**Documentație**

Disciplina

Tehnici de programare

**Tema 4**

**RESTAURANT MANAGEMENT SYSTEM**

**Pîrvu Andreea Gabriela**

An academic:2019-2020

Cuprins

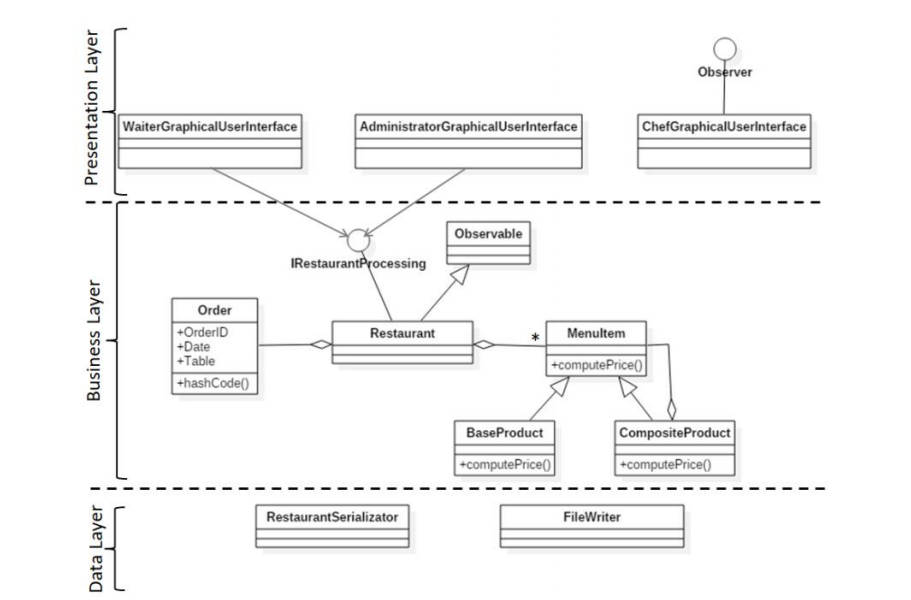
1. Obiectivul temei
2. Analiza problemei, modelare, scenarii, cazuri de utilizare
3. Proiectare
4. Implementare
5. Concluzii
6. Bibliografie

**1.Obiectivul temei**

* 1. **Obiectivul principal**

Se cere implementarea unui sistem de gestionare a restaurantelor. Sistemul ar trebui să aibă trei tipuri de utilizatori: administrator, ospătar și bucătar. Administratorul poate adăuga produse de bază sau produse compuse din cele de bază, de asemenea poate șterge produsele, însă dacă produsul de baza face parte dintr-un produs compus, acesta nu poate fi șters decât după ștergerea produsului compus și poate modifica prețul produselor existente din meniu. Chelnerul poate crea o nouă comandă și poate calcula factura pentru fiecare comandă. Bucătarul este înștiințat de fiecare dată când se primește o nouă comandată printr-un ospătar.

Luați în considerare sistemul de clase din diagrama de mai jos:



Pentru a simplifica aplicația, puteți presupune că sistemul este folosit de un singur administrator, un chelner și un bucătar și nu este nevoie de un proces de autentificare. Rezolvați următoarele:

1. Definiți interfața RestaurantProcessing care conține principalele operațiuni care pot fi executate de chelner sau de administrator, după cum urmează:

• Administrator: creează un articol de meniu nou, șterge elementul de meniu, editează elementul de meniu

• Ospătar: creează o nouă comandă; calculați prețul pentru o comandă; generați factura în format .txt.

2. Definiți și implementați clasele din diagrama de clasă prezentată mai sus:

• Utilizați modelul de design compus pentru definirea claselor MenuItem, BaseProduct și CompositeProduct

• Utilizați modelul de proiectare a observatorului pentru a anunța bucătarul de fiecare dată când se adaugă o nouă comandă care conține un produs compozit.

