Universitatea Tehnica Cluj-Napoca

Facultatea Automatica si Calculatoare

Specializare Calculatoare si Tehnologia Informatiei

Tehnici fundamentale de programare

*~Tema 4~*

Proiect Restaurant Management System

Nume: Ștef David-Alexandru

Grupa: 30224

Cuprins:

1. Cerința Proiect …………………………………………………………. 2
2. Analiza Cerinței si Idei de Implementare.….…………………………… 3
3. Organizarea in pachete …………………………………………………. 4
4. Implementarea Cerinței ………………………………………………… 4
5. Diagrame ……………………………………………………………….. 9
6. Concluzii ………………………………………………………………. 10
7. Bibliografie ……………………………………………………………. 11

**1. Cerința Proiect**

Această temă consă în implementarea unui sistem de management al restaurantelor. Sistemul ar trebui să aibă trei tipuri de utilizatori: administrator, chelner și bucătar. Administratorul poate adăuga, șterge și modifica produsele existente din meniu. Chelnerul poate crea o nouă comandă pentru un tabel, poate adăuga elemente din meniu și poate calcula factura pentru o comandă. Bucătarul este notificat de fiecare dată când trebuie să gătească mâncarea comandată printr-un chelner. Se presupune, de asemenea, că sistemul este folosit de un singur administrator, un chelner și un bucătar.

Obiectivele principale ale acestei teme:

* Implementarea diagramei clasei din specificațiile temei;
* Interfața grafică: fereastra pentru administrator operations, fereastră pentru waiter operations și fereastră pentru Chef user;
* Salvarea informațiilor din clasa restaurantului într-un fișier utilizând serializarea. Încărcarea informațiilor la pornirea aplicației;
* Proiectare prin contract: condiții prealabile și postcondiții în interfața RestaurantProcessing. Implementați-le în clasa Restaurant folosind instrucțiunea de afirmare. Definirea unei invariante pentru restaurantul de clasă;

Alte obiective:

* Definiți interfața RestaurantProcessing care conține operațiunile principale care pot fi executate de chelner sau administrator, după urmează:
* Administrator: crează element nou al meniului, șterge element din meniu, editează element din meniu, vede meniul;
* Chelner: crează comandă nouă, calculează prețul pentru o comandă, editează elemente din meniu, vede comenzile;
* Utilizați modelul de proiectare compus pentru definirea clasei MenuItem, BaseProduct și CompositeProduct
* Utilizați modelul de design Observator pentru a notifica bucătarul de fiecare dată când se adaugă o nouă comandă care conține un produs compus
* Implementați clasa Restaurant utilizând o colecție JCF predefinită care utilizează o structură de date hashtable. Cheia hashtable va fi generată pe baza ordinii de clasă, care poate avea asociate mai multe MenuItems
* Utilizați un JTable pentru a afișa informații legate de restaurant

**2. Analiza Cerinței si Idei de Implementare**

Pentru această temă, am create trei funcționalități diferite, bazate pe utilizatorul care le folosește: Administrator, Chelner sau Bucătar. Aplicația are 3 View-uri diferite, bazate pe cei 3 utilizatori și oferă operații diferite:

*Administratorul:*

* Poate **adăuga** element nou în Meniu. Acest element poate fi un produs de bază, caz în care utilizatorul trebuie să ofere prețul sau poate fi un produs compus, în acest caz prețul nu este necesar, dar utilizatorul trebuie să ofere o listă a elementelor din meniu care sunt parte a produsului compus.
* Poate **edita** elementele existente din Meniu: poate schimba numele sau descrierea, de asemenea să schimbe prețul, dacă elemental este un produs de bază sau să schimbe lista de elemente din meniu, dacă este un produs compus.
* Poate **șterge** un element din Meniu. Bineînțeles, daca Administratorul șterge un produs de bază și există un produs compus care este format doar din produsul de bază, produsul compus va fi șters de asemenea.
* Pentru **adaugarea** unui element avem ca si conditii ca datele introduse de catre administrator sa fie valide si sa fie completate obligatoriu campul “nume” si campul “pret”. In caz contrar aplicatia il va informa in acest sens.
* Pentru **editarea** unui element este necesar ca articolul introdus sa existe, in caz contrar aplicatia il va informa pe administrator in acest sens. De asemenea, conditiile de la adaugare trebuie respectate si aici.
* Pentru **stergerea** unui element este necesar sa existe articolul introdus, iin caz contrar aplicatia il va informa pe administrator in acest sens. De asemenea, conditiile de la adaugare trebuie respectate si aici.

