DOCUMENTATIE TEMA 4

Restaurant Management

Pop Alin

Grupa 30227

# Cuprins

1. Obiectivul temei . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 3
2. Analiza problemei, modelare, scenarii . . . . . . . . . . . . . . 4
3. Proiectare . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 5
4. Implementare . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 4
5. Concluzii si posibile dezvoltari ulterioare . . . . . . . . . . . . 12

# Cerinte fundamentale

## Obiectivul principal

Luați în considerare implementarea unui sistem de management al restaurantelor. Sistemul ar trebui să aibă trei tipuri de utilizatori: administrator, chelner și bucătar-șef. Administratorul poate adăuga, șterge și modifica produsele existente din meniul. Chelnerul poate crea o comandă nouă pentru o masă, poate adăuga elemente din meniu și poate calcula factura pentru o comandă. Bucătarul este informat de fiecare dată când trebuie să gătească alimente comandate prin intermediul unui chelner.

## Obiective secundare

* Dezvoltarea de use case-uri si scenarii
* Alegerea structurilor de date
* Impartirea pe clase si pachete
* Dezvoltarea algoritmilor
* Implementarea solutiei
* Testare
* Proiectare prin tehnici de programare a contractului
* Polimorfism
* Modele de design: observator, compozit
* Implementări JCF HashMap și HashSet
* Serializare

# 1.Analiza problemei

Se cere crearea unui program care poate sa managerieze actiunile dintr – un restaurant . In restaurant avem chef , care trebuie sa fie notificat de fiecare data cand apare o comanda noua , administartor care poate sa introduca produse noi in meniu , sa editeze produsele existente si sa stearga produse din meniu si nu in ultimul rand avem clientul care poate sa faca o comanda si sa ceara bonul pentru comanda creata .

Composite Design Pattern :

Modelul compozit este un model de despărțire și descrie un grup de obiecte care sunt tratate la fel ca o singură instanță a aceluiași tip de obiect. Intenția unui compozit este de a "compune" obiecte în structuri de copaci pentru a reprezenta ierarhiile parțiale. Vă permite să aveți o structură arborescentă și să cereți fiecărui nod din structura copacului să efectueze o sarcină.

Așa cum a fost descris de Gof, "Compune obiecte în structura copacilor pentru a reprezenta ierarhiile parțiale. Compozitul permite clientului să trateze obiecte individuale și compoziții de obiecte uniform ".

Atunci când se ocupă de date structurate pe arbori, programatorii trebuie adesea să facă diferența între un nod de frunze și o ramură. Acest lucru face ca codul să fie mai complex și, prin urmare, predispusă la erori. Soluția este o interfață care permite tratarea uniformă a obiectelor complexe și primitive.

În programarea orientată pe obiecte, un compozit este un obiect proiectat ca o compoziție a unuia sau mai multor obiecte similare, toate prezentând funcționalități similare. Aceasta este cunoscută ca o relație "a-a" între obiecte.

Conceptul cheie este că puteți manipula o singură instanță a obiectului așa cum ați manipula un grup de ele. Operațiile pe care le puteți efectua pe toate obiectele compozite au deseori o relație de numitor comun.

Observer Desing Pattern:

Modelul de observator este un model de proiectare software în care un obiect, numit subiectul, menține o listă a dependenților săi, numiți observatori, și îi notifică automat despre orice schimbări de stare, de obicei apelând una dintre metodele lor. Este folosită în principal pentru a implementa sisteme distribuite de gestionare a evenimentelor, în software "driven event". Majoritatea limbilor moderne, cum ar fi C #, au construite "eveniment" constructe care implementează componentele modelului de observator. Modelul de observator este, de asemenea, o parte esențială în modelul arhitectural familiar model-view-controller (MVC). Modelul de observator este implementat în numeroase biblioteci de programare și sisteme, inclusiv aproape toate seturile de instrumente GUI.

