DOCUMENTATIE TEMA 4

RESTAURANT

Nume si prenume: Fagadar Ionela-Catalina

Grupa: 30226

Profesor Laborator: Marin Oana

Contents

[**1.Obiectivul temei** 5](#_Toc8218281)

[**2.Analiza problemei, modelare, scenarii, cazuri de utilizare** 5](#_Toc8218282)

[**3.Proiectare** 6](#_Toc8218283)

[**3.1. Decizii de proiectare** 6](#_Toc8218284)

[**3.2. Diagrame UML** 7](#_Toc8218285)

[**3.3. Structuri de date** 8](#_Toc8218287)

[**3.4. Proiectare clase** 8](#_Toc8218288)

[**3.5. Proiectare interfete** 9](#_Toc8218289)

[**3.6. Relatii** 9](#_Toc8218290)

[**3.7. Pachete** 9](#_Toc8218291)

[**3.8. Algoritmi** 9](#_Toc8218292)

[**3.8.0. Algoritmul de deserializare** 9](#_Toc8218293)

[**3.8.1. Algoritmul de producere a facturii** 10](#_Toc8218294)

[**3.8.2. Algoritmul de interogare a bazei de date** 10](#_Toc8218295)

[**4. Implementare** 11](#_Toc8218296)

[**4.1. Clasa MenuItem – clasa Abstracta** 11](#_Toc8218297)

[**4.2. Clasa BaseProduct** 11](#_Toc8218298)

[**4.3. Clasa CompositeProduct** 11](#_Toc8218299)

[**4.4. Clasa Order** 11](#_Toc8218300)

[**4.5. Clasa Restaurant** 12](#_Toc8218301)

[**4.6. Clasa RestaurantSerializator** 12](#_Toc8218302)

[**4.7. Clasa Controller, View, AdministratorGraphicalUserInterface, BaseGUI, CompositeGUI, ChefGraphicalUserInterface, WaiterGraphicalUserInterface, ViewJTable** 12](#_Toc8218303)

[**4.8. Clasa Main** 12](#_Toc8218304)

[**5.Rezultate** 12](#_Toc8218305)

[**6.Concluzii si dezvoltari ulterioare** 12](#_Toc8218306)

[**7.Bibliografie** 13](#_Toc8218307)

# **1****.Obiectivul temei**

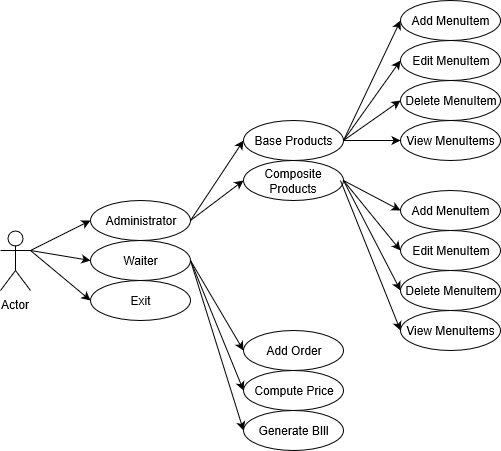
Sa se implementeze un sistem de management al unui restaurant. Sistemul ar trebui sa aiba 3 tipuri de utilizatori: Administratorul, Chelnerul si Bucatarul. Administratorul poate adauga, sterge sau modifica produsele existente din meniu. Chelnerul poate crea o noua comanda pentru tabela, adauga elemente din meniu si sa produca factura pentru comanda. Bucatarul e notificat de fiecare data cand tree sa gateasca o mancare comandata prin chelner.

|  |  |  |
| --- | --- | --- |
| Obiectiv | Descriere | Capitol |
| dezvoltarea de use case-uri si scenarii |  | 2 |
| alegerea structurilor de date |  | 3 |
| impartirea pe clase |  | 3 |
| dezvoltarea algoritmilor |  | 3 |
| implementarea solutiei |  | 4 |
| testare |  | 5 |

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

Programul este conceput astfel incat sa se poata introduce in meniu, produse de baza sau produse compuse din produse de baza, sa se poata edita produsele existente, sau sa se poata sterge. Totodata, se pot face comenzi de produse din meniu, se poate verifica pretul si se poate produce o factura pentru aceste comenzi.