3. Implementați Restaurantul clasei folosind o colecție JCF predefinită care utilizează o structură de date hashtable. Cheia hashtable va fi generată pe baza comenzii clasei, care poate avea asociate mai multe meniuri de meniu. Folosiți JTable pentru a afișa informațiile legate de restaurant.

• Definiți o structură de tip <Map> pentru stocarea informațiilor referitoare la comandă în clasa Restaurant. Cheia Hărții va fi formată din obiecte de tip Order, pentru care se va suprascrie metoda hashCode () pentru a calcula valoarea hash din hartă din atributele Comenzii (OrderID, data etc.)

• Definiți o structură de tip Colecție care va salva meniul restaurantului. Alegeți tipul de colecție adecvat pentru implementarea dvs.

• Definiți o metodă de tip „bine formată” pentru clasa Restaurant.

• Implementați clasa folosind metoda Design by Contract (implicând condiții pre, post, invariante și afirmații).

4. Elementele de meniu pentru popularea obiectului Restaurant vor fi încărcate / salvate dintr-un fișier utilizând Serialization.

* 1. **Obiective secundare**

Pentru îndeplinirea scopului principal:

1. Crearea claselor necesare;
2. Stabilirea caracteristicilor și comportamentelor obiectelor necesare;
3. Implementarea metodelor necesare unei clase;
4. Extragerea informațiilor din interfața grafică;
5. Generarea rezultatelor în interfață precum și vizualizarea facturii în fișierul bills.txt
6. **Analiza problemei, modelare, scenarii, cazuri de utilizare**

**2.1 Analiza problemei**

Pentru acest proiect, am creat o interfață MenuItem pe care o implementează clasele BaseProduct și CompositeProduct și care trebuie să implementeze metodele din interfața MenuItem și anume:

int computePrice();

String getName();

void setName(String n);

void setPrice(int pret);

Practic, un BaseProduct este un element fără ingrediente. În timp ce un CompositeProduct este un element format din BaseProduct. Vom folosi modelul de design compus pentru a descrie aceste grupuri de obiecte în structuri de arbori pentru a reprezenta ierarhii întregi. În programul nostru, prețul unui produs compus va fi calculat în funcție de prețul articolelor din care sunt compuse.

* De exemplu:

- Avem: banane(preț: 4 ron), mere(preț: 5 ron), kiwi(pret: 6 ron), frisca(pret: 1 ron), ananas(pret: 7 ron), mango(pret: 9 ron).

- Vom crea un articol nou – salata de fructe , care va conține toate produsele de mai sus.

- Folosind metoda computePrice din clasa compositeProduct, vom serializa noul element din meniu sub forma: salata de fructe(pret: 32 ron).

În plus, administratorul restaurantului poate: adăuga, elimina sau edita elementele situate în meniu.

Acest program va genera, de asemenea, o fereastră pentru ospătarul restaurantului. Chelnerul poate: crea o comandă, poate genera factură într-un fișier.txt. Chelnerul poate crea o nouă comandă prin introducerea selectarea produselor care se cer.

**2.2 Modelarea**

Pentru gestionarea unui sistem de gestionare a comenzilor, trebuie să ne creăm clasele în conformitate cu tot ceea ce este necesar pentru a deține informațiile pe care le presupune un astfel de calcul. Prin urmare, programul constă în primul rând dintr-o clasele numite „BaseProduct” și „CompositeProduct” care stau la baza acestei aplicații, efectuând operații asupra acestor produse.

De asemenea avem clasele cu ajutorul cărora utilizatorul poate interacționa cu aplicația, clasele care formeaza Interfata Grafică a aplicației, fiind 3 ferestre: una pentru administrator, una pentru chelner și una pentru bucătar, fiecare cu comenzile corespunzătoare, conform cerinței.

* **Cerințe funcționale:**

Sistemul de gestionare a comenzilor trebuie sa functioneze corect.

