Universitatea Tehnica
Din Cluj-Napoca

Sistem de management

Restaurant

Nume: Chiras Valentin-Fanica

Grupa: 30223

Universitatea Tehnica
Din Cluj-Napoca

Cuprins:

Universitatea Tehnica
Din Cluj-Napoca

Tema 4:

Consider implementing a restaurant management system. The system should have three types of users: administrator, waiter and chef. The administrator can add, delete and modify existing products from the menu. The waiter can create a new order for a table, add elements from the menu, and compute the bill for an order. The chef is notified each time it must cook food that is ordered through a waiter.

1.Obiectivul temei

Acest gen de aplicatie software eficientizeaza procesarea informatiilor specifice unui restaurant si, in acelasi timp, automatizeaza toate procesele din cadrul acestuia. Acestea se gasesc la punctele de plata dintr-un food business. Aceast tip de aplicatie este util din urmatoarele motive:

- Management al personalului imbunatatit
- Comunicarea mai buna intre administrator, ospatar si bucatar
- Tranzactiile sunt salvate intr-o baza de date

In aplicatia curenta se va prezenta acest concept intr-un mod mult simplificat. Avand cateva functionalitati care permit operatii de intrare/iesire pentru gestiunea unui restaurant:

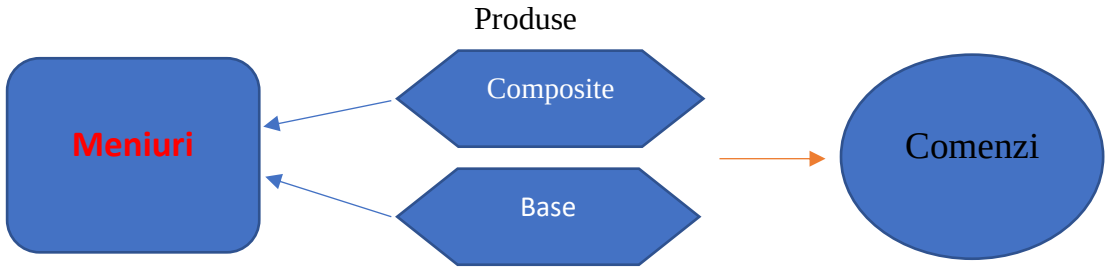
-Adaugare/Stergere/Editare de produse
-Generare de comenzi pe baza meniurilor create
-Generare de bonuri fiscare

2.Analiza problemei

Ca prim obiect ce necesita cuantificat pentru a lucre ulterior aplicatia, este produsul. Fiecare produs va fi de doua tipuri: produs de baza sau produs compus, aceste elemente reprezentand in cele din urma meniul restaurantului.

Comenzile se vor cuantifica printr-in id, data si numarul mesei unde va fi servita. De la aceste date se va dezvolta functionalitatea aplicatiei.

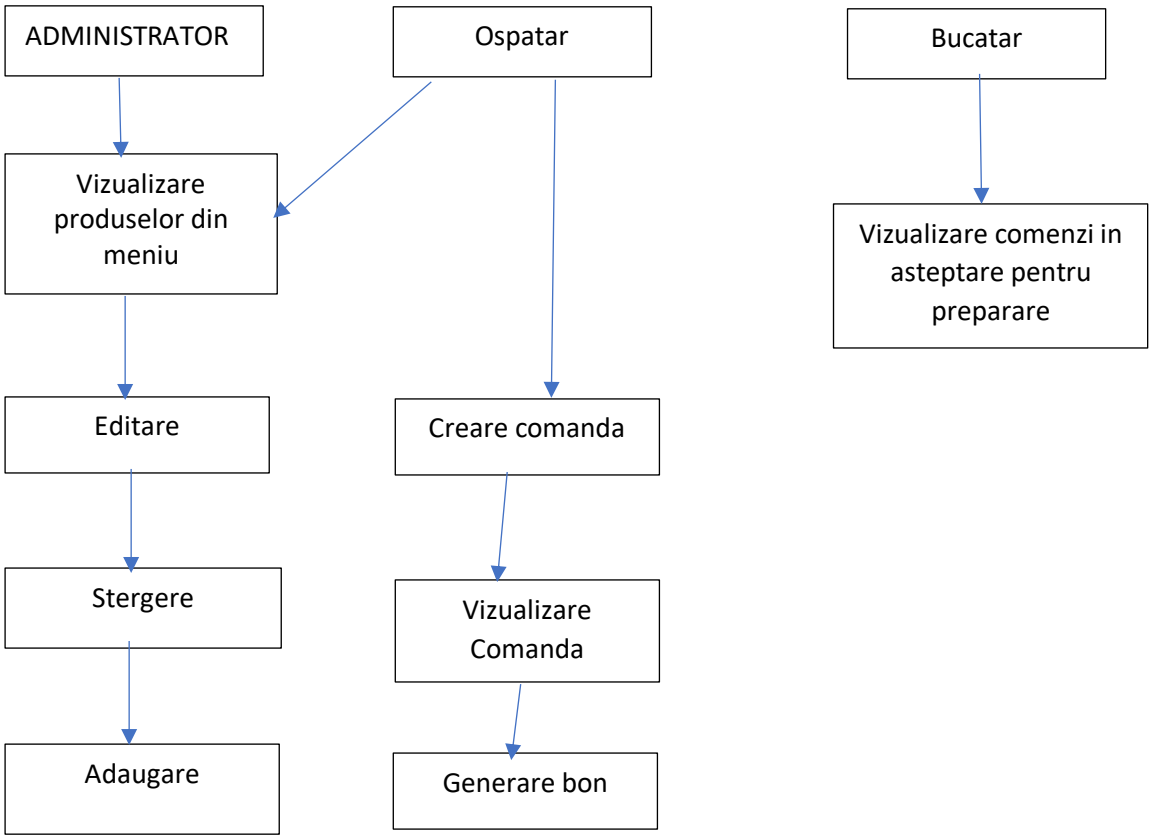
Pentru ca datele sa persiste se va folosi un mechanism prin care acestea vor fi accesibile mereu. In cazul de fata, se va folosi un fiser binar drept baza de date, astfel se vor proteja datele.



