MINISTERUL EDUCAȚIEI NAȚIONALE

UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

RESTAURANT MANAGEMENT SYSTEM

Documentatie

Facultatea: Automatica si Calculatoare

Roman Alexandru | Grupa 30228

CUPRINS

1. Obiectivul si descrierea proiectului

Pe masura ce tehnologia tot avenaseaza, toate task-urile incep a se automatize, aplicatii sau roboti preluand activitati, altfel desfasurate de oameni. Odata cu avansarea tehnologiei cumparaturile, comenzile oamenilor au inceput sa fie preluate de catre aplicatii si programe care pot diminua nummarul de angajat ai unei firme si pot reduce din timpul de prelucare a informatiilor. Cumparaturile online exsta de mult timp dar mai nou incep sa apara tot mai multe servicii precum aplicatii de comandat mancare de la restaurante/fast food-uri. Este mult mai usor sa comanzi de pe internet sau de pe o aplicatie mancare decat sa mergi pana la magazin sa o comanzi de acolo. De asemenea ar fii mult mai usor si pentrumanagerul restaurantului sa vada meniul undeva fiind capabil sa il modifice usor

Aplicatia creata de mine este un sistem de managament pentru un restaurant prin care se poate modifica meniul, si se pot face comenzi, avand si un sistem pentru crearea unei facturi intr-un fisier .txt.

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

Aplicatia pe care am creat-o are dupa cum vom vedea mai jos, 3 interfete grefice, fiecare cu un anumit rol. Acestea nu sunt lipite unele de altele si pot fii pozitionate oricum pe ecran. O interfata este dedicata modificarii meniului, si anume Administrator.O interfata este destinata crearii de comenzi si gestionarea lor, si anume Waiter. A treia interfata este interfata pentru Chef.Aici se pot vedea toate comenzile facute si se modifica instant, imediat dupa ce a fost facutra o modificare.

Aplicatia are un meniu initial, care se incarca de fiecare data cand pornesti aplicatia si se poate modifica dupa plac si contine 14 elemente(produse).

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

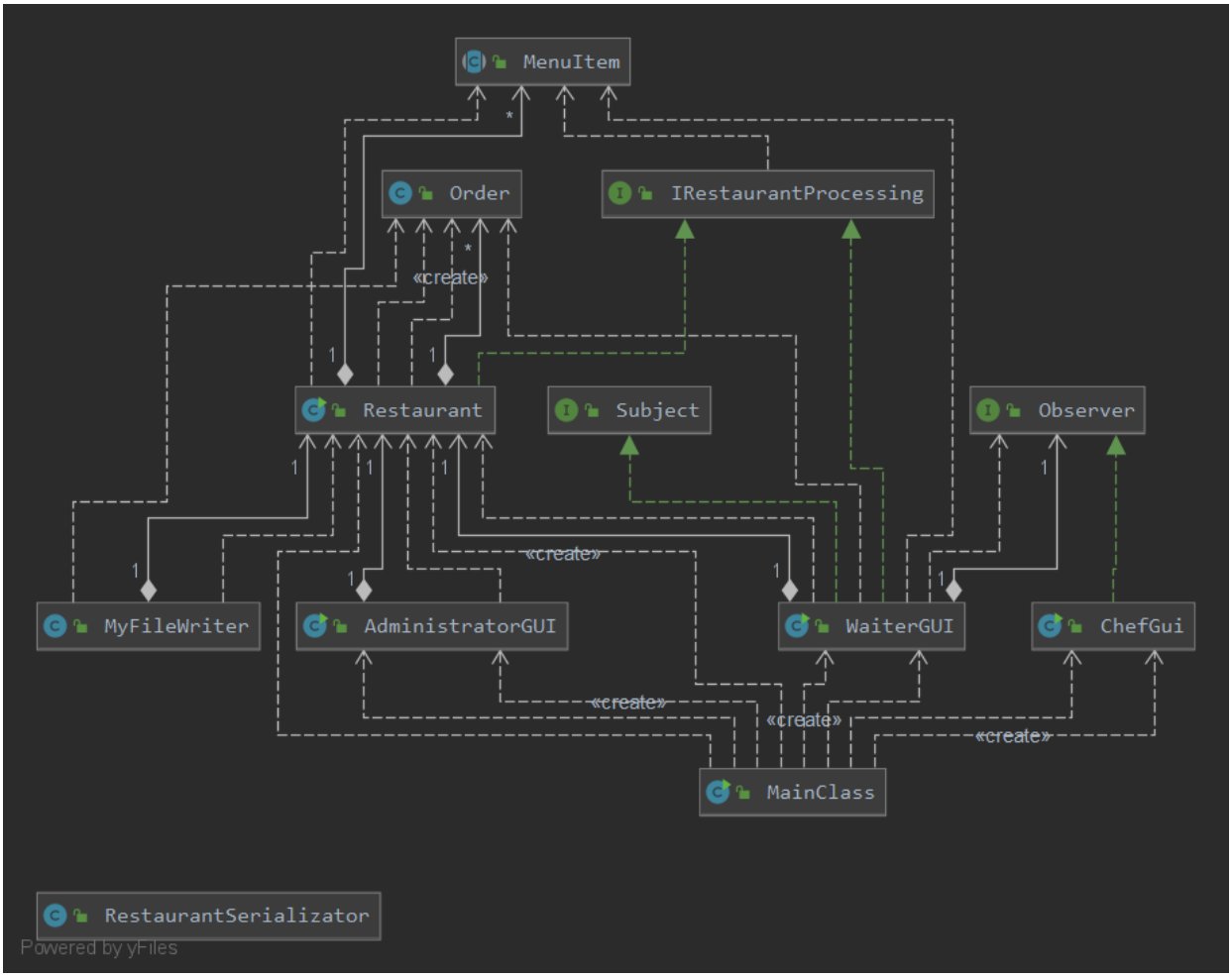
Ca si decizie de proiectare am decis sa structurez proiectul in 3 pachete :