Aceasta se intelege că realizarea unei comenzi trebuie sa fie corectă, în funcție de produsele disponibile in tabela cu produse disponibilă pentru vizualizare în fereastra administratorului, care cuprinde atât produse de bază cât și produse compuse. Totodată, prețul produselor compuse trebuie să fie calculate corect, precum și factura pentru fiecare comandă, în funcție de produsele introduse.

De asemenea, informatiile transmise de utilizator prin intermediul interfeței grafice trebuie interpretate corect, stocate corespunzator.

* Cerinte:

- adăugarea produselor de bază

-adăugarea produselor compuse din produsele de bază

- editarea prețului unui produs deja introdus

-ștergerea produselor astfel încât dacă acesta este un produs de baza care este folosit in alcătuirea unui produs compus, acesta nu poate fi șters până când produsul compus nu este șters

-vizualizarea tabelului cu produsele care sunt disponibile, sub forma(nume, preț, tip).

-adăugarea unei comenzi

-vizualizarea comenzii în tabelul corespunzător comenzilor, care este afișat în fereastra chelnerului.

-realizarea unui fișier de ieșire .txt cu facture comenzii

-înștiințarea bucătarului atunci când apare o nouă comandă.

-salvarea informațiilor din clasa Restaurant într-un fișier restaurant.ser folosind serialization.

**2.3 Use-cases**

Entry conditions: administratorul introduse produsele de bază precum și produsele compuse din podusele de bază, ospatarul introduce comenzile și solicita facturarea acestora, iar bucătarul este mereu anunțat atunci când apare o nouă comandă.

Exit conditions: programul insereaza produsele de baza și cele compuse, calculând prețul produselor compuse în funcție de produsele de bază pe care le conține și de prețul acestora. În cadrul unei comenzi, programul inserează produsele care se cer precum și cantitatea dorită. De asemenea, programul generează un fisier bills.txt unde calculeaza totalul comenzii. Totodata, programul notifică bucătarul atunci când se inserează o nouă comandă.

Flow of events:

1) Inserarea produselor de bază sau compuse

2) Inserarea unei comenzi

3) Se vizualizează rezultatele in tabelele din fereastra administratorului(tabela cu produse), respectiv fereastra chelnerului(tabela cu comenzi), unde se pot vizualiza produsele introduse precum și comenzile.

4) Se vizualizeaza fișierul de ieșire bills.txt unde este calculat prețul total al fiecărei comenzi.

1. **Proiectare**

Programarea Orientata pe Obiecte este o paradigmă de programare care utilizează obiecte și interacțiuni între acestea pentru a modela arhitectura unui program.

Aplicația creată respectă principiile OOP:

-orice obiect are un tip (este o instanță a unei clase);

-încapsulare;

-un obiect nu poate modifica starea alui obiect decât prin intermediul metodelor;

-abstractizarea (fiecare obiect are rolul unui actor abstract care iși poate modifica valoarea, poate realiza acțiuni și poate comunica cu alte obiecte din sistem fără a-și dezvălui implementarea facilităților);

-obiectele pot comunica între ele;

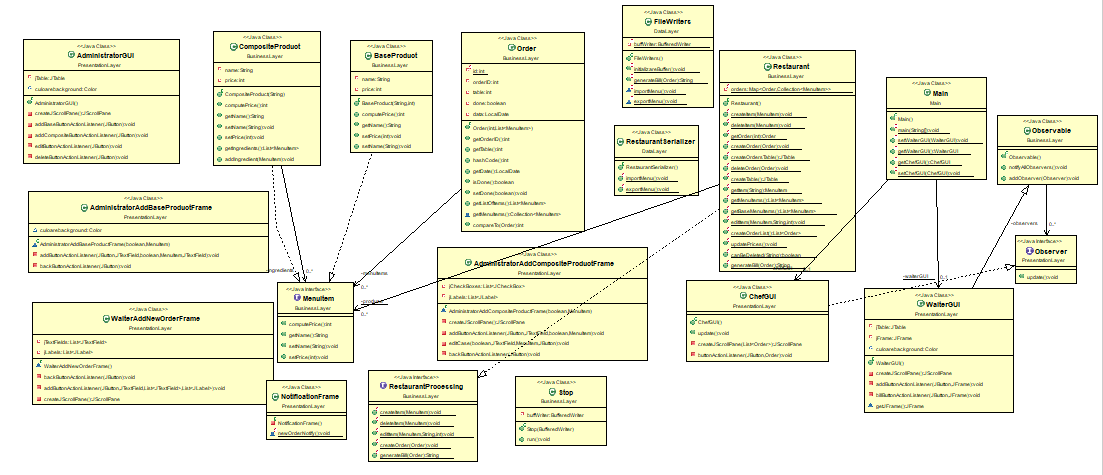
Aplicația lucrează cu comenzi a căror funcționalitate este procesată de o interfață grafică.