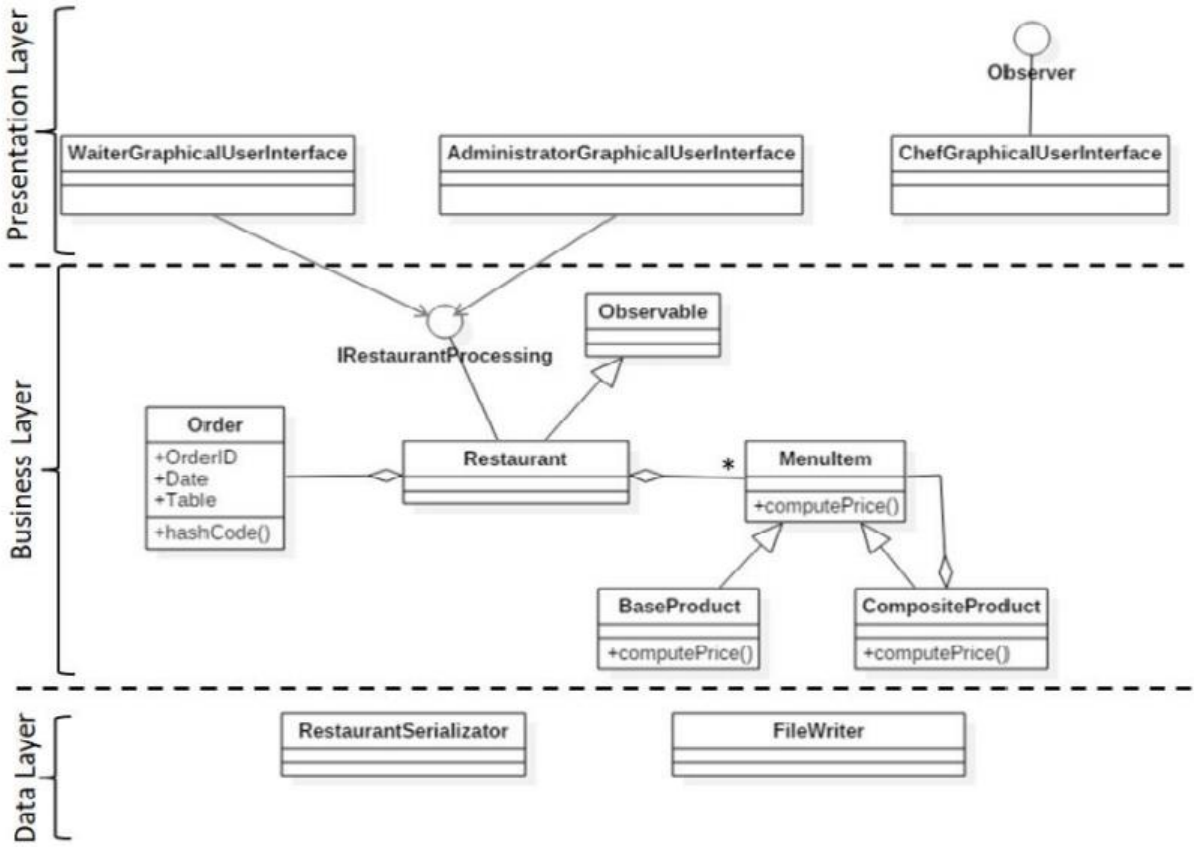
Administratorul se ocupa cu adaugarea si stergerea produselor din meniu, precum si editarea acestor prin intermediul interactiunii persoana-interfata GUI.

Ospatarul se va ocupa de procesarea comenziilor iar bucatarul va fi informat de comanda procesata de ospatar prin intermediul Observer Design Pattern, folosind interfata

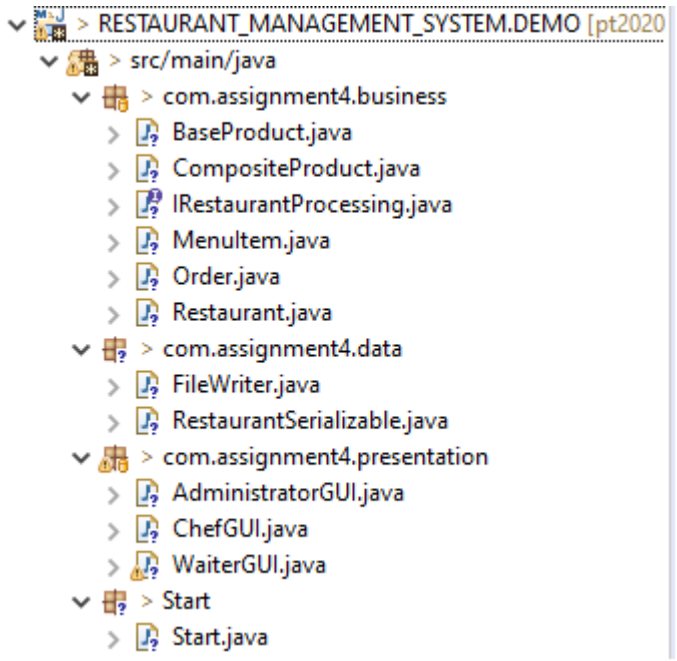
Observable, care este “deprecated” odata cu JDK 9 in prezent. Acest lucru ducand totodata la limitarea eficientei aplicatiei de fata.



