

**FACULTATEA: Automatica si Calculatoare**

**SPECIALIZAREA: Calculatoare si tehnologia informatiei**

**DISCIPLINA: Tehnici de Programare**

**AN UNIVERSITAR: 2019/2020**

**PROIECT: Restaurant Management System**

**Realizat de :**

Ghiura Darius

**Grupa:** 30227

**CUPRINS**

1. **Obiectivul temei**
2. **Analiza problemei, modelare, scenarii, cazuri de utilizare**
3. **Proiectare**
4. **Implementare**
5. **Rezultate**
6. **Concluzii**
7. **Bibliografie**
8. Obiectivul temei

Obiectivul temei este proiectarea si implementarea unui sistem de management pentru un restaurant. Sistemul va avea 3 tipuri de utilizatori : administrator, ospatar si bucatar. Fiecare utilizator va putea efectua anumite operatii :

* Administrator : inserarea, stergerea si modificarea unui produs din meniul restaurantului;
* Ospatar : crearea unei comenzi si generarea notei de plata;
* Bucatar : este notificat de fiecare data cand o comanda este plasata.

Obiective secundare :

* Folosirea Composite Design Pattern si Design by contract
* Serializarea obiectului Restaurant

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

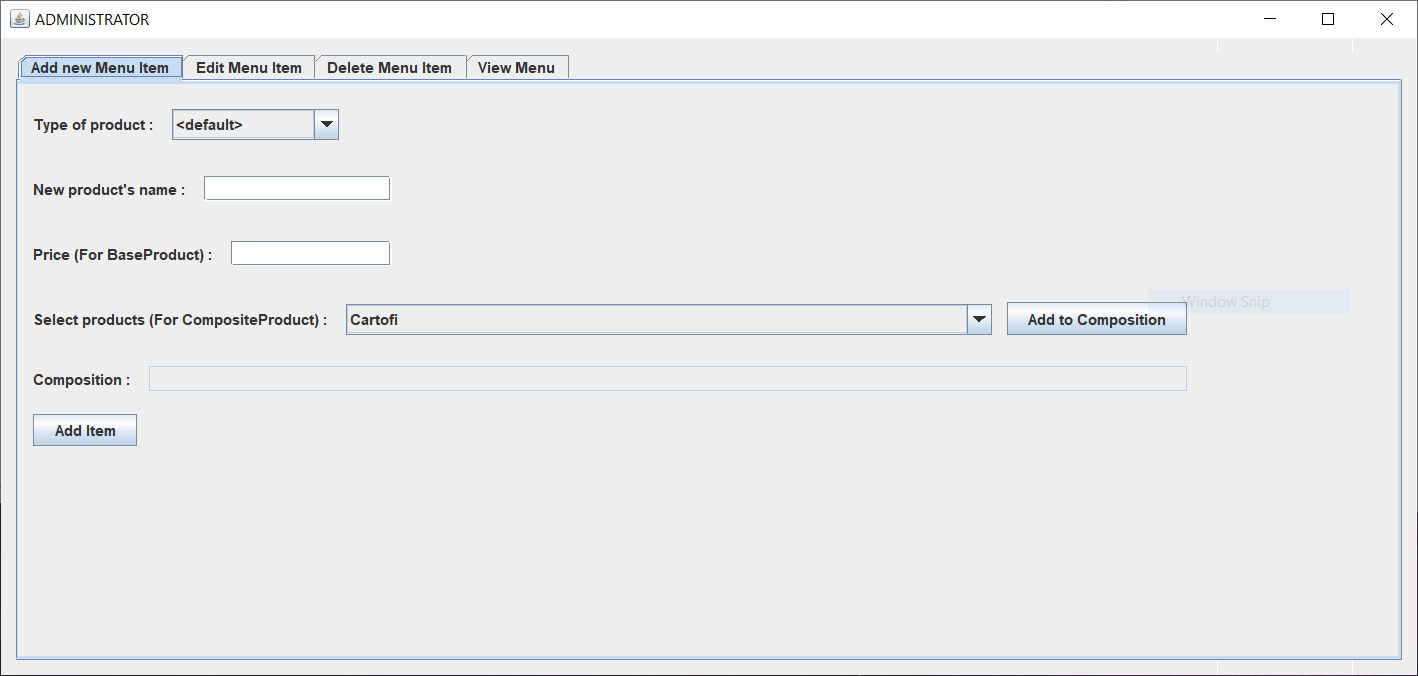
Un restaurant are un meniu si o lista de comenzi. Meniul restaurantului este compus din produse care pot fi BaseProduct sau CompositeProduct. Un CompositeProduct este un produs alcatuit din mai multe produse care pot fi tot Base sau Composite. Fiecare comanda are asociata o masa si o lista de produse.

Pentru a executa actiunile utilizatorilor, fiecare va avea o interfata grafica.

Aplicatia se ruleaza prin executarea urmatoarei comenzi in consola : ***java -jar PT2020\_Group\_LastName\_FirstName\_Assignment\_4.jar*** ***restaurant.ser***

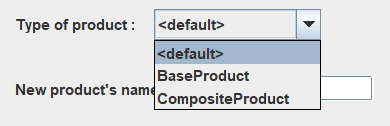
La rularea aplicatiei vor porni cele 3 ferestre.

