

**FACULTATEA DE AUTOMATICĂ ŞI CALCULATOARE**

**~SPECIALIZAREA: CALCULATOARE SI TEHNOLOGIA INFORMATIEI~**

**Cuprins**

**RESTAURANT MANAGEMENT SYSTEM**

**Grupa 30226**

**An 2, Semestrul 2**

**Arseniuc Anamaria**

**-Documentatie-**

C:\Users\Admin\AppData\Local\Temp\ksohtml\wps7BAC.tmp.png

1 . Obiectivul temei . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . pg 3-4

2 . Analiza problemei . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .. . .pg 4-10

3 . Proiectare. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .. . . . .. . . . . .pg 11-12

4 . Implementare si testare . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .pg 12-16

5 . Rezultate . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .. . . . . . . . .. . . . . . . .pg 16

6 . Concluzii si dezvoltari ulterioare . . . . . . . . . . . . . . . . . . . . . . . . .. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .pg 16-17

7 . Bibliografie . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .. . . . . . . . . . . . .pg 17

**1.Obiectivul temei**

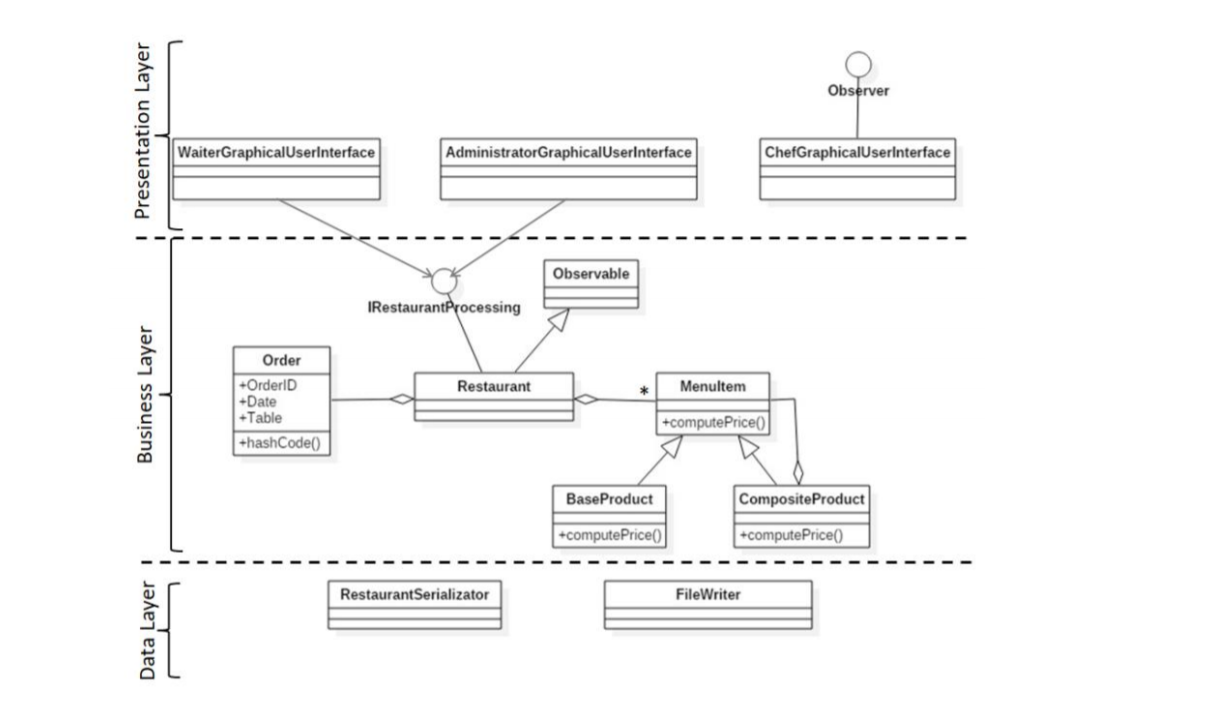
*Se va prezenta obiectivul principal al temei printr-o fraza si un tabel sau o lista cu obiectivele secundare. Obiectivele secundare reprezinta pasii care trebuie urmati pentru indeplinirea obiectivului principal. Fiecare obiectiv secundar va fi descris si se va indica in care capitol al documentatiei va fi detaliat.*

**Cerinta temei:**

**“***Proiectati si implementati un sistem de gestionare a unui restaurant .”*

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 ordered through a waiter.

Consider the system of classes in the diagram below.



To simplify the application you may assume that the system is used by only one administrator, one waiter and one chef, and there is no need of a login process.

Solve the following:

1. Define the interface RestaurantProcessing containing the main operations that can be executed by the waiter or the administrator, as follows:

• Administrator: create new menu item, delete menu item, edit menu item

• Waiter: create new order; compute price for an order; generate bill in .txt format.

1. Define and implement the classes from the class diagram shown above:

• Use the Composite Design Pattern for defining the classes MenuItem, BaseProduct and CompositeProduct

• Use the Observer Design Pattern to notify the chef each time a new order containing a composite product is added.

1. Implement the class Restaurant using a predefined JCF collection which uses a hashtable data structure. The hashtable key will be generated based on the class Order, which can have associated several MenuItems. Use JTable to display Restaurant related information.

• Define a structure of type Map> for storing the order related information in the Restaurant class. The key of the Map will be formed of objects of type Order, for which the hashCode() method will be overwritten to compute the hash value within the Map from the attributes of the Order (OrderID, date, etc.)

• Define a structure of type Collection which will save the menu of the restaurant. Choose the appropriate collection type for your implementation.

• Define a method of type “well formed” for the class Restaurant.

• Implement the class using Design by Contract method (involving pre, post conditions, invariants, and assertions).

4. The menu items for populating the Restaurant object will be loaded/saved from/to a file using Serialization.

* Obiectivul acestei teme este de a implementa o aplicatie cu interfata

grafica pentru a fi folosita ca simulator al unui restaurant, cu trei tipuri de utilizatori: administratorul, chelnerul si bucatarul restaurantului.