3.Proiectare - Diagrama UML



-Proiectare pe clase



In aplicatia de fata s-au folosit 2 sabloane de proiectare care sunt importante.

Un sablon de proiectare descrie o problema care se intalneste in mod repetat in proiectarea programelor si silutia generala pentru problema respective, astfel incat sa poata fi reutilizata oricand este nevoie cu mici adaptari si schimbar in functie de problema. Solutia este exprimata folosind clase si obiecte.

In general, sablonul are 4 elemente esentiale:

-Un nume prin care poate fi referit. Prin atribuirea de nume se creaza un vocabular de proiectare.

-Descrierea problemei. Se explica contextual in care apare, pentru a se cunoaste aplicabilitatea sablonului.

-Descrirea solutiei. Sunt descrise elementele care compun proiectul, relatiile dintre ele si responsabilitatile.

-Consecintele aplicarii sablonului.

Pentru stocarea produselor din meniu s-a folosit "**Composite Design Pattern**":

Unul din scopurile sablonului este acela de a permite clientilor sa lucreze cu obiectele Leaf si Composite (BaseProduct si CompositeProduct, in cazul nostru) in mod uniform. Pentru aceasta, clasa Component, clasa Component trebuie sa defineasca cat mai multe dintre operatiile ce pot sa apara inLeaf si Composite. De obicei, Component ofera implementari implicite la aceste operatii, urmand ca Leaf si Composite sa le redefineasca.

Component:

- declara interfata pentru obiectele ce vor forma ierarhia;
- ofera o implementare implicita, comuna tuturor claselor, pentru operatiile intergetei;
- delcara o interfata pentru accesul si gestionarea obiectelor care compun un container;
- optional, defineste o interfata pentru accesul la obectul container dinspre elementele componente.

Leaf:

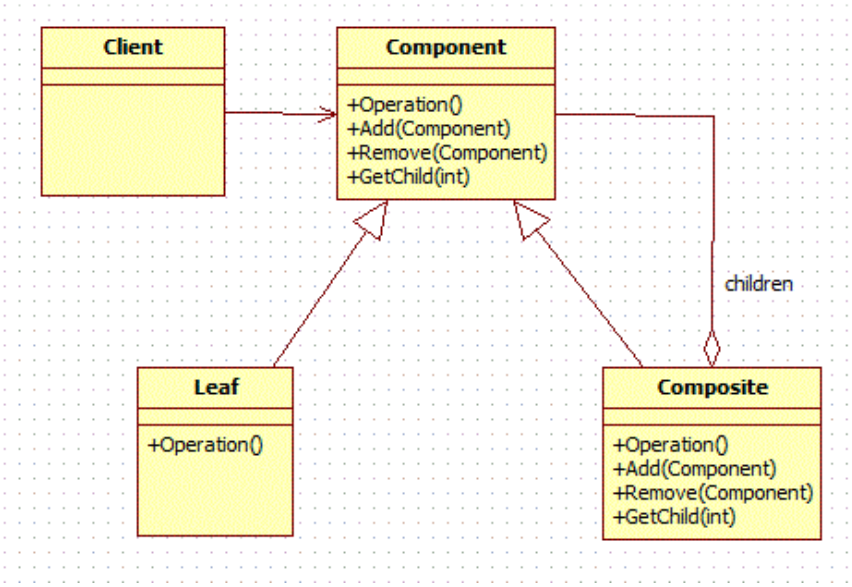
- reprezinta obiecte elementare ale ierarhiei;
- defineste comportamentul specific unui obiect primitv;

Composite:

- defineste comportamentul obiectelor de tip container;

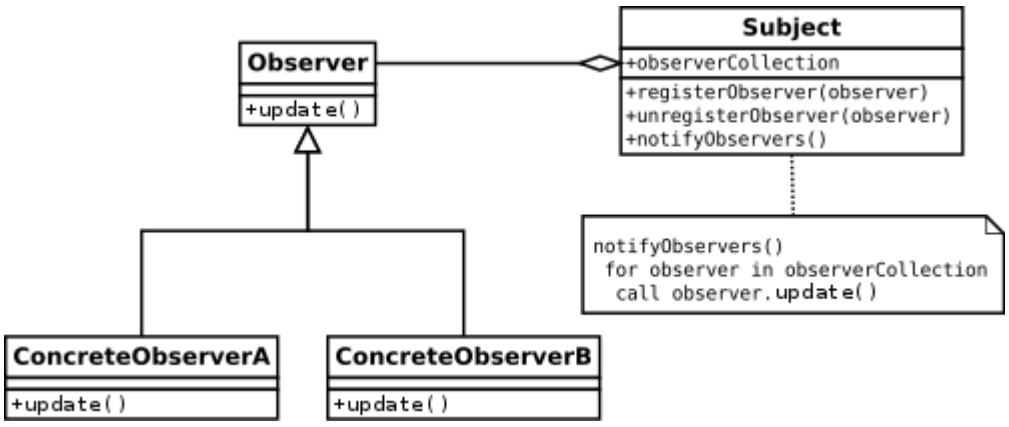
Client:

- manipuleaza obiectele ierarhiei utilizand interfata **Component**.



