**Tehnici de programare fundamentale**

**Tema 4**

**Restaurant Management System**

Pentek Tamas

Grupa: 30228

**1. Obiectivul temei**

Obiectul principal al temei este crearea unei aplicatii pentru un restaurant care poate sa aiba trei tipuri de utilizatori: administrator, chelner si bucatar. Fiecare utilizator are acces la diferite date si pot sa fac diferite operatii. Aplicatia are o interfata grafica prin care administrator poate sa adauge, sa modifice si sa stearga un meniu; chelnerul poate sa faca o comanda si sa creeze o factura; iar bucatarul este notificat de fiecare data cand trebuie sa pregateasca ceva mancare.

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

Utilizatorul aplicatiei cand deschide aplicatia prima data trebuie „sa faca log in”, adica trebuie sa aleaga ce tip de utilizator este: administrator, chelner sau bucatar. Fiecare tip de utilizator poate sa faca operatii diferite, de exemplu un administrator paote sa adauga un nou meniu, dar un chelner nu are voie sa faca asa ceva. Administratorul este utlizitorul cu cele mai multe operatii, poate sa actualizeze orice meniu si poate sa vada orice meniu din restaurant. Chelnerul are mai putine posibilitati, poate sa faca o comanda si are voie sa vada toate comenzile facute in restaurant. Practic, bucatarul nu poate sa faca nici o operatie, el primeste doar informatii despre comenzi, adica efectiv mancare pe care trebuie sa pregateasca.

Utilizatorul este ajutat prin mai multe guidelineuri, de exemplu: cele trei tipuri de utilizator sunt afisate cu culori diferite, prin aceasta avem o sansa mai mica ca un utilizator sa faca log in unde nu trebuie, unde nu are voie. Un alt feedback pentru utilizator este o fereastra cu un mesaj care apare cand utilizatorul a introdus ceva gresit sau nu a introdus nici o valoare.

Utilizatorul poate sa aleaga din trei operatii mai complexe: operatii cu meniuri, daca este vorba despre un administrator; operatii cu comenzi si cu facturi, daca utilizatorul este un chelner; iar daca este vorba de un bucatar atunci poate sa faca o singura operatie: sa citeasca ce fel de mancare trebuie sa gateasca.

In ceea ce urmeaza, sunt prezentate cateva operatii ca o diagrama use-case si prin descriere use-case sub forma unei liste, pe care daca utilizatorul urmareste va efectua operatia dorita. Fiindca avem 3 tipuri de utilizator, avem o diagrama use-case pentru fiecare tip.

**Use Case: Adaugare meniu**

**Primary Actor: administrator**

**Main Success Scenario:**

1. Utilizatorul deschide aplicatia
2. Utilizatorul face log in ca un administrator
3. Apasa butonul „Add new menu”
4. Dupa accea va aparea 2 butoane: „Base Product” si „Composite Product”
5. Utilizatorul apasa butonul „Composite Product”
6. Administratorul introduce in tabelul respectiv numele si pretul elementelor din ce e compus meniul
7. Utilizatorul introduce numele meniului si apasa butonul „Add”
8. Apare o fereastra cu mesajul urmator: „New menu added successfully!”
9. Administratorul apasa „OK” si meniul a fost adaugat

**Alternative Sequences:**

a.) Administratorul face log in ca un chelner:

1. Utilizatorul apasa butonul „Back” dupa care ajunge in pagina principala
2. Selecteaza butonul „Administrator”
3. Utilizatorul continua cu punctul 3. din Main Success Scenario

b.) Administratorul introduce un meniu care exista deja:

1. Administratorul apasa butonul „OK”
2. Apare un mesaj de eroare: „Duplicate Menu Item!”
3. Utilizatorul apasa butonul „OK”
4. Schimba datele introduse si apasa butonul „OK”
5. Utilizatorul continua cu punctul 8. din Main Success Scenario