* Administratorul poate adauga, sterge si modifica produsele existente din meniu.
* Chelnerul poate crea o noua comanda pentru o masa, adauga elemente din meniu si sa calculeze bonul pentru o coamnda.
* Bucatarul restaurantului este anuntat (notificat) de fiecare data

**2.Analiza problemei,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. **Analiza problemei:**

* **Scopul unui restaurant este acelasi atat in lumea reala, cat si in**

**programul realizat in cadrul acestui proiect. Restaurantul ofera servicii clientilor, deci din acest punct de vedere trebuie realizate clase care sa stocheze informatii suficiente despre produsele din meniu si despre comenzile acestora.**

* **Administrator are posibilitatea de a crea un nou produs in**

**meniu, de a sterge un produs din meniu si de a edita produsele din meniu.**

* **Chelnerul are posibilitatea de a crea o noua comanda, calcula**

**pretul pentru acea comanda si de a genera factura in format .txt. Rapiditatea aplicatiei sta in folosirea unui hash-table, care faciliteaza cautarile si in general operatiile asupra produselor din meniu.**

1. **Modelarea:**

* **Pentru realizarea proiectului meu am creat mai multe clase cu**

**scopul de a avea functionalitati specifice clasei respective si de a putea adauga cu usurinta noi functionalitati in urmatoarele versiuni ale aplicatiei, astfel am incercat pe tot posibilul sa generalizez pentru fiecare clasa metodele si atributele acesteia. In cele ce urmeaza va voi prezenta clasele care sunt mai importante cat si metodele din acestea.**

* **Am structurat proiectul pe mai multe clase si mai multe pachete**

**pentru a reda fiecarei clase o singura functionalitate si le-am impartit pe pachete in functie de functionalitatea comuna, astfel pachetele si clasele proiectului meu sunt: model ( cu clasele BaseProduct, Composite Product, MenuItem, Order ), data ( cu clasa RestaurantSerialization ), config ( cu clasa PersistenceProperties), business ( cu clasele IRestaurantProcessing, Restaurant ), presentation( cu clasele ChefGUI, GUI ) si app ( cu clasa App ) .**

1. **Cazuri de utilizare:**

* **Utilizatorului aplicației trebuie să i se ofere o listă de produse**

**disponibile care apartin unui meniu din restaurant**

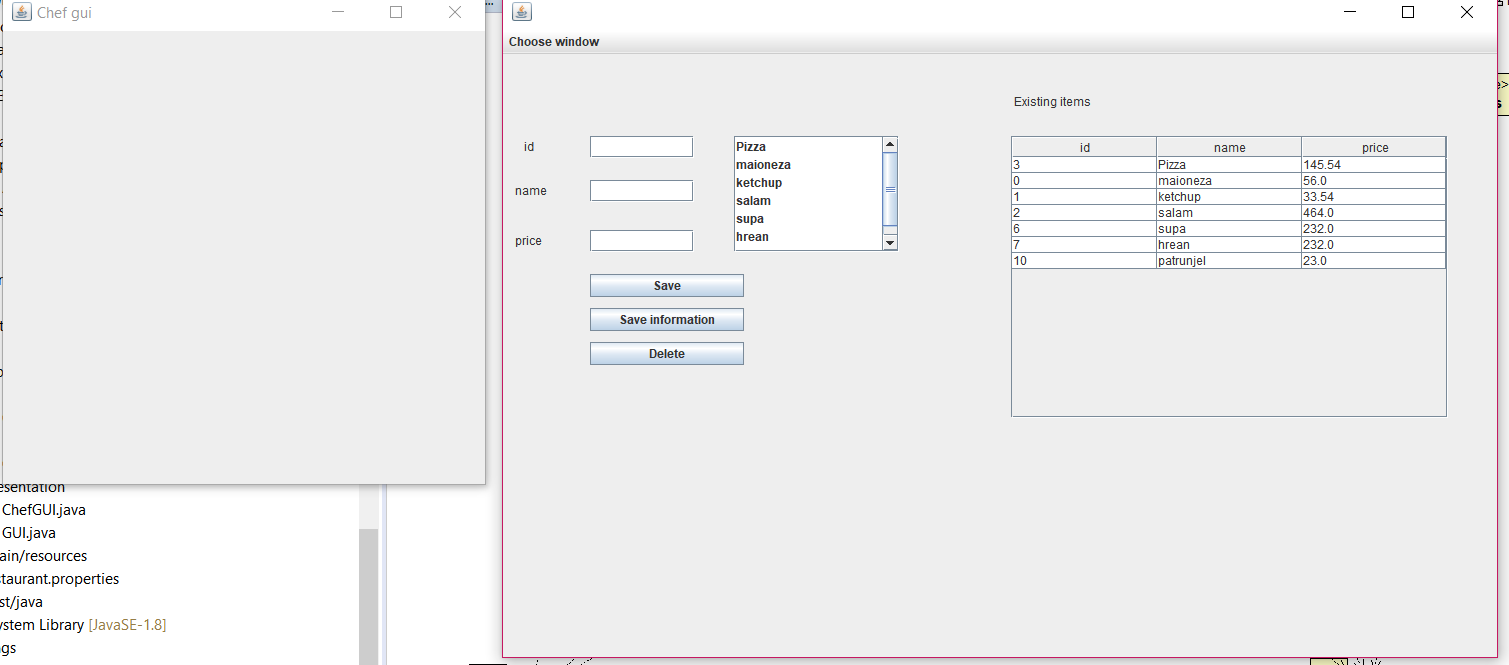
**și o listă de comenzi efectuate de clienti si preluate de chelnerul restaurantului. Toate actiunile din cadrul acestui loc vor fi notificate bucatarului, care este un observator. Ambele liste trebuie să conțină toate informațiile legate de despre comenzi și produse. Utilizatorul se poate afla in ipostaza administratorului și să poata adauga un nou produs in meniu, sterge si edita lista produselor din meniu. Utilizatorul, de asemenea, se poate afla si in ipostaza chelnerului si sa creeze o noua comanda, sa calculeze pretul pentru o comanda data de client si sa genereze factura in format .txt**

