs o schimbare prin apelarea metodiei update()

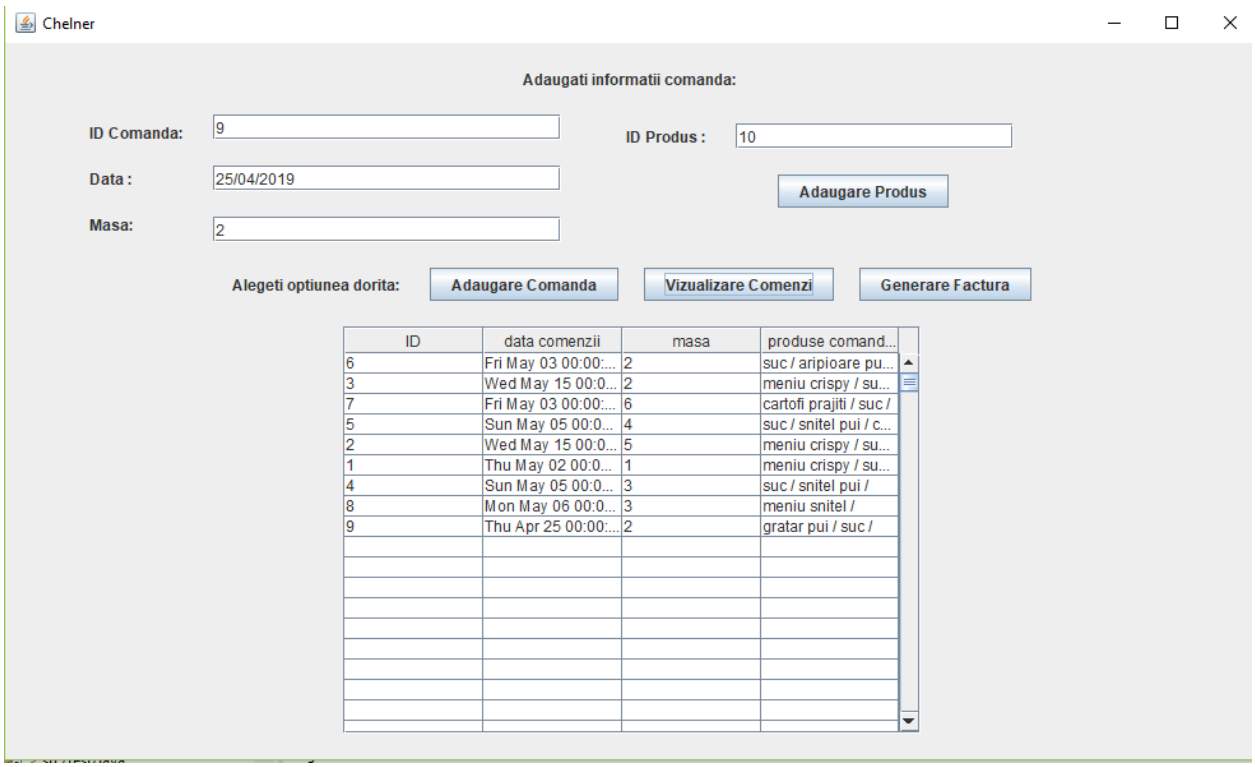
getObservers( ) -> metoda ce returneaza lista de observeri din clasa *Restaurant*

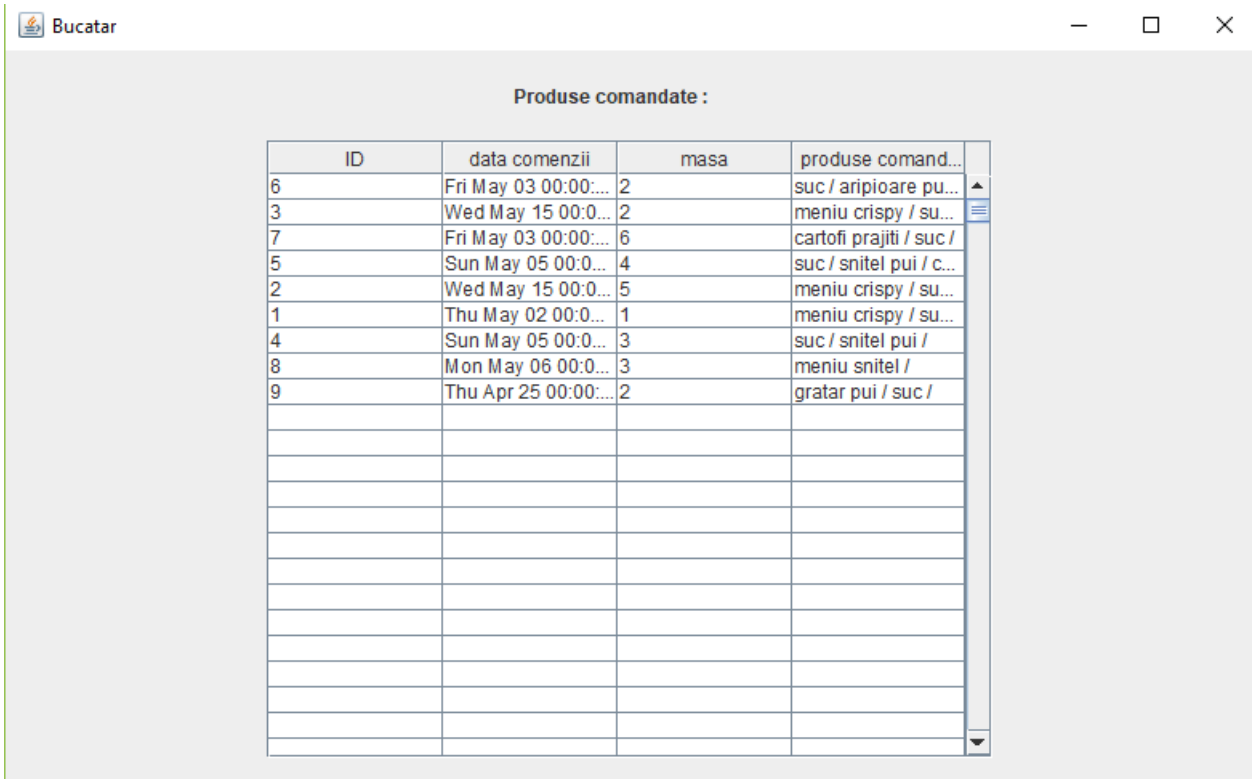
Metodele clasei *Validator* corespund fiecarei metode din clasa *Restaurant* si au rolul de a verifica daca datele de intrare sunt corespunzatoare si de a notifica utilizatorul in caz contrar.