**Use Case: Adaugare o comanda noua**

**Primary Actor: chelner**

**Main Success Scenario:**

1. Utilizatorul deschide aplicatia
2. Face log in ca si „Waiter/Waitress”
3. Chelnerul selecteaza butonul „Add new order”
4. Selecteaza meniurile si dupa fiecare meniu selectat apasa butonul „Add”
5. Selecteaza numarul mesei
6. Apasa butonul „Finish”
7. Apare o fereastra cu mesajul urmator: „Order succesfully added!”
8. Chelnerul apasa butonul „OK” si comanda a fost adaugata cu succes

**Alternative Sequences:**

a.) Chelnerul genereaza o factura:

1. Utilizatorul face pasii pana la punctul 5. din Main Success Scenario
2. Selecteaza optiunea „Compute Bill”
3. Apasa butonul „Finish”
4. Apare o fereastra cu mesajul urmator: „Order successfully added and bill created!”
5. Chelenrul apasa butonul „OK” si comanda a fost adaugat cu succes, factura a fost generata

b.) Chelnerul apasa butonul „Finish” fara a selecta si a adauga un meniu:

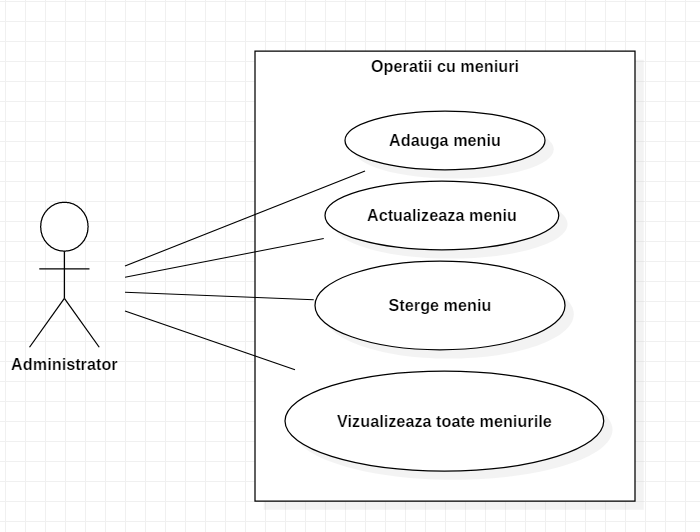
1. Apare un mesaj de eroare: „Error! Order contains nothing!”
2. Apasa butonul „OK”
3. Utilizatorul continua cu punctul 4. din Main Success Scenario

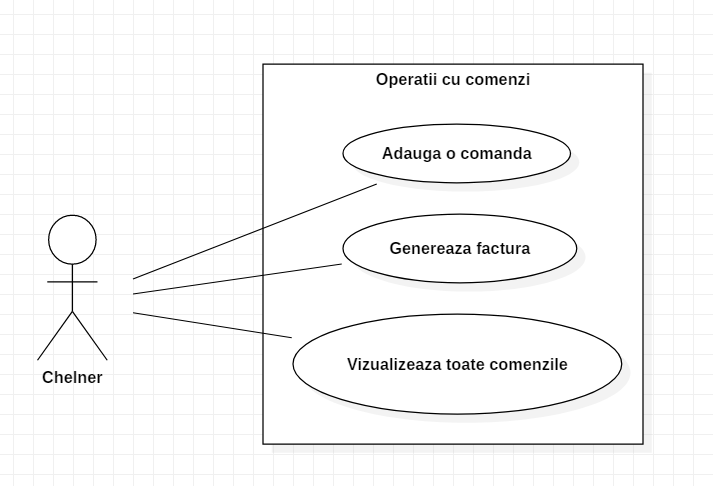
**Use Case: Vizualizeaza meniurile comandate**