BusinessLayer, DataLayer, PresentationLayer.

- Pachetul ***BusinessLayer*** este pachetul care contine majoritatea claselor pe care la vom explica. Este format atat din clasele care se ocupa de initializarea aplicatiei precum MainClass, Controller cat si din toate clasele care implementeaza ActionListener. Contine si clasele mici folosite pentru blueprintul lor precum MenuItem si interfetele IRestaurantProcessing, Observer si Subject.

- Pachetul **DataLayer** contine clasele care se ocupa de managementul datelor si anume MyFileWriter si RestaurantSerializator(care a ramas o clasa inca neimplementata) dar urma a fii folosita pentru a stoca datale despre meniul si comenzile restaurantului intr-un fisier si a le incarca inapoi in el la pornire.

- Pachetul **PresentationLayer** contine toate interfetele grafice din proiect, adica cele trei principale precum si interfetele care se deschid la anumite operatii precum AddItem sau AddOrder, etc.



Mai sus, putem vedea diagrama UML a proiectului.Proiectul contine foarte multe clase dar am ales doar clasele principale pentru a putea observa mai bine relatiile dintre ele. Sagetile cu verde reprezinta relatia de implementare a interfetelor IRestaurantProcessing, Observer, Subject.Majoritaeaa sunt relatii de depententa intre obiectele care nu pot exista unele fara celelalte si putem observa si cateva relatii de agregare(cand un obiect este parte a altui obiect, de exemplu cum este Observer fara de WaiterGui).

Ca si structuri de date am ales sa folosesc HashMap pentru comenzi care are o complexitate buna, iar in rest ArrayList pentru siruri de date.

Proiectul, dupa cum am zis si mai sus, are 3 interfete grafice principale: AdministratorGui, WaiterGui, ChefGui. In total sunt 7 interfete, restul fiind folosite la operatii folosite la editarea meniului/adaugare de comenzi. Celelalte 4 interfete se numesc : ItemAddGui, ItemDeleteGui, ItemModifyGui.

4. Implementare

La capitolul implementare voi explica cum am implementat clasele enumerate mai sus.

Clasa ***MainClass:*** *e*ste clasa care contine metoda main(). Aceasta instantiaza clasa restaurant, apoi cele 3 interfete enumerate mai sus, iar apoi clasa Controller.

Clasa ***Restaurant:*** este probabil cea mai importanta clasa din proiect. Ea sticheaza atat meniu cat si comenzile intr-un arrayList, respectiv hashMap. In connstructorul ei doar se initializeaza aceste structuri si se adauga 14 elemente in meniu.Aceasta clasa contine multe metode necesare precum addBaseItem sau AddCompositeItem. Clasa Restaurant implementeaza interfata IRestaurantProcessing si folosind strategia Design by contract se foloseste de niste asserturi pentru a verifica niste preconditii si postconditii ce trebuie indeplinite.

Clasa ***MenuItem:*** Clasa MenuItem este o clasa abstracta care are doua varialbile boolean si anume base, respectiv composite, care devin true atunci cand obiectul respectiv este de tip BaseProduct, respectiv CompositeProduct. Clasele BaseProduct si CompositeProduct extind clasa abstracta MenuItem si seteaza flagul corespunzator pe true si desigur este declarat de tip MenuItem. Atunci cand vrem sa accesam obiectul trebuie mai intai sa verificam prin aceste variabile de tip boolean daca este BaseProduct sau CompositeProduct.Desigur, nu este singura varianta rin care se poate verifica asta. Se putea folosi instanceof, totul tine de alegerea de implementare.