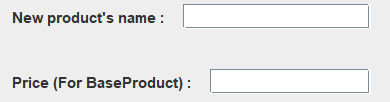
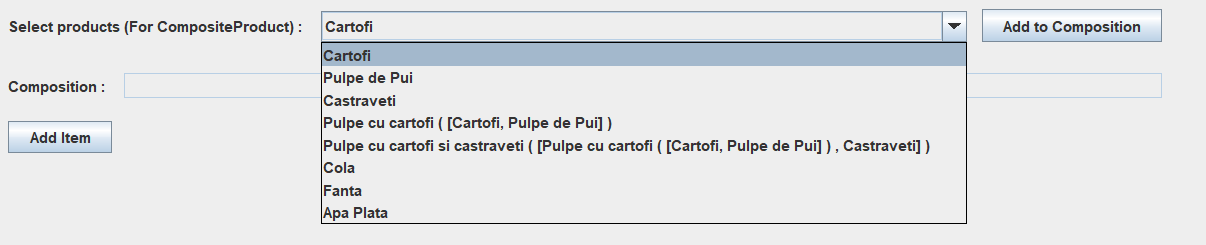
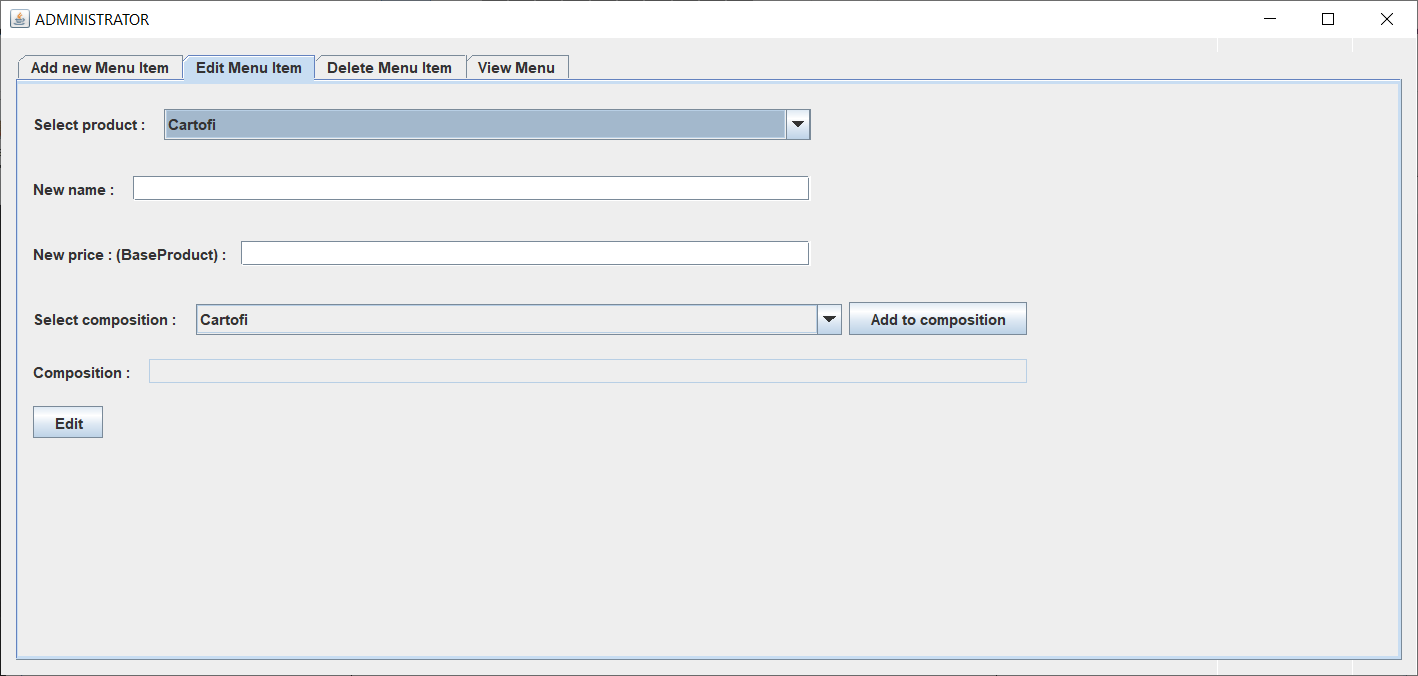
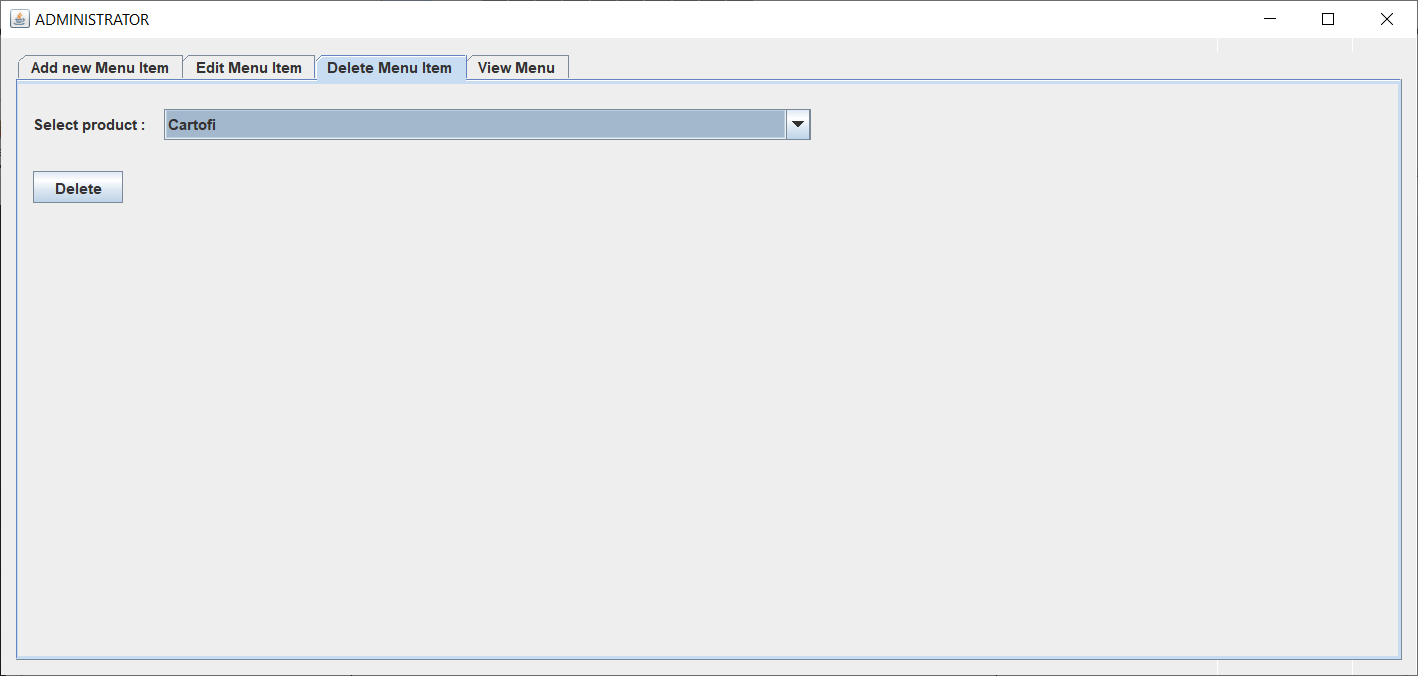
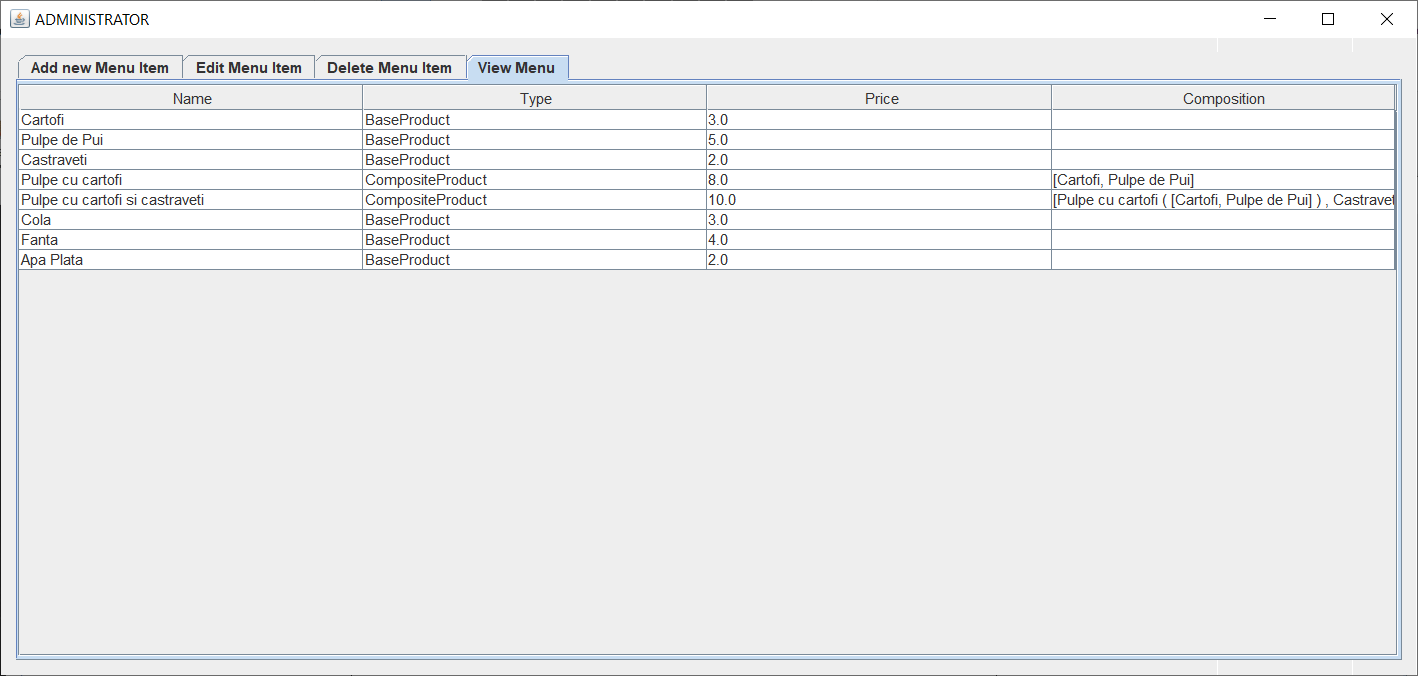
**Cazuri de utilizare :**

 **Administrator**

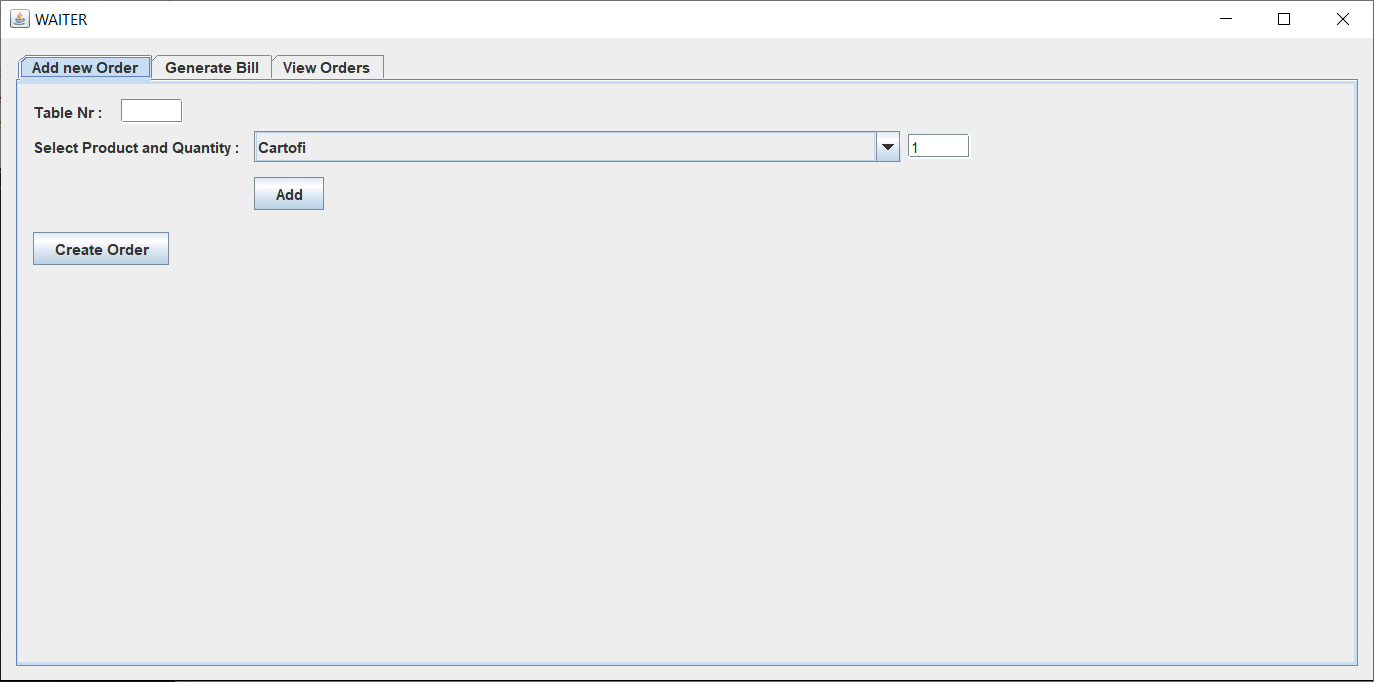
Pentru selectarea operatiei de executat se selecteaza unul dintre tab-urile „Add new Menu Item”, „Edit Menu Item”, „Delete Menu Item”, „View Menu”.