*Chelnerul:*

* Poate **adăuga** o comandă nouă, unde trebuie să ofere un ID, tabel și o listă a elementelor din Meniu care au fost comandate.
* Poate solicita aplicației să genereze **prețul** pentru o comandă existentă
* Poate genera un **bon** pentru o commandă existentă în format .txt, unde se pot vedea ID-ul comenzii, data, numarul mesei, produsele comandate si pretul total.
* Pentru **adaugarea** unei comenzi este necesar sa existe produsul comandat in Meniu. De asemenea, campurile ID si numarul mesei sunt obligatorii si chelnerul trebuie sa introduca date valide in acestea. Cazurile in care ID-ul sau masa sunt ocupate deja, nu sunt introduse sau sunt introduse eronat, vor fi semnalate de catre aplicatie.

*Bucătarul:*

* Utilizatorul bucătar nu poate efectua nicio acțiune în această fereastră, dar **este** **anunțat** când chelnerul adaugă o comandă care constă într-un produs compus.

**3. Organizarea in pachete**

**3.1. Pachetul** **businessLayer**

În pachetul businessLayer am definit clasele ProdusDeBaza, ProdusCompus, ArticolMeniu, Comanda, Restaurant si Interface InterfaceRestaurant. Aceste clase reprezintă datele care sunt gestionate în această aplicație.

**3.2. Pachetul** **dataLayer**

Acest pachet include clasele FileWriterTxt și SerializatorRestaurant, utilizate pentru scrierea în fișiere .txt și pentru descărcarea respectiv încărcarea in fisierul Restaurant.ser.

**3.3. Pachetul** **presentationLayer**

Acest pachet conține cele trei clase responsabile pentru generarea și manipularea operațiunilor din cele trei tipuri de ferestre care fac parte din această aplicație. AdministratorGUI, ChefGUI și WaiterGUI. Există, de asemenea, o clasă View, care leagă toate aceste ferestre împreună și le initializeaza.

**3.4. Pachetul test**

Conține o singură clasă: MainClass care definește principala metodă care pornește aplicația.

**4. Implementarea cerinței**

**4.1. Clasa ArticolMeniu**

Aceasta clasa reprezinta produsul din meniul restaurantului care are ca atribute numele sau si pretul. Este o clasa care sta la baza aplicatiei, utilizata in toate celelalte clase intr-un fel sau altul. Aceasta clasa implementeaza interfata Serializable prin care se face serializarea informatiilor legate de articol. De asemenea, ea face Override metodei „equals()” care este rescrisa pentru a compara doua produse in mod corect si metodei „hasCode()” care ajuta la gasirea unui „cod” unic pentru fiecare variabila.

**4.2. Clasa ProdusDeBaza**

Aceasta clasa reprezinta un produs de baza care are doar cele doua atribute: numele articolului si pretul, iar ea extinde Clasa ArticolMeniu.

**4.3. Clasa ProdusCompus**

Aceasta clasa reprezinta de asemnea un produs insa in plus fata de cele doua atribute amintite in clasele precedente aceasta mai are ca atribut un ArrayList de tipul ArticolMeniu, produsele care exista deja in meniu si care compun obiectul Clasei ProdusCompus. De asemenea, aceasta clasa extinde ArticolMeniu.

**4.4. Clasa Comanda**

Aceasta clasa implementeaza interfata Serializable care este utilizata la stocarea unei comenzi in fisierul.ser. Un obiect de tipul comanda are ca atribute, id-ul comenzii, numarul mesei la care s-a efectuat o comanda, timpul la care s-a realizat(de tipul „Date”) si un ArrayList de tipul ArticolMeniu care reprezinta produsele care au fost comandate. De asemenea, clasa face Override metodei „equals()”, utilizata pentru a verifica egalitatea a doua metode in mod corect si a metodei „hasCode()”, utilizata pentru a completa id-ul si numarul mesei in mod unic. Pentru a se putea vedea produsele comandate se implementeaza metoda „showItems()” care afiseaza numele numele fiecarui produs din cadul comenzii.