Clasa ***BaseProduct:*** clasa BaseProduct extine clasa MenuItem si contine un id, un String si adica numele si un pret(double). Adica caracteristicile unui produs care nu este compus din mai multe produse.

Clasa ***CompositeProduct:*** contine pe langa BaseProduct un ArrayList care se numeste subproducts si contine elementele din compositie.De exemplu produsul “meniul zilei”poate avea in compozitie ciorba de burta, piept de pui, apa plata, etc.

Clasa ***Order:*** este folosita pentru a stoca un irderId si numarul mesei la care s-a facut comanda. De amintit faptul ca comenzile facute sunt stocate intr-un hashMap de tipul:

```
Map<Order, Collection<MenuItem>>
```

Asta inseamna ca instantele nostre a clasei order vor fii folosite pentru a accesa o comanda care este o colectie(in cazul nostru ArrayList) de produse dorite de client;

Clasa ***Controller:*** in aceasta clasa se vor apela metodele din administrator respectiv waiter(Gui) pentru a adauga ActionListenerurile necesare.Cele din imaginea de mai jos sunt pentru Add/delete/modify item si pentru addorder si generate bill.

```
        this.restaurant = restaurant;
        this.administrator = administrator;
        this.waiter = waiter;

        administrator.setAddItemListener(new AddItemListener(restaurant, administrator));
        administrator.setDeleteItemListener(new DeleteItemListener(restaurant, administrator));
        administrator.setModifyItemListener(new ModifyItemListener(restaurant, administrator));

        waiter.setAddOrderListener(new AddOrderBtnListener(restaurant, administrator, waiter));
        waiter.setGenerateBill(new GenerateBillListener(restaurant, administrator, waiter));
```

Clasa ***Data:*** este folosite pentru a stoca date(Stringuri) folosite de catre Chef. Din moment ce chef doar observa este indeajuns sa ii trimitem informatia necesara afisarii in tabel a datelor. Chef primeste un ArrayList de obiecte tip Data(liniile din tabel), iar Data are 3 stringuri

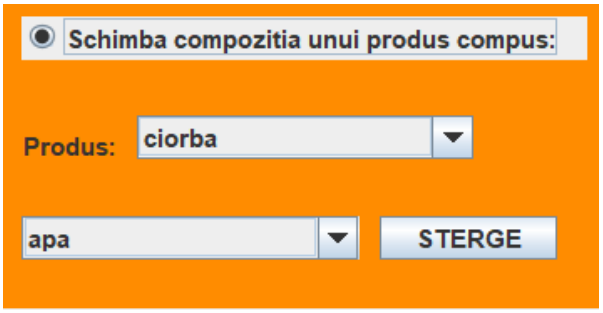
Cate una pentru fiacre coloana.

Restul claselor sunt toate clase care implementeaza ActionListener si metoda AcrionPerformed. Ele primesc, dupa cum se poate vedea si in poza de mais sus instanta clasei Restaurant si a clasei administrator, nuele clase chiar si a clasei waiter.

Clasele care implementeaza ActionListener sunt pentru actiunile facute atat atunci cand apesi pe un buton cat si atunci cand modifici un JComboBox

Mai jos putem vedea actionPerformedul clasei ModifyItemListener instanta fiind adaugata butonului Modify Item.

```
@Override
public void actionPerformed(ActionEvent e) {
    modifyItemView = new ItemModifyGui(administrator.getModel(), restaurant);
    modifyItemView.setVisible(true);

    modifyItemView.setListenerSchimba(new ItemBtnChangeListener(restaurant, administrator, modifyItemView));
    modifyItemView.setListenerSchimbaCompozitie(new ItemBtnAddSubProducts(restaurant, administrator, modifyItemView));
    modifyItemView.setListenerStergeSubProdus(new ItemBtnDeleteSubProducts(restaurant, administrator, modifyItemView));
    modifyItemView.setListenerRefreshSubProductComboBox(new RefreshSubProductComboBoxListener(restaurant, administrator, modifyItemView));

}
```



Casuta de mai sus este din interfata pentru a modifica un item.Aici am folosit un actionListener pentru a putea seta al doilea JComboBox. Al doilea JComboBox reprezinta

produsele din compozitia produsului principal. Acesta este gol pana ce va fii selectat produsul dorit iar apoi se vor adauga produsele din compozitie in aldoilea JComboBox.