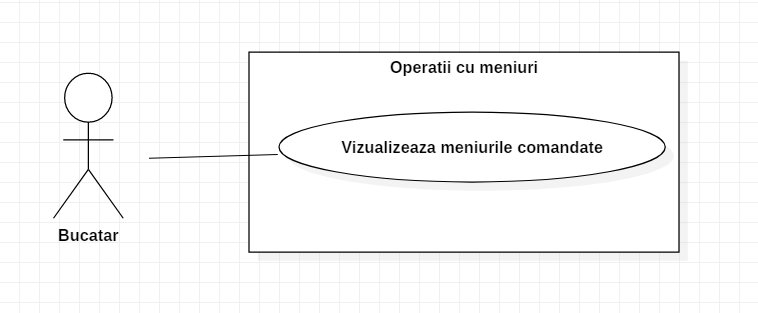
**Primary Actor: bucatar**

**Main Success Scenario:**

1. Utilizatorul face log in ca si bucatar, apasand butonul „Chef”
2. Apare o lista cu meniuri comandate







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

Acest proiect este impartit in 4 pachete. Pachetele sunt facute in asa fel incat sa respecte proiectarea Layered Architecture. Clasele respecta proiectarea OOP, prin structura si prin componente, variabile si operatii folosite.

Primul pachet este RestaurantProject.ManagementSystem.DataLayer care contine 2 clase: FilWriter si RestaurantSerializator. Clasa FileWriter interactioneaza cu fisirere, creeaza un fisier si inchide un fisier. Acesta este folosit cand se genereaza o factura care este slavata pe hard disk. Clasa RestaurantSerializator este folosit la serializarea si deserializarea meniului restaurantului. Cand se deschide aplicatia are loc deserializarea, adica aducerea datelor din fisier cu extensia .ser in aplicatie. Cand se inchide aplicatia are loc serializarea datelor, adica salvarea meniului intr-un fisier.

Al doilea pachet este RestaurantProject.ManagementSystem.BusinessLayer in care gasim 4 clase si 2 interfete. Interfata RestaurantProcessing contine metodele care sunt implementate de clasa Restaurant. Aceste metode sunt folosite de exemplu pentru a genera o factura, pentru a adauga o comanda noua, pentru a adauga un nou meniu. Clasa Restaurant are doau atribute foarte importante: o lista cu meniurile si un Map cu comnezile facute. Interfata cealalta este MenuItem care foloseste Composite Design Pattern, prin care celalate doua clase BaseProduct si CompositeProduct implementeaza aceasta interfata. BaseProduct reprezinta elementele de baza din care sunt preparate meniurile. CompositeProduct reprezinta un meniu intreg, fiecare obiect de tip CompositeProduct este alcatuit la randul lui dintr-o lista de MenuItem, adica un Composite Product poate da fie alcatuit din produse de baza si din alte produse compuse. Clasa Order contine informatii despre comanda, de exemplu data comenzii si masa la care s-a facut comanda.

Al treilea pachet este RestaurantProject.ManagementSystem.PresentationLayer, in acest pachet avem 4 clase. Avem o clasa separata pentru fiecare tip de utilizator si inca o clasa in care se face legatura intre aceste. Clasa AdministratorGraphicalUserInterface contine interfata grafica pentru un utilizator de tip administrator si toate operatiile pe care poate sa faca un administrator. In aceasta clasa gasim foarte multe elemente grafice si pe langa asta aici facem legatura intre Model si Presentation, aici apelam metodele care de exemplu introduc un nou meniu in lista de meniuri, genereaza o factura.

Clasa ChefGraphicalUserInterface contine interfata grafica pentru un utilizator de tip chelner, si aici gasim toate metodele apelate din clasa Restaurant pentru a face o noua comanda sau pentru a vizualiza toate comenzile. Atat clasa AdministratorGraphicalUserInterface cat si clasa ChefGraphicalUserInterface are un contstructor cu un parametru de tip RestaurantProcessing, aceste doau clase sunt create in clasa MainWindow care are un obiect de tip Restaurant, aceste obiect este transmis in cele doua clase, prin care ne asiguram ca atat administratorul cat si chelnerul, bucatorul sa lucreze cu acelasi restaurant, sa nu avem probleme ca de exemplu administratorul introduce un nou meniu si chelnerul nu—l vede pentru ca la chelner am folosit un alt obiect de tip Restaurant. Clasa ChefGraphicalUserInterface este o clasa destul de mica, contine putine elemente de grafica, bucatarul avand o singura operatie: sa vizualizeze toate meniurile comandate.