5. Rezultate:

Rezultatele obtinute in urma plasarii unei comenzi sunt prezentate prin intermediul intergetei grafice.


Administrator
Adaugati informatii produs:
Adaugare produs de baza
Adaugare in Produs Compus
Finalizare Adaugare
ID :
13
Denumire :
salata greceasca
Pret :
7
Stergere Produs
Editare Denumire Produs
Editare Pret Produs
Vizualizare Produse
ID
denumire
pret
1 cartofi prajiti 5
2 snitel pui 8
3 crispy pui 10
4 aripioare pui 10
5 gratar pui 12
6 gratar porc 15
7 salata varza 4
8 sos 3
9 paste 15
10 suc 5
11 meniu snitel 22
12 meniu crispy 27
13 salata greceasca 7


Chelner
Adaugati informatii comanda:
ID Comanda:
9
Data :
25/04/2019
Masa:
2
ID Produs :
10
Adaugare Produs
Alegeti optiunea dorita:
Adaugare Comanda
Vizualizare Comenzi
Generare Factura
ID
data comenzii
masa
produse comand...
6 Fri May 03 00:00:... 2 suc / aripioare pu...
3 Wed May 15 00:0... 2 meniu crispy / su...
7 Fri May 03 00:00:... 6 cartofi prajiti / suc /
5 Sun May 05 00:0... 4 suc / snitel pui / c...
2 Wed May 15 00:0... 5 meniu crispy / su...
1 Thu May 02 00:0... 1 meniu crispy / su...
4 Sun May 05 00:0... 3 suc / snitel pui /
8 Mon May 06 00:0... 3 meniu snitel /
9 Thu Apr 25 00:00:... 2 gratar pui / suc /


Bucatar
Produse comandate :
ID
data comenzii
masa
produse comand...
6
Fri May 03 00:00:...
2
suc / aripioare pu...
3
Wed May 15 00:0...
2
meniu crispy / su...
7
Fri May 03 00:00:...
6
cartofi prajiti / suc /
5
Sun May 05 00:0...
4
suc / snitel pui / c...
2
Wed May 15 00:0...
5
meniu crispy / su...
1
Thu May 02 00:0...
1
meniu crispy / su...
4
Sun May 05 00:0...
3
suc / snitel pui /
8
Mon May 06 00:0...
3
meniu snitel /
9
Thu Apr 25 00:00:...
2
gratar pui / suc /


Order_bill.txt
Data comenzii : Fri May 17 00:00:00 EEST 2019
Masa cu numarul : 8
Produsele comandate sunt:
10. suc : 5 lei
12. meniu crispy : 27 lei
Meniul contine :
-> 1. cartofi prajiti
-> 3. crispy pui
-> 7. salata varza
-> 8. sos
-> 10. suc
2. snitel pui : 8 lei
1. cartofi prajiti : 5 lei
Pret total :45

6.Concluzii :

In urma acestei teme am invatat sa imi structurez mai bine codul in clase si pachete de lucru, astfel incat acesta sa respecte pricipiile POO. Am fost pusa in situatia de a realiza o interfata grafica compusa din mai multe clase de tip *View* si *Controller* ce afiseaza rezultatele actualizate din baza de date intr-un tabel de tip *JTable*, ceea ce a adus un plus considerabil cunostiitelor legate de lucrul cu modelul MVC, cat si in ceea ce priveste modului de a scrie cod. De asemenea, am dobandit cunostiinte legate de tehnica *de serializare* , cat si in privinta lucrului cu design pattern-urile *Observer* si *Composite*.

Ca imbunatatiri ulterioare, aplicatia mea ar putea beneficea de imbunatatirea interfetei grafice prin adaugarea unor parole si a logarii in functie de cheler, administrator sau bucatar. Totodata, ca o dezvoltare ulterioare ar fi introducerea unei functionalitati ce sterge automat o comanda din tabelul afisat in fereastra bucatarului odata ce comanda a fost finalizata .

7. Bibliografie :

- Indrumator de laborator POO
- Curs POO
- Curs TP
- https://stackoverflow.com/
- http://coned.utcluj.ro/~salomie/PT_Lic/4_Lab/HW4_Tema4/HW4_Tutorial_Hashing_In_Java.pdf
- https://www.tutorialspoint.com/design_pattern/composite_pattern.htm
- https://www.tutorialspoint.com/design_pattern/observer_pattern.htm

- https://www.tutorialspoint.com/java/java_polymorphism.htm
- https://www.geeksforgeeks.org/serialization-in-java/
- https://www.javaworld.com/article/2074956/icontract--design-by-contract-in-java.html