Mai jos putem vedea cum arata in interiorul actionPerformedului apelat cand vrei sa schimbi oretul unui produs:

```
} else if (modifyItemView.getChooseNameOrPrice().getSelectedItem().equals("pretul")) {//----------set price

    Double price = Double.parseDouble(modifyItemView.getTextFieldEnterChange().getText());  //pretul nou
    Object obj = modifyItemView.getComboBoxSelect().getSelectedItem(); //produsul selectat
    for (MenuItem itm : restaurant.getMeniu()) {//cautare + schimbare

        if (itm.base) {
            if (((BaseProduct) itm).getName().equals(obj)) {
                ////((BaseProduct) itm).setPrice(price);
                restaurant.schimbaPret(itm, price);
            }
        } else if (itm.composite) {
            if (((CompositeProduct) itm).getName().equals(obj)) {
                ////((CompositeProduct) itm).setPrice(price);
                restaurant.schimbaPret(itm, price);
            }
        }
    }
}

modifyItemView.refreshTable(); //set table
modifyItemView.setFirstComboBox( option: 0, restaurant);//set combobox
modifyItemView.setThirdComboBox( option: 0, restaurant);//set the other combobox
modifyItemView.setFourthComboBox( option: 0,restaurant,(String)modifyItemView.getComboBoxProduct().getSelectedItem());
```

Putem observa de la inceput modul de verificare a instantei produsului pentru a putea face cast la clasa respectiva, este doar o metoda de implementare, si dezavantajul este ca trebuie sa repeti parti din cod de 2 ori doar ca pe un cast diferit.

La final apelam metode din modifyItemView(instanta a interfetei grefice) pentru a da refresh tabelului dupa o schimbare precum si a seta(tot un refresh) JComboBoxurile. Astfel nu vor aparea erori. Aceste metode de setComboBox presului stergerea si refacerea lui.

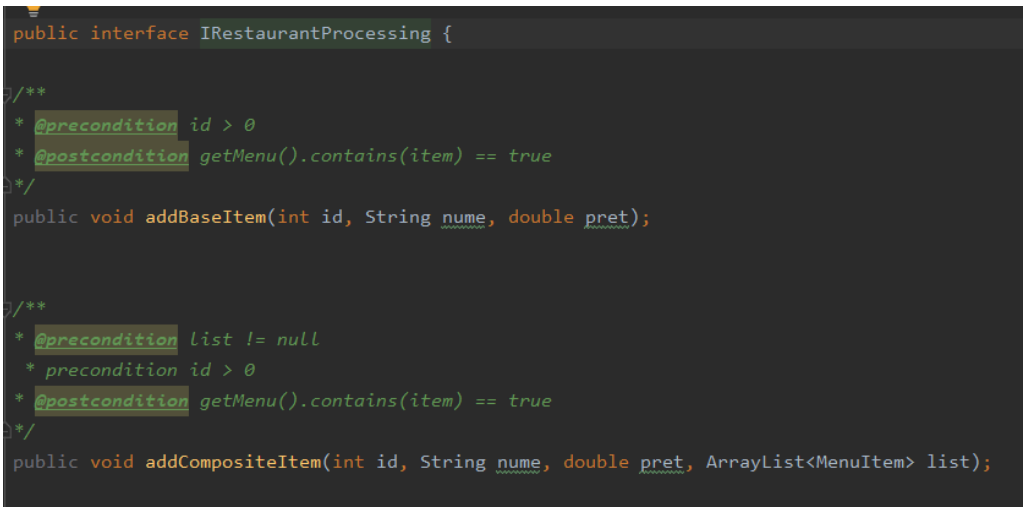
Tabelul foloseste un DefaultTableModel , metoda freshTable seteaza setRowCount pe 0 iar apoi folosind metoda addRow reconstruieste tabelul cu meniul nou.

Medoele actionPerformed din clasele ce implementeaza interfata ActionListener sunt foarte asemenatoare diferind in mare doar operatiile facute pe structuri de date.

Clasa ***FileWriter:*** este o clasa folosita prin apasarea butonului Make Bill. Constructorul sau este apelat in metoda actionPerformed a clasei generateBillListener. Acesta foloseste un FileWriter pentru a scrie in fisier comenzile facute si calculeaza de asemenea costul final al comenzilor. Toate comenzile sunt trecute intr-un fisier text numit Bill.

```
**************************BILL**************************


====Chitanta masa 0====
Coca-Cola-----------------------5.0
Cascaval------------------------11.0
Orez Risotto--------------------9.0
==================================== total 25.0 ron
```

Interfata ***IRestaurantProcessing:*** este folosita pentru a declara metodele folosite de administrator care sunt implementate de catre Restaurant