* **Un use-case realist care arata funcționarea acestei aplicații este**

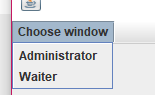
**simularea funcționării unui restaurant, a cărui gestionier este un administrator, chelner sau chiar o persoană reală (utilizatorul) care să introducă datele produselor si comenzilor de produse. Administratorul aplicației nu trebuie decât să adauge un nou produs in meniu, dandu-i un nume si un pret; sa stearga sau sa editeze lista de produse din meniu. Chelnerul poate să efectueze urmatoarele operațiuni : sa creeze o noua comanda , sa calculeze pretul pentru respective comanda si sa genereze factura in format .txt . Se mai remarcă accesul la tabela produselor si a comenzilor, asupra carora se fac modificari .**

Cadrul de cerinte functionale formalizat

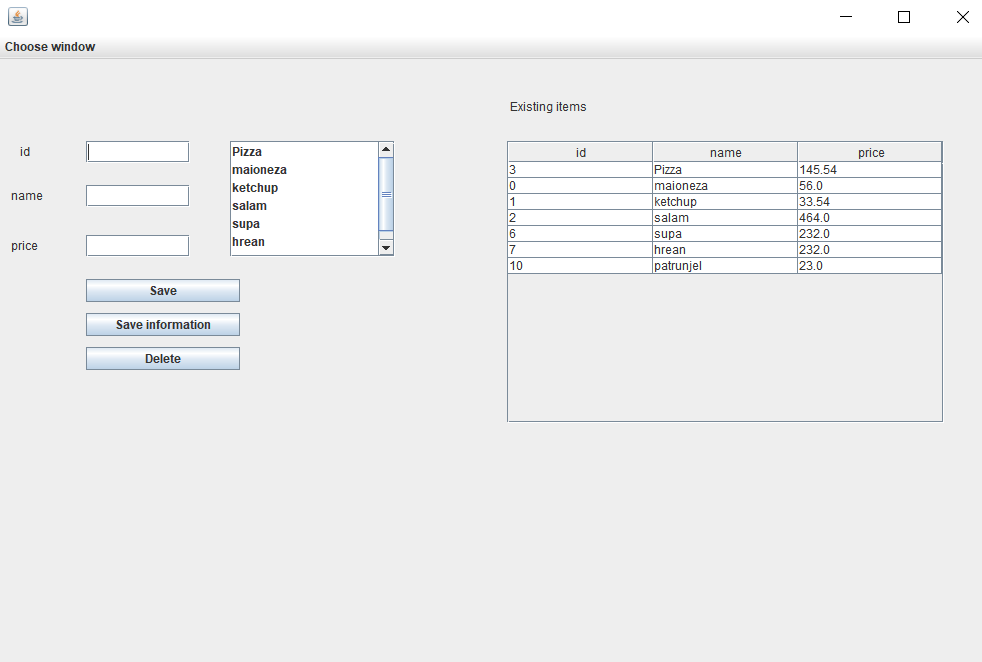
* **Aplicatia se va deschide cu doua interfete grafice:**



1. **prima contine 2 ferestre (Administrator si Waiter)**

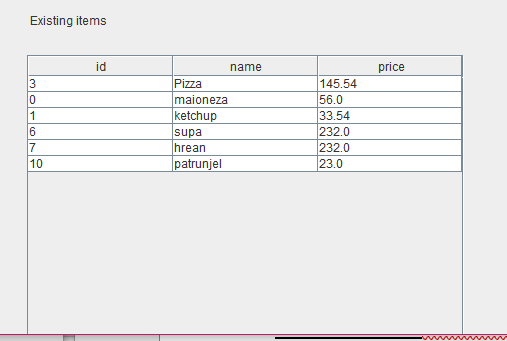


**care e formata din urmatoarele campuri:**

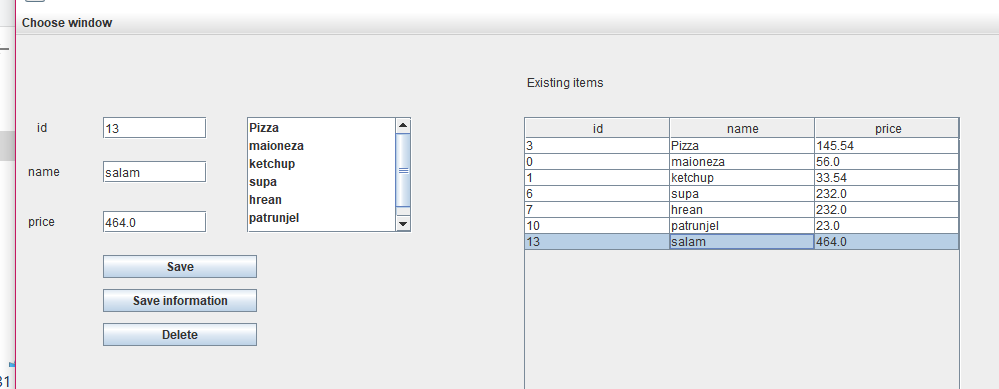


1. **iar a 2 a contine fereastra de “Observer” a bucatarului, care e notificat la fiecare actiune a chelnerului.**