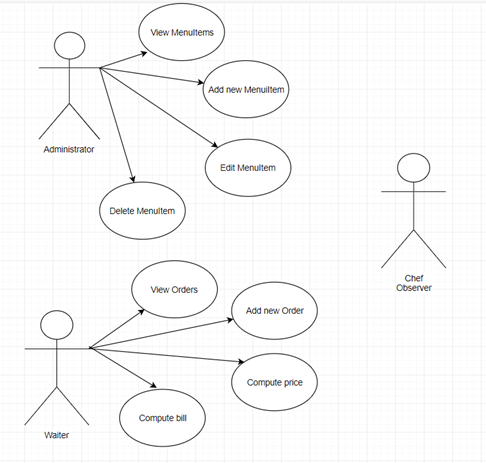
Scenariul in care programul va rula cu success :

* Utilizatorul ruleaza clsa Launch
* Utilizatorul introduce datele necesare efectuarii operatiilor
* Utilizatorul allege ce operatii vrea sa faca
* Metodele ruleaza cu success si se afiseaa rezultatele dorite

Scenarii in care programul nu va rula cu success :

* Utilizatorul va introduce date necorespunzatoare metodelor pe care vrea sa le execute
* Utilizatorul va descoperi un bug nou

Diagrama use-case:



# Proiectare

Proiectul este alcatuit din 4 pachete si 12 clase .

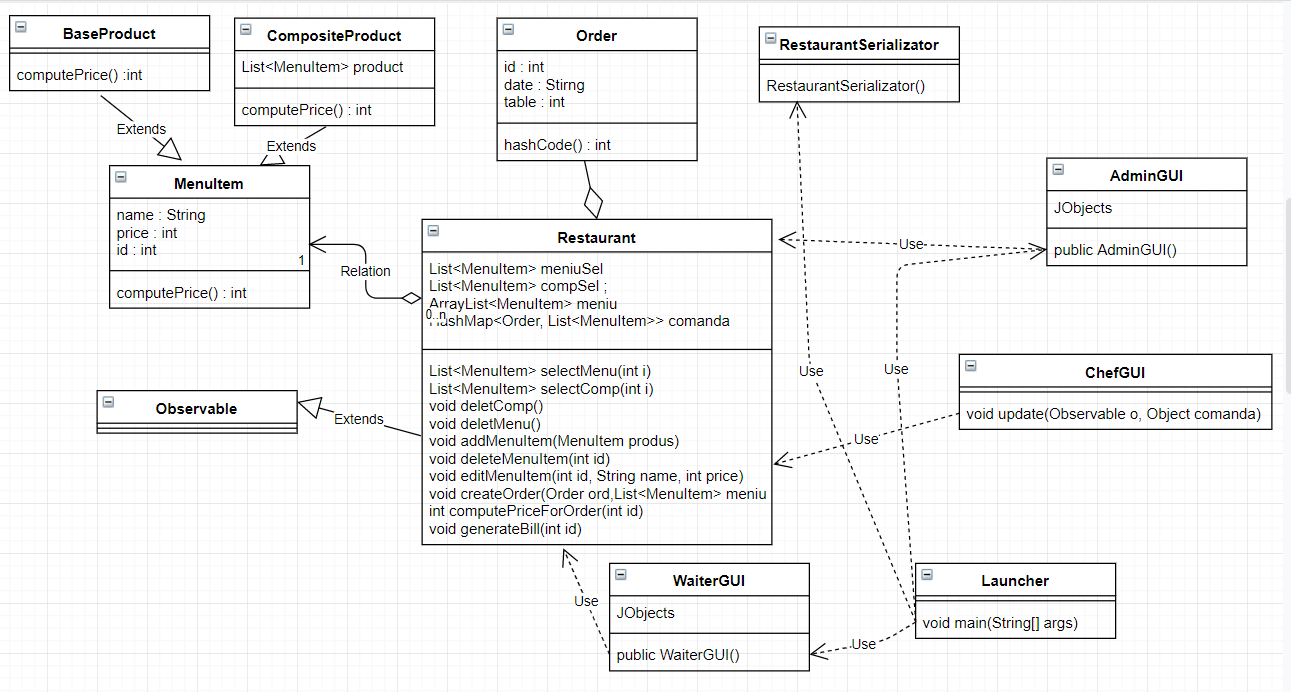
Pachetele din proiect sunt :