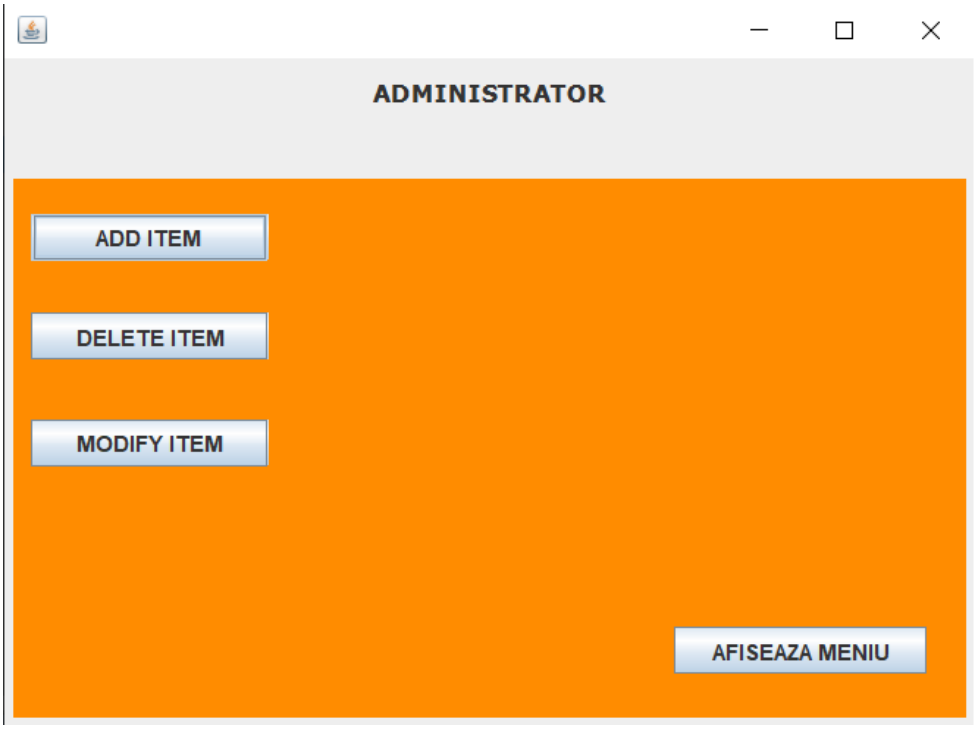
```
public interface IRestaurantProcessing {

/**
 * @precondition id > 0
 * @postcondition getMenu().contains(item) == true
*/
public void addBaseItem(int id, String nume, double pret);


/**
 * @precondition list != null
  * precondition id > 0
 * @postcondition getMenu().contains(item) == true
*/
public void addCompositeItem(int id, String nume, double pret, ArrayList<MenuItem> list);
```

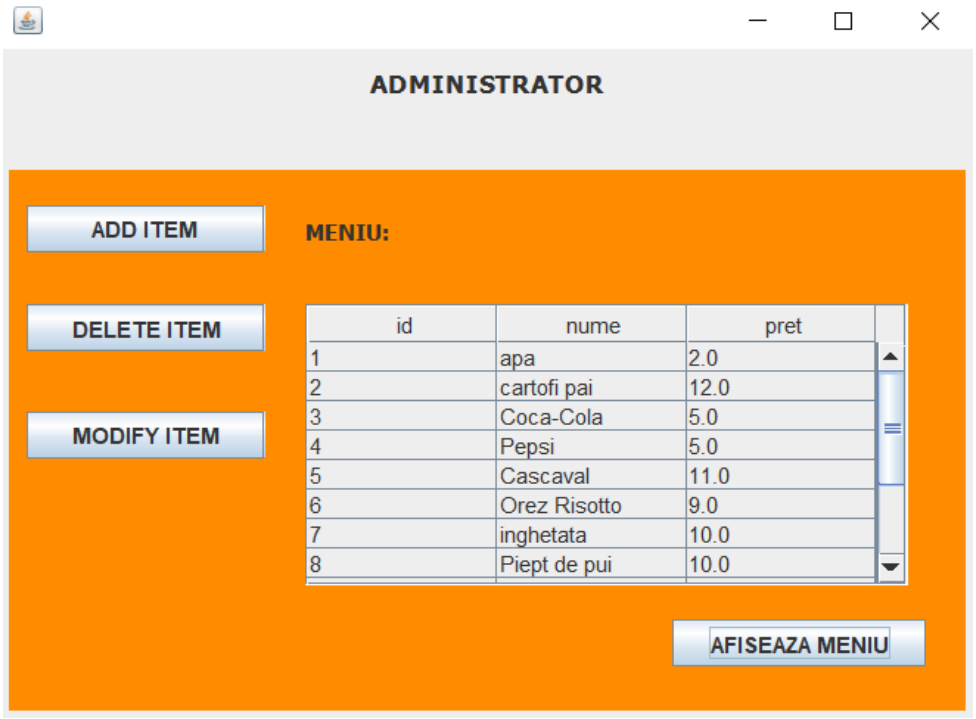
Interfetele ***Observer*** si ***Subject:***

Observe Design Pattern este un software in care un obiect, numit Subject, tine un dependent sau o lista de dependenti numiti Observeri si ii nitifica automat la orice schimbare, deobicei apeland una dintre metodele lor. In acest caz, clasa care implementeaza observerul este ChefGui pentru ca el “observa ” adica primeste informatia, iar Subject este implementat de catre WaiterGui care il are pe Chef ca si Observer in compositie si ii da notify.

