**RESTAURANTUL**

* Cerinta

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.

2. 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.

3. 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 temei:

Obiectivul temei este de a crea si de a programa o aplicatie care proceseaza meniul si comenzile unui restaurant. De asemenea obiectivul acestei teme este si de a implementa interfete simple si usor de utilizat prin intermediul carora se poate gestiona activitatea unui restaurant cu privire la meniul sau si comenzile sale.

Aplicatia gestioneaza actiunile effectuate intr-un restaurant fictiv si anume actiunile de creare, update si delete a unui menu-item si de creare, view si compute bill pentru o comanda.

* Conceptul de serializare a datelor

Serializarea este macanismul prin care starea obiectelor poate fi citită din memorie şi trimisă printr-un flux către o destinaţie (o altă aplicaţie, fişier, reţea, etc.).

Procesul invers prin care un obiect este reîncărcat în memorie prin citirea acestuia dintr-un flux se numeşte deserializare. Pentru serilaizarea şi deserializarea obiectelor limbajul java pune la dispoziţie fluxurile ObjectInputStream şi ObjectOutputStream.

Pentru implementarea mecanismului de serializare / deserializare se poate folosi şi interfaţa Externalizable ce permite controlul mai fin al procesului de salvare şi încărcare a obiectelor. Această interfaţă extinde interfaţa Serializable şi defineşte metodele writeExternal() şi readExternal() în cadrul cărora trebui implementat codul pentru scrierea şi citire obiectelor. Clasa ce implementează această interfaţă trebuie obligatoriu să aibă definit constructorul implicit (având specificatorul de acces public).

Dacă se doreşte ca un anumit atribut al unui obiect serializat să nu fie salvat atunci se poate folosi cuvântul cheie transient în faţa declaraţiei atributului respecitv.

* Analiza problemei

Proiectul a fost realizat cu ajutorul a 16 clase impartite in 4 pachete :

BusinessLayer, DataLayer, PresentationLater, Main. Pachetul BusinessLayer contine urmatoare clase (BaseProduct, CompositeProduct, MenuItem, Order, Restaurant, IObservable, IObserver). In acest pachet se implementeaza clasele de baza, clasele esentiale pentru implementarea proiectului, respective clasa meniurilor de baza si cea a meniurilor compuse precum si clasa MenuItem care este extinsa de BaseProduct si CompositeProduct.

Pachetul DataLayer contine doar o clasa si anume RestaurantSerializator, clasa cu ajutorul careia se va realiza serializarea respective deserializarea datelor.

Pachetul PresentationLayer este pachetul in care se afla clasele cu ajutorul carora realizam interfata grafica. Pachetul prezinta urmatoarele 7 clase : AdministratorGraphicalUserInterface, ChefGraphicalUserInterface, WaiterGraphicalUserInterface, MainInterface, Tabela, TabelaOrder, si Invoice. Primele 4 clase sunt pentru realizarea interfetei grafice iar ultimele clase sunt clasele cu ajutorul carora cream JTable-uri pentru vizualizarea datelor respectiv clasa Invoice cu ajutorul careia putem creaza o factura in format .txt.

Pachetul Main contine clasa App care de fapt este main-ul programului nostru.