La inceput, se va alege ce utilizator suntem: Administrator sau Chelner. Daca se alege administratorul, trebuie sa se aleaga pe ce produse se face operatia de adaugare sau editare: produse de baza sau produse compuse. Astfel, am ales ca produsele sa aiba un id deja calculat, dupa ultimul produs introdus. Daca id-ul se schimba si nu este unul unic, se va afisa un mesaj: „Id nu e unic”, iar daca este zero, se va afisa „Id trebuie sa fie diferit de zero!”. Daca nu se introduc toate datele necesare, se va afisa un mesaj „Introdu toate datele”. Pentru a se putea iesi din aplicatie, este necesar sa se apese butonul Exit si apoi butonul X al ferestrei principale. Altfel, nu se va putea inchide aplicatia.



# **3.Proiectare**

## **3.1. Decizii de proiectare**

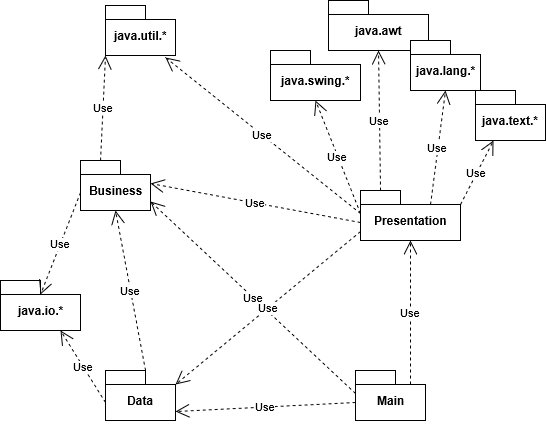
Am ales sa proiectez acest proiect in modul 3-tier architecture, cu layerele: Presentation Layer, Business Logic, Data Access, impartindu-l in 4 pachete: Business, Data, Main si Presentation. Am decis ca, clasele din pachetul Business sa fie accesate de orice pachet, astfel rezultand o buna functionare a proiectului. Am ales ca pachetul Data sa contina doar serializatorul pentru Resataurant, serializatorul tinand locul unei baze de date. Pachetul Presentation sa contina Controllerul si Interfetele grafice alese pentru a implementa aplicatia.

## **3.2. Diagrame UML**

Diagrama de clase a acestui proiect:

## 

Diagrama de pachete a acestui proiect:



## **3.3. Structuri de date**

Se folosesc doua structuri de date in acest proiect: Lista si HashTable. Lista este folosita in clasa CompositeProduct, deoarece un produs compus poate contine mai multe produse de baza, cat si in clasa Restaurant, unde se retin toate produsele introduse ( atat produsele de baza, cat si cele compuse) si toate comenzile depuse de Chelner. HashTabelul este folosit pentru a face corespondenta intre comanda si produsul comandat, astfel pastrandu-se in clasa Restaurant, o tabela cu comenzile si produsele aferente comenzilor.

## **3.4. Proiectare clase**

Se folosesc 5 clase pentru proiectarea acestei aplicatii.Singura clasa abstracta este clasa MenuItem care contine metodele ce trebuie implementate de cele doua clase: BaseProduct si CompositeProduct. Alta clasa este clasa BaseProduct care este reprezentata de un produs de baza ce are un id, un nume si un pret, clasa CompositeProduct ce are un id, un nume, si contine o lista de produse de baza. Clasa Order care e reprezentata de un id, o data la care se face comanda, si masa la care se face comanda. Clasa Restaurant reuneste toate clasele implementate, pentru a se putea face operatii pe acestea. Pentru memorarea datelor ce s-au efectuat de cand s-a creat aplicatia pana in prezent, se foloseste clasa RestaurantSerializator, care, la inceput este folosita pentru a deserializa datele si a le citi din fisier, apoi, la sfarsit, serializeaza datele si le scrie in acelasi fisier. Clasele din pachetul Presentation sunt clase ce se ocupa de Interfata grafica a acestei aplicatii.