* bussinessLayer : este pachetul in care avem implementate metodele si structurile pe care le folosim in proiect
* dataLayer : este pachetul in care avem calsele pentru serializare si crearea de fisiere TXT
* presentationLayer : este pachetul in care avem calsele care implementeaza interfata grafica
* start : este pachetul in care avem calsa care contine metoda main , adica clasa de unde rulam programul

Clasele pe care le regasim in proiect sunt :

* BaseProduct ;
* CompositeProduct ;
* MenuItem ;
* Order ;
* Restaurant ;
* RestaurantProcessing ;
* FileWriter ;
* RestaurantSerialization ;
* AdminGUI ;
* ChefGUI ;
* WaiterGUI ;
* Launcher ;

Diagrama UML :



# Implementare

## BaseProduct

Aceasta clasa extinde clasa MenuItem , avand rolul de a creea un produs simplu , care nu este derivate din alte produse .

Avem un constructor cu parametric String name , int price si int id . Acest constructor apeleaza super ( ) cu acesti parametric pentru a folosi constructorul din clasa pe care o exinde .

Getterele si setterele din aceasta clasa au rolul de a seta valorile campurilor din clasa , respective de a face posibil acecesul la valorile acestor campuri .

Meoda toString() are rolul de a face afisarea Obiectelor de tip BaseProduct cat mai lizibila .

## CompositeProduct

Aceasta clasa extinde clasa MenuItem , avand in plus fata de aceasta un ArrayList de MenuItem , pentru a putea creea produse compuse din mai multe produse existente deja .

Avem un constructor cu parametri String name , int price si int id . Acest constructor apeleaza super ( ) cu acesti parametric pentru a folosi constructorul din clasa pe care o exinde . In plus fata de consturctorul calsei pe care o extinde mai creeaza si un nou ArrayList de MenuItems .

Getterele si setterele din aceasta clasa au rolul de a seta valorile campurilor din clasa , respective de a face posibil acecesul la valorile acestor campuri .

Meoda toString() are rolul de a face afisarea Obiectelor de tip CompositeProduct cat mai lizibila .

Metoda computePrice() are rolul de a calcula pretul total al produsului compus , prin adunarea tuturor preturilor produselor din care este compus .

## MenuItem

Aceasta este o clasa abstract , care are un constructor cu parametri String name , int price si int id , pentru a putea creea obiecte noi de tipul MenuItem .

Campurile acestei calse sunt name de tip String , price de tip nt si id de tip int , pentru a stoca numele produsului , pretu si id – ul acestuia .

Getterele si setterele din aceasta clasa au rolul de a seta valorile campurilor din clasa , respective de a face posibil acecesul la valorile acestor campuri .

Meoda toString() are rolul de a face afisarea Obiectelor de tip MenuItem cat mai lizibila .

Metoda computePrice() este o metoda abstract , aceasta neavand o implementare in aceasta clasa , dar fiin implementata in clasele care o extend .

## Order

Clasa Order are rolul de a stoca elementele unei comenzi .

Campurile din aceasta clasa sunt orderId de tip int , date de tip Stirng si table de tip int , pentru a stoca id-ul comenzii , data in care s-a effectual comanda si numarul mesei la care a avut loc comanda .

Getterele si setterele din aceasta clasa au rolul de a seta valorile campurilor din clasa , respective de a face posibil acecesul la valorile acestor campuri .

Meoda toString() are rolul de a face afisarea Obiectelor de tip MenuItem cat mai lizibila .

Metoda hashCode() are rolul ca atunci cand este folosit un obiect de tip Order , intr-un hashMap , ca si id , acesta face ca Orderul respective sa aiba un ahshCode unic .