Ultimul pachet este RestaurantProject.ManagementSystem.Main in care avem singura clasa, MainClass, care contine metoda main() de unde este pornita aplicatia.

Din figura urmatoare rezulta ca pachetul Main depinde de toate celelalte pachete, ceea ce este absolut normal, fiindca fara celelalte pachete nu putem rula aplicatia, daca rulam fara un pachet nu vom avea rezultatul dorit. Un alt lucru ce rezuta de pe figura este faptul ca pachetul BusinessLayer depinde de pachetul DataLayer, pentru ca fara acest pachet BusinessLayer nu poate sa faca crearea fisierilor si inchiderea fisierilor. PresentationLayer depinde de DataLayer pentru ca fara acest pachet nu putem sa facem serializarea si deserializarea, ceea ce ar insemna ca la fiecare pornire de aplicatie ar trebui sa introducem tot meniul.

Diagrama UML de pachete arata in felul urmator:

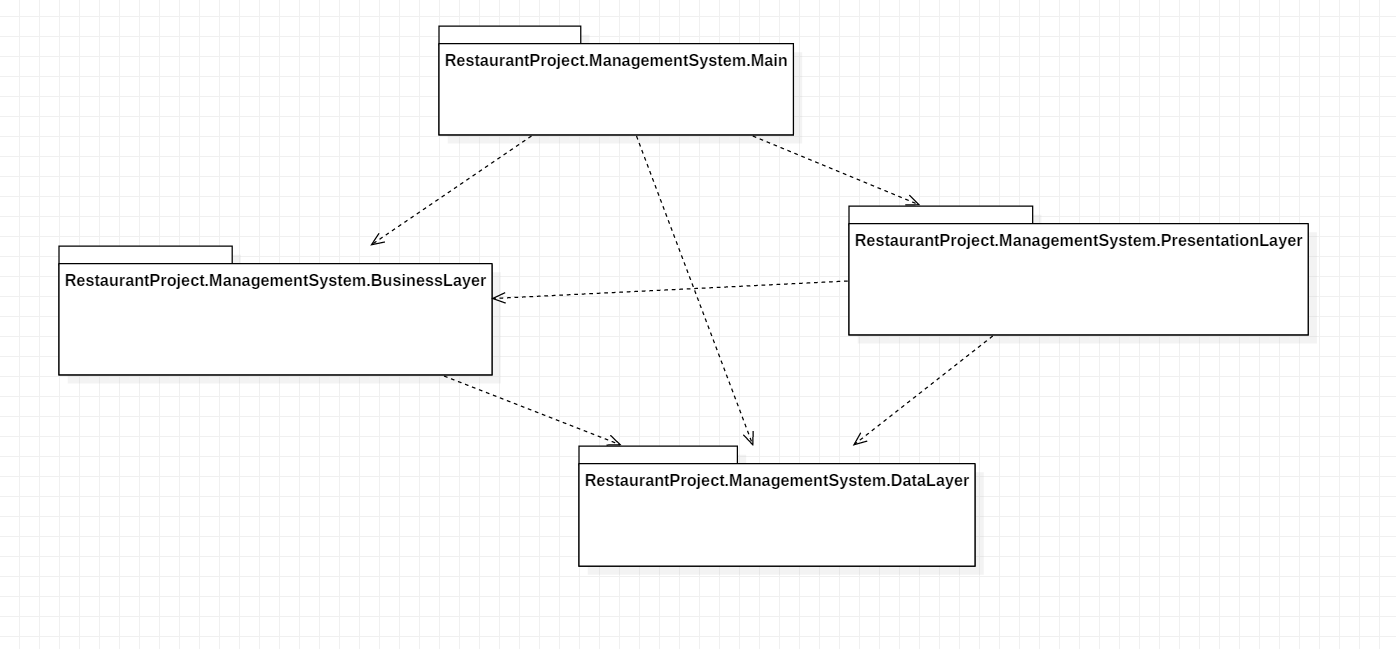
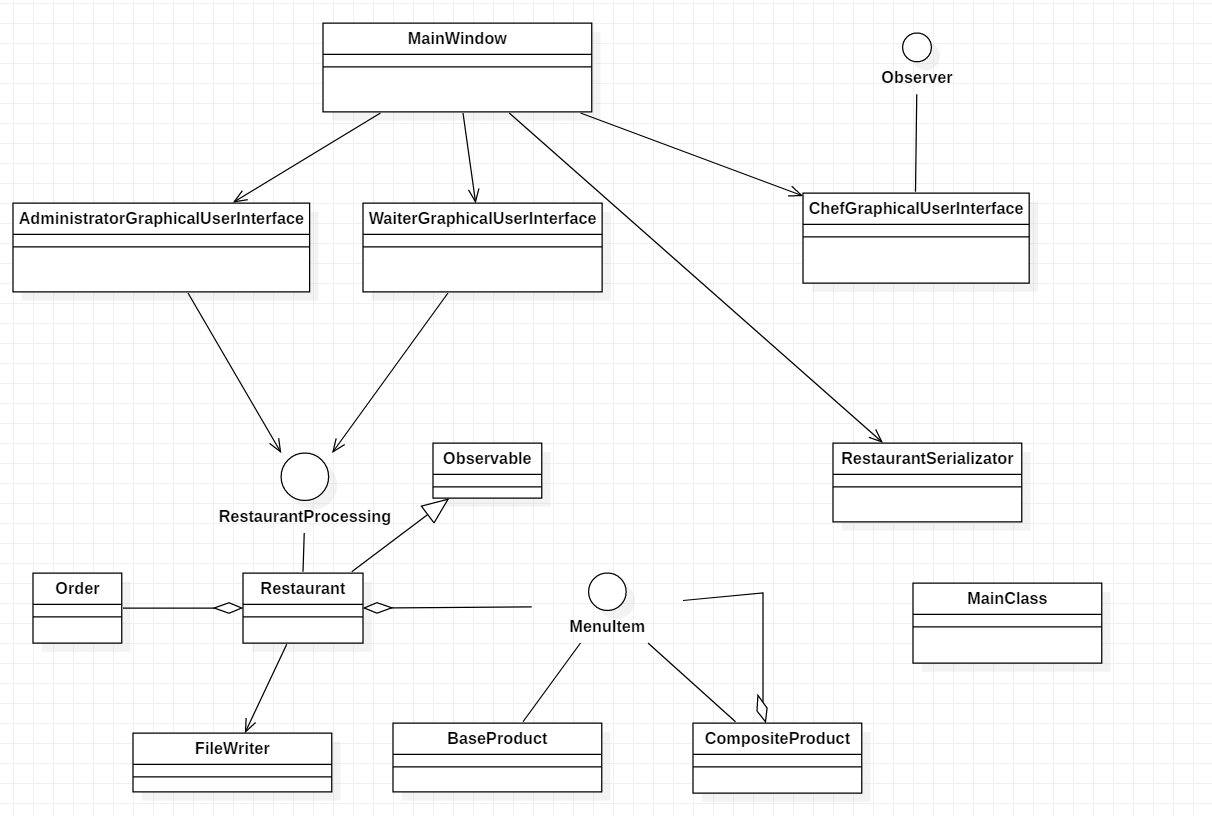


Diagrama UML de clase este prezentat pe figura urmatoare.