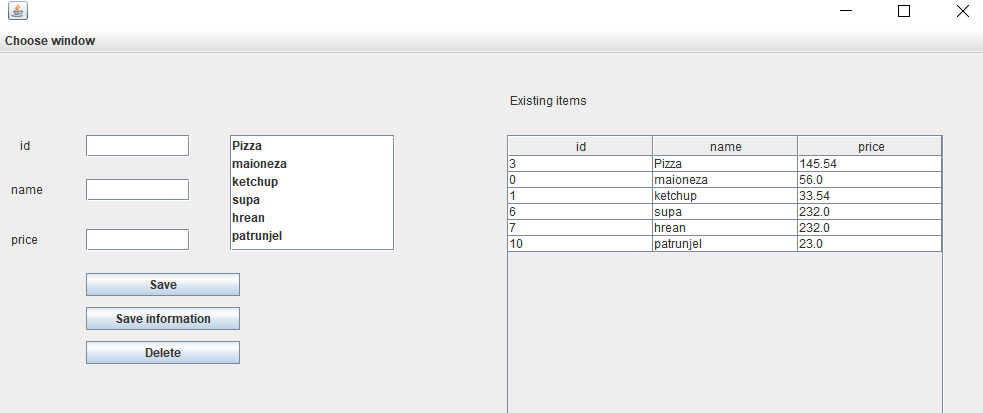
**Pe fereastra de “Administrator” vor aparea 3 field-uri : id-ul produsului existent in tabel, numele produsului si pretul acestui (individual pentru “Base Product” si compus (daca vrem), pentru “Composite Product”). Butonul “Save” va adauga noile informatii in tabelul cu produsele existente din partea dreapta, insa temporar; butonul de “Save information” va salva produsul in tabelul cu produse existente definitive(nu va disparea cand vom inchide fereastra); iar butonul “Delete” – sterge un item din tabelul existent de produse , la selectarea sa. In partea dreapta, apare un table cu toate produsele existente.**

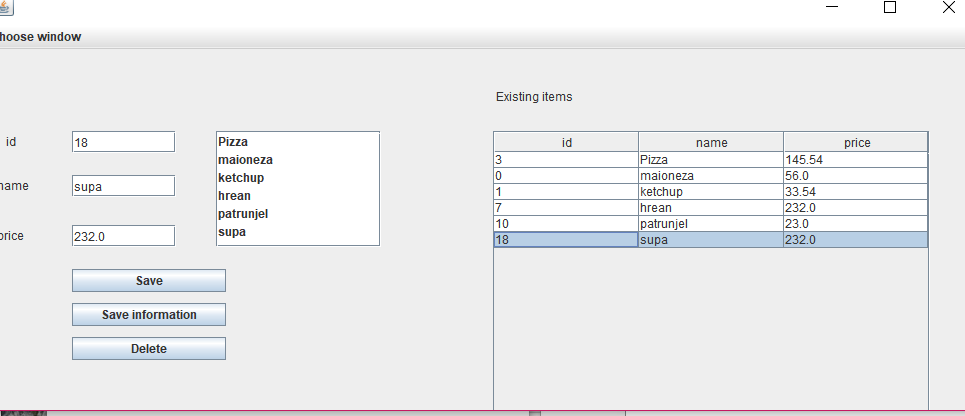


**EXEMPLU: Pentru a sterge un item din tabelul cu produse existente , se selecteaza un rand din tabel si se apasa pe butonul “Delete”. Am selectat produsul cu id-ul 13.**

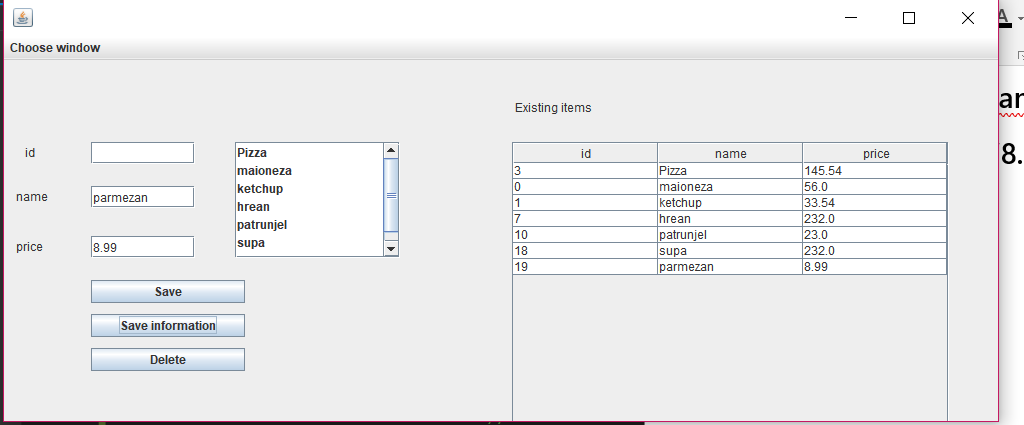


Si l-am sters.