Intr-o superclasa trebuie definite doar acele operatii care au sens pentru subclasele ei. Tinand cont de aceasta, Component are anumite operatii care nu par sa aibe sens pentru Leaf (operatiile de gestionare a elemntelor unui container). Aici se pune problema daca se poate defini o implementare implicita pentru asemenea operatii. De exemplu, in cazul metodei GetChild, daca vom considera un obiect Leaf este un caz particular de conainer, adica unul care nu are niciodata o componenta, putem sa impunem ca metoda Component::GetChild sa nu returneze nici o componenta, urmand ca Leaf sa utilizeze aceasta implementare, iar Composite sa o redefineasca corespunzator.

Pentru a anunta bucatarul sef despre noile comenzi s-a utilizat “ **Observer Design Pattern**”:



Acest sablon defineste o relatie de dependenta 1…* intre obiecte astfel incat cand un obiect isi schimba starea, toti dependentii lui sunt notificati si actualizati automat.

Acest sablon are ca aplicabilitate:

- cand o abstractie are doua aspect, unul depinzand de celalalt. Incapsuland aceste aspect in obiecte separate, permitem reutilizarea lor in mod independent.

- cand un obiect necesita schimbarea altor obiecte si nu stie cat de multe trebuie schimbate
- cand un obiect ar trebui sa notifice pe altele, fara sa stie cine sunt acestea

Contine urmatorii participant:

- **Subject** → cunoaste observatorii
- **Observer** → defineste o interfata de actualizare a obiectelor ce trebuie notificate de schimbarea subiectelor
- **ConcreteObserver** → memoreaza starea care ar trebui sa fie consistenta cu subiectii

Consecintele folosirii acetui sablon:

- abstractizarea cuplarii dintre subiect si observatory
- notificarea ca un subiect si-a schimbat starea nu neceista cunoasterea destinatarului (comunicare de tip broadcast)
- o schimbare la prima vedere inocenta poate provoca schimbarea in cascada a starilor obiectelor.

Pentru crearea aplicatiei s-a folosit modelul arhitectural MVC care consta in folosirea a 3 pachete diferite care impart logica aplicatiei in 3 elemente interconetate.

Model este cea mai importanta componenta, pe baza careia se construieste aplicatia

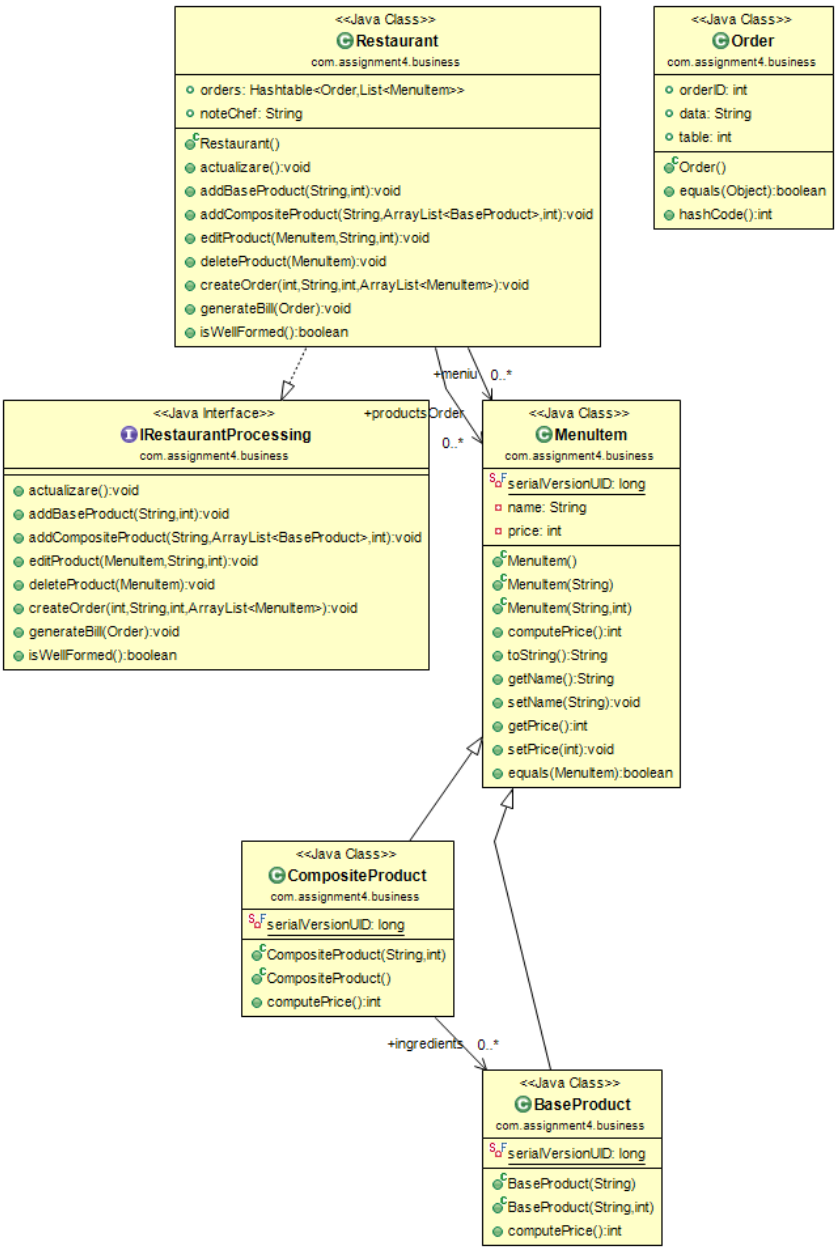
View repezinta partea grafica a plicatiei.