## **3.5. Proiectare interfete**

Singura interfata este IrestaurantProcessing ce contine metode ce se pot efectua in aceasta aplicatie.

## **3.6. Relatii**

## **3.7. Pachete**

Proiectul e structurat in 4 pachete: Business, Data, Main si Presentation. Business contine clase si interfete ce ajuta la modelarea obiectelor din lumea reala, continand partea logica a acestei aplicatii. Pachetul Data contine o clasa in care se face deserializarea si serializarea datelor, acest pachet fiind important pentru a pastra in memorie datele de la infiintarea proiectului si pana in prezent. Pachetul Main contine clasa Main care porneste Aplicatia, iar pachetul Presentation contine clase ce extind clasa Jframe, clase responsabile de interfata grafica a acestui proiect.

## **3.8. Algoritmi**

Algoritmii utilizati pentru acest proiect sunt algoritmii utilizati pentru deserializare si pentru serializare, cat si algoritmii utilizati pentru producerea facturii.

## **3.8.0. Algoritmul de deserializare**

Acest Algoritm citeste date serializate dintr-un fisier serializat. Se creeaza un obiect de tipul FileInputStream, care primeste ca argument, numele fisierului de unde vrem sa citim date serializate. Se creeaza un alt obiect de tipul ObjectInputStream unde se adauga ca argument, celalalt obiect creat de tipul FileInputStream. Obiectului de tip Restaurant, care se vrea a fi citit, ii se atribuie Obiectul citit din fisier, cu ajutorul functiei: readObject(). Se inchid cele doua obiecte create, cu ajutorul functiei close(). Se returneaza obiectul de tip Restaurant citit.

**public** Restaurant deserialize(){

Restaurant r= **null**;

**try** {

FileInputStream fileIn = **new** FileInputStream("fisierR.ser");

ObjectInputStream in = **new** ObjectInputStream(fileIn);

r = (Restaurant) in.readObject();

in.close();

fileIn.close();

} **catch** (IOException i) {

i.printStackTrace();

**return** **null**;

} **catch** (ClassNotFoundException e) {

e.printStackTrace();

}

"Deserialized Restaurant...");

**return** r;

}

## **3.8.1. Algoritmul de producere a facturii**

Serializarea este asemanatoare cu deserializarea. Astfel, se creeaza un obiect de tipul FileOutputStream de aceasta data, care primeste, la fel, ca argument, numele fisierului in care vrem sa scriem date serializate, apoi se creeaza un alt obiect de tipul ObjectOutputStream, care primeste ca argument obiectul precedent creat de tipul FileOutputStream. Se scrie in fisier datele serialzate cu ajutaorul functiei writeObject(Restaurant), si se inchid cele doua obiecte cu functia close().

**public** **void** serialize(Restaurant r) {

**try** {

FileOutputStream fileOut =**new** FileOutputStream("fisierR.ser");

ObjectOutputStream out = **new** ObjectOutputStream(fileOut);

out.writeObject(r);

out.close();

fileOut.close();

System.***out***.printf("Serialized data is saved in fisierR.ser");

} **catch** (IOException i) {

i.printStackTrace();