**4. Implementare**

O clasa importanta, din punctul de vedere a stocarii datelor, mai ales meniul restaurantului este clasa RestaurantSerializator in care avem 2 metode care face serializarea si deserializarea datelor. Acest lucru este posibil prin faptul ca interfata MenuItem extinde interfata Serializable, prin care va fi posibil serializarea si deserializarea meniului. Altfel ar fi foarte inutil aplicatia, deoarece fiecare data cand pornim aplicatia ar trebui sa introducem toate meniurile in lista de meniuri ceea ce ar fi foarte costisitor, dar prin serializare si deserializare nu trebuie sa pierdem timpul ca acesta. Metoda serialization arata in felul urmator:

**public** **void** serialization(List<MenuItem> menus) {

**try** {

FileOutputStream file = **new** FileOutputStream(**this**.filename);

ObjectOutputStream outputObject = **new** ObjectOutputStream(file);

**for** (MenuItem m : menus) {

outputObject.writeObject(m);

}

outputObject.close();

file.close();

} **catch** (IOException e) {

System.***out***.println("InputOutput exception - serialization");

}

}

Metoda prima data creeaza un obiect de tip FileOutputStream pe baza numele fisierului care este salvat in variabila filename. Dupa aceea este creat un obiet de tip ObjetOutputStream pe baza obiectului de tip FileOutputStream. Urmatoarea instructiune scrie fiecare meniu din lista de meniuri in fisierul respective, iar la final se inchide fisierele. Serialziarea poate arunca o exceptie de tipul IOException de accea trebuie sa tratam si cazul acesta: daca apare vreo problema afisam un mesja de eroare.

O alta metoda importanta este metoda createNewOrder din clasa Restaurant.

**public** **void** createNewOrder(Order key, List<MenuItem> value) {

**assert** key != **null** && value != **null** && **this**.getSizeOrder() >= 0;

**assert** isWellFormed();

**int** sizePre = **this**.getSizeOrder();

orders.put(key, value);

**for** (MenuItem m : value) {

**if** (m **instanceof** CompositeProduct) {

**this**.~~setChanged~~();

**this**.~~notifyObservers~~(**new** CompositeProduct(m.getName()));

}

}

**int** sizePost = **this**.getSizeOrder();

**assert** sizePost == sizePre + 1;

**assert** **this**.contains(key, value) == **true**;

**assert** isWellFormed();}

Acesta creeaza pe baza unei comanda si unei liste cu meniuri o noua comanda. Comanda este slavata intr-un Map in care putem sa salvam valori de tip (cheie, valoare). In acest caz cheia este un obiect de tip Order, mai exact in clasa Order este suprascrisa metoda hashcode(), care genereaza un hashcode si practice acesta va fi cheia. Valoarea reprezinta continutul comenzii, adica meniurile care au fost comandata la masa respectiva. Clasa Restaurant extinde clasa Observable, prin metodele clasei Observable, bucatarul va fi notificat cand un o comanda contine un Composite Product. Prima data verificiam daca este vorba despre un Composite Product, in cazul in care este doar un Base Product, bucatarul nu are ce se pregateasca. Daca este un Composite Product atunci semnalizam o schimbare si notificam Observer-ul, care este bucatarul, ca s-a produs o schimbare, si anume a aparut un nou Composite Product.

Clasa ChefGraphicalUserInterface, adica bucatarul implementeaza interfata Observer si prin metoda update() este actualizat o lista care contine meniurile pe care bucatarul trebuie sa o pregateasca. In clasa MainWindow este adaugat la un obiect de tip Restaurant un observer, prin care bucatarul observa orice meniu adaugat de catre chelnerul.