* Proiectare

DIAGRAMA DE CLASE

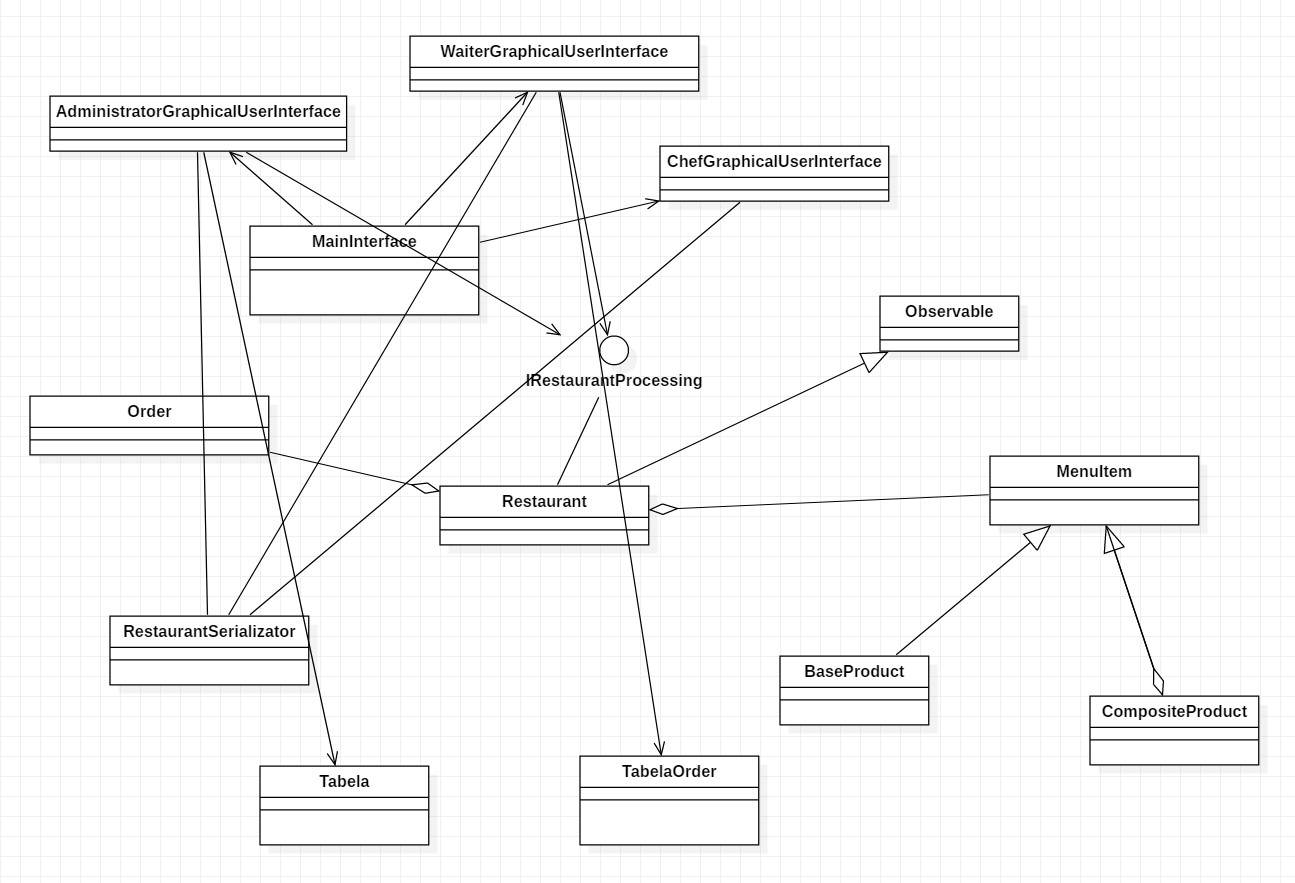
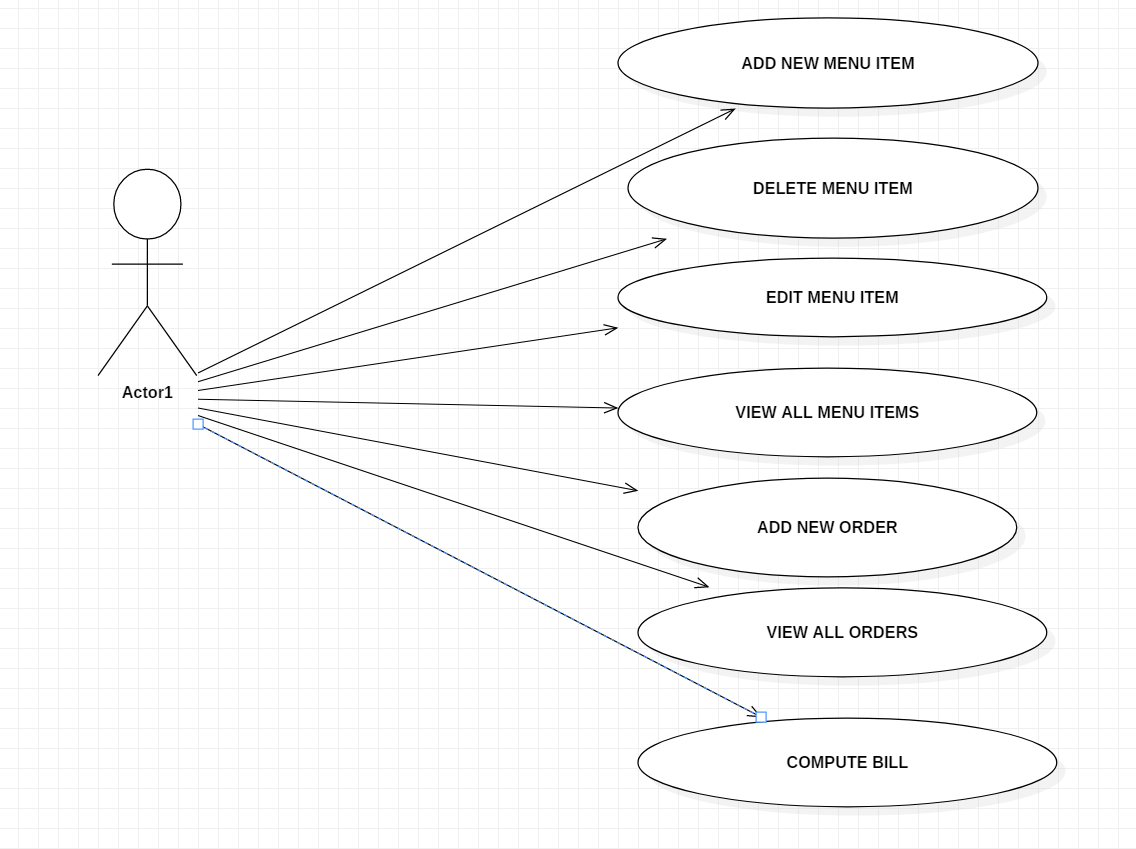