* **Adaugarea unui produs la meniul restaurantului**

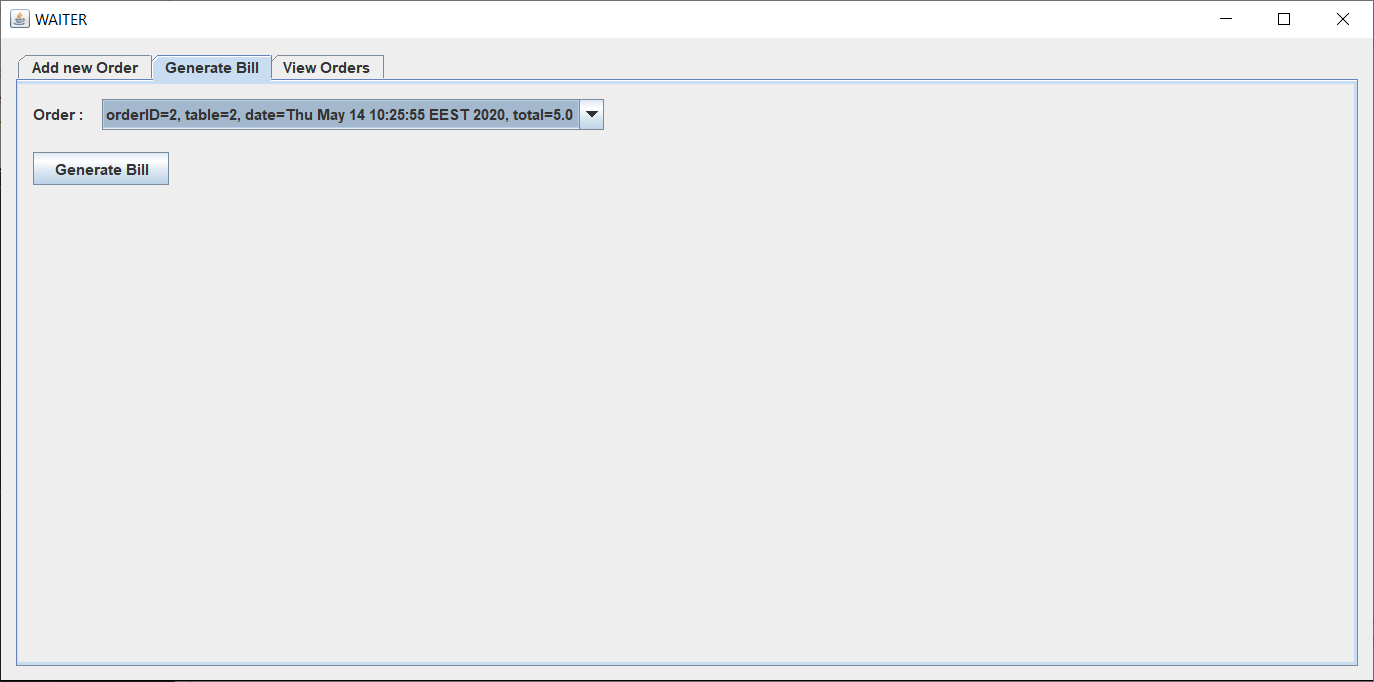
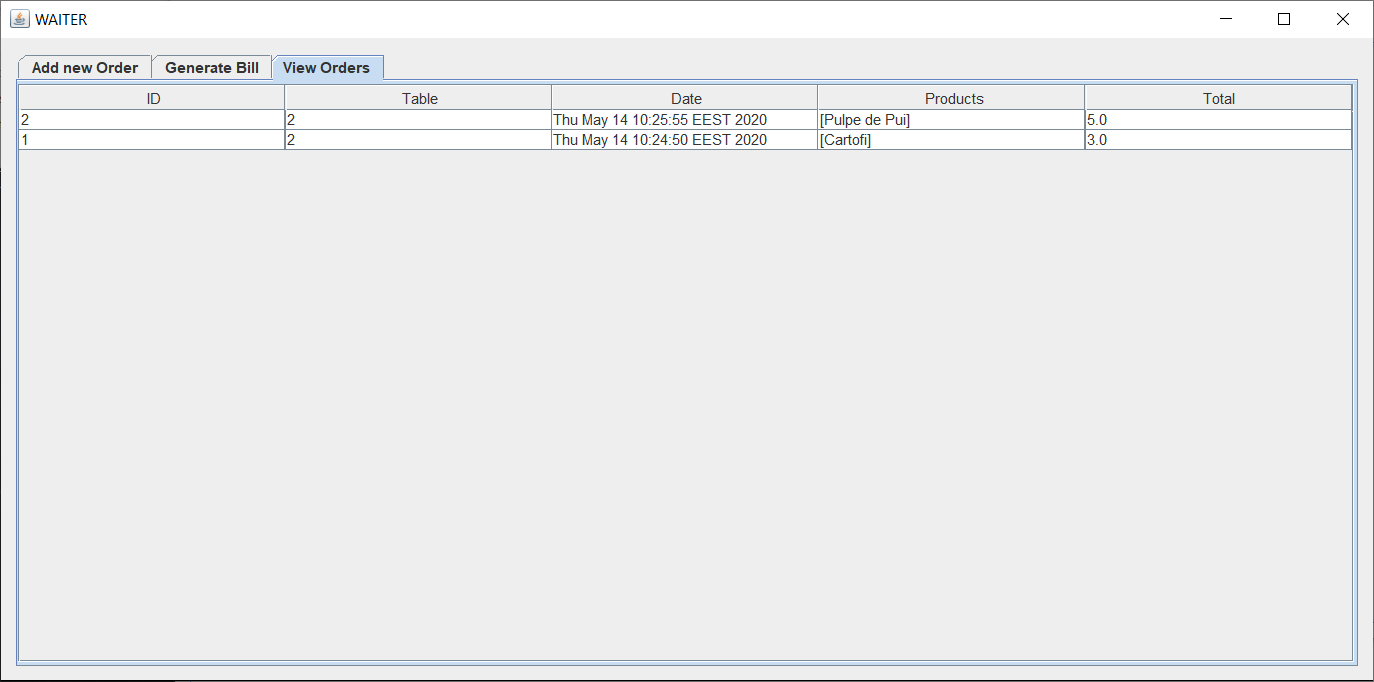
Pentru adaugarea unui produs, in primul rand se selecteaza tipul produsului :

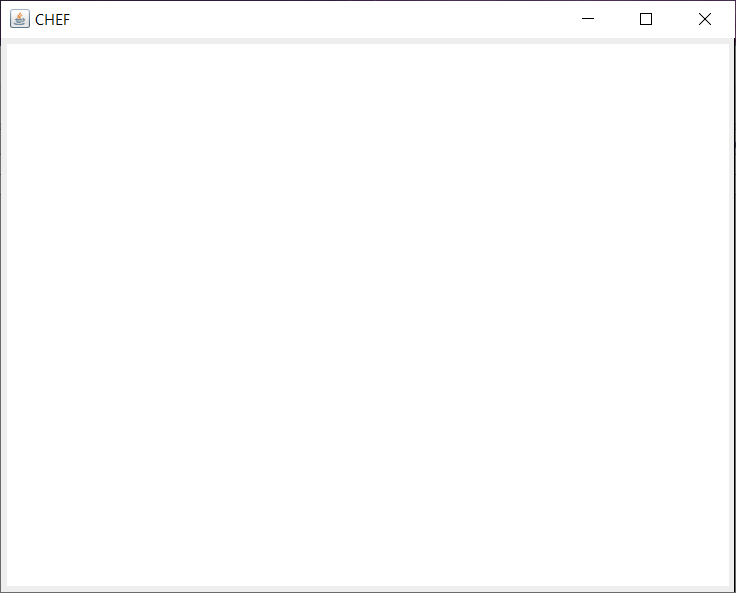
* Daca produsul de agaudat este BaseProduct :
  + Se insereaza numele produsului si pretul
  + Se apasa pe Add Item
* Daca este CompositeProduct :
  + Se insereaza numele produsului
  + Se alege un produs de adaugat in compozitia lui si se apasa pe butonul Add To Composition
  + Se repeta pasul anterior pentru fiecare produs din compozitie
  + Se apasa pe Add Item
* **Editarea unui produs existent in meniul restaurantului**
  + Se selecteaza produsul care trebuie modificat.
  + Se insereaza campurile care trebuie modificate (Nume si/sau pret pentru BaseProduct)
  + Pentru CompositeProduct se adauga produse in lista de compozitie ca la Add
  + Se apasa pe Edit
* **Stergerea unui produs din meniul restaurantului**
  + Se alege un produs din lista si se apasa pe Delete
* **Vizualizarea meniului**

**Ospatar**

****

Pentru selectarea operatiei de executat se selecteaza unul dintre tab-urile „Add new Order”, „Generate Bill”, „View Orders”.

* **Crearea unei comenzi**
  + Se insereaza numarul mesei
  + Se alege un produs din meniu si se insereaza cantitatea si se apasa pe Add
  + Se repeta pasul anterior pentru toate produsele comandate
  + Se apasa pe Create Order
* **Generearea unei facturi**
  + Se selecteaza comanda pentru care sa se genereze facture
  + Se apasa pe Generate Bill
* **Vizualizarea comenzilor**

**Bucatar**

La crearea unei comenzi, aceasta va aparea pe interfata bucatarului.

1. Proiectare

Proiectul este impartit in 4 pachete :

* BusinessLayer : Contine logica aplicatiei si clasele pentru restaurant, produse si comenzi
* DataLayer : Contine clasele care interactioneaza cu fisiere : BillGenerator si RestaurantSerializator
* MainClass : Contine metoda main
* PresentationLayer : contine 3 pachete in care sunt interfetele grafice si controller-ele lor

Clasele MenuItem, BaseProduct si CompositeProduct au fost construite folosind Composite Design Pattern. Clasa MenuItem este abstracta si are campurile pentru numele si pretul produsului si are metoda computePrice. Clasele BaseProduct si CompositeProduct extind clasa MenuItem si implementeaza metoda din clasa parinte.