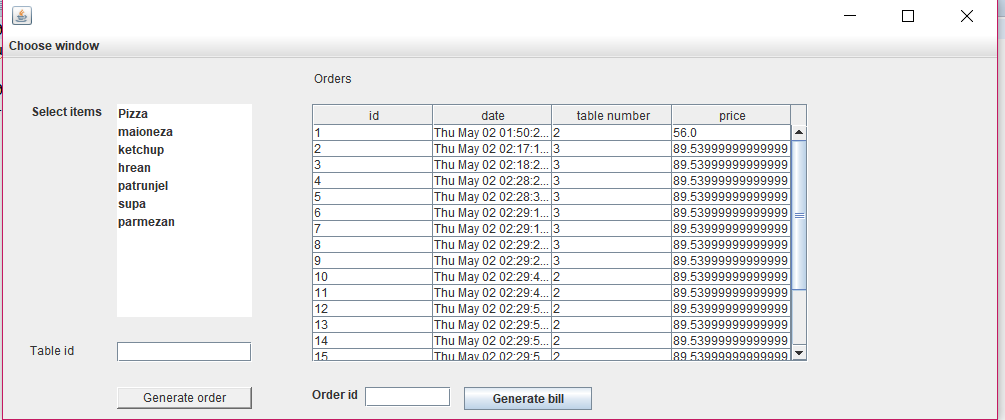


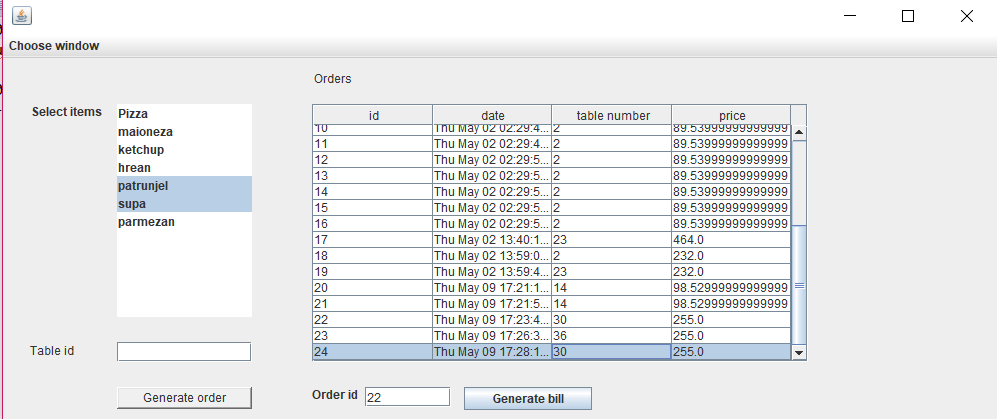
Presupunem ca editam randul cu id-ul produsului 18 si schimbam supa in “parmezan” si pretul in “8.99”, apoi dam “Save”, iar daca vrem sa pastram acest produs in continuare in tabelul nostrum, apasam “Save information”.



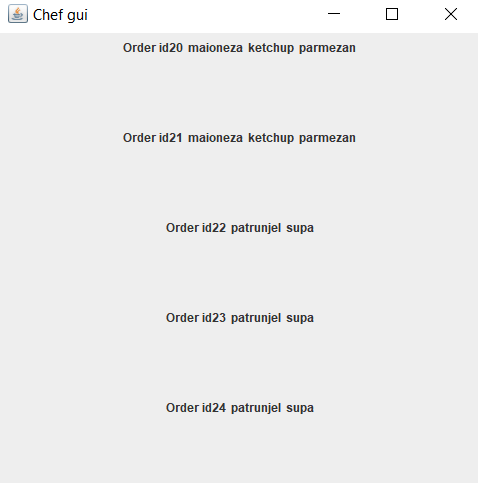
Pentru a adauga un produs nou, completam campurile de “name” si “price” cu date noi, iar field-ul “id” il lasam gol.

In fereastra de “Waiter” avem lista de produse si istoricul comenzilor pana in prezent. Avem field-ul de “Table id”, care reprezinta masa la care s-a comandat si putem introduce orice numar vrem. Tot aici avem si field-ul de “Order id”. Pentru a se genera un nou id, trebuie sa selectam intai un id existent din tabel (la un order se genereaza un nou order id) si produsele dorite din lista din stanga (se vor selecta tinand apasata tasta “ctrl”) si sa adaugam un numar la “Table id” . Lista comenzilor se va actualiza de fiecare data.