**4.5. Interfata InterfaceRestaurant**

Aceasta interfata declara metodele care au rolul de a raspunde cerintelor fiecarui utilizator si anume: adaugarea unui articol in meniu („adaugaArticol()”), stergerea unui articol din meniu („stergeArticol()”), actualizarea uni articol („actualizeazaArticol()”), crearea unei noi comenzi („creareComanda()”), calcularea pretului total al comenzii („calculeazaPretComanda()”), metoda care elibereaza bonul pentru o comanda („creareFacturaComanda()”).

**4.6. Clasa Restaurant**

Aceasta clasa implementeaza interfata Serializable utilizata pentru stocarea si preluarea informatiilor despre comenzi si meniu si interfata InterfaceRestaurant facand Override metodelor declarate in interfata. Are ca variabile de instanta cate un ArrayList de comenzi si unul de articole, un HashMap care are ca si cheie un obiect de tipul comanda si valoare, un ArrayList de comenzi, un Obiect de tipul SerializatorRestaurant prin care se face serializarea si deserializarea datelor si un obiect de tipul FileWriterTxt utilizat pentru generarea bonului fiscal.

Metoda isValid() reprezinta invariantul clasei, metoda de tipul „well formed” care verifica ca listele de articole si comenzi sa nu fie nule si totodata sa se modifice in mod corect lungimea lor conform operatiei efectuate asupra fiecaruia.

In metoda adaugaArticol() se utilizeaza pentru prima data instructiunea assert verficand ca fiecare componenta utilizata pentru implementarea metodei sa existe sau valoarea ei sa fie corecta. Avem preconditia utilizand la inceput metoda isValid() mai apoi se parcurge lista de articole, iar daca articolul nu exista in meniu atunci el se va adauga, in cazul in care acesta exista deja metoda va returna „false”. Mai apoi se aplica serializarea listei de articole care se stocheaza in fiserul .ser, iar la final se apeleaza din nou metoda isValid() care reprezinta postconditia. Ca si exemplu de utilizarea a instructiunii assert, pre si postconditiei si a adaugarii articolului este atasat un model al codului:

*assert art != null;*

*assert isValid();*

*int dimens = 1 + articole.size();*

*for (ArticolMeniu articol : articole) {*

*if (articol.getNumeArticol().equals(art.getNumeArticol()))*

*return false;*

*}*

*articole.add(art);*

serial.writeToFile(articole);

assert articole.size() == dimens;

assert isValid();

Metoda sterge Articol() este pe acelasi model ca si metoda adaugaArticol() in ceea ce priveste utilizarea pre si postconditiilor, a instructiunii assert si a serializarii datelor despre articole. Ca diferenta, aceasta metoda parcurge lista de articole, iar daca articolul dat ca si parametru se gaseste in lista atunci acesta este adaugat intr-o lista auxiliara de articole, iar ulterior din lista de articole sunt sterse articolele din lista auxiliara.

Metoda actualizeazaArticol() este tot pe acelasi model ca si metodele precedente in ceea ce priveste utilizarea pre si postconditiilor, a instructiunii assert si a serializarii datelor despre articole. Ca diferenta, metoda verifica daca articolul exista in lista si in caz afirmativ i se va face o actualizare a pretului sau cu noul pret. In cazul in care articolul cautat nu exista metoda returneaza „false”.

Metoda creareComanda() Metoda actualizeazaArticol() este tot pe acelasi model ca si metodele precedente in ceea ce priveste utilizarea pre si postconditiilor, a instructiunii assert si a serializarii datelor despre comenzi. Aceasta verifica daca exista deja id-ul sau numarul mesei in lista, iar in caz negativ adauga comanda in lista de comenzi, in caz negativ returneaza „false”.

Metoda calculeazaPretComanda() parcurge lista de produse comandate ale comenzii si adauga pretul fiecarui produs la pretul total al comenzii pe care ulterior il returneaza.