Diagrama USE-CASE



* Implementare

Asa cum am precizat si anterior, programul este format din 16 clase dintre care o clasa este main-ul programului.

* Clasa Base Product

In cadrul acestei clase avem un constructor special cu ajutorul caruia reusim sa instantiem un obiect de tipul Base Product cu atributele sale si anume nume, cantitate, pret.

private String numeBaseProduct;

private int cantitateBaseProduct;

private int pretBaseProduct;

public BaseProduct(String numeBaseProduct, int cantitateBaseProduct, int pretBaseProduct) {

this.numeBaseProduct = numeBaseProduct;

this.cantitateBaseProduct = cantitateBaseProduct;

this.pretBaseProduct = pretBaseProduct;

}

* Clasa Composite Product

La fel ca si clasa de mai sus, clasa CompositeProduct extinde clasa MenuItem si implementeaza metoda computePrice. Pe langa BaseProduct, CompositeProduct mai are un atribut in plus si anume o lista de MenuItem. Clasa aceasta are 2 constructori cu ajutorul caruia diferentiem produsele.

In momentul in care se doreste crearea unui produs composite product care are atributele asemanatoare cu cele ale BaseProduct se apeleaza primul constructor, iar in momentul in care se doreste crearea unui CompositeProduct ce contine o lista de BaseProduct atunci se apeleaza cel de-al doilea constructor.

private String nume;  
private int cantitate;  
private int pret;  
private ArrayList<MenuItem> items;  
  
public CompositeProduct(String name,int cant,int pret) {  
 this.nume = name;  
 this.cantitate = cant;  
 this.pret = pret;  
 items = new ArrayList<MenuItem>();  
 //CompositeProduct c = new CompositeProduct(name,cant,pret);  
 //items.add(c);  
}  
  
public CompositeProduct(String name, int cant, int pret, ArrayList<MenuItem> x) {  
 this.nume = name;  
 this.cantitate = cant;  
 this.pret = pret;  
 this.items = new ArrayList<MenuItem>();  
 items = x;  
}

* Clasa MenuItem

Aceasta clasa este o clasa abstracta care nu are nici un atribut ci doar un constructor si antetul unei metode computePrice. Aceasta metoda va fi implementata de toate clasele care vor mosteni clasa MenuItem, respectiv BaseProduct si CompositeProduct. Singura diferenta de implementare este ca BaseProduct este un simplu obiect pe cand CompositeProduct are ca si atribut o lista de obiecte de tipul BaseProduct sau de tipul sau insusi si tocmai de aceea calcularea pretului difera.

public abstract class MenuItem implements Serializable {  
 public MenuItem() {  
 }  
 public abstract int computePrice();}

* Clasa Restaurant

Aceasta clasa este una dintre cele mai importante clase din proiect deoarece ea este responsabila de implementarea metodelor necesare pentru realizarea proiectului. Clasa dispune de 2 ArrayList-uri cu ajutorul carora se stocheaza comenzile si menu item-urile si de un hash-map in interiorul caruia salvam order-urile.