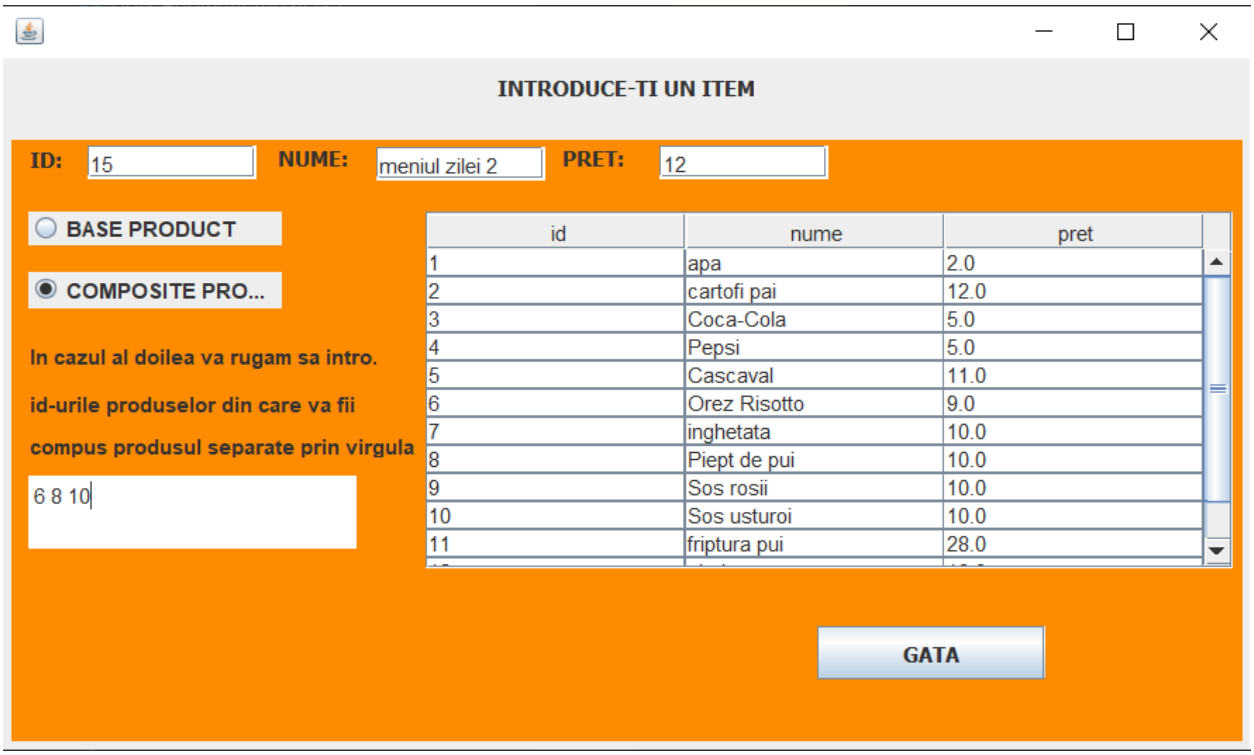
5. Rezultate

Interfara grafica pentru Administrator arata ca mai jos. Are butoane pentru add, delete, modify item iar fiecare deschid cate o interfata noua si de asemenea are butonul afiseaza meniu care este ca un on/ off, adica daca il apesi iti va aparea tabelul de produse iar daca il apesi din nou dispare iar. Toate tabelele sunt adaugate in componenta unui JScrollPane.