## Restaurant

Aceasta este clasa unde sunt implementate toate actiunile care se pot realiza intr-un restaurant .

Campurile din aceasta clasa sunt meniuSel , care este un ArrayList de MenuItem , in care se salveaza elementele din meniu pentru care se creeaza o comanda noua . compSel este un ArrayList de MenuItem in care se salveaza elementele din care este format un compositeProduct . meniu este un ArrayList de MenuItem in care se salveaza meniul restaurantului , adica toate produsele , atat cele baseProduct cat si cele CompositeProduct , din aceste produse fiin posibila alegerea pentru creearea unei comenzi .

Comanda este un HashMap<Order, List<MenuItem>> in care se salveaza o comanda printr-o cheie de tip Order , si valoarea asociata acesteia cu un ArrayList de MenuItem in care regasim elementele din comanda .

Getterele si setterele din aceasta clasa au rolul de a seta valorile campurilor din clasa , respective de a face posibil acecesul la valorile acestor campuri .

Meoda toString() are rolul de a face afisarea Obiectelor de tip MenuItem cat mai lizibila .

Metoda selectMenu are un parametru de tip int , si pune in meniuSel elementele din meniu care au id-ul egal cu numarul transmis ca si parametru .

Metoda selectComp are un parametru de tip int si are rolul de a selecta acele produse din meniu care au id-ul egal cu parametrul functiei , si de a le aduga in lista compSel , pentru a selecta din ce produse este compus un produs de tipul CompositeProduct .

Metoda deleteComp are un parametru de tip int , si are rolul de a sterge din lista de component a produsului de tip CompositeProduct , acel produs care are id-ul egal cu intregul transmis ca si parametru .

Metoda deleteMenu nu are nici un parametru , si are rolul ca atunci cand este apelata sa stearga toate datele din lista de component alese pentru a creea un order nou .

Metoda addMenuItem are un parametru , un produs de tipul MenuItem , si are rolul de a adauga in lista meniu , adica lista in care se afla produsele restaurantului , MenuItem-ul transmis ca si parametru .

Metoda deleteMenuItem are ca si parametru o variabila de tip int , si are rolul ca atunci cand este apelata sa se stearga din meniu produsul care are id-ul egal cu intregul transmis ca si parametru functiei .

Metoda editMenuItem are rolul de a edita acel element din meniu care are id-ul egal cu id-ul transmis ca si parametru functiei .

Metoda createOrder are ca si parametru un Order si o lista de MenuItems , si are rolul de a creea o noua copmanda cu cheia orderul transmis ca parametru si cu valoare produsele transmise ca parametru prin ArrayList .

Metoda computePriceForOrder are ca si parametru un intreg , si are rolul de a calcula pretul produselor din comanda care are id-ul comenzii egal cu intregul transmis ca si parametru .

Metoda generateBill are ca parametru un intreg si are rolul de a creea un bon , intr-un fisier de tip txt , pentru comanda cu id-ul orderului egal cu intregul transmis ca si parametru .

## AdminGUI

In aceasta clasa regasim interfata grafica pentru administrator , cu posibilitatea de a apela , prin apasarea butoanelor , operatiile de adaugare produs , editare produs , stergere produs sis a vedem toate produsele din meniu .

## WaiterGUI

In aceasta clasa regasim interfata grafica pentru waiter , cu posibilitatea de a apela , prin apasarea butoanelor corespunzatoare , operatiile de create order , generate bill si view all orders .

## Launcher

In aceasta lasa regasim metoda main in care sunt apelati constructorii pentru interfetele grafice si serializare . Aceasta este clasa care trebuie rulata pentru a folosi programul .

# 4.Cocluzii si posibile dezvoltari ulterioare

Aceasta a fost o tema foarte folositoare deoarece am invatat multe lucruri noi , utilizarea Design Patternurilor , in special a celor de Composite si Observer si serializarea programelor .

Programul poate fi dezvoltat prin utilizarea lui pentur un intreg lant de magazine , in care avem mai multi waiter , chefi si administartori .