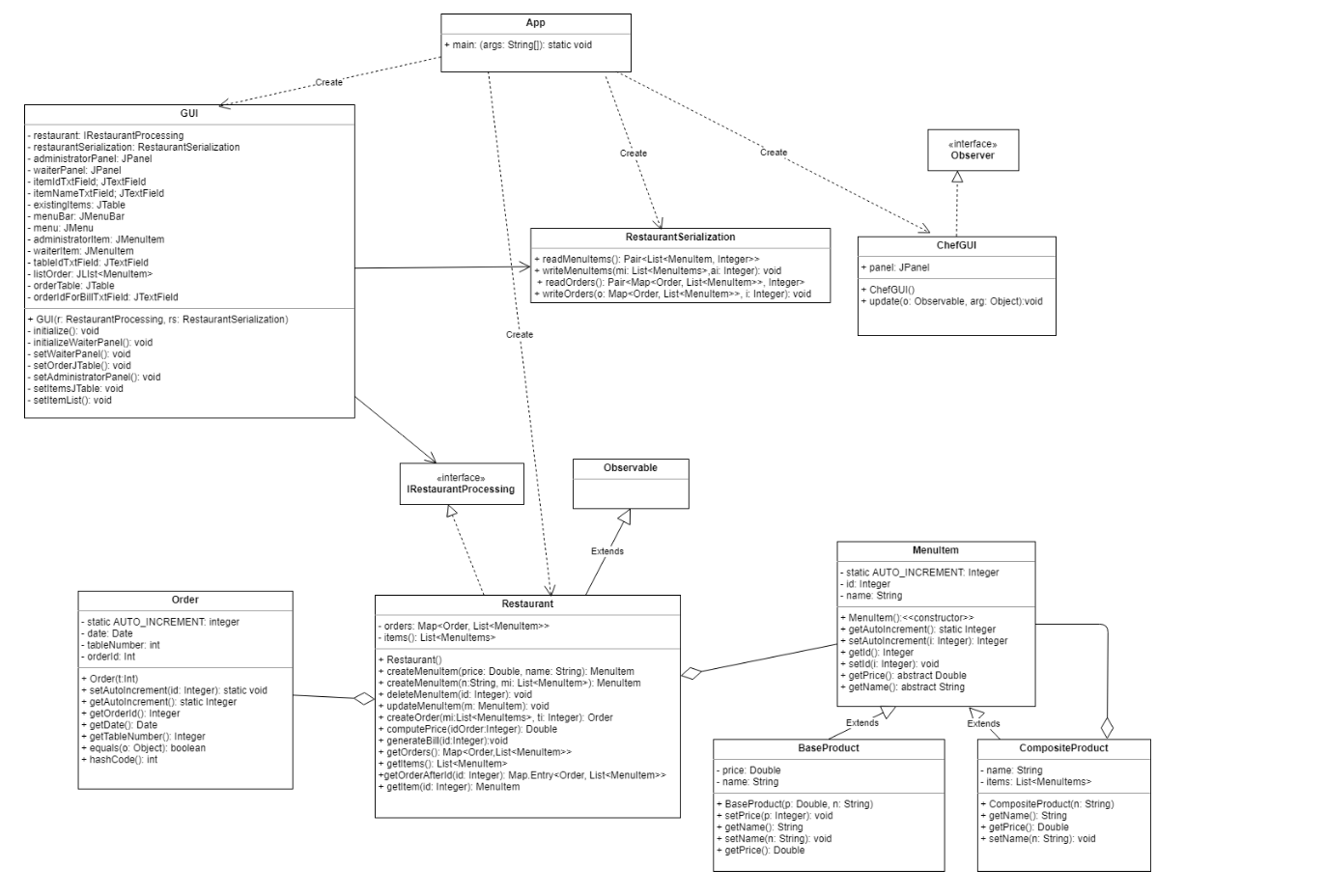


Bucatarului , in fereastra “Chef gui”, ii vor aparea toate comenzile preluate de chelner (id-ul comenzii si produsele selectionate de client).

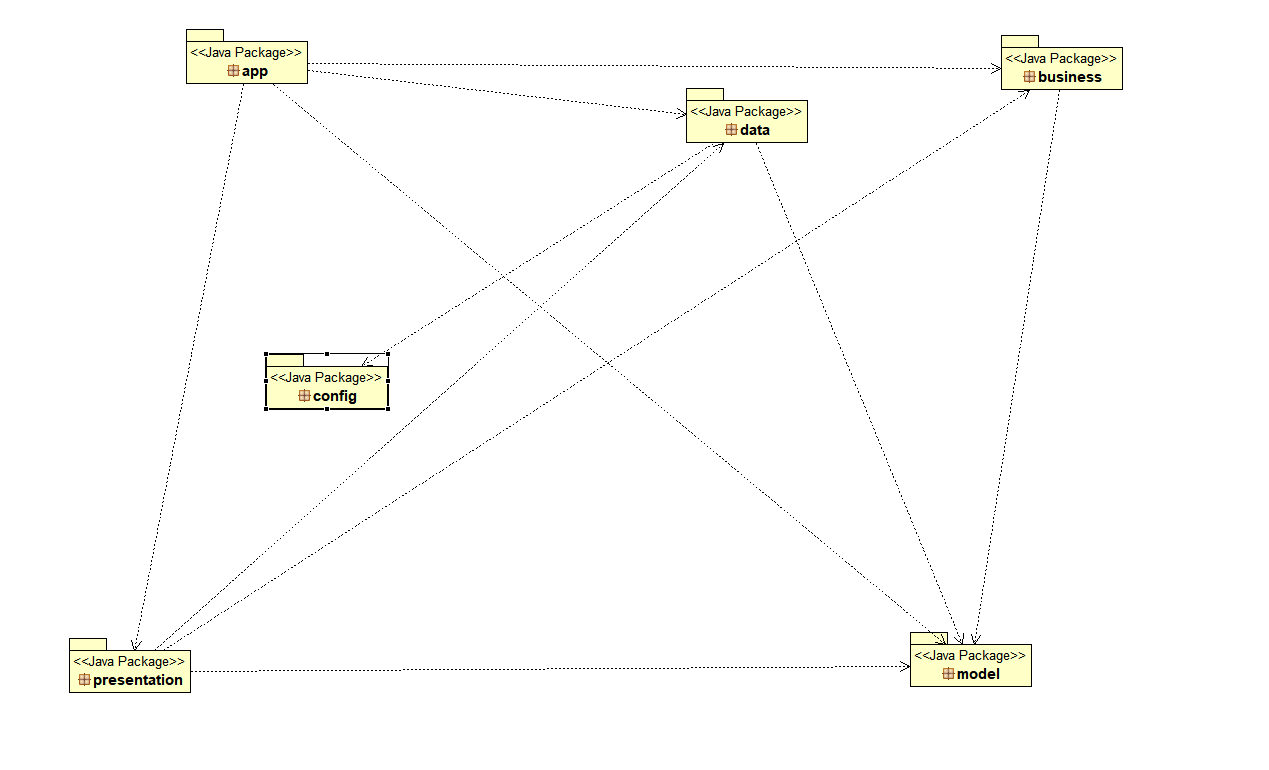


**3.** **Proiectare (decizii de proiectare, diagrame UML, structuri de date, proiectare clase, interfete, relatii, packages, algoritmi, interfata utilizator)**

* **Diagrama UML de clase**



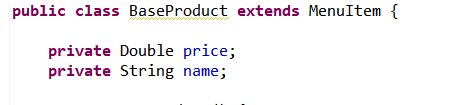
* **Diagrama de pachete**



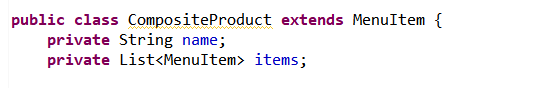
**4. Implementare**

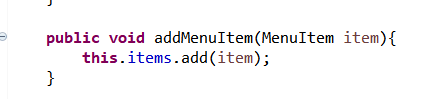
*Se va descrie fiecare clase cu campuri si metodele importante. Se va descrie implemantarea interfetei utilizator.*

* Clasa abstracta MenuItem contine campurile: id si name impreuna cu gettere si settere pentru acestea . Pe langa acestea mai avem variabila statica AUTO\_INCREMENT care apartine de clasa , folosita pentru fiecare produs creat, se genereaza un id unic . Aceast camp are de asemenea getter si setter . Mai avem doua metode abstracte getPrice() si getName() care vor fi implementate de subclase .
* Clasa BaseProduct extinde clasa MenuItem si contine campurile name si price impreuna cu gettere si settere pentru acestea .

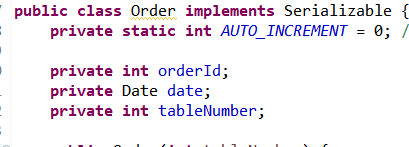


* Clasa CompositeProduct extinde clasa MenuItem si contine campurile name si o lista de MenuItem impreuna cu gettere si settere pentru acestea . In plus, avem o metoda de adaugare a unui item in lista .