}

}

## **3.8.2. Algoritmul de interogare a bazei de date**

In acest algoritm, se creeaza un obiect de tipul BufferedWriter pentru a scrie in fisier. Acest constructor primeste ca argument un FileWriter, care e construit cu numele fisierului unde vrem sa scriem factura („file.txt”). Se initializeaza variabile ca: prima, care e folosita pentru a verifica daca aceasta este prima comanda procesata din factura, price care contine pretul final al facturii, si i care este folosit pentru a memora pasul la care ne aflam. Astfel, pentru fiecare comanda pe care o avem, se verifica daca numarul mesei acestei comenzi corespunde cu numarul mesei pentru care se face comanda ( se primeste ca argument la functie). Daca corespunde, se verifica daca este prima comanda procesata, iar in caz afirmativ, se scriu in fisier urmatoarele informatii: data, masa, si nr comenzii. Pentru fiecare produs din meniu ce corespunde cu comanda, se scrie nr ei si toate informatiile legate de acel produs din meniu. Se scrie si pretul individual pentru produsul din meniu, iar pretul total se actualizeaza. Comanda se sterge din tabela cu comenzi si produse (HashTable), cat si din lista de comenzi. Variabila prima se face false, deoarece prima comanda s-a terminat de procesat. Se procedeaza la fel pentru toate comenziile. Daca in schimb, nr mesei unei comenzi nu corespunde cu nr mesei pentru factura, se incrementeaza pasul. Nu se incrementeaza pasul daca nr mesei de comanda corespunde cu numarul mesei pentru factura, deoarece comenzile sunt sterse la sfarsitul procesarii. La sfarsit, dupa ce s-au verificat toate comenzile, se scrie in fisier totalul facturii, si se inchide fisierul.

**public** **void** generateBill(**int** masa) {

BufferedWriter out;

**try** {

out = **new** BufferedWriter(**new** FileWriter("file.txt"));

**boolean** prima = **true**;

**int** price = 0;

**int** i=0;

**while**(i<**this**.getComenzi().size()){

**if** (**this**.getComenzi().get(i).getTable() == masa) {

**if** (prima) {

out.write("Data: "+**this**.getComenzi().get(i).getDate()+"\n");

out.write("Masa: "+masa+"\n");

out.write("COMANDA:\n");

}

//Order o = new Order(id,data,masa);

ArrayList<MenuItem> mancaruri = **this**.getTabelComenzi().get(**this**.getComenzi().get(i));

**if** (mancaruri!=**null**) {

out.write("NrComenzii: "+**this**.getComenzi().get(i).getOrderID()+"\n");

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

out.write(m.toString()+"\n");

}

out.write("Pret: "+computePrice(**this**.getComenzi().get(i))+"\n");

price+=computePrice(**this**.getComenzi().get(i));

}

**else** price+=0;

**this**.getTabelComenzi().remove(**this**.getComenzi().get(i));

**this**.getComenzi().remove(**this**.getComenzi().get(i));

prima = **false**;

}

**else** i++;

}

out.write("\nTotal:"+price+"\n");

out.close();

}

**catch** (IOException e) {

e.printStackTrace();

}

}

# **4. Implementare**

## **4.1. Clasa MenuItem – clasa Abstracta**

Metode abstracte: computePrice(), getId(), getProduseBaza(), setProduseDeBaza(List<BaseProduct>), setNume(String), setPret(int), addProduseBaza(BaseProduct), getNume(), getPret().

## **4.2. Clasa BaseProduct**

Atribute: id, nume, pret

Constructori , Getter, Setter, metoda toString(), computePrice() si celelalte metode ce trebuie implementate datorita extinderii clasei abstracte MenuItem.

## **4.3. Clasa CompositeProduct**

Atribute: id, nume, produseDeBaza

Constructori, Getter, Setter, metoda toString(), addProdusDeBaza(BaseProduct) si celelalte metode ce trebuie implementate datorita extinderii clasei abstracte MenuItem.

## **4.4. Clasa Order**

Atribute: id, date, table

Constructori, Getter, Setter, metoda toString(), hashCode() si equals(Object).

## **4.5. Clasa Restaurant**

Atribute: mancaruri, comenzi, tabelComenzi.

Metode: Constructori, addMenuItem( MenuItem), deleteMenuItem( MenuItem), editMenuItem( MenuItem), addComanda( Order, ArrayList<MenuItem), computePrice( Order), generateBill( int, String, int), CautaUltimulId(), cautaUltimulIdMenu(), generateBill( int), toString().

## **4.6. Clasa RestaurantSerializator**

Metode: serialize( Restaurant), deserialize().