* **Diagrama UML**

UML este un limbaj comun utilizat pentru a descrie, specifica, proiecta și documenta procesele, structura și comportamentul de afaceri existente sau noi, a artefactelor sistemelor software.

Pentru a arăta designul programului, am ales să creez și să prezint diagrama clasei, deoarece este cea mai specifică și mai utilizată diagrama UML folosită în general. Este blocul principal al oricărei soluții orientate pe obiecte. Prezintă clasele dintr-un sistem, atributele și operațiile fiecărei clase și relația dintre fiecare clasă. Prin urmare, toate detaliile de implementare necesare despre legăturile făcute între clase și modul în care se folosesc reciproc pot fi citite din aceasta.

În majoritatea instrumentelor de modelare, o clasă are trei părți, nume în partea de sus, atribute în mijloc și operații sau metode în partea de jos.



* **Structuri de date**

Dupa cum am spus, aplicatia este creata pe baza interactiunii dintre utilizatori cu clientii si produsele pe care acestea le solicită. Astfel, principalele structuri de date cu care se opereaza sunt structurile BaseProduct și CompositeProduct precum și interfața MenuItem pe care cele 2 clase o implementeaza.

Structura BaseProduct are urmatoarele atribute : private String name și private int price.

CompositeProduct are ca si proprietati: private String name și private int price și o lista de ingrediente de tipul MenuItem private List<MenuItem> ingredients.

Interfața MenuItem constă în metodele: int computePrice(), String getName(), void setName(String n), void setPrice(int pret) care sunt implemetate de clasele BaseProduct și CompositeProduct care o implementează.

* **Pachete**

Aplicația este structurata pe 4 pachete: BusinessLayer, DataLayer, Main și PresentationLayer. De asemene cuprinde 16 clase și 3 interfețe.

1. BussinessLayer

Acest pachet conține modelele logicii din spatele funcțiilor pe care administratorul le poate executa ca operațiuni principale, același lucru poate fi aplicat și în cazul cazierului. IRestaurantProcessing este definit ca o interfață care conține definițiile metodelor care implementează logica din spatele programului. De asemenea, obiectele cu care funcționează aplicația sunt definite în acest pachet, în principal în cadrul interfeței MenuItem pe care o implementează clasele l: BaseProduct, CompositeProduct. În general, clasele cuprinse în BussinessLayer sunt: ​​BaseProduct, CompositeProduct, Order, Restaurant, Observable, Stop și interfețele RestaurantProcessing, MenuItem.

2. DataLayer

DataLayer conține clasa FileWriters, care este folosită prin metoda noastră GenerateBill situată în clasa Restaurant, pentru a tipări detaliile fiecărei comenzi într-un fișier .txt. RestaurantSerialization stochează fiecare obiect conținut în baza de date într-un fișier, care poate fi utilizat pentru a deserializa ulterior și scrie informațiile anterioare care au fost salvate în utilizările anterioare ale aplicației. Fiecare modificare a HashMap și a meniului Item ArrayList va fi serializată ca o modalitate de a fi memorată de program.

3. Main

Clasa Main execută clasele GUI și afișează aplicația pentru a putea fi utilizată de către un utilizator.