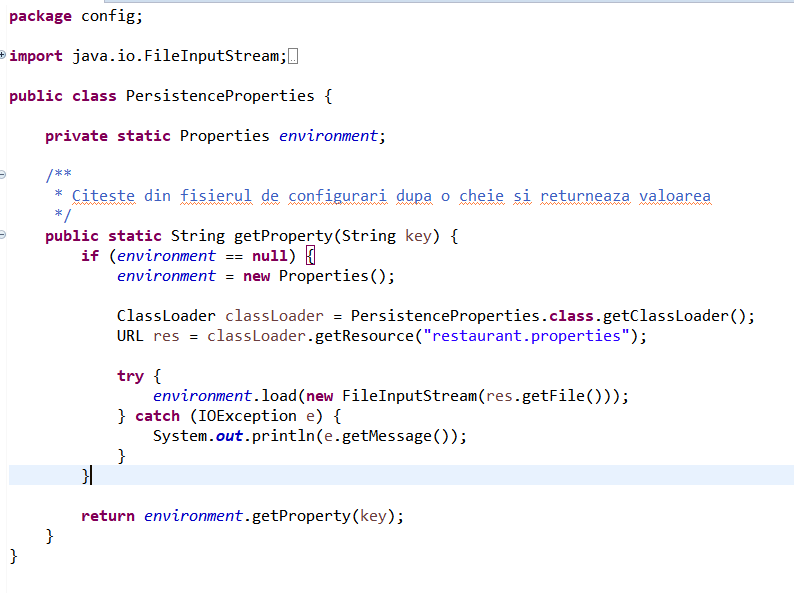


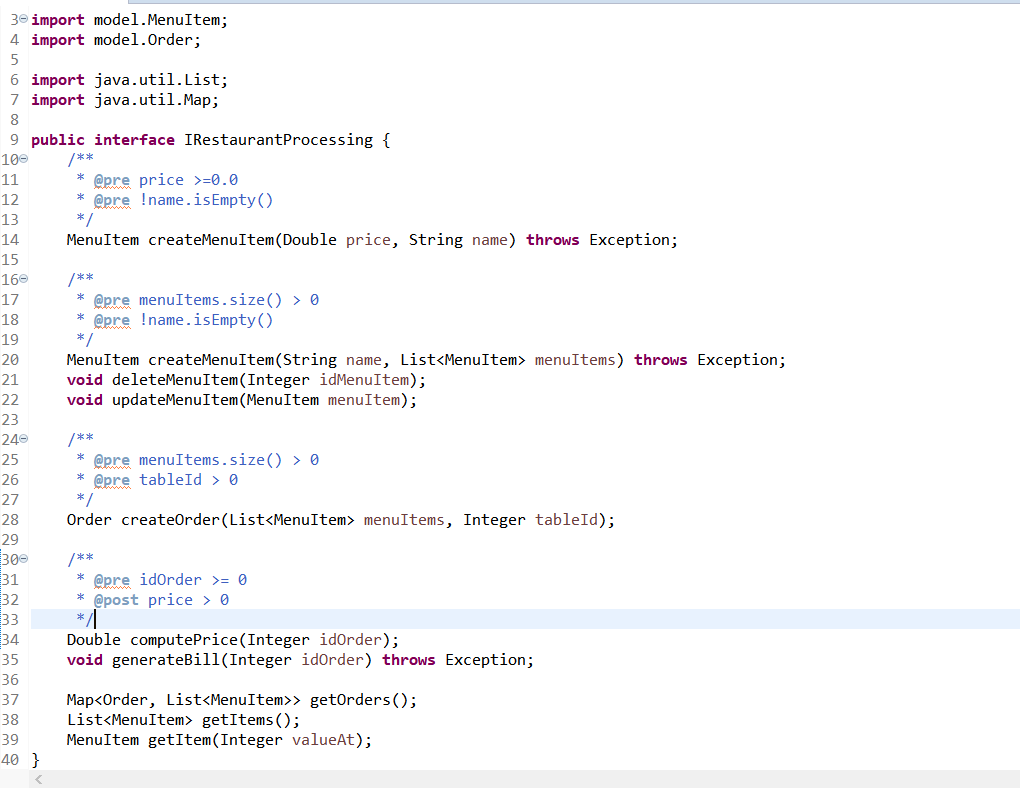
* Clasa Order contine variabila statica AUTO\_INCREMENT (care apartine de clasa; pentru fiecare order creat, se va genera un id unic), id-ul order-ului, data si indicele de tabel . Pe langa gettere si settere, am suprascris metode de equals() si hashCode() .

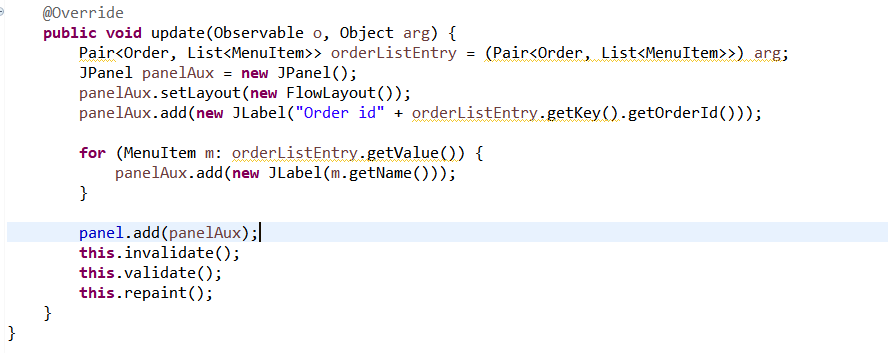


* In clasa RestaurantSerialization avem metodele de citire si scriere din fisier. Metoda readMenuItems() citeste produsele din fisier si ultimul id al vreunui produs creat in aplicatie(autoincrement). Metoda writeMenuItems() scrie in fisier produsele restaurantului si ultimul id al produselor create in aplicatie. Metoda readOrders() citeste orderele din fisier si ultimul id al vreunui order creat in aplicatie. Metoda writeOrders() scrie orderele din fisier si ultimul id al vreunui order creat in aplicatie.
* In clasa PersistenceProperties se afla metoda getProperty() care incarca cele 2 fisiere .txt din *restaurant.properties* : items.txt si orders.txt (citeste din fisierul de configurari dupa o cheie si returneaza valoarea.)