Controller leaga modelul de view.

In aplicatia curenta pachetul **business** contine clasele principale ale aplicatie si este o instantiere a obiectelor din lumea reala. Pachetul **data** contine clasele care contin metodele de input/output. Iar pachetul **presentation** este, la baza, pachetul ce contine clasele care creaza interfata cu care va interactiona persoana cu aplicatia, si leaga celelalte pachete cu clasele implicite. Poate fi asemanat cu componenta Controller + View din MVC.

Pachetul **Business:**

Diagrama UML:

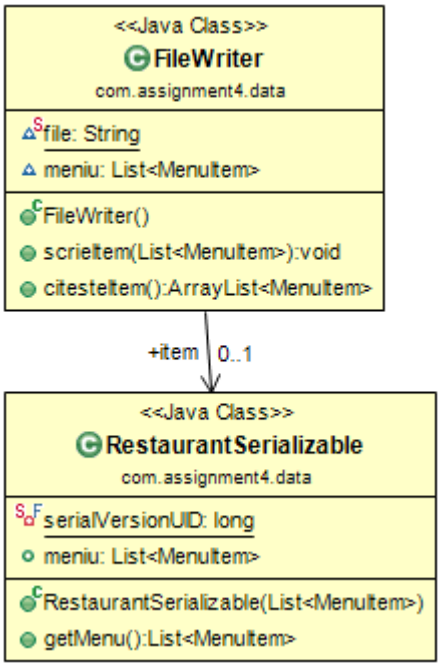


Aici se afla logica din spatele aplicatiei. Clasele **CompositeProduct** si **BaseProduct** sunt clase care exting clasa **MenuItem.** Acest lucru se datoreaza faptului ca intr-un meniu se pot afla aceste doua categorii de produse. Un produs de “baza” cum ar “Cartofi prajiti” sau un produs compus din mai multe ingrediente cum ar fi “Ciorba de cartofi”.

Clasa **Restaurant** implementeaza intergata ***IRestaurantProcessing***. Aceasta clasa este cea mai importanta clasa deoarece toata functionalitatea aplicatie se gaseste aici. In aceasta clasa se gasesc metodele ce reprezinta actiunea fiecarui user: administrator, ospatar sau chef bucatar. Tot aici se genereaza si facturile create de chelneri.

Pachetul **data**:

Diagrama UML:



Acest pachet contine clasele care stocheaza informatia procesata de aplicatie intr-un fisier binar numit “restaurant.ser”. Crearea bonurilor fiscale fiind realizate in totalitate de metoda `public void generateBill(Order order)` din clasa Restaurant().

Pachetul **presentation**:

Diagrama UML: →

Acest pachet contine clasele ce creaza interfata grafica prin intermediul careia se va interactiona cu acesata.

Pentru fiecare responsabil exista cate un panou de control:

-administrator

-ospatar

-chef bucatar

Toate aceste clase extind clasa JFrame pentru creearea Interfetei fiacre clasa reprezentand un Frame specific.

4.Principii OOP

Serializarea

Pentru salvarea datelor, Java permite un mecanism avansat pentru acest lucru si anume serializarea obiectelor. Acest lucru reprezinta salvarea si restaurarea starii obiectelor. Obiectele oricarei clase care implementeaza intergata Serializable, pot fi salvate intr-un

stream si restaurate din acesta. Pachetul java.io contine 2 clase speciale pentru serializarea tipurilor primitive: ObjectInpuStream si respective ObjectOutputStream. Subclasele claselor serializabile vor fi si ele, la randul lor, serializabile. Daca un obiect este serializat, atunci orice alt obiect care contine o referinta la obiectul respective va fi el insusi serializat.

Interfata **public interface Serializable** nu declara metode sau atribute. Aceasta serveste doar pentru indentificarea obiectelor serializabile. Precum exista mecanismul de serializare, totodata exista si mecanismul invers, cel de deserializare. Acest mecanism presupune ca clasa obiectului poseda si un constructor cu vizibiliatate publica si fara argumente. Atributele fiind initializate dintr-un flux de date.

Instantierea

Un obiect care apartine unei clase se numeste instanta a clasei.

Polimorfismul

Permite ca aceasi operatie sa se realizeze in mod diferit in subclase diferite.

Avantajele acestei capacitati este ca putem trata colectii de obiecte ale aceleiasi ierarhii intr-o maniera unitara deoarece functia apelata este indentificata la rulare cu functia membra din clasa careia ii apartine obiectului si ca putem asigura independenta implementarilor.