Clasa CompositeProduct are in plus ca variabila instanta unArrayList<MenuItem>, deoarece un CompositeProduct poate fi compus atat din BaseProduct-uri cat si din alte CompositeProduct-uri.

Clasa Restaurant si interfata pe care o implementeaza, care contine operatiile de efectuat pe meniu si comenzi, sunt implementate folosind Design By Contract. Aceasta metoda presupune folosirea assert-urilor pentru a verifica preconditii si postconditii, astfel asigurand ca metoda se executa corect. Exista si un invariant, care este o conditie care poate fi folosita ca pre si post conditie. Metoda wellFormed este o metoda care trebuie sa returneze true tot timpul pentru a asigura construirea corecta a clasei Restaurant. Preconditiile si postconditiile sunt inserate ca si custom tag-uri in JavaDoc

Interfata grafica pentru Bucatar este implementat folosind Observer Desing Pattern. Acest pattern presupune folosirea unei interfete Subject care reprezinta obiectul de observat si o interfata Observer care face ceva cand observa ca o anumita actiune este realizata de Subject. Astfel clasa Restaurant implementeaza interfata Subject si clasa Chef interfata Observer. De fiecare data cand o noua comanda este adaugata observer-ul este notificat, iar in interfata grafica de la Bucatar apare comanda plasata.

Pentru a salva starea Restaurantului cand se termina executia programului se foloseste Serializarea. Serializarea este procedeul prin care datele încapsulate în instanța unei clase sunt trimise și primite cu ajutorul unui flux de I/O. Clasele care realizează acest lucru sunt java.io.ObjectOutputStream și java.io.ObjectInputStream iar metodele cele mai utilizate sunt writeObject(Object) și readObject(). Urmatoarea data cand se porneste aplicatia obiectul care a fost serializat anterior va fi deserializat si folosit ca Restaurant, astfel meniul si comenzile raman salvate.

Serializarea obiectului se executa doar cand toate ferestrele de GUI au fost inchise, asa se asigura ca starea obiectului care se salveaza este cea dorita.

Pentru a serializa obiectul de tip Restaurant toate clasele variabilelor instanta, precum si clasa Restaurant, trebuie sa implementeze interfata Serializable

Pentru salvarea meniului restaurantului a fost folosit un ArrayList de MenuItem, iar pentru salvarea comenzilor a fost folosit un HashMap<Order, ArrayList<MenuItem>>. Acest HashMap are ca si cheie un obiect de tipul Order, care are HashCode-ul facut dupa : OrderId, Table, Date. Fiecarui Order ii corespunde un ArrayList de MenuItem care reprezinta produsele comandate.

Interfetele grafice folosesc ComboBox-uri pentru selectarea produselor si a comenzilor, acestea fiind capabile sa retina referinta catre un obiect.

Se folosesc TextField-uri pentru introducerea datelor despre produse sau comenzi.

Pentru afisarea meniului si comenzilor se folosesc JTable-uri

**Diagrama UML**

<https://imgur.com/ZuWpR45>

1. Implementare

**Pachetul BusinessLayer :**

**Clasa : Order**

**Variabile instanta :**

* **orderID**
* **table**
* **date –** Data si ora cand a fost plasata comanda
* **total –** Pretul total al comenzii
* **serialVersionUID –** Folosita la serializare

Implementeaza interfata Serializable, suprascrie metodele de equals si hashCode care se folosesc la HashMap-ul cu comenzi. HashCode-ul este calculat folosind OrderId, Table si Date.

**Clasa : MenuItem**

**Variabile instanta :**

* **name**
* **price**
* **serialVersionUID**

Clasa abstracta folosita pentru construirea produselor din meniu cu Composite Design Pattern

**Clasa : BaseProduct**

**Variabila instanta :**

**serialVersionUID**

Extinde MenuItem, reprezinta un produs de baza care are doar nume si pret

**Clasa : CompositeProduct**

**Variabile instanta :**

* **products –** Lista care contine compozitia produsului
* **serialVersionUID**

Extinde MenuItem, reprezinta un produs care este compus din alte produse. Pretul este calculat adunand pretul produselor din lista de compozitie.

**Interfata : IRestaurantProcessing**

Contine operatiile care trebuie implementate de Restaurant. Contine custom tag-uri de JavaDoc care arata pre si post conditiile pentru fiecare metoda.

**Interfata : Subject**

Interfata care face parte din Observer Design Pattern. Contine metode pentru adaugarea, stergea unui Observer si notificarea Observer-ului cand se intampla un anumit eveniment.

**Clasa : Restaurant**

**Variabile instanta :**

* **orders –** HashMap-ul care contine comenzile restaurantului
* **menu –** ArrayList cu produsele meniului
* **observerList –** ArrayList folosit in Observer Design Pattern, care contine obiectele care vor fi notificate daca in Restaurant se intampla un anumit eveniment
* **orderId**
* **serialVersionUID**

Clasa implementeaza interfetele IRestaurantProcessing, Subject si Serializable. Pentru fiecare metoda din IRestaurantProcessing se folosesc assert-uri la inceputul si la sfarsitului metodei, care asigura functionarea corecta.

Metoda createMenuItem adauga un MenuItem in meniu.

Metoda deleteMenuItem sterge obiectul primit ca parametru din lista de meniu, apoi parcurge lista si verifica daca un CompositeProduct contine produsul sters anterior in lista de compozitie. Daca da, adauga CompositeProduct-ul la o lista de elemente care trebuie sterse. La finalul parcurgerii, pentru fiecare obiect din lista de stergere se apeleaza din nou metoda de deleteMenuItem cu acel obiect trimis ca si parametru. Astfel daca se sterge un produs, toate produsele care contin acel produs in lista de compozitie vor fi sterse.

Metoda editMenuItem modifica :

* Numele si/sau pretul pentru un BaseProduct
* Numele si/sau compozitia pentru un CompositeProduct

La final metoda recalculeaza preturile pentru CompositeProduct-urile din meniu

Metoda createOrder introduce o comanda in HashMap, calculeaza pretul total al comenzii adunand pretul produselor si apeleaza metoda care notifica observatorii.

Metoda generateBill apeleaza metoda generateBill din clasa BillGenerator pentru o comanda.

Metoda wellFormed verifica daca fiecare comenzi din HashMap ii este asociata o lista de produse.

**Pachetul DataLayer :**

**Clasa : BillGenerator**

Metoda generateBill creeaza un fisier cu formatul bill\_orderId.txt.

O factura contine informatii despre comanda precum : Id, masa, data si ora, lista de produse comandate si totalul comenzii

**Clasa : RestaurantSerializator**

Metoda serialize primeste ca parametrii un obiect de tipul Restaurant si numele unui fisier. Realizeaza serializarea obiectului intr-un fisier cu numele primit.

Metoda deserialize returneaza un obiect de tipul Restaurant rezultat dupa deserializarea unui fisier care contine un obiect serializat.

**Pachetul DataLayer :**

**Pachetul Admin :**

**Clasa : AdministratorView**

Interfata grafica pentru Administrator. Extinde JFrame. Pe JFrame este pus un JTabbedPane. Fiecare tab are un JPanel, tab-ul pentru View Menu are un JScrollPane.

Metoda comboBoxInitializer introduce obiectele din meniul restaurantului in ComboBox-uri

Metoda updateCombo sterge continutul ComboBox-urilor si apeleaza comboBoxInitializer

Metoda tableSetter formeaza JTable-ul.

**Clasa : AdministratorController**

Contine clasele care implementeaza ActionListener.

DeleteListener : cand se apasa butonul de Delete se preia produsul selectat, se apeleaza metoda deleteMenuItem din Restaurant, se apeleaza metodele pentru a updata ComboBox-urile din Administrator si din Waiter.

TypeComboListener : cand se selecteaza tipului produsului pentru operatia de Add Menu item, daca tipul este BaseProduct : ComboBox-ul pentru selectarea compozitiei si butonul de Add to Composition sunt dezactivate, iar daca tipul este CompositeProduct se dezactiveaza textField-ul pentru introducerea pretului.

AddToCompositionListener : cand se apasa butonul de Add To Composition se adauga intr-o lista produsele din compozitia unui CompositeProduct

AddItemListener : cand se apasa butonul de Add se preiau informatiile din textField-uri, se apeleaza metoda createMenuItem din Restaurant si se rezeteaza ComboBox-urile si textField-ul de la compozitie.

EditItemListener : cand se apasa pe butonul de Edit se preiau informatiile din textField-uri, se apeleaza metoda editMenuItem din Restaurant si se rezeteaza ComboBox-urile si textField-ul de la compozitie.

**Pachetul Waiter :**

**Clasa : WaiterView**

Interfata grafica pentru ospatar. Extinde JFrame. Structura JFrame-ului este aceeasi cu cea de la AdministratorView.

Metoda productsComboBoxSetter seteaza ComboBox-ul pentru operatia de Create New Order de unde se aleg produsele.

Metoda orderComboBoxSetter seteaza ComboBox-ul pentru operatia de Generate Bill de unde se selecteaza comanda.

Metoda tableSetter formeaza JTable-ul.

**Clasa : WaiterController**

Contine clasele care implementeaza ActionListener.

GenerateListener : Apeleaza generateBill din Restaurant cu comanda selectata

AddListener : Cand se apasa pe butonul Add adauga la lista de produse comandate \* cantitatea introdusa.

CreateOrderListener : Cand se apasa pe Create Order se apeleaza metoda createOrder din Restaurant.