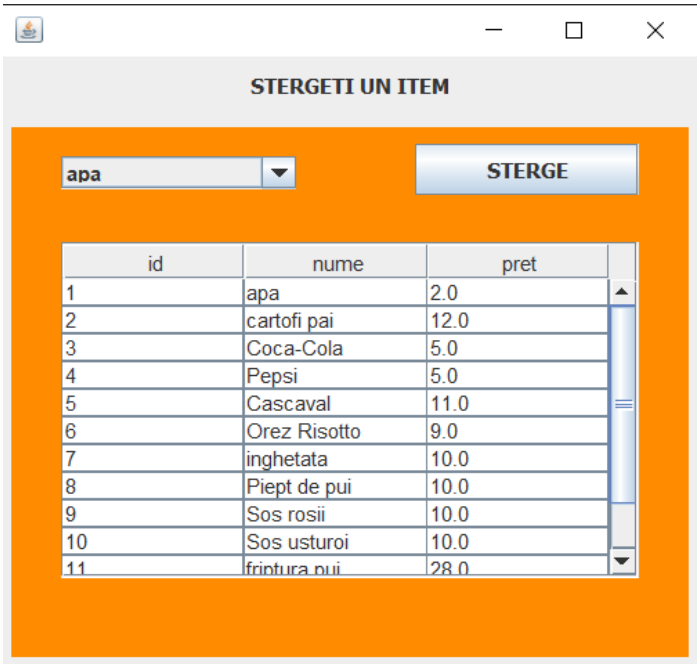




Pentru a adauga un produs trebuie introdus un item, apoi selectat una dintre optiunile base product/ composite product iar in casuta de jos se vor introduce id-urile produselor din compositie in caz de CompositeProduct(separate doar prin spatii).

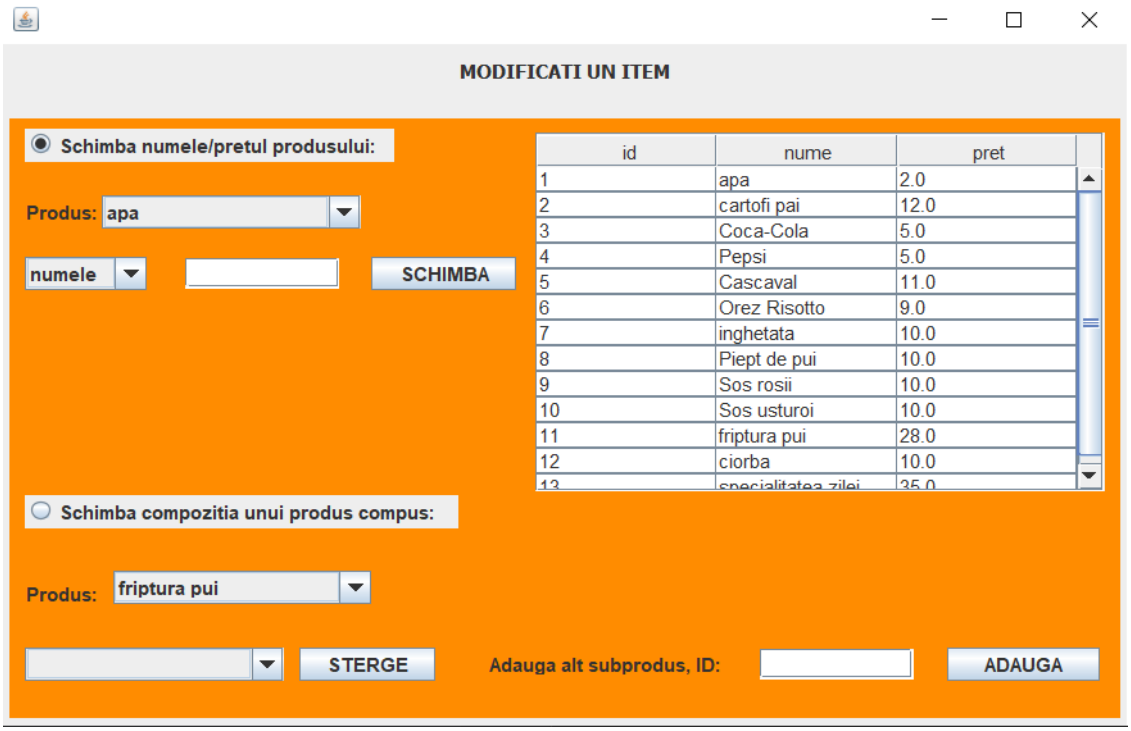


Pentru delete item avem o interfata foarte simpla. Se selecteaza produsul din JComboBox si apoi se apasa pe “sterge”. Toate JComboBoxurile din interfete se vor actualiza, desigur, unde e cazul. In acest caz, cand se va sterge un item, va disparea si din JCombobox.