**public** **void** update(~~Observable~~ o, Object arg) {

**if** (arg **instanceof** CompositeProduct) {

**this**.index = **this**.*TEMP*++;

String element = **this**.index + ". " + ((CompositeProduct) arg).getName();

foods.add(element);

Collections.*sort*(foods);

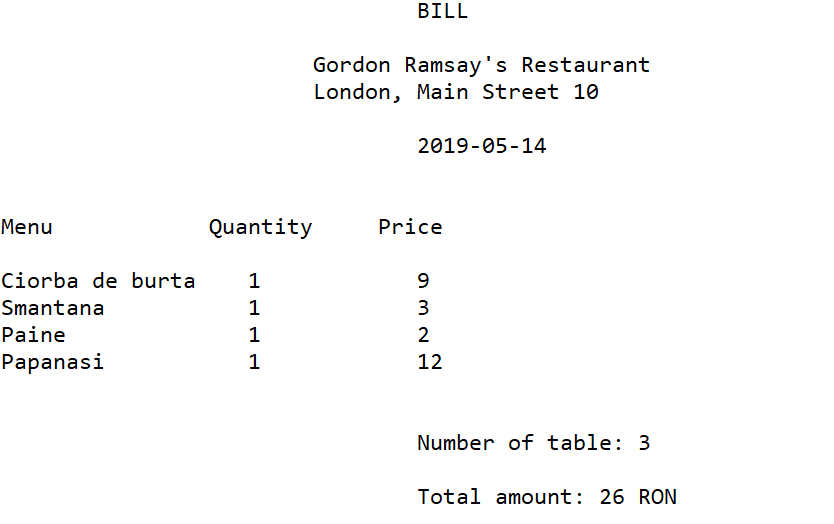
foodList.setListData(foods.toArray());

updateUI();

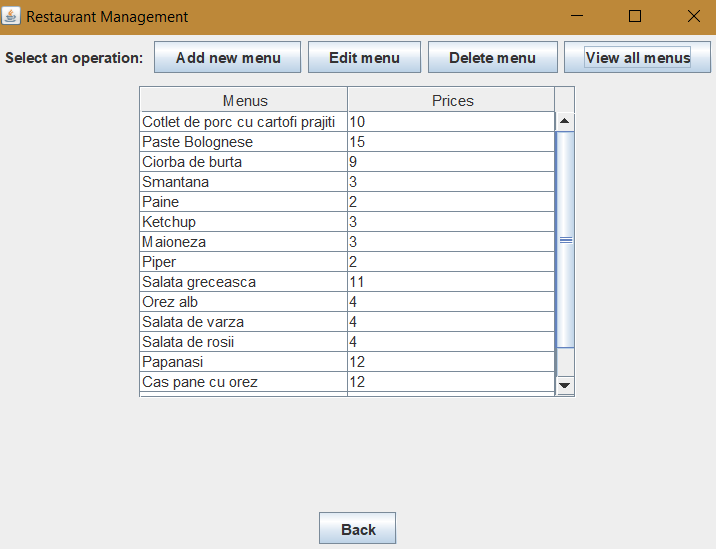
}

}

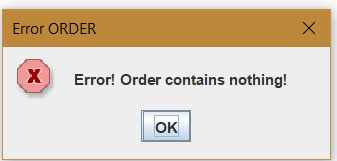
Un chelner are posibilitatea sa genereze o factura pe baza comenzii. Acesta se face prin metoda generateBill() si rezultatul, adica factura este salvata pe hard disk si arata in felul urmator:



Daca utilizatorul este un administrator atunci are posibilitatea sa vada toate meniurile. Meniurile sunt afisate intr-un tabel cu numele si cu pretul. Acest tabel arata in felul urmator:



Daca un chelner din greseala vrea sa faca o comanda care nu contine nici un meniu, se deschide o fereastra si apare un mesaj de eroare.