**Pachetul Chef :**

**Interfata : Observer**

Face parte din Observer Design Pattern, contine metoda de implementat update

**Clasa : ChefGraphicalUserInterface**

Implementeaza interfata Observer. Interfata grafica contine un JTextArea in care sunt afisate comenzile cand sunt adaugate.

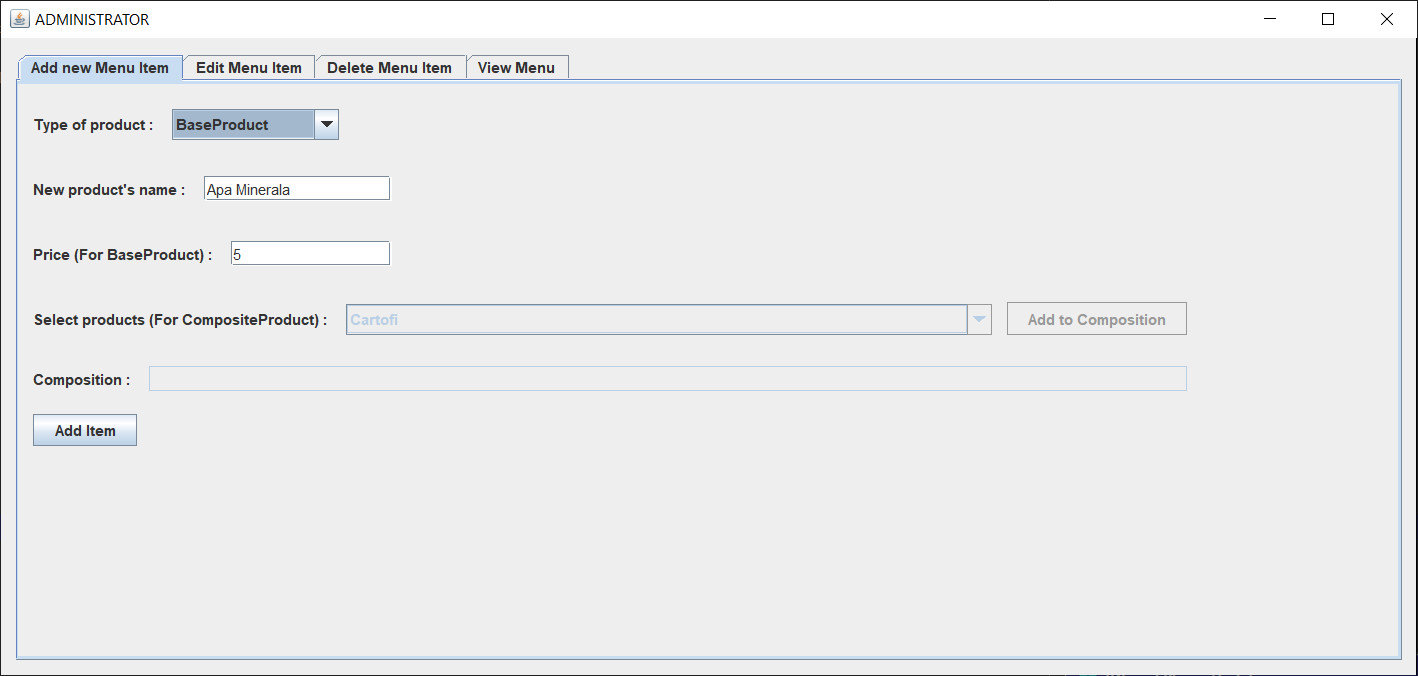
**Pachetul MainClass :**

**Clasa : MainClass**

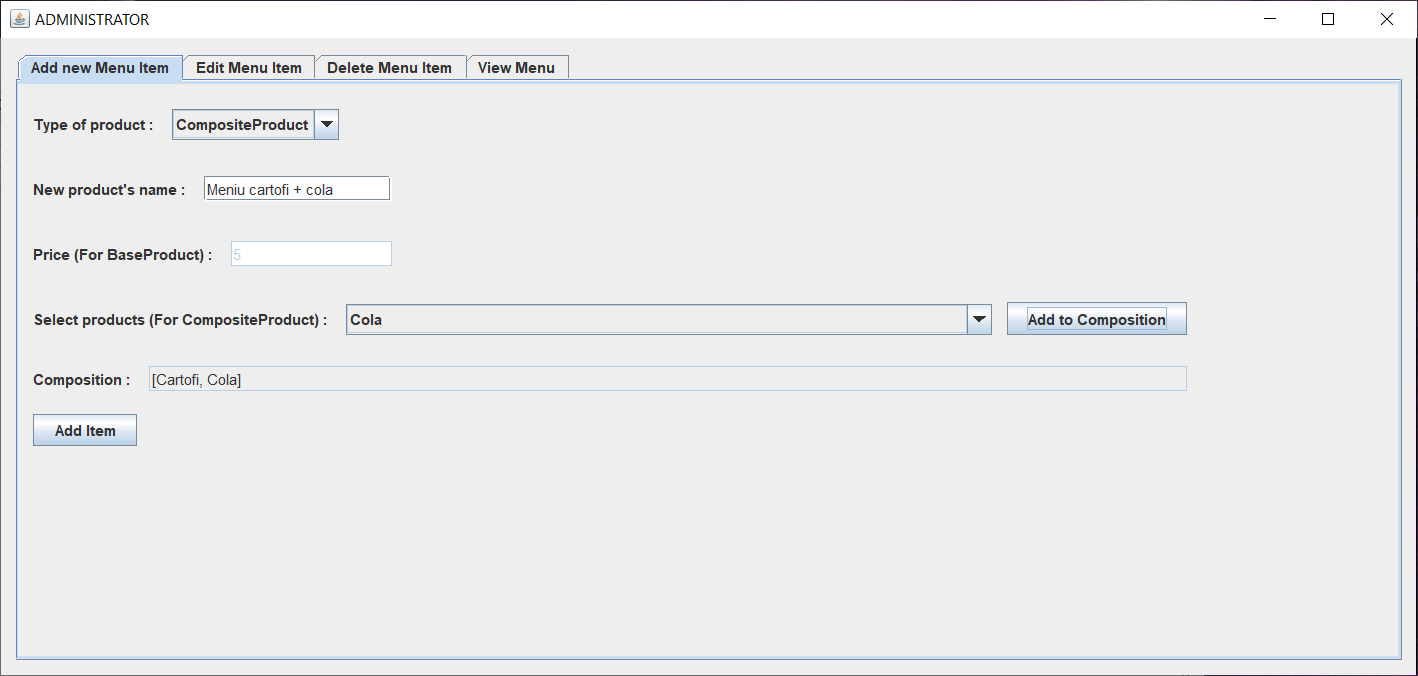
Contine metoda main a proiectului. La inceputul metodei obiectul de tip Restaurant este initializat folosind deserializarea. Pentru a verifica daca interfetele grafice sunt inca vizibile se foloseste un while in care se verifica metodele isDisplayable ale JFrame-urilor. Cand se iese din while se face serializarea si apoi aplicatia isi opreste executia.

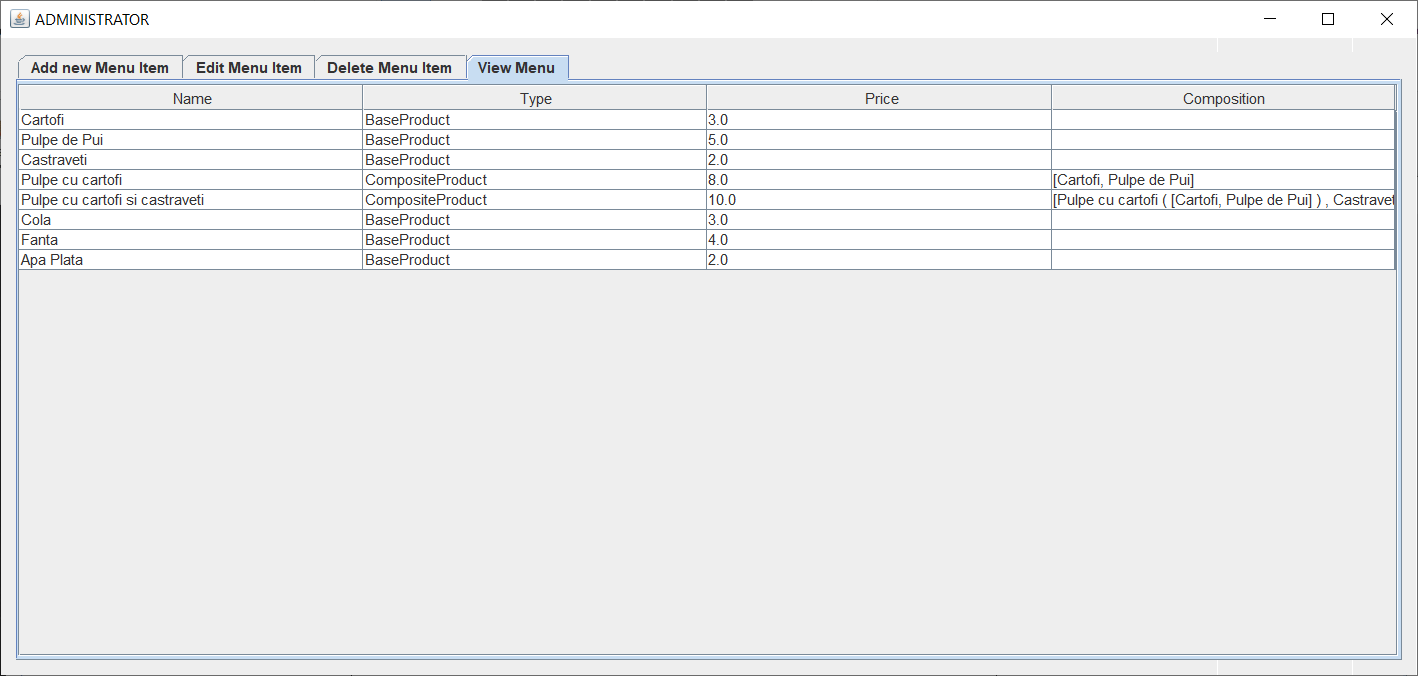
1. Rezultate

Aplicatia a fost testata prin executarea operatiilor din interfetele grafice si verificand rezultatele din tabelele pentru meniu si comenzi, precum si prin verificarea fisierelor .txt generate, reprezentand facturile comenzilor.