* Metoda createNewMenu

public void createNewMenu(MenuItem item) {  
  
 assert item != null;  
 System.*out*.println(item.toString());  
 int alex = items.size();  
 try {  
  
 items.add(item);  
 } catch (Exception e) {  
 System.*out*.println(e.getMessage());  
 }  
 if (alex + 1 == items.size()) {  
 JOptionPane.*showMessageDialog*(null, "SUCCES ! ");  
 } else {  
 JOptionPane.*showMessageDialog*(null, "FAIL ! ");  
 }  
 assert alex+1 == items.size();  
}

Cu ajutorul assert-urilor aceasta metoda verifica pre si post conditiile necesare adaugarii in lista a unui nou meniu. In cazul in care inserarea a avut loc cu success se va afisa pe ecran un buton de success, iar in cazul in care a esuat se va afisa un buton de fail.

* Metoda deleteMenuItem

public void deleteMenuItem(String item) {

assert item != "";

int alex = items.size();

int res = 0;

for (MenuItem x : items) {

if (x instanceof BaseProduct) {

if (((BaseProduct) x).getNumeBaseProduct().equalsIgnoreCase(item)) {

items.remove(x);

res++;

JOptionPane.showMessageDialog(null, "SUCCESFUL DELETE ! ");

}

} else {

if (x instanceof CompositeProduct) {

if (((CompositeProduct) x).getNume().equalsIgnoreCase(item)) {

items.remove(x);

res++;

JOptionPane.showMessageDialog(null, "SUCCESFUL DELETE ! ");

}

}

}

assert alex - 1 == items.size();

}

if (res == 0) {

JOptionPane.showMessageDialog(null, "THIS MENU ITEM ISN'T IN OUR LIST !");

}

Aceasta metoda primeste ca parametru un String reprezentand numele meniului de sters. Nestiind tipul menuItem-ului, in interiorul metodei se parcurge toata lista de menuItem pana cand se gaseste numele produsului primit ca parametru. In momentul in care s-a gasit se face remove din lista la item-ul acela.

Daca numele produsului primit ca parametru de metoda delete nu se gaseste in lista de meniuri inseamna ca acesta nu a fost niciodata introdus si se va afisa un mesaj de eroare.

* Metoda editMenuItem

public void editMenuItem(String item, int cantitate, int pret) {  
 assert item != "";  
 int res = 0;  
 for (MenuItem x : items) {  
 if (x instanceof BaseProduct) {  
 if (((BaseProduct) x).getNumeBaseProduct().equalsIgnoreCase(item)) {  
 int alu = items.indexOf(x);  
 items.set(alu, new BaseProduct(item, cantitate, pret));  
 res++;  
 JOptionPane.*showMessageDialog*(null, "SUCCESFUL UPDATE BASE PRODUCT! ");  
 }  
 } else {  
 if (x instanceof CompositeProduct) {  
 if (((CompositeProduct) x).getNume().equalsIgnoreCase(item)) {  
 int balu = items.indexOf(x);  
 items.set(balu, new CompositeProduct(item, cantitate, pret));  
 res++;  
 JOptionPane.*showMessageDialog*(null, "SUCCESFUL UPDATE COMPOSITE PRODUCT! ");  
 }  
 }  
 }  
  
 }  
 if (res == 0) {  
 JOptionPane.*showMessageDialog*(null, "THIS PRODUCT ISN'T IN OUR BASE");  
 }  
  
}

Aceasta metoda este responsabila de editarea unui meniu din lista noastra de meniuri. Metoda functioneaza pe principiul metodei de delete, adica in primul rand cauta numele meniului in lista de meniuri. In cazul in care gaseste meniul efectueaza asupra lui un update, iar in cazul in care nu gaseste meniul in lista de meniuri inseamna ca acesta nu exista si nu se poate efectua nici o modificare asupra lui, trebuie sa se creeze un meniu cu acest nume.