Mostenirea

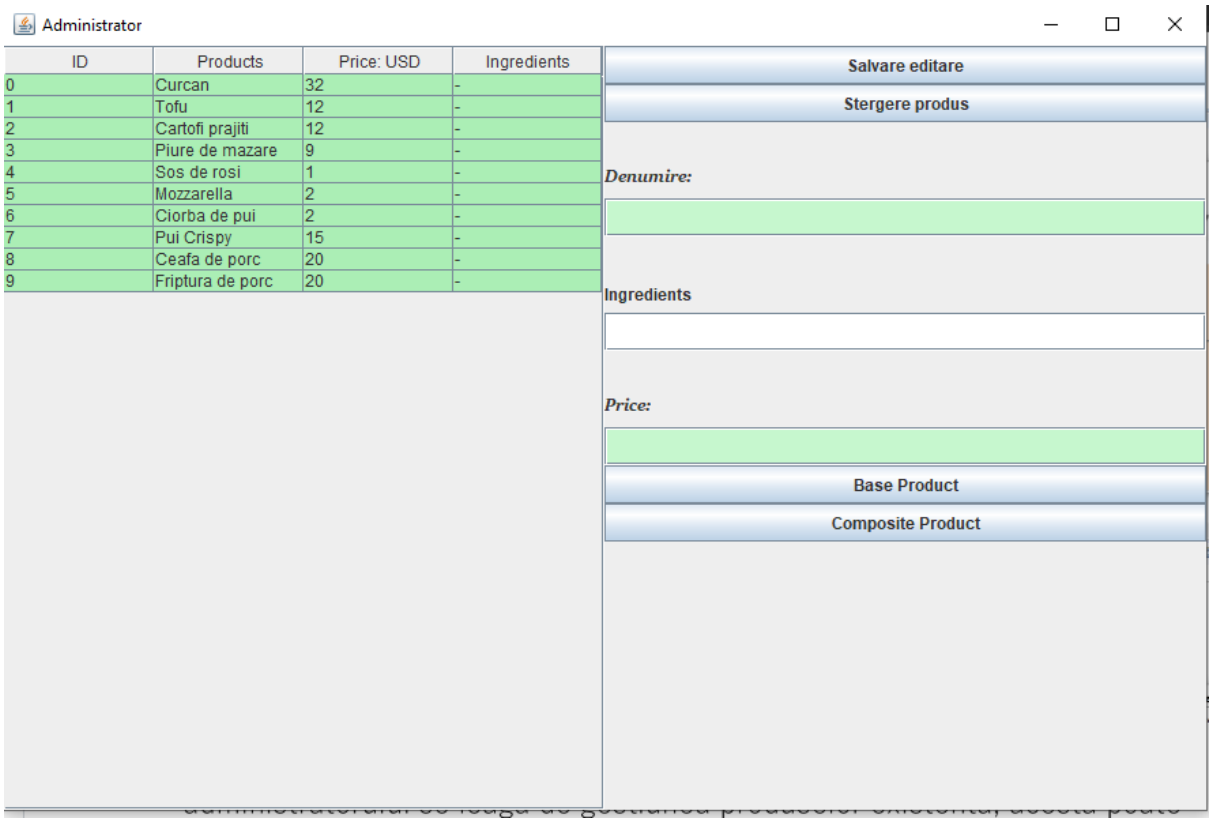
Reprezinta mecanismul prin care o clasa preia structura si comportamentul unei alte clase care adauga elemente specifice.

Clasa de baza reprezinta clasa de la care se preia structura si comportamentul iar clasa derivata este clasa care preia structura si comportamentul.

Avantajele mecanismului sunt date de posilibilitatea reutilizarii codului, obtinerea exstensiei unei clase fara a fi necesara o recompilare a clasei initiale, si utilizarea polimorfismului in timpul executiei programului prin folosirea functiilor virtuale.

5.Implementare

Interfata cu utilizatorul:



Interfata pentru administrator pune la dispozitie genstionarea produselor din meniu. Astfel acesta are obligatia de a adauga, sterge sau edita un produs.

Pentru a adauga un produ acesta trebuie sa scrie numele produsului in campul “Denumire” si pretul acestuia “Price” in cazul in care doreste un produs “Base” in cazul in care doreste un produs care contine mai multe ingrediente in compozita sa trebuie sa adauge si ingredientele in campul “Ingredients” sub forma “ingredient1, ingredient2, …”.

Acestea vor fi adaugate in tabelul din stanga si totodata vor fi salvate in fisierul “restaurant.ser”.