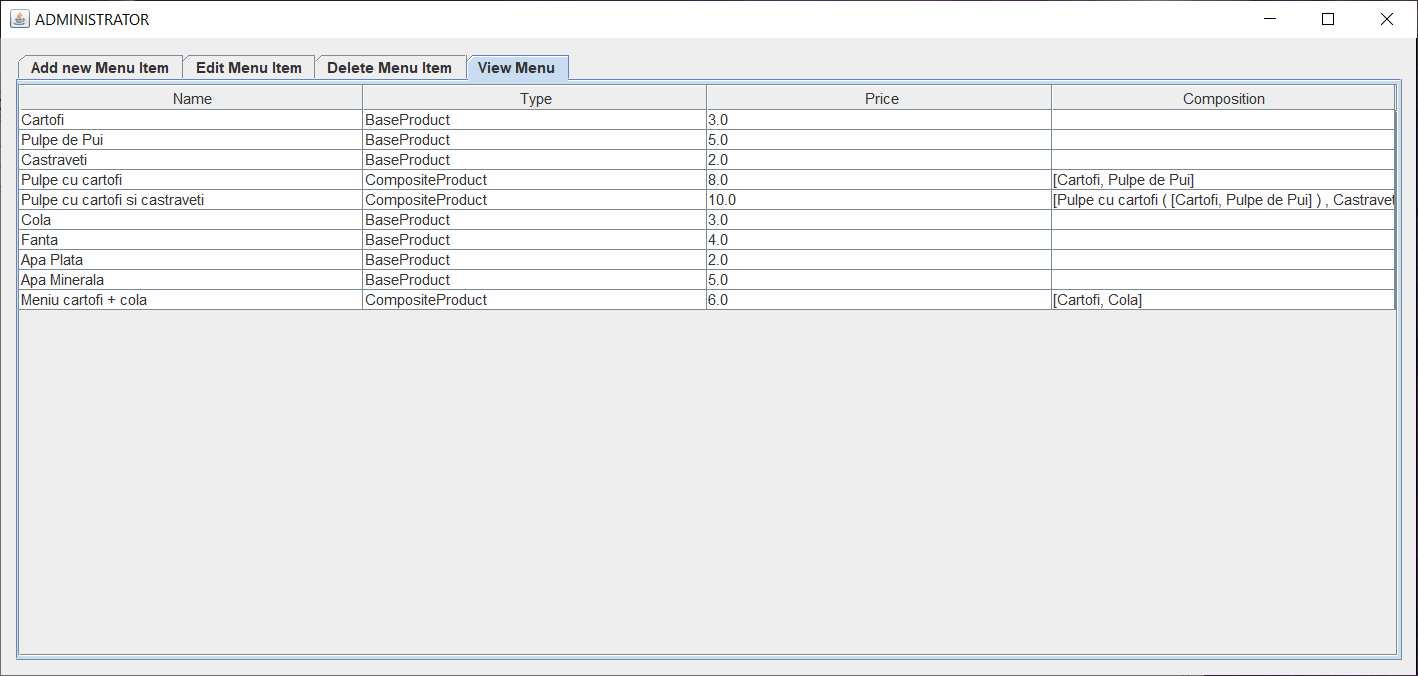
Inserarea unui BaseProduct :

Inserarea unui CompositeProduct :

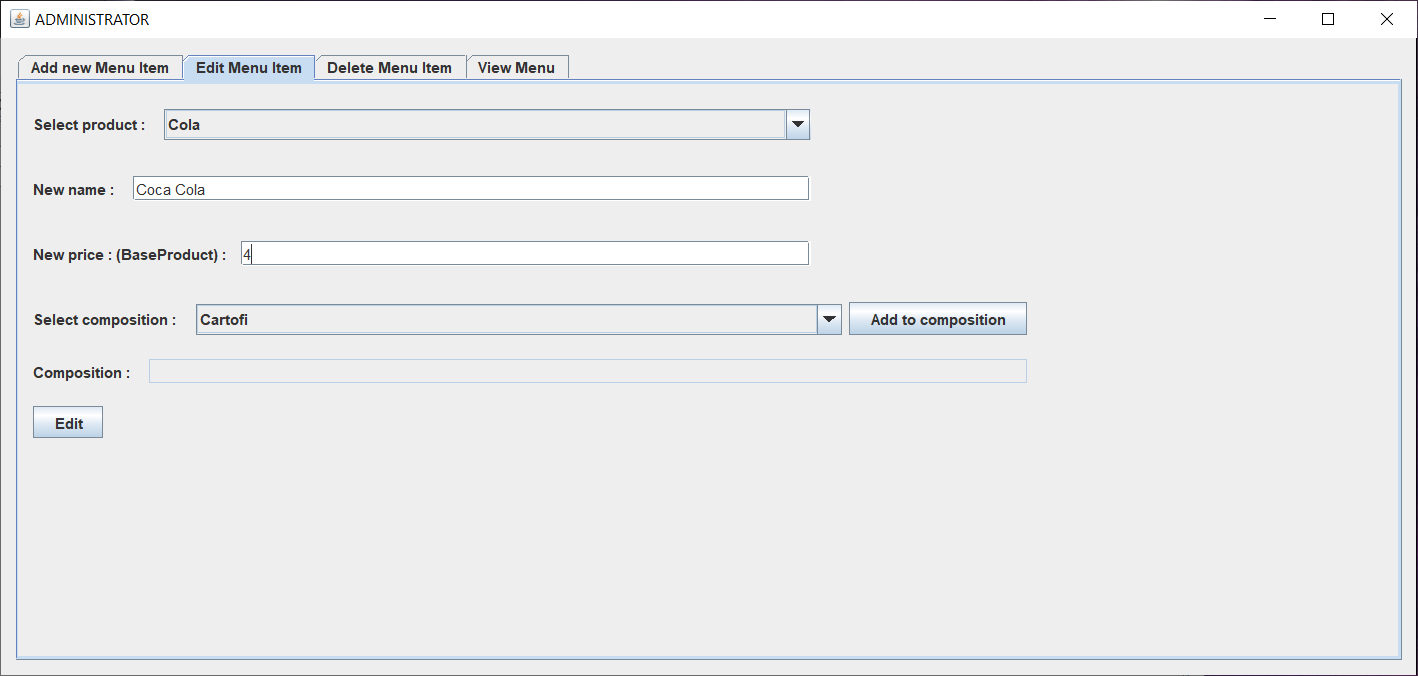


****Meniul inainte de inserari :

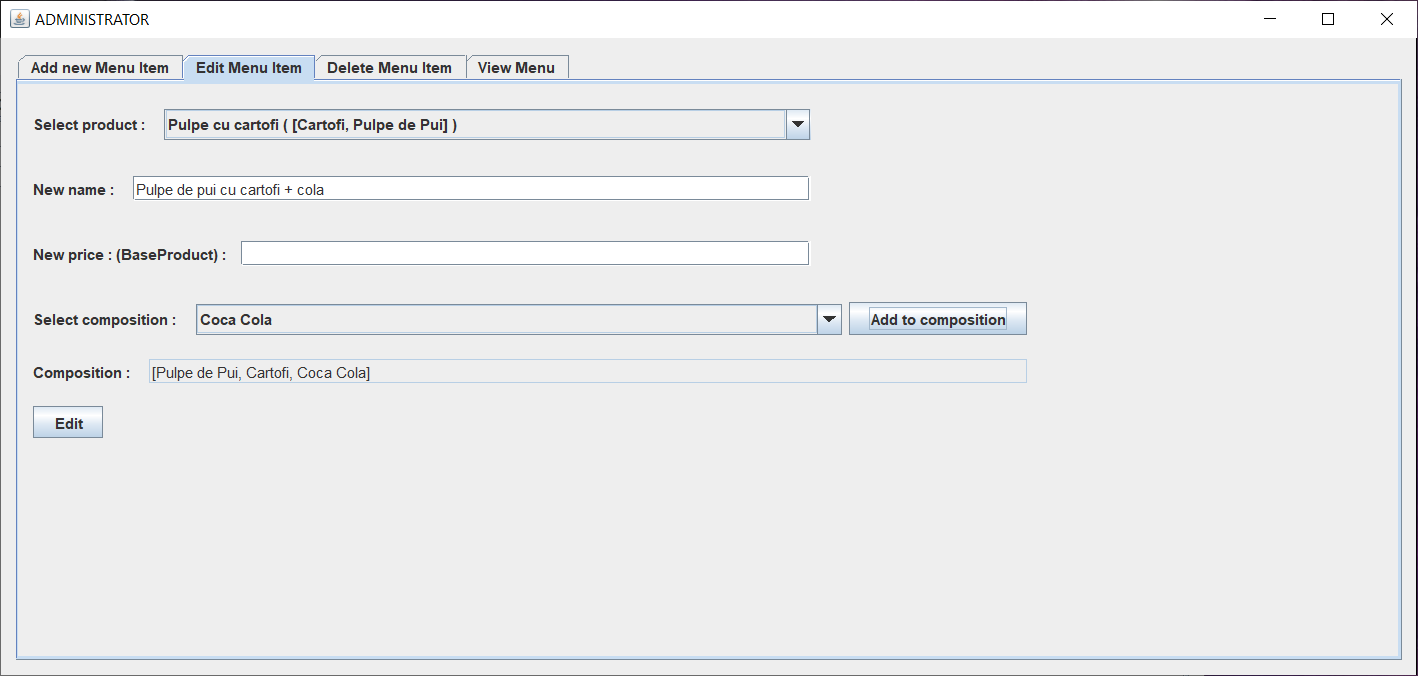
Meniul dupa inserari :



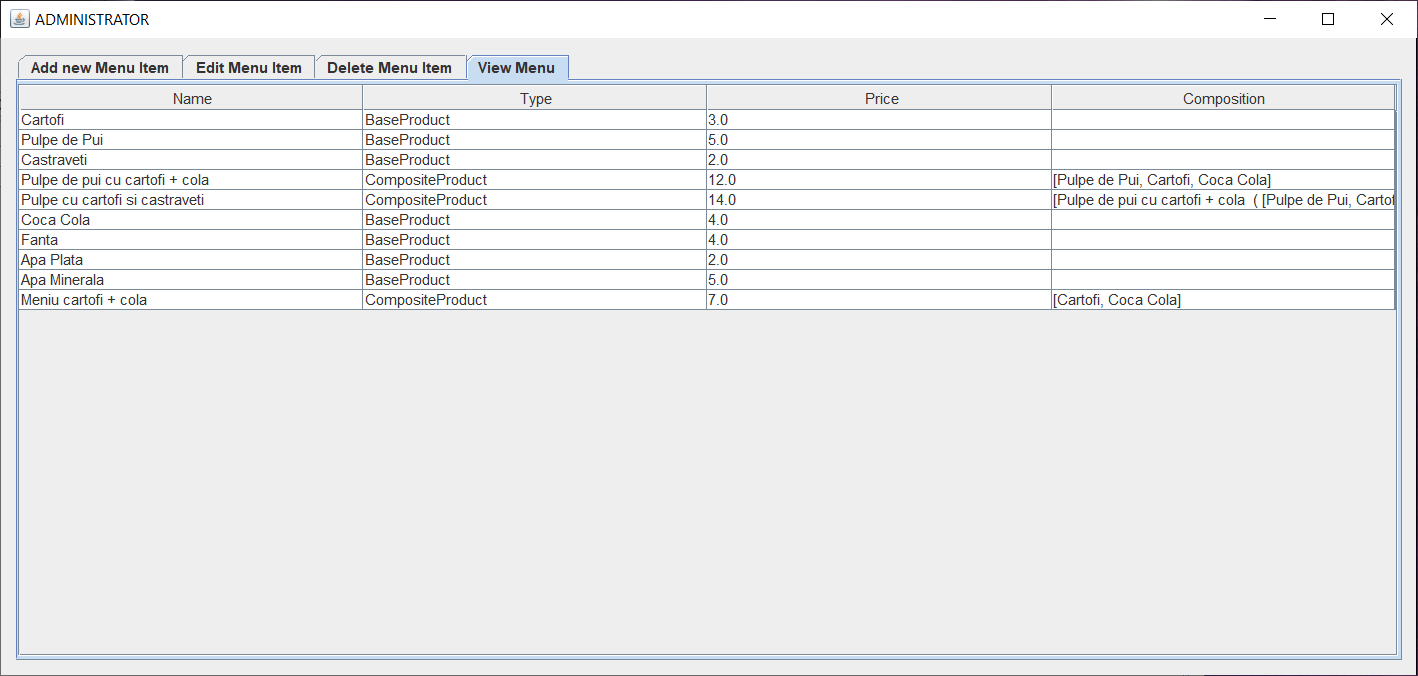
Editarea unui BaseProduct :



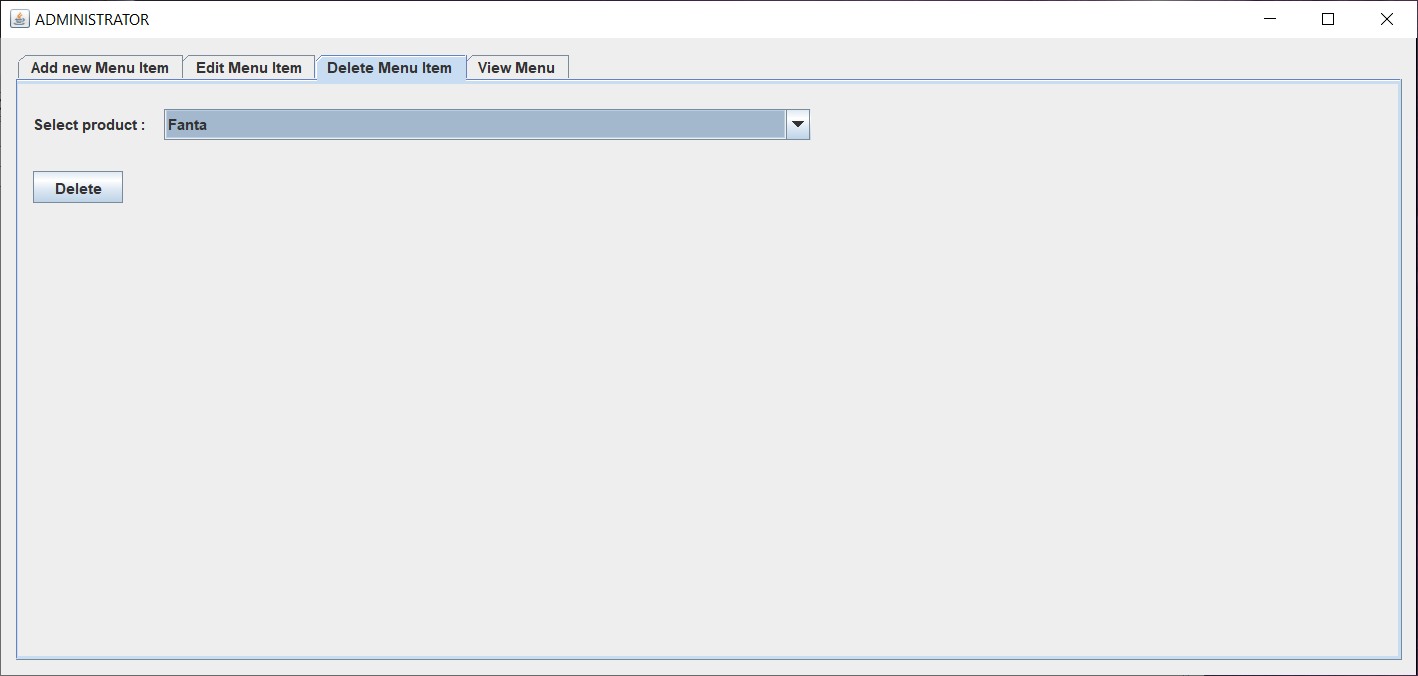
Editarea unui CompositeProduct :



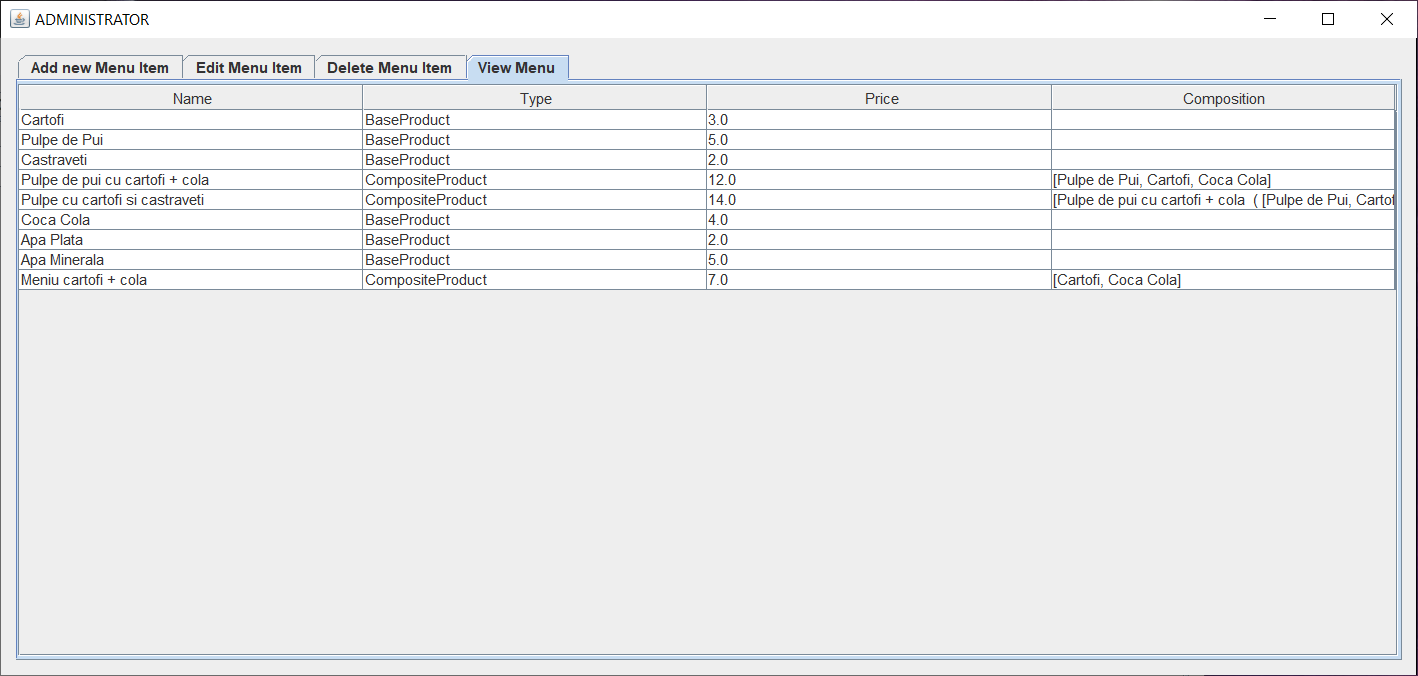
Meniu dupa editari :



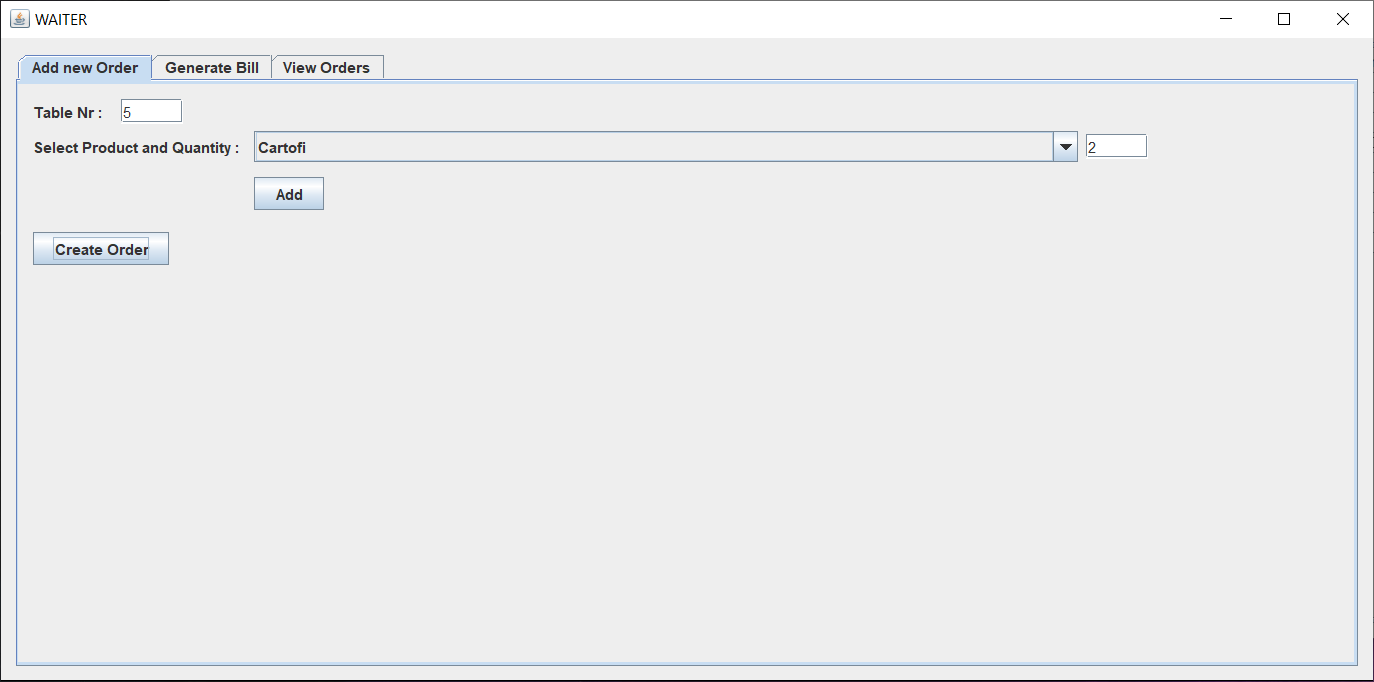
Stergerea unui produs :



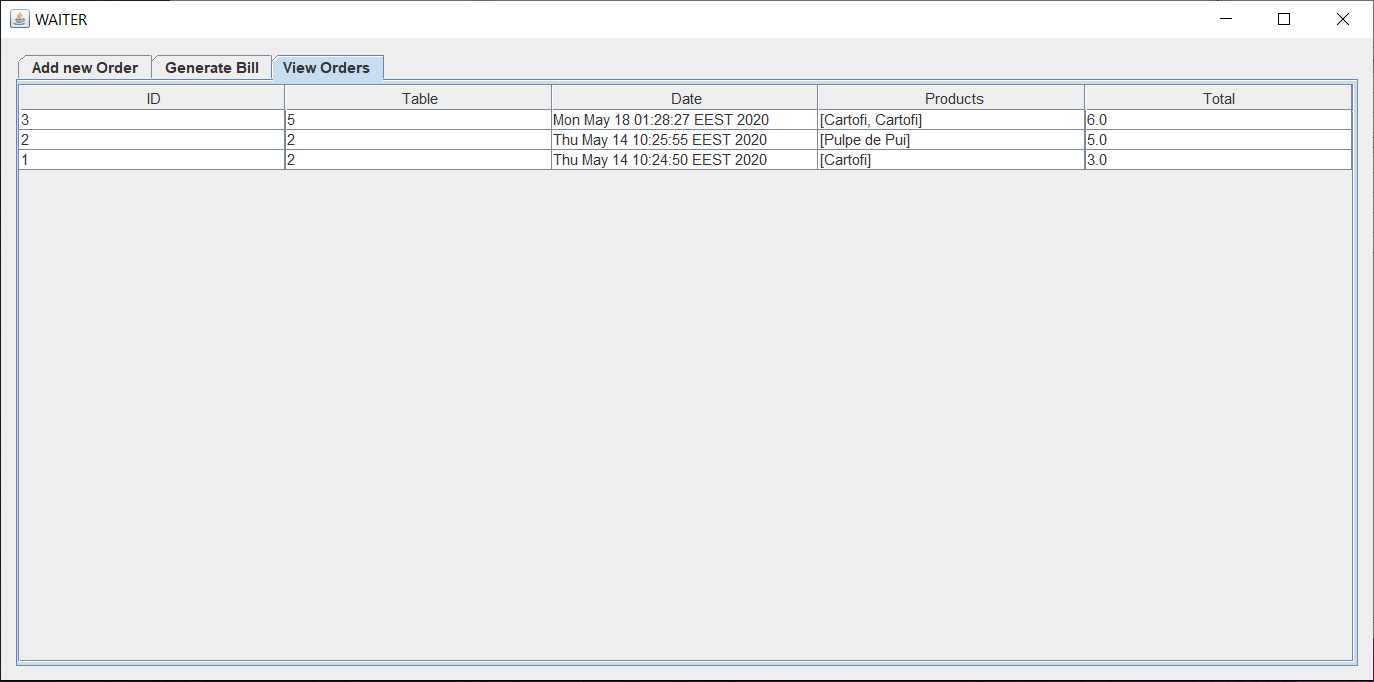
Meniu dupa stergere :



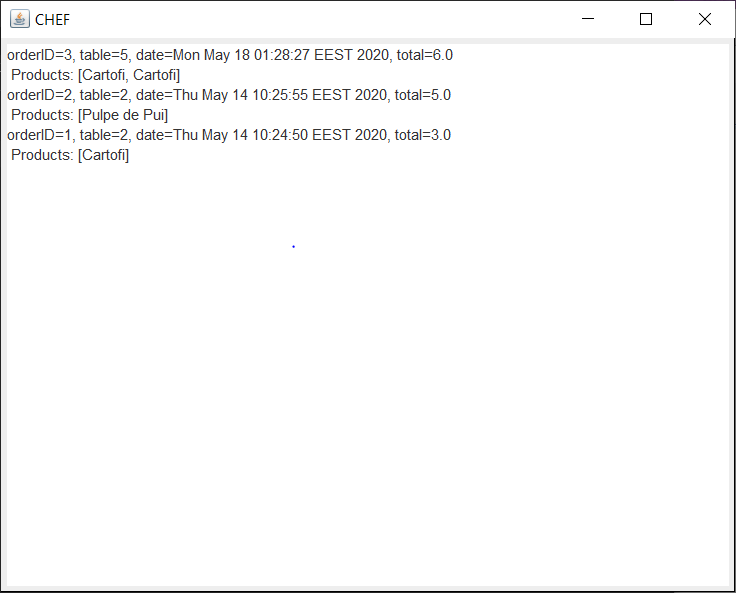
Creare unei comenzi :



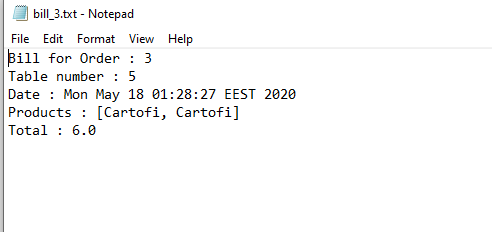
Comenzile dupa plasarea comenzii de mai sus :



Interfata bucatarului dupa plasarea comenzii :



Factura generata pentru comanda anterioara :



1. Concluzii

Aplicatia implementeaza cu succes un sistem de management al unui restaurant in care sunt diferiti utilizatori care pot executa diferite operatii precum : manipularea meniului si comenzilor.

Proiectul a fost folositor pentru invatarea tehnicilor de Observer Design Pattern, Design By Contract, Composite Design Pattern.

1. Bibliografie

<https://www.geeksforgeeks.org/observer-pattern-set-2-implementation/>

<https://ms.sapientia.ro/~manyi/teaching/oop/oop_romanian/curs15/curs15.html>

<https://docs.oracle.com/javase/7/docs/technotes/tools/windows/javadoc.html#tag>

<https://www.tutorialspoint.com/java/java_serialization.htm>

<https://en.wikipedia.org/wiki/Composite_pattern>