4. PrezentationLayer

Aici este formatată interfața grafică și unde este introdusă logica aplicației (în spatele actionListeners). Mai multe detalii în implementare.

1. **Implementare**

Obiectele de bază care vor fi utilizate de logica aplicației noastre se află în pachetul BusinessLayer, bazat pe clasele BaseProduct și CompositeProduct care implementează interfețele MenuItem și Serializable. Acestea implementează metodele din interfața MenuItem amintite mai sus. Produsele compuse au o metoda de calculare a prețului în funcție de prețul produselor de bază din care sunt compuse.

Restaurantul conține logica principală a aplicației noastre, prin definirea principalelor metode care vor fi încorporate în gui, cum ar fi: createItem, deleteItem, createOrder, createOrdersTable, deleteOrder, createTable, getMenuItems, editItem, updatePrices, canBeDelete, generateBill.

Comenzile create în cadrul aplicației vor fi notate într-un HashMap din tastele de tip comandă și elemente ca ArrayLists din meniuItems (care ar putea conține CompositeProducts sau BaseProducts). De fiecare dată când se face o modificare în structura bazei de date, modificările sunt amintite de program prin serializarea obiectelor modificate.

Clasa AdministratorGUI este compusă dintr-un tabel care conține produsele(nume, preț, tip) și 4 JButtons care comandă modificările solicitate către tabelul nostru.

La apăsarea butonului AddBaseMenuItem, utilizatorul trebuie să introduca numele produsului si prețul acestuia, apoi produsul va apărea in tabelul de mai jos. La apăsarea butonului AddCompositeMenuItem, utilizatorul trebuie să introducă numele produsului și să selecteze din ce ingrediente de bază este alcătuit produsul compus pe care îl creează. La apăsarea butonului Edit Menu Item putem modifica prețul unui produs de bază, iar la apăsarea butonului Delete Menu Item ștergem produsul dorit.

Comanda de adăugare a comenzilor funcționează într-un mod similar cu comanda de adăugare a produsului, cu toate acestea, un chelner poate genera o factură la comenzile deja create. La apăsarea butonului New Order, apare o ferestră unde utilizatorul trebuie să completeze numărul mesei si cantitatea de ingrediente pe care o dorește, apoi se apasă butonul Add. La apăsarea butonului Bill Order putem vedea factura comenzii care este de asemenea generată și în fișierul de ieșire bills.txt.

1. **Concluzii**

Specificația problemei s-a prezentat ca un subiect interesant, cu multe posibilități. Scrierea algoritmilor a ajuns să fie mai complexă atunci când avem de-a face cu mai mulți utilizatori, în cazul nostru: administrator, chelner și bucătar și folosirea Serialization care ne ajută la menținerea informațiilor.

* Cunoștințe dobandite

În primul rând, cred că această misiune m-a ajutat să înțeleg mult mai bine conceptele de programare orientată pe obiecte, precum și să folosesc Serializarea.

* Ca și dezvoltări ulterioare:

Există încă mult spațiu pentru îmbunătățiri pentru a face programul mai eficient. Unii dintre algoritmi ar putea fi optimizati și pot fi implementate într-un mod diferit. Pe intrare pot fi impuse mai multe constrângeri, ceea ce înseamnă că se pot gestiona mai multe excepții și se pot crea mai multe operații pentru datele din tabele.

1. .**Bibliografie**

<https://www.tutorialspoint.com/java/java_serialization.htm>

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

<https://www.geeksforgeeks.org/serialization-in-java/>

<https://docs.oracle.com/javase/8/docs/api/java/io/Serializable.html>

<https://javarevisited.blogspot.com/2011/02/how-hashmap-works-in-java.html>

<https://stackoverflow.com/questions/11415160/how-to-enable-the-java-keyword-assert-in-eclipse-program-wise>

<https://docs.oracle.com/javase/8/docs/technotes/guides/language/assert.html>