Metoda creareFacturaComanda() creeaza bonul fiscal pentru o comanda folosind comanda, produsele comandate si metoda care calculeaza pretul total.

**4.7. Clasa SerializatorRestaurant**

Aceasta clasa are rolul de a realiza serializarea si deserializarea datelor. Clasa implementeaza metoda de serializare si deserializare a datelor. Se introduce numele fisierului .ser asupra caruia se efectuaza aceste operatii. Exemplu serializare:

*ObjectOutputStream obj = new ObjectOutputStream(new FileOutputStream(ser));*

*obj.writeObject(articole);*

*obj.close();*

Exemplu deserializare:

*ArrayList<ArticolMeniu> articole = new ArrayList<ArticolMeniu>();*

*ObjectInputStream objInput = new ObjectInputStream(new FileInputStream(ser));*

*articole = (ArrayList<ArticolMeniu>) objInput.readObject();*

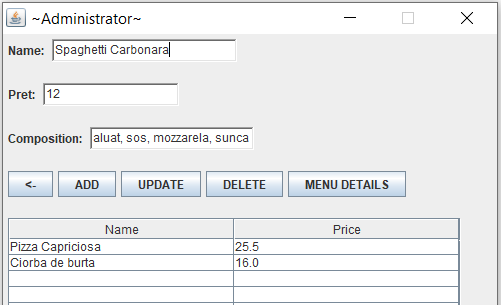
*...*

*return articole;*

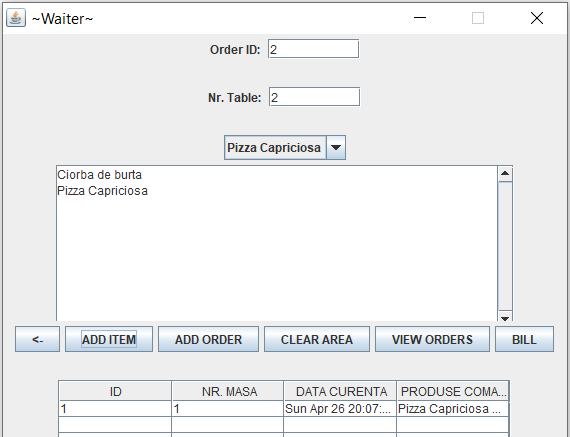
**4.8. Clasa** **FileWriterTxt** implementeaza metoda care genereaza bonul fiscal al comenzii in format .txt. Aceasta primeste ca si parametrii lista de articole, comanda si pretul ei si folosind aceste informatii se construieste un String pe modelul unui bon care contine datele despre comanda (id, numarul mesei, data), datele despre produsele comandate (numeles si pretul) si pretul total al comenzii.

In ceea ce urmeaza vor fi prezentate interfetele grafice cu utilizatorii, explicandu-se pentru fiecare utilizator modul de utilizare al interfetei sale conform cerintei

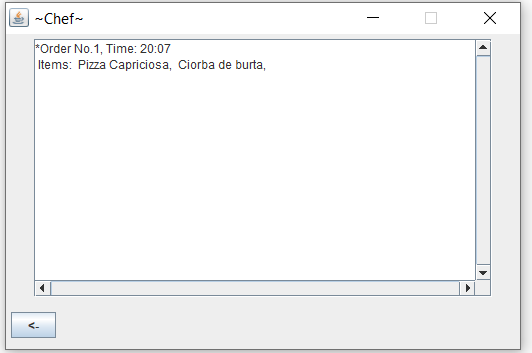
**4.9. Clasa AdministratorGUI** reprezinta o componenta a interfetei grafice cu utilizatorul si anume componenta care ii corespunde administratorului restaurantului. Astfel aceasta clasa folosind InterfaceRestaurant si implementand interfata ActionListener creeaza imaginea si functionalitatea fiecarui buton si textField de pe interfata grafica. Pentru a se realiza operatiile de adaugare, stergere si actualizarea se vor introduce date valide in cele 3 textField-uri. Introducerea compozitiei articolului este optionala. La apasarea butonului „ADD” comanda se va aduga articolul, la apasarea butonului „UPDATE” se va face actualizarea, iar la pasarea butonului „DELETE” se va sterge articolul. In urma efectuarii acestor operatii se va putea actiona butonul „MENU DETAILS” care afiseaza un tabel cu articolele cu existente is pretul lor.