* Clasa Invoice

Cu ajutorul acestei clasa se creeaza o chitanta in formatul .txt. Clasa Invoice are o sinura metoda, createInvoice, care primeste doi parametrii, ambii de tipul int. Primul reprezinta id-ul mesei care are comanda respectiv iar al doilea reprezinta suma care trebuie achitata de masa care a facut acea comanda.

Tot ceea ce este afisat in interiorul apelului outputStream.println() se va afisa in fisierul text.

public class Invoice {

/\*\*

\* Creates the invoice.

\*

\* @param Id the id

\* @param sumaDeAchitat the suma de achitat

\*/

public static void createInvoice(int Id, int sumaDeAchitat) {

String fileName = "Chitanta.txt";

try {

PrintWriter outputStream = new PrintWriter(fileName);

outputStream.println("CHITANTA");

outputStream.println("Masa cu numarul " + Id + " are de platit suma de " + sumaDeAchitat + " lei");

outputStream.close();

} catch (FileNotFoundException e) {

e.printStackTrace();

}

}

}

* Clasa Tabela

Cu ajutorul acestei clase vom reusi sa afisam datele din tabele in interfata grafica prin intermediul unui JTable.

public class Tabela {  
  
 public Tabela() {  
  
 }  
  
 public String[][] retrieveProperties(ArrayList<MenuItem> menuItems) {  
  
 int i = 0;  
  
 String[][] data = new String[50][50];  
  
 for (MenuItem item : menuItems) {  
 if (item instanceof BaseProduct) {  
 data[i][0] = "Base Product";  
 data[i][1] = ((BaseProduct) item).getNumeBaseProduct();  
 data[i][2] = String.*valueOf*(((BaseProduct) item).getCantitateBaseProduct());  
 data[i][3] = String.*valueOf*(((BaseProduct) item).getPretBaseProduct());  
  
 } else if (item instanceof CompositeProduct) {  
 data[i][0] = "Composite Product";  
 data[i][1] = ((CompositeProduct) item).getNume();  
 data[i][2] = String.*valueOf*(((CompositeProduct) item).getCantitate());  
 data[i][3] = String.*valueOf*(((CompositeProduct) item).getPret());  
 }  
 i++;  
 }  
 return data;  
 }  
  
 public void tableItems(String[][] dataItems) {  
 JFrame frame = new JFrame("VIEW MENU ITEMS");  
  
 String[] columnNames = { "Tip produs", "Nume produs", "Cantitate produs", "Pret produs" };  
  
 JTable menu = new JTable(dataItems, columnNames);  
 menu.setBounds(30, 40, 300, 300);  
  
 JScrollPane scrollPane = new JScrollPane(menu);  
 frame.add(scrollPane);  
 frame.setSize(500, 400);  
 frame.setVisible(true);  
 }  
  
}

* Concluzii

Concluzia acestei teme este ca am creat o aplicatie usor de utilizat de catre orice utilizator. Implementand aceasta tema am invatat sa lucrez mult mai bine cu interfetele grafice si cu aceasta ocazie am invatat notiuni noi, cum ar fi serializarea. Pe langa serializare am lucrat si cu hash-map intr-un proiect destul de mare, lucru nemaifacut pana acum. Desigur ca am intalnit dificultati, dar am reusit sa le depasesc. Pe langa acestea am folosit JTable cu ajutorul caruia am facut display-ul produselor mele.

Ca si dezvoltari ulterioare as adauga:

* Reimplementarea interfetei
* Posibilitatea de a exista useri, accesul facandu-se diferit (administrator si normal user)
* Adaugarea mai multor informatii referitoare atat la produsele de baza cat si a produselor compuse
* Reimplementarea chitantei, aceasta facandu-se cu ajutorul unui document PDF personalizat
* Etc.
* Bibliografie
* <https://www.geeksforgeeks.org/serialization-in-java/>
* <https://www.tutorialspoint.com/java/java_serialization.htm>
* <https://beginnersbook.com/2014/07/java-serialization/>
* <https://docs.oracle.com/javase/8/docs/api/java/util/HashMap.html>
* <http://coned.utcluj.ro/~salomie/PT_Lic/4_Lab/HW4_Tema4/HW4_Tutorial_Hashing_In_Java.pdf>
* <https://stackoverflow.com/questions/33401739/how-do-i-display-an-array-on-a-jtable-in-java>
* <https://stackoverflow.com/questions/33401739/how-do-i-display-an-array-on-a-jtable-in-java>