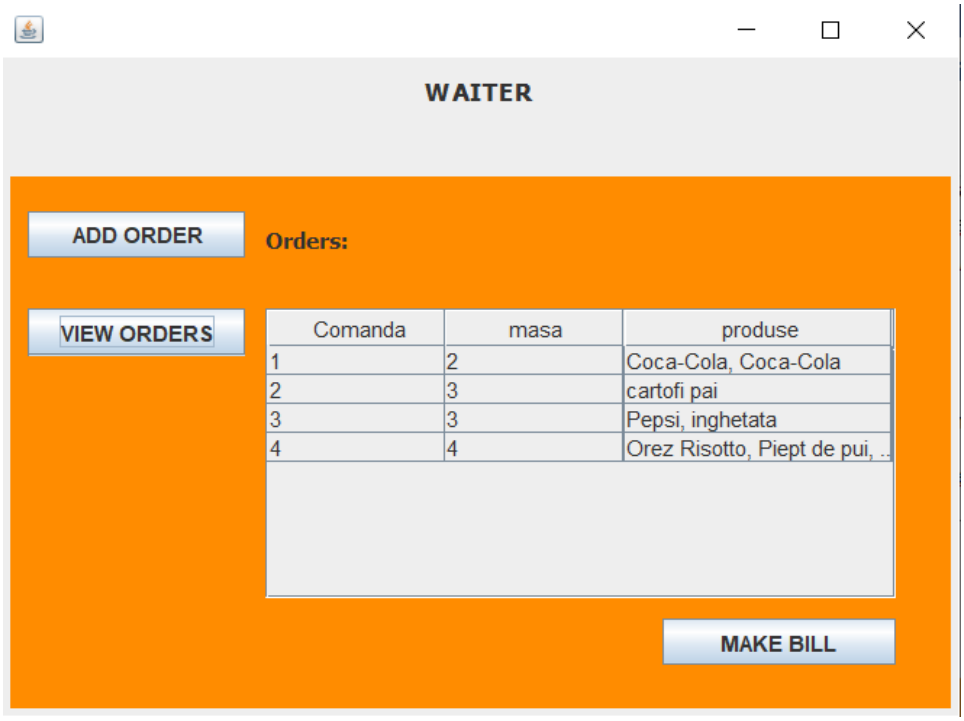
Interfata pentru modificare item pare putin mai dificila dar credeti-ma, nu este. Butonul “schimba” este pentru a schimba pretul/numele unui Produs.

Daca vrei sa stergi un subprodus sau sa adaugi unul vei selecta a doua casuta, iar apoi, un produs. Apoi se va selecta un subprodus pentru stergere, respectiv se va introduse id-ul unui produs pe care vrei sa-l introduci.



Interfata Waiter are doar 3 butoane:

butonul add order, care deschide o interfata noua, butonul View orders care functioneaza si aici ca un switch si poate face tabelul cu comenzi sa dispara si sa apara, la alegere. Avem de asemenea si butonul Make bill care va genera o chitanta intr-un fisier .txt cu toate comenzile facute. Desigur este doar o alegere de implementare, as fii putut sa fac cate un .txt pentru fiecare comanda dar in acest caz s-ar fii creat foarte multe fisiere, cate unul pentru fiecare comanda.



Si avem de asemenea tabelul Chef. Aceste fiind un “Observer” mai mult, nu are niciun buton.

Este o fereastra care se actualizeaza la fiecare comanda prin Observer Design Pattern.

Culorile din interfata le-am ales sa fie mai stridente pentru a iesii in evidenta.

6. Concluzii si posibilitati de dezvoltare ulterioara

In concluzie, pot zice ca dupa fiecare proiect imi aprofundez cunostintele in java tot mai mult. Din acest proiect pot spune ca am invatat unele tehnici de programare si algoritmi precum folosirea unei interfete pentru Observer Design Pattern. De asemenea mi-am aprofundat cunostintele in java Swing si am folosit pentru prima oara un HashMap.

Problemele intampinate de mine la acest proiect au avut de-a face cu niste buguri legate de faptul ca ArrayListul care il foloseam in HaspMap il modificam si astfel imi modificam lista de comenzi.De asemenea maimulte buguri mici, adica greseli care faceau anumite lucruri din interfata sa nu se actualizeze bine dar le-am rezolvat.Singura problema sau baiate de cap de-a mea este crearea Jar-ului si facerea lui sa mearga in cmd, desii mergea de la inceput

Un astfel de proiect ar putea fii folosit prin adaugarea catorva implememementari pentru a manageria un restaurant. Ar putea fii chiar transformat intr-o aplicatie.O aplicatie care are un cont pentru ownerul magazinului care va face modificarile necesare meniului si preturilor produselor(asa cum putem noi modifica in Administrator). In administrator desigur, ar trebuii sa poata intra doar patronul.Clientii ar trebuii sa poata depuna comenzi(ca in Waiter) iar bucatarul ar avea o fereastra a lui care se modifica si ii afiseaza comenzile in timp real.Astfel nu ar mai fii nevoie de prezenta waiterului doar pentru servire eventual ,nu si pentru preluare comenzilor.

Desigur proiectul meu nu este perfect, mai este mult de munca pentru a-l transforma intr-o aplicatie dar o aplicatie de genul ar putea avea mult succes si cu siguranta ar fii utila pentru multe branduri deoarece mai nou, totul se comanda online, este mult mai rapid si necesita mai putina forta de munca din parcea firmalor si mai putin eform din partea cumparatorilor.