Utilizatorul care are cele mai multe posibilitati este administratorul, el poate sa adauge un meniu, sa stearga un meniu, sa updateze un meniu sau sa vizualizeaza toate meniurile. Aceste operatii sunt foarte simple de facute, administratorul are o interfata grafica prin care aceste operatii devin foarte simple. Metoda care sterge un meniu din lista de meniuri se afla in clasa Restaurant si arata in felul urmator:

**public** **void** deleteMenuItem(MenuItem m) {

**assert** **this**.getSizeMenu() > 0 && **this**.contains(m) == **true**;

**assert** isWellFormed();

**int** sizePre = **this**.getSizeMenu();

menus.remove(m);

**int** sizePost = **this**.getSizeMenu();

**assert** sizePost == sizePre - 1;

**assert** **this**.contains(m) == **false**;

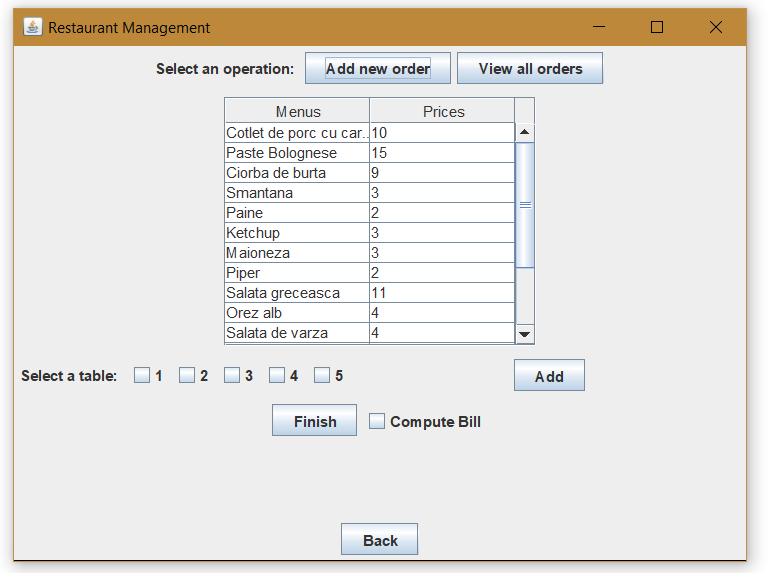
**assert** isWellFormed();

}

Pe langa codul functional avem si niste asserturi care sunt folosite ca si preconditii si postconditii si invariantul clasei, o metoda „well formed” care verifica mereu atributele clasei Restaurant ca la fiecare moment sa avem date corecte. Preconditiile si postconditiile sunt folosite pentru ca clasa Restaurant este implemetata folosind metoda Design by Contract.

Stergerea meniului se face foarte simplu, apeland metoda listei de stergere cu un paramentru, care este meniul care dorim sa stergem.

Interfata grafica folosita de un chelner este user-friendly, fiecare chelner poate sa foloseasca fara probleme si daca greseste ceva, aplicatia da ceva feedback prin care singur rezolva problema. Interfata arata in felul urmator:



**5. Concluzii**

Aceasta aplicatie este foarte folositoare, mai ales daca utilizatorii sunt persoane care lucreaza la un restaurant. Aplicatie reprezinta o versiune mai simpla a unor aplicatii pe care si restaurantele mai mari le folosesc. Aceasta tema a fost folositoare deoarece am recapitulat notiunile legate de serializarea si deserializarea datelor, am invatat cum functioneaza clasa Observable si interfata Observer, cum trebuie sa proiectam cu tehnica Design by Contract.

Aceast aplicatie se poate dezvolta cu foarte mult, se mai pot adauga si alte operatii, de exemplu

sa calcualm la sfarsitul zilei cat a fost venitul restaurantului sau sa facem si o extensie care calculeaza salarul fiecarui angajat. Oalta inbunatatire ar fi sa facem o metoda de log in, astfel fiecare angajat are acces la datele corespunzatoare.

**6. Bibliografie**

* Pentru diagrame StarUML
* HW4\_Tutorial\_Hashing\_In\_Java
* https://stackoverflow.com/questions/11415160/how-to-enable-the-java-keyword-assert-in-eclipse-program-wise
* https://docs.oracle.com/javase/7/docs/api/javax/swing/JTable.html
* https://docs.oracle.com/javase/7/docs/api/java/util/Observable.html
* https://docs.oracle.com/javase/7/docs/api/java/util/Observer.html
* https://howtodoinjava.com/java/io/java-write-to-file/