* In interfata IRestaurantProcessing contine antetul metodelor pentru create, actualizare si stergere de produse, respective de ordere. De asemenea contine antetul pentru functiile de generare factura si calculare pret, dar si metodele pentru obtinerea comenzilor, produselor sau a unui produs.
* Clasa Restaurant extinde clasa Observable si implementeaza interfata IRestaurantProcessing. Aceasta contine lista de produse(meniul) dar si comenzile. Aici sunt implementate metodele din interfata IRestaurantProcessing. Clasa foloseste Design by Contract(pre si post conditii si asserturi). 
* Clasa App contine metoda main, unde se creeaza instantele pentru restaurantSerialization, menuItems, orders si unde se creeaza fereastra pentru bucatar, obiectul de restaurant si se adauga ChefGUI ca si observer al restaurantului. Apoi se seteaza menuItems si orders citite anterior la restaurant si se instantiaza GUI-ul principal.
* Clasa ChefGUI reprezinta interfata grafica unde bucatarul vede comenzile aparute. Aceasta clasa implementeaza interfata Observer si este folosita in metoda *update*. Metoda apelata de catre observable atunci cand se creaza un order in aplicatie si se adauga un noul label cu id-ul order-ului si produsele comandate ca bucatarul sa fie notificat cand apare o comanda noua .



* Clasa GUI reprezinta interfata unde utilizatorul poate sa joace rolul de administrator, respectiv de chelner. Administratorul poate sa adauge un produs nou in meniu, sa-l modifice sau sa-l stearga. In aceasta fereastra se afla si un tabel cu toate produsele existente. Chelnerul poate sa adauge produse la comanda si sa trimita order-ul la bucatar. De asemenea acesta poate vedea tabelul cu toate order-ele.

**5.Rezultate**

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

Rezultatele aplicatiei sunt corecte si simuleaza corect un sistem de gestionare a unui restaurant. Dupa cum se observa din faza de testare, aplicatia poate sa adauge produse, pretul acestora sau sa le stearga din meniu. De asemenea , se realizeaza comenziile si se genereaza factura. Astfel este demonstrata buna functionalitate a aplicatiei.

**6.Concluzii**

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

* **In concluzie, acest proiect m-a ajutat sa imi aprofundez**

**cunostintele de programare orientata pe obiect si sa invat noi concepte ale limbajului Java. Particular,** **acest sistem de este implementat folosind limbajul de programare Java si conceptele de programare orientata obiect. Mediul de dezvoltare este Eclipse, intrucat este usor de utilizat si contine o multitudine de mecanisme de avertizare in caz de erori, extrem de utile pentru un programator aflat in stadiul de invatare.**

* **Rezultatele obtinute in urma simualarii restaurantului,**

**in conformitate cu datele furnizate de utilizator pot fi observate in interfata grafica si verificate prin urmarirea atenta a operatiilor efectuate asupra comenzilor si produselor .**

* **De asemenea, am inclus tabele in cadrul carora se poate**

**observa in timp real starea produselor si comenzilor incluse in restaurant. Fiecare produs poate fi deci updatat, iar rezultatul se vede ori in tabel, ori in text label-ul dedicat informatiilor suplimentare legate de nume sau pret .**

* **De asemenea, este esential sa observam ca pentru datele**

**limitate furnizate , algoritmul este cat se poate de eficient , deoarece desi orice sistem de management a unui restaurant este complex si are multe dificultati de gestionare , programul reuseste sa opereze corect , intrucat a fost conceput intr-un mod cat mai logic si simplu , pentru utilizator , fie jucand rolul de chelner , fie cel de administrator .**

* **In plus , folosirea colectilor JCF si design pattern-urilor**

**Design by Contract, Observer si Composite ofera o valoarea mai mare proiectului si ii sporeste profesionismul cu care a fost realizat .**

* **In cadrul dezvoltarilor ulterioare se pot aminti**

**urmatoarele: imbunatatirea interfetei grafice (prin folosirea de model view controler) , implementarea a diverse metode ce ajuta la securizare ( logarea ca chelner , administrator sau bucatar ) . Astfel, aplicatia ar avea un impact mult mai potrivit asupra realitatii , deoarece proiectul actual nu asigura o prea mare siguranta utlizatorilor, rezumandu-se strict la partea teoretica a ideii temei .**

* **Bineinteles , proiectul e simplist , iar cum un restaurant**

**in zilele noastre ofera un numar tot mai mare de servicii, este ideal ca anumite metode generale sa fie introduse in clasele ce au legatura cu acestea .**

* **Nu in ultimul rand, proiectul mai poate fi imbunatatit in**

**ceea ce priveste cautarea anumitor informatii , in ideea reducerii timpilor .**

**7. Bibliografie**

* https://stackoverflow.com/
* https://docs.oracle.com/javase/7/docs/api/
* **https://dzone.com/articles/the-observer-pattern-using-modern-java**
* **http://www.tutorialspoint.com/java/java\_serialization.html**
* **http://javarevisited.blogspot.ro/2011/02/how-hashmap-works-in-java.html**
* **http://docs.oracle.com/javase/8/docs/technotes/guides/language/assert.html**
* **http://javarevisited.blogspot.ro/2012/01/what-is-assertion-in-java-java.html**
* **http://stackoverflow.com/questions/11415160/how-to-enable-the-java-keywordassert-in-eclipse-program-wise**