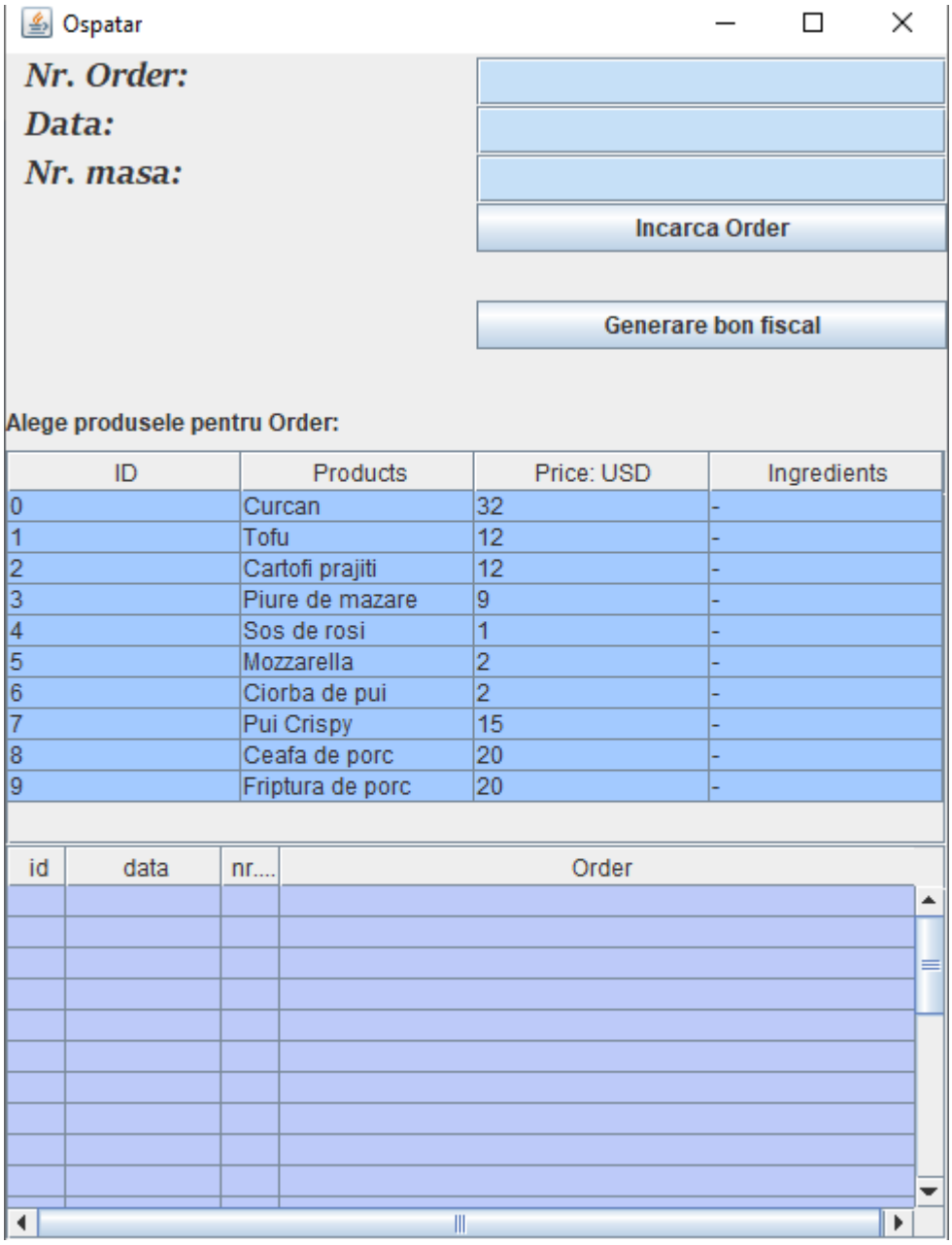
Pentru a sterge un produs, administratorul trebuie sa selecteze cu Mouse-ul randul dorit pentru a fi sters apoi sa dea clik pe butonul “Sterge produs”.

Editarea produslui consta in schimbarea numelui sau al pretului unui produs. Acest lucru se face dand dublu-click pe numele sau pretul produsului, se modifica campul, apoi se

apasa pe butonul “Salvare editare”. Acesta actiune lucreaza in felul urmator: se va sterge produsul din Lista de produse, si se va reintroduce produsul editat!

Adugarea se face cu ajutorul metodelor `addBaseProductButtonListener()` si `addCompositeProductButtonListener()` care contin metoda suprascrisa `actionPerformed(ActionEvent e)` responsabila de executarea actiuniilor specificarte mai sus. Acesta va adauga in un nou BaseProduct sau CompositeProduct si va scrie in fiserul binar noul produs.

Acelaeasi principii sunt folosite si cand vine vorba de stergere sau editare.



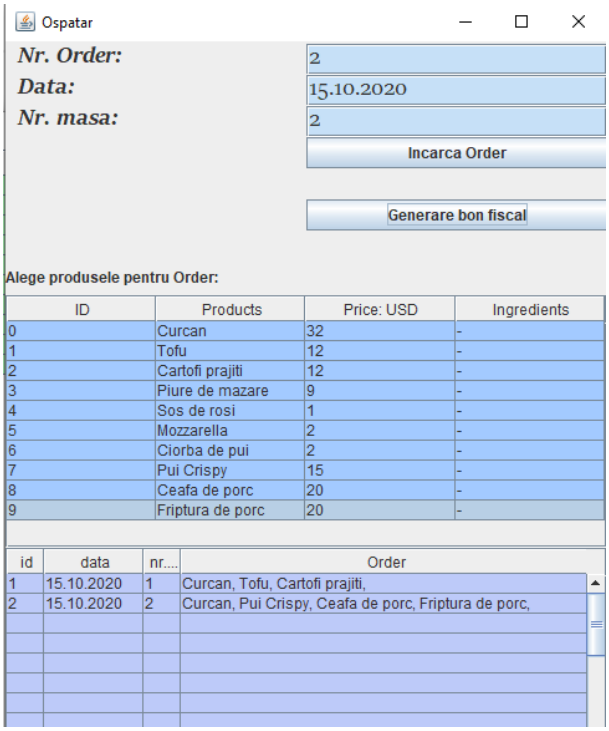
Interfata chelnerului contine in primul rand tabelul de produse al administratorului. Prima problema la aplicatia curenta este faptul ca aceasta nu se actualizeaza daca administratorul adauga sau sterge un produs. Pentru realizarea acestui lucru trebuie redeschisa aplicatia deoarece aceasta interfata creaza o copie a tabelului initial al administratorului.