## **4.7. Clasa Controller, View, AdministratorGraphicalUserInterface, BaseGUI, CompositeGUI, ChefGraphicalUserInterface, WaiterGraphicalUserInterface, ViewJTable**

Clase ce compun interfata grafica a aplicatiei, astfel: Controller-ul detine controlul interfetei grafice, adaugand ascultatori butoanelor din interfata grafica. Clasa View contine o fereastra cu 3 butoane in care se alege ce utilizator suntem, si astfel, ce fel de operatii de adaugare, editare, stergere sau afisare avem voie sa facem. Aceasta este prima fereastra ce se deschide atunci cand executam programul. Daca se alege Butonul pentru Administrator, se pot face adaugari, editari, stergeri si afisari de produse, pe produsele de baza sau cele compuse. Astfel, dupa apasarea butonului de Administrator, se deschide o fereastra cu 2 butoane, in care se alege ce produs vrem sa adaugam, sterge, sau edita: produs de baza sau produs compus. Pentru fiecare buton se afiseaza o noua fereastra cu casete text, etichete, butoane sau JcomboBox, in care trebuie introduse datele ce se vor procesa. Daca se alege butonul pentru Chelner, se vor deschide doua ferestre: una pentru a adauga o comanda, cu campurile respective, cu butoane pentru adaugare, pentru compunere pret si pentru generare de factura. Celalta fereastra este fereastra bucatarului, si este impartita in 2 parti: prima parte in care se scrie cand bucatarul pregateste o comanda, si a doua parte in care se scrie ce comenzi se afla in prezent neplatite. De fiecare data cand se apasa pe butonul de adaugare comanda, bucatarul este atentionat si astfel, se scrie pe ecran. Aceasta operatie se poate efectua datorita implementarii clasei Observer a clasei ChefGraphicalUserInterface, cat si a extinderii clasei Observable a clasei Restaurant. Atentionarea se face prin utilizarea unor functii mostenite, ca : SetChanged() in care se atentioneaza ca s-a produs o modificare, si a functiei: notifyOberver( Order).

Clasa ChefGraphicalUserInterface contine o metoda update( Observable arg0, Object arg1) care se executa de fiecare data cand se notifica Observatorul prin notifyObserver( Order), adica de ficare data cand se face o comanda.

Atentie! La inchiderea aplicatiei, se va apasa pe butonul de EXIT din prima fereastra aparuta, apoi pe butonul X al ferestrei, deoarece, este blocata iesirea din fereastra, pana nu se face serializarea prin apasarea pe butonul EXIT, butonul X al ferestrei luandu-si apoi proprietatile prestabilite .

## **4.8. Clasa Main**

Metoda main() ce porneste aplicatia

# **5.Rezultate**

Aceasta aplicatie descrie o solutie optima de a simula comenzile dintr-un restaurant, care isi poate schimba produsele din meniu oricand.

# **6.Concluzii si dezvoltari ulterioare**

In concluzie, consider ca acest program simuleaza un restaurant destul de folositor. Ca si dezvoltari ulterioare, s-ar putea schimba interfetele grafice deoarece nu sunt foarte user-friendly, si s-ar putea schimba modul in care se stocheaza datele de comanda si produsele din meniu, in HashTable.

# **7.Bibliografie**

* http://coned.utcluj.ro/~salomie/PT\_Lic/4\_Lab/HW4\_Tema4/HW4\_Tutorial\_Hashing\_In\_Java.pdf
* https://stackoverflow.com/questions/5509082/eclipse-enable-assertions?fbclid=IwAR3okKLWoKHYQq6jyt\_hcjwC1dklm8hjhbymxkPyFlbjVYhc-jYHkkvgzU8
* <https://stackoverflow.com/questions/696782/retrieve-column-names-from-java-sql-resultset>
* https://www.baeldung.com/java-observer-pattern?fbclid=IwAR0tsFUe\_7zd3F2DMmIJFdJ3aVjjHcrUSjbJNsP7-EQoq9Ah8uKzSsf9Cdo