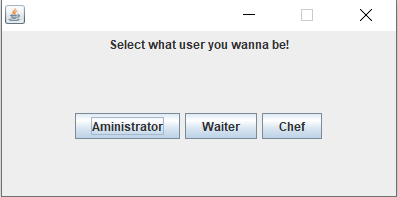
**4.10. Clasa** **WaiterGUI** reprezinta o componenta a interfetei grafice cu utilizatorul si anume componenta care ii corespunde chelnerului restaurantului. Astfel aceasta clasa folosind InterfaceRestaurant si implementand interfata ActionListener creeaza imaginea si functionalitatea fiecarui buton si textField de pe interfata grafica. De asemenea, WaiterGUI extinde clasa Observable si detine un ArrayList de tipul Observer cu ajutorul caruia se trimite „notificare” bucatarului in momentul in care este efectuata o comanda de catre chelner. Interfata grafica se utilizeaza in felul urmator, chelnerul introduce un id si un numar al mesei neocupat, apoi selecteaza din ComboBox produsele cerute de client care exista in Meniu. Se selecteaza produsul dorit si se actioneaza butonul „ADD ITEM”, la fel se face pentru fiecare produs al unei comenzi. Cand se doreste crearea comenzii dupa introducerea articolelor se actioneaza butonul „ADD ORDER”. Dupa efectuarea comenzii se va „curata” textArea in care au fost introduse produsele pentru a plasa o noua comanda. Actionand butonul „VIEW ORDERS” chelnerul poate vizualiza detaliile despre fiecare comanda. Dupa efectuarea comenzii se actioneaza butonul „BILL” pentru a se elibera bonul fiscal al comenzii (pentru aceasta trebuie introduse corect ID-ul si numarul mesei comenzii careia se doreste generarea bonului).



**4.11. Clasa** **ChefGUI** reprezinta o componenta a interfetei grafice cu utilizatorul si anume componenta care ii corespunde bucatarului restaurantului. Totodata, ChefGUI implementeaza interfata Observer facand *Override* la metoda *update()* aacestei interfete acest lucru realizeaza notificarea bucatarului in momentul in care se efectueaza o comanda. De asemenea bucatarul poate vedea comenzile pe care trebuie sa le pregateasca in ordinea in care s-au efectuat.