Pentru ca ospatarul sa creeze o comada acesta trebuie sa introduca “Nr. Order”, “Data” si “Nr.Masa”. Apoi apasa pe butonul “Incarca Order” dupa care selecteaza, dand click cu Mouse-ul pe numele produselor necesare crearii comenzii. Pentru o noua comanda se va repeta actiunile anterioare. Butonul “Generare bon fiscal” va crea bon pentru ultima comanda creata, acest lucru putand fii vazut ca un dezavantaj al aplicatiei.

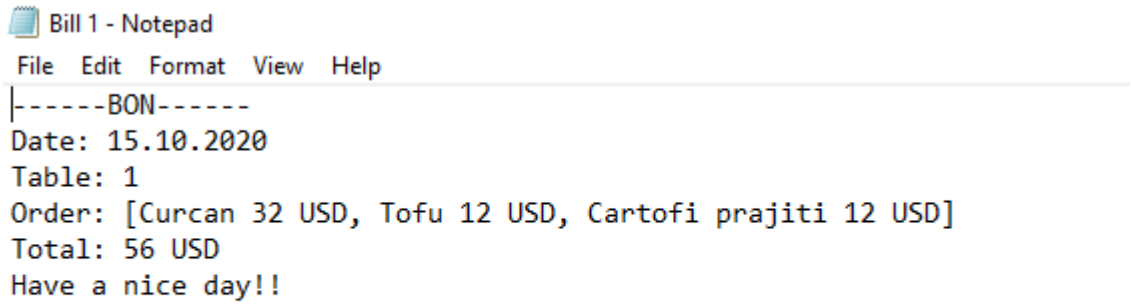
Aceasta este interfata bucatarului chef. In aplicatia curenta nu este functionala decat daca se foloseste un JDK < 9 deoarece am folosit pattern-ul predefinit al interfetei Observable care odata cu JDK 9 a fost `deprecation.`

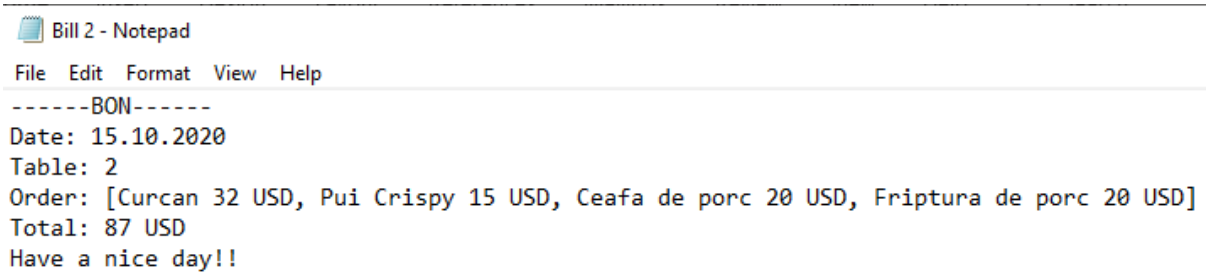
!! De asemenea daca se introduce un produs compus acesta se va scrie corect in tabel in schimb daca se redeschide aplicatia, acel produs compus isi va scrie ingredientele in coloanal “Products” in loc de coloana “Ingredients”.

6.Rezultate



Pentru comenzile de mai sus s-au generat urmatoarele bonuri:





Aceste bonuri sunt fisiere .txt care se salveaza in interiorul proiectului Java.

7.Concluzii

In urma acestei teme am ajuns sa inteleg mai bine anumite concepte OOP si modularizarea unui program.

Am invatat cum sa dezvolt clase pe baza sablonului Composite, si cum se foloseste paradigma Observable-Observer.

Se pot imbunatatii:

Eficienta interactiunii user-interface, prin actualizarea panoului “Ospatar” odata cu acutalizarea panoului de produse a administratorului.

Compunerea sablonului Observable-Observer astfel incat sa nu fie limitata de versiunea JDK-ului.

Sa se genereze bon fiscal pentru toate comenzile nu doar pentru ultima comanda procesata.

8.Bibliografie

- https://docs.oracle.com/javase/7/docs/api/java/util/Observable.html
- https://javarevisited.blogspot.com/2011/02/how-hashmap-works-in-java.html
- https://www.google.ro/search?q=Observable+Java&hl=ro&source=lnms&tbm=isch&sa=X&ved=2ahUKEwiwuoax-bDpAhUMZxUIHVV9DekQ_AUoAXoECA0QAw&biw=1366&bih=647
- https://www.tutorialspoint.com/java/java_serialization.htm
- https://www.baeldung.com/java-serialization
- https://www.geeksforgeeks.org/serialization-in-java/
- https://docs.oracle.com/javase/8/docs/api/java/io/Serializable.html
- http://inf.ucv.ro/~mihaiug/courses/poo/slides/Curs%2012%20-%20Sabloane%20de%20Proiectare.pdf