**DOCUMENTATIE TEMA 4**

**Nume prenume Tigarean Lucian**

**Grupa 302210**

**Profesor Laborator Assist Antal Marcel**

Contents

[1. Cerinte Functionale 3](#_Toc476131445)

[2. Obiective 3](#_Toc476131446)

[2.1. Obiectiv Principal: 3](#_Toc476131447)

[2.2. Obective Secundare: 3](#_Toc476131448)

[3. Analiza Problemei 3](#_Toc476131449)

[4. Proiectare 3](#_Toc476131450)

[4.1. Structuri de date 3](#_Toc476131451)

[4.2. Diagrama de clase 3](#_Toc476131452)

[4.3. Algoritmi 3](#_Toc476131453)

[5. Implementare 4](#_Toc476131454)

[6. Testare 4](#_Toc476131455)

[7. Concluzii si Dezvoltari Ulterioare 4](#_Toc476131456)

[8. Bibliografie 4](#_Toc476131457)

# Cerinte Functionale

Dezvoltarea unui sistem de management al unui restaurant. Sistemul trebuie sa aiba 3 tipuri de utilizatori administrator, chelner si bucatar. Cerintele functionale pentru cele 3 tipuri de utilizatori sunt:

* Administrator: administratorul poate sa adauge, sa stearga, sa modifice un produs din meniu.
* Chelner: acesta poate sa creeze o noua comanda pentru o masa, sa adauge elemente din meniu si sa compuna factura pentru produsele comandate.
* Bucatarul: acesta are un sistem prin care este notificat daca o noua comanda a fost creata si primeste lista cu produsele.

# Obiective

## Obiectiv Principal:

Obiectivul principal este de a reazliza acest sistem de management al restauranului utliziand tehnicile de programare specificate in cerinta: Composite Design Pattern pentru realizarea meniului, Observer Design Pattern pentru notificarea bucatarului cand o comanda a fost creata, utilizarea structurii de date Map pentru stocarea in mod eficient a comenzii si a listei cu produsele asociate comenzii, implementarea clasei Restaurant utlizand Design by Contract.

## Obective Secundare:

|  |  |  |
| --- | --- | --- |
| **Obiectiv Secundar** | **Descriere** | **Capitol** |
| Dezvoltarea de use case-uri si scenarii | Sunt descrise use case-uri pentru efectuarea operatiilor specifice celor 3 tipuri de utlizatori: administrator, chelner si bucatar | 3 |
| Alegerea structurilor de date | Alegerea unei structuri de date convenabile atat din punct de vedere al eficientei cat si a memoriei ocupate | 4 |
| Impartirea pe clase | Impartirea pe pachete si clase | 4 |
| Dezvoltarea algoritmilor | Dezvloatarea algoritmilor pentru calcularea facturii si a operatiilor efectuate asupra meniului | 4 |
| Implementarea solutiei | Implