DOCUMENTATIE

TEMA 4

RESTAURANT MANAGEMENT SYSTEM

Vamvu Denisa-Elena

Grupa 30229

CUPRINS

1. Obiectivul temei

-Se va prezenta obiectivul principal al temei printr-o fraza si un tabel sau o lista cu obiectivele secundare. Obiectivele secundare reprezinta pasii care trebuie urmati

pentru indeplinirea obiectivului principal. Fiecare obiectiv secundar va fi descris si

se va indica in care capitol al documentatiei va fi detaliat.

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

-Modelare, scenarii, cazuri de utilizare . Se va prezenta cadrul de cerinte functionale formalizat si cazurile de utilizare ca si diagrame si descrieri de use-case.

Descrierile use-case-urilor se vor face sub forma unui flow-chart ori sub forma unei liste continand pasii executiei fiecarui use-case.

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

-Se va prezenta proiectarea OOP a aplicatiei, diagramele UML de clase si de pachete, structurile de date folosite, interfetele definite si algoritmii folositi (decizii de proiectare, diagrame UML, structuri de date, proiectare clase, interfete, relatii, packages, algoritmi, interfata utilizator).

4. Implementare

-Se va descrie fiecare clasa cu campurile si cu metodele importante. Se va descrie implemantarea interfetei utilizator.

5. Rezultate

-Se vor prezenta scenariile pentru testare cu Junit sau alt framework de testare.

6. Concluzii

-Se vor prezenta concluziile, ce s-a invatat din tema, dezvoltari ulterioare.

7. Bibliografie

-Se vor mentiona resursele bibliografice care au fost folosite pentru dezvoltarea temei.

1. Obiectivul temei

Obiectivul temei este realizarea unei aplicatii pentru managementul unui restaurant. Sistemul are 3 tipuri de utilizatori: administrator, chelner si bucatar. Administratorul poate adauga, sterge sau modifica produse existente din meniu. Chelnerul poate crea o comanda noua pentru o anumita masa, poate adauga elemente din meniu in comanda, si poate genera facture pentru o comanda. Bucatarul este atentionat de fiecare data cand trebuie sa gateasca mancare care este comandata cu ajutorul unui ospatar.

Aplicatia va fi realizata folosind:

* Design by Contract
* Design pattern-urile Observer si Composite
* JCF HashMap
* Serialization

Obiective secundare*:*

* Dezvoltarea de use-case-uri
* Alegerea corecta a structurilor de date
* Impartirea pe clase
* Dezvoltarea algoritmilor
* Implementarea solutiilor
* Testarea programului

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

Aplicatia trebuie sa permita implementarea unui restaurant cu date serializate citite dintr-un fisier care poate fi primit ca si argument in linia de comanda sau se poate apela direct folosindu-se fisierul “restaurant.ser” unde se afla meniul restaurantului. Trebuie de asemenea sa prezinte printr-o interfata grafica meniul pentru cei 3 useri si sa implementeze functionalitatile specifice fiecaruia . Pentru administrator mai trebuie sa afiseze intr-un tabel meniul restaurantului, iar pentru chelner trebuie sa afiseze intr-un tabel comenzile primite. Bucatarul va fi notificat si va incepe procesarea produselor o data ce chelnerul a finalizat o comanda. De mentionat faptul ca desi meniul, pentru administrator, va contine toate produsele, atat cele compuse, cat si cele de baza si are rol si de stoc, meniul pe care chelnerul il prezinta pentru comenzi este alcatuit din produse compuse, care necesita sa fie procesate de catre bucatar.

**Use case**

Programul va fi apelat din linia de comanda cu ajutorul unui fisier .jar ce se afla in folderul proiectului. Se executa comanda “java -jar” urmata de numele fisierului .jar si numele fisierului “.ser” care contine datele despre restaurant (optional). Aplicatia poate fi rulata si fara un fisier ca si argument, iar in acest caz fisierul de unde se vor lua datele este fisierul “restaurant.ser” care se afla deja populat in folderol proiectului. Ca si rezultat se va deschide aplicatia prezentand initial 3 butoane pentru a putea selecta tipul de utilizator.

1. Proiectare

Decizii de poiectare: am folosit JavaSwing pentru realizarea interfetei grafice, toate pattern-urile specificate iar datele despre restaurant au fost retiute prin serializare.

Structuri de date folosite: am folosit mai multe structuri de tip ArrayList pentru a retine lista de produse a meniului, lista de comenzi. De asemenea am folosit o structura de tip Map<Order, Collection<MenuItem>> pentru a stoca informatiile legate de o anumita comanda. Implementarea va fi detaliata in sectiunea 4.

Diagrama use-case:

A close up of a map

Description automatically generated

Diagrama de pachete:

A screenshot of a cell phone

Description automatically generated

Diagrama UML de clase:

A close up of a map

Description automatically generated

1. Implementare

In continuare voi prezenta clasele si pachetele pe care le-am folosit in acest proiect, dar si metodele cele mai importante din fiecare clasa.

* **Business –** pachetul contine clasele necesare pentru implementarea restaurantului cu tot ce contine el: produse si comenzi.
* **MenuItem**

Clasa MenuItem reprezinta, asa cum ii spune numele, un produs din meniu. Are ca si atribute un nume si un pret, un intreg. Este superclasa pentru clasele BaseItem si CompositeItem si implementeaza interfata Serializable care alaturi de atributul de tip static final long serialVersionUID permit obiectelor de acest tip sa fie serializate. Serializarea este conversia starii unui obiect intr-un bytestream care poate fi salvat si mai tarziu deserializat chiar si de pe alta platforma.

* **BaseItem**

Clasa BaseItem reprezinta un produs de baza, mostenind atributele din clasa MenuItem, fiind tratata ca si un “ingredient” pentru produsele compuse. Desi acestea sunt salvate in meniul restaurantului, se considera ca nu pot face parte dintr-o comanda.

* **CompositeItem**

Clasa CompositeItem defineste un produs compus, asa cum se gasesc in meniurile restaurantelor. Ca si BaseItem, mosteneste atributele name si price din superclasa si are in plus un ArrayList<MenuItem> care reprezinta produsele ce il compun. Cele 3 clase prezentate pana acum sunt definite folosind pattern-ul Composite. Astfel, un composite product poate fi format atat din base products cat si din alte composite products.

Pretul unul CompositeItem este recalculat de fiecare data cand un produs nou este introdus in lista sa de componente, acesta reprezentand suma tuturor preturilor produselor componente.

* **Order**

Clasa Order retine datele despre o comanda: id-ul comenzii, numarul mesei din restaurant, si data, care va fi data curenta. Cea mai importanta metoda este metoda hashCode() care returneaza un int representant hashcode-ul unui obiect de tip order. Rezultatul acestei metode va fi folosit ca si cheie pentru HashMap-ul din restaurant unde se vor lega obiectele de tip order cu listele de menuItems.

* **IRestaurantProcessing**

Interfata IRestaurantProcessing contine antetele tuturor functiilor de baza pe care un restaurant le poate implementa: crearea unui produs, editarea unui produs, stergerea unui produs, crearea unei comenz, calcularea pretului unei comenzi si generarea unei facturi in format txt pentru o anumita masa. In aceasta interfata se implementeaza modelul Design by Contract folosind preconditii si postconditii pentru fiecare metoda. Acestea vor fi verificate in implementarile din clasa Restaurant folosind instructiuni de tip assert.

* **Restaurant**

Clasa Restaurant contine ca si atribute un meniu, ca si o lista de MenuItems, o lista de comenzi, si un HashMap unde sunt retiute legaturile dintre o comanda si produsele comandate, un observer care va fi bucatarul si un string ca si nume al fisierului de unde se va face deserizarea datelor si incarcarea lor in meniu (in constructor) . Clasa extinde Observable pentru a se putea implementa design Pattern-ul Observer ce presupune un subiect ( in cazul nostru restaurantul ) care notifica toti observatorii sai ( in cazul nostrum un bucatar ) atunci cand asupra sa este efectuata o schimbare. Invariantul clasei il reprezinta meniul deoarece este cel mai de baza atribut al sau si trebuie sa fie mereu intr-o stare stabila atunci cand se fac operatii pe el.

Deoarece clasa implementeaza interfata definita anterior, aceasta trebuie sa implementeze si toate metodele definite in aceasta, care sunt:

* **createMenuItem** care este implementata in doua variante: pentru un baseItem, unde primeste un nume si un pret, sau pentru un compositeItem, caz in care se trimit ca si parametrii un nume si o lista de menuItems reprezentand compozitia acestuia. Functionarea este asemanatoare: obiectele sunt create si adaugate la meniu.
* **editMenuItem** aduce si ea doua implementari: pentru un produs de baza unde se trimite numele vechi dupa care se face cautarea in meniu, numele nou si pretul nou si pentru un produs compus unde se trimite numele vechi cu acelasi scop, numele nou, si o lista total noua de ingredient. Atunci cand un produs compus este editat, nu se mai tine cont de lista lui anterioara de ingrediente.
* **deleteMenuItem** primeste ca si argument un nume dupa care este cautat produsul care se doreste a fi sters si il sterge pe el ca si produs singular, si toate produsele compuse din care face parte s.a.m.d in mod recursiv.
* **createOrder** primeste ca si argumente o masa si o lista de produse din meniu. Metoda creeaza o comanda pentru masa respectiva, cu id-ul corespunzator folosind dimensiunea listei de comenzi si data curenta initializata cu metoda System.currentTimeMillis(). Comanda este adaugata la lista de comenzi si la structura de tip HashMap, realizand legatura intre comanda si lista de produse comandate. De asemnea, instiinteaza bucatarul ca are preparat ceva. In cazul in care fereastra de bucatar nu este deschisa, acesta nu poate procesa produsele.
* **computePrice** primeste ca si parametru numarul unei mese si returneaza pretul total al comenzii corespunzatoare mesei respective.
* **generateBill** primeste ca si paramtru numarul unei mese si creeaza o factura in format txt continand numele produselor, pretul fiecaruia, pretul total si masa.

O alta metoda este metoda **isWellFormed**() care se asigura ca invariantul este intr-o stare stabila. Este verificat alaturi de pre si post conditii in fiecare metoda folosind assert. In cazurile in care ceva este gresit se va arunca o exceptie si se va afisa un mesaj de eroare.

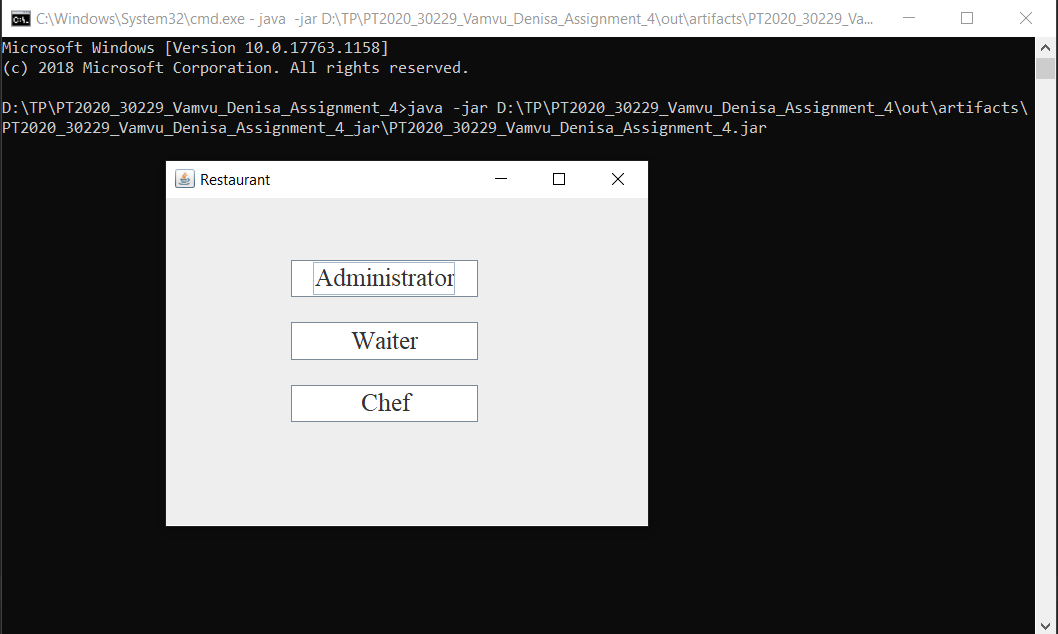
Metoda **notifyObserver** primeste ca si parametru un obiect de tip Observable care este in cazul nostrum restaurantul si un ArrayList ce reprezinta o comanda. Aceasta apeleaza metoda de update din clasa ChefGUI.

* **Data** contine clasele care se ocupa cu managementul datelor si anume serializarea si deserializarea acestora.
* **FileWriter –** contine o metoda statica ce parcurge meniul unui restaurant si foloseste un obiect de tipul ObjectOutputStream care poate scrie tipuri primitive ale unor obiecte intr-un fisier de tip FileOutputStream ca un stream de bytes folosind metoda writeObject(). Prima data se scrie size-ul meniului sim ai apoi lista de MenuItems care implementeaza Serializable.
* **RestaurantSerializator –** contine o metoda statica ce cu ajutorului un ObjectInputStream citeste datele dintr-un FileInputStream si le retransforma in obiecte cu ajutorul functiei readObect(). Se va citi prima data size-ul cu ajutorul functiei readInd() pentru ca se stie ca asta s-a scris, deoarece in aceasta aplicatie folosim acelasi fisier si pentru serializare si pentru deserializare, iar mai apoi se citesc size obiecte de tip MenuItem.
* **Presentation** contine clasele necesare pentru definirea interfetei utilizator. Acestea au fost grupate in doua subpachete, View si Controller. Pachetul View contine definirea cu JavaSwing a celor 4 interfete ale aplicatiei: cea de inceput, cea de administrator, cea de chelner si cea de bucatar. Pachetul controller contine clasele care implementeaza metodele pentru toate actiunile ce sunt efectuate asupra aplicatiei.
* **FrontGUI** contine definirea interfetei de start, cu 3 butoane, cate unul pentru fiecare user.
* **AdministratorGUI** contine definirea interfetei pentru utilizatorul administrator ce contine cate un tab pentru fiecare operatie pe care o poate realiza administratorul. Am ales sa separ operatiile de creare de produs si de editare pentru a face interfata cat de friendly posibila si am pus doar field-urile necesare pentru fiecare tip in parte. Deci vor fi doua tab-uri pentru crearea de produse, doua pentru editare, unul pentru stergere, si unul in care va fi afisat un tabel cu produsele disponibile din meniu.
* **ChefGUI** va contine un mesaj atunci cand bucatarul va fi notificat ca a fost efectuata o comanda.
* **WaiterGUI** contine de asemenea cate un tab pentru crearea unei comenzi, afisarea pretului si generarea facturii in acelasi tab, si un tab care contine un tabel cu toate comenzile.
* **Controller** contine clasele care implementeaza ActionListener pentru cele 3 butoane din FrontGUI. La apasarea unuia dintre ele este deschisa o noua fereastra care corespune unei clase din pachetul view pentru fiecare user, si un controller aferent acesteia, mai putin la butonul pentru bucatar, unde nu avem nevoie de un controller.
* **AdministratorController** contine implementarea actionListenerelor pentru fiecare buton din interfata de administrator, o metoda care populeaza tabelul de meniu, metode ce populeaza un combobox, ce este folosit pentru a pune la dispozitia administratorului produsele din meniu pentru a fi selectate si folosite la operatii.
* **WaiterContoller** este similar controllerului de administrator.

1. Rezultate

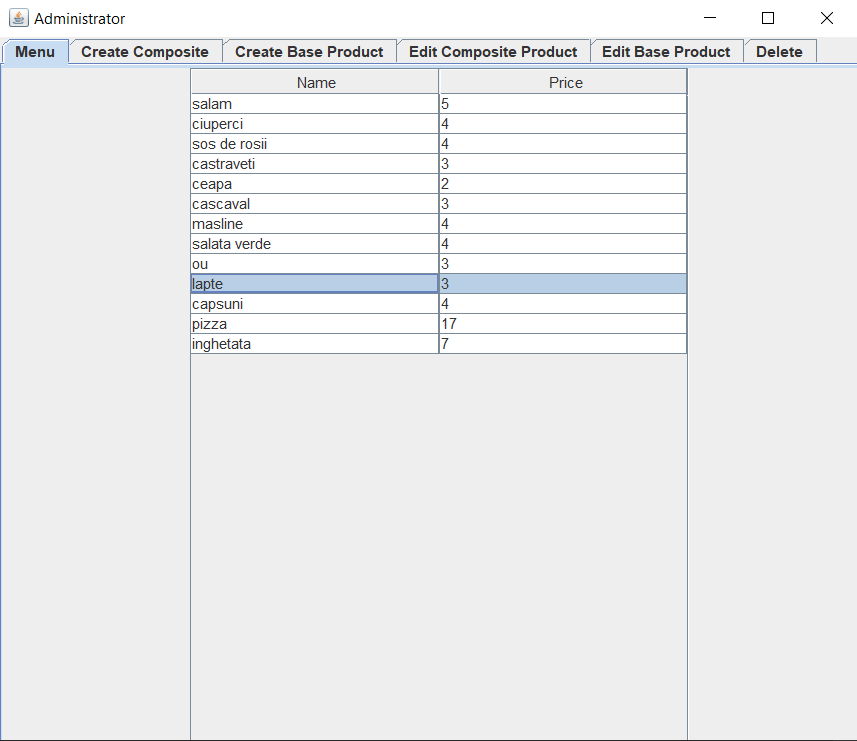
Ca si rezultate am observant ca aplicatia functioneaza conform cerintei iar in continuare voi exemplifica un model de rulare si utilizare.

Se ruleaza .jar-ul fara niciun argument, asa ca se foloseste fisierul deja exitent. Se deschide meniul principal.

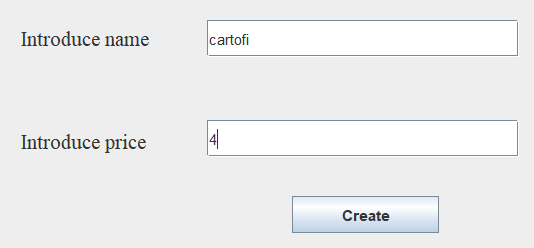


Pentru simplitatea aplicatiei, se considera ca avem cate un user din fiecare tip, deci nu exista un meniu de login.

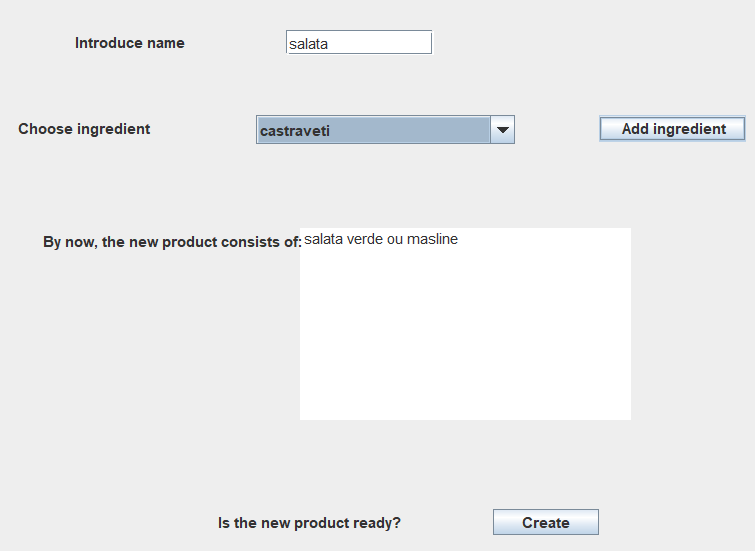
Pentru administrator:



Se deschide interfata. In primul tab este prezentat meniul, iar in urmatoarele sunt implementate operatiile pe care acesta le poate efectua.

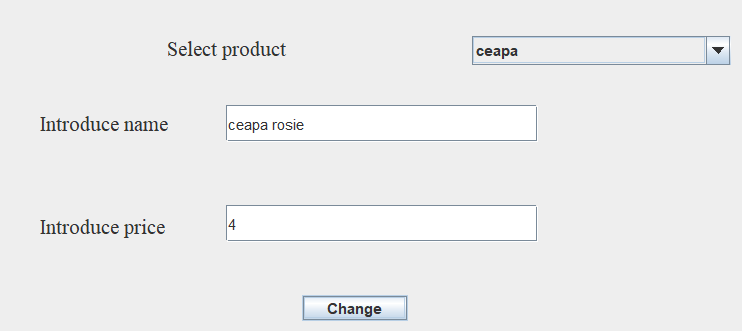


Pentru crearea unui produs de baza de introduce un nume si un intreg reprezentand pretul. Se dupa se apasa butonul, produsul trebuie sa apara atat in tabul de meniu, cat si in combobox-uri.



Pentru crearea unui produs compus se introduce un nume si se alege cate un produs dupa care se apasa pe butonul de add, iar acesta va fi adaugat in lista din text area. Cand se apasa butonul de create, produsul este creat si disponibil in meniu, ca si mai sus.

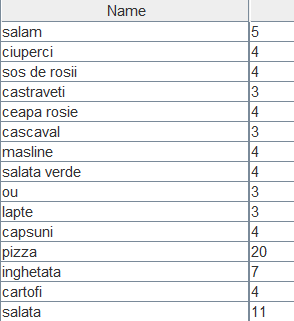
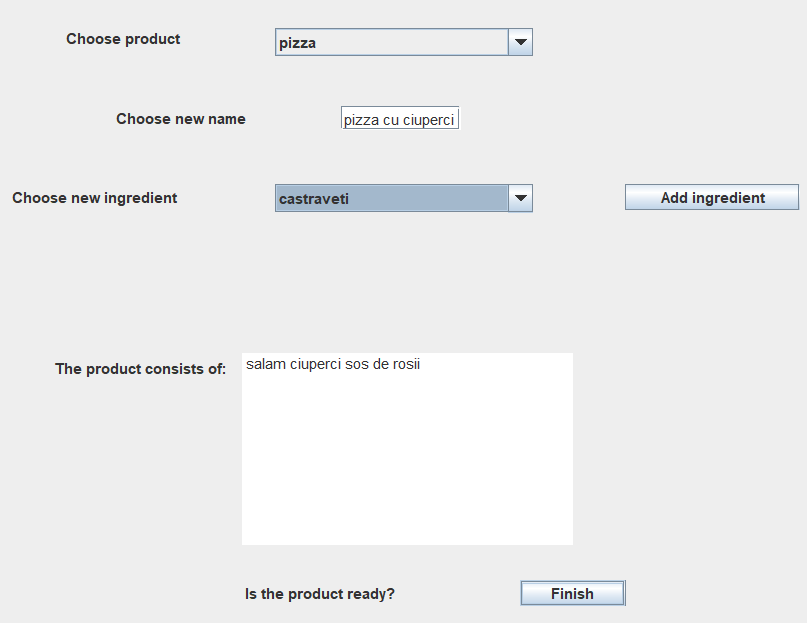
Pentru editarea unui base product, se selecteaza produsul dorit si se introduce noul nume si noul pret.



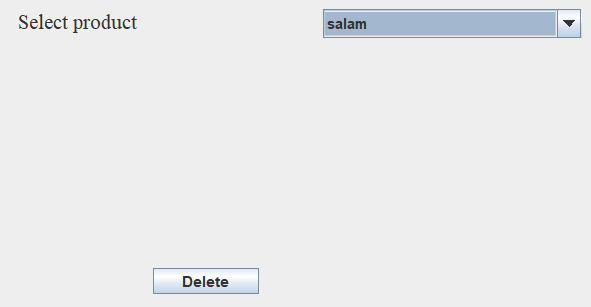
Dupa ce se apasa butonul schimbarea este vizibila in meniu:



Pentru editarea unui composite product se selecteaza acesta, noul nume si noua lista de ingrediente din meniu, slectata din combobox si afisata dupa fiecare update. Pretul este recalculat automat in functie de ingrediente.

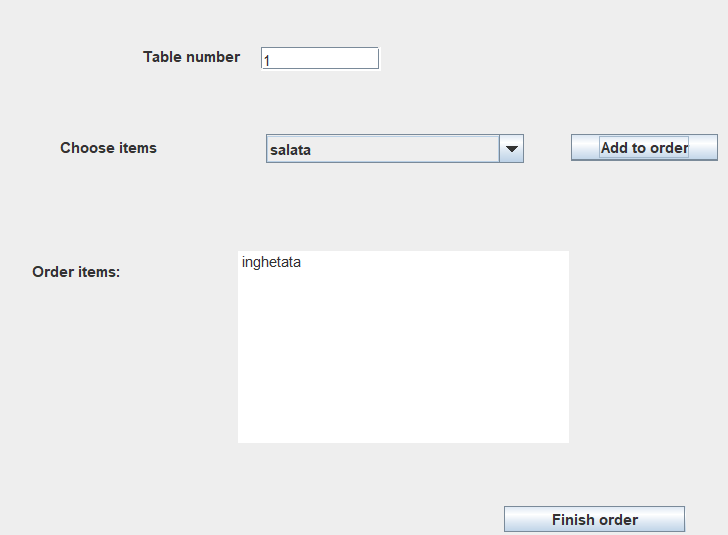


Pentru stergerea unui produs se selecteaza acesta si se apasa pe buton.

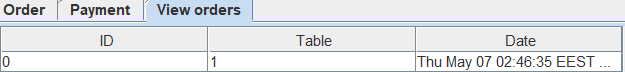


Se observa ca s-a sters atat produsul salam cat si produsul pizza, care continea salam.

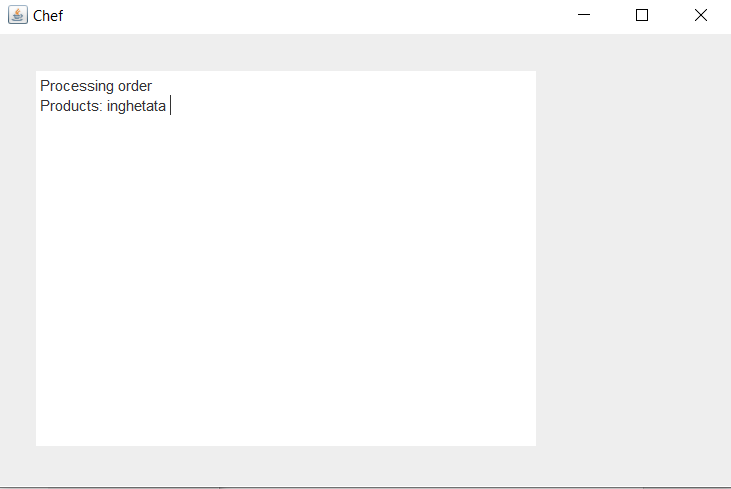
Interfata Waiter deschide 3 tab-uri: tab-ul pentru crearea unei comenzi, tabu-ul de payment care permite afisarea pretului pentru o comanda si generarea facturii in format pdf si tabelul cu order-urile, initial gol.

Crearea unei comenzi functioneaza similar cu cea a unui composite: se introduce numele mesei si se adauga produsele dorite. De mentionat ca se pot alege numai produse de tip composite si numai cate unul din fiecare. 

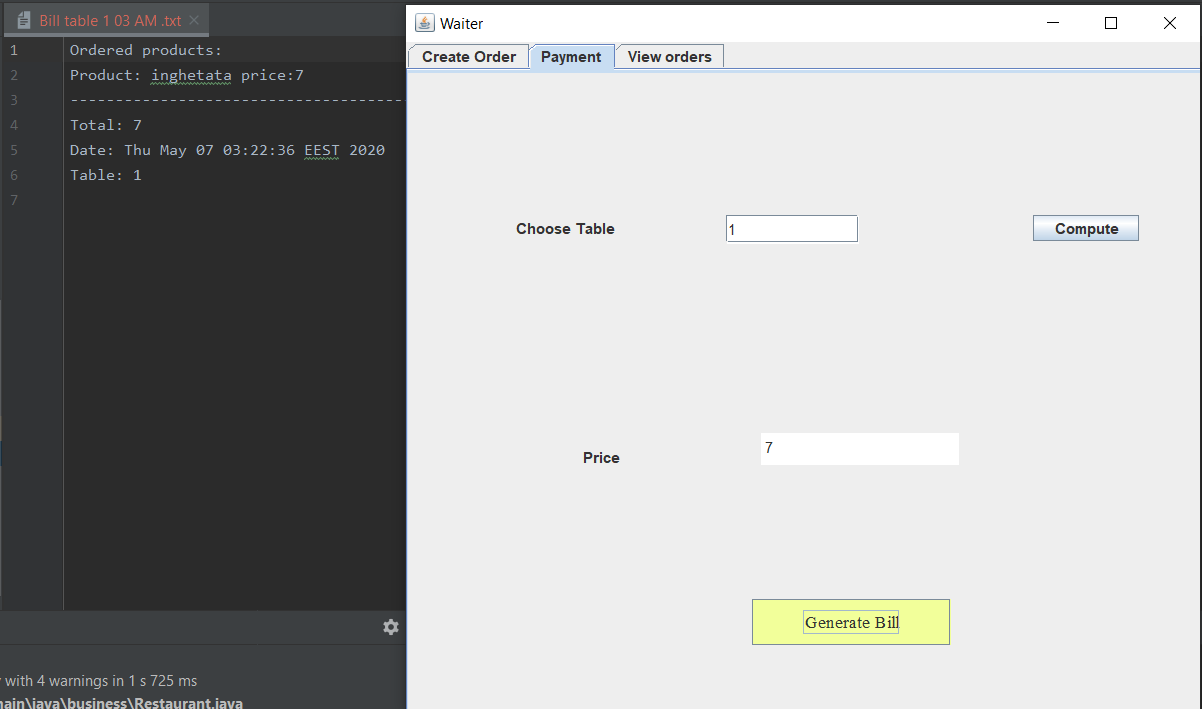
Dupa ce se apasa Finish Order, aceasta va fi disponibila in tabel.



Interfata pentru bucatar trebuie si ea sa fie pornita ca acesta sa poata fi notificat.



In tab-ul de payment se selecteaza masa. Daca se apasa pe butonul de compute se afiseaza pretul in meniu, iar daca se apasa pe generateBill se genereaza factura in format txt.



1. Concluzii

Ca si dezvoltare ulterioara as dori sa introduc un meniu special unde sa pot afisa in interfata si lista de componente ale un CompositeItem.

1. Bibliografie
   1. <https://www.baeldung.com/java-serialization>
   2. <https://intellij-support.jetbrains.com/hc/en-us/community/posts/207014815-How-to-enable-assert>
   3. <https://docs.oracle.com/javase/7/docs/technotes/tools/windows/javadoc.html#tag>
   4. <http://coned.utcluj.ro/~salomie/PT_Lic/5_Courses/>