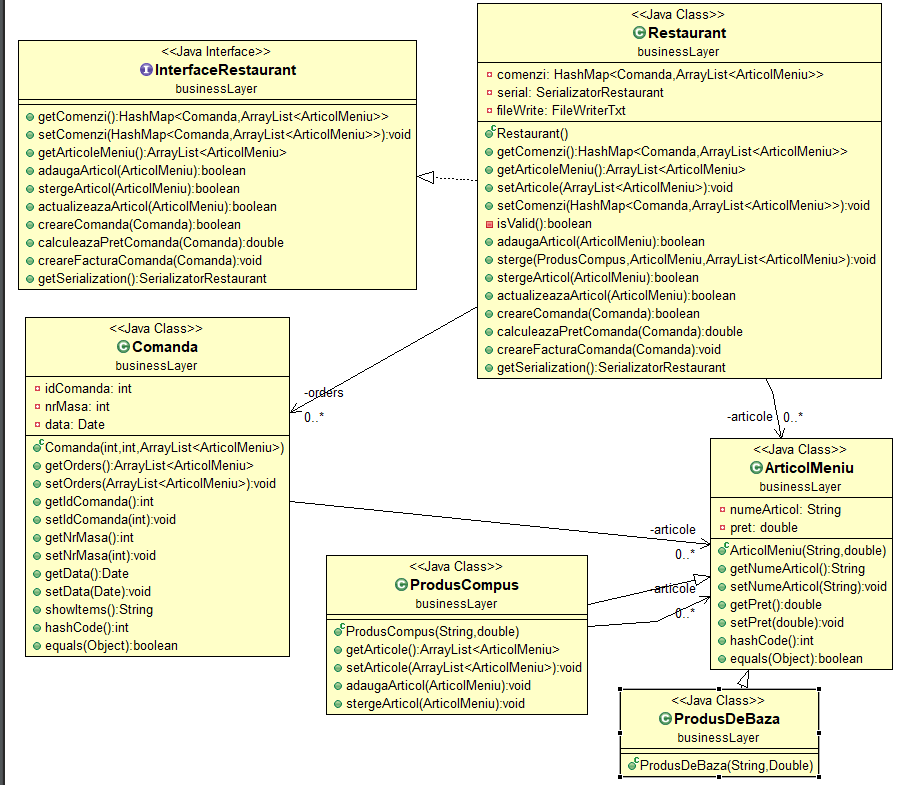


**4.12. Clasa View** reprezinta o componenta a interfetei grafice cu utilizatorul si anume componenta care ii corespunde chelnerului restaurantului. Astfel aceasta clasa implementand interfata ActionListener creeaza imaginea si functionalitatea fiecarui buton. Aceasta clasa creeaza prima fereastra care ii apare utilizatorului la rularea aplicatiei. Aceasta fereastra contine trei butoane “Administrator”, “Waiter”, “Chef”, iar in urma actionarii unui buton se va deschide fereastra corespunzatoare utilizatorului selectat.

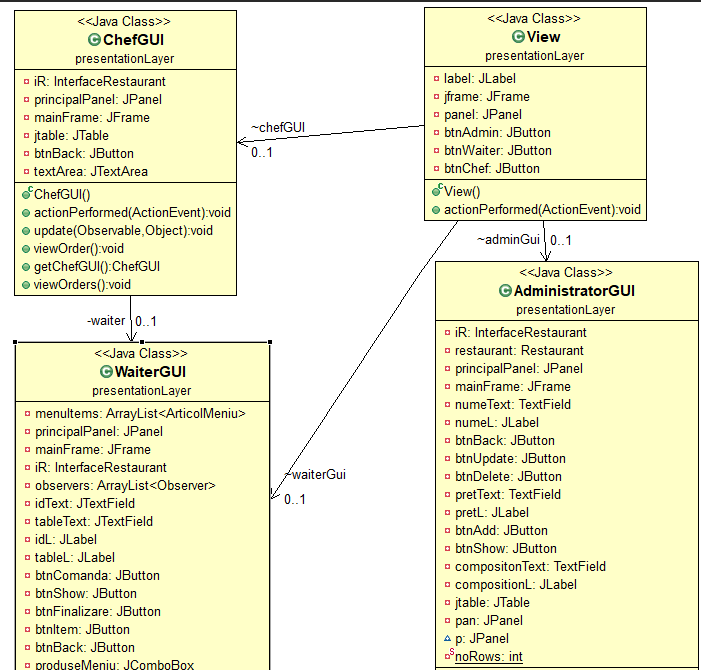


**5. Diagrame UML**

Package dataLayer:



Package presentationLayer:



**6. Concluzii**

In concluzie, acest proiect a aratat modul in care se utilizeaza modelele Composite Design Pattern is Observe, cum se utilizeaza serializarea si deserializarea asupra unui fisier de tip .ser pentru a se prelua si stoca date. Proiectul poate fi preluat si utilizat in mod corespunzator de catre un restaurant, deoarece functioneaza in mod corespunzator.

Desigur, proiectul ar putea fi extins și dezvoltat și mai mult. De exemplu, aplicația ar putea beneficia de un proces Log in, în care utilizatorul s-ar putea conecta ca Administrator, Chelner sau Bucătar, în loc să aibă toate cele trei ferestre împreună. De asemenea, aplicația ar putea fi programată să fie folosită de mai mult de un Administrator, Chelner sau Bucătar. În acest fel, devine chiar posibil să fie utilizată într-o situație din viața reală, ca un Restaurant Management System (sistem de management al unui restaurant).

**7. Bibliografie**

http://www.tutorialspoint.com/java/java\_serialization.htm

https://www.baeldung.com/java-serialization

https://www.geeksforgeeks.org/serialization-in-java/ o https://docs.oracle.com/javase/8/docs/api/java/io/Serializable.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 o https://intellij-support.jetbrains.com/hc/en-us/community/posts/207014815-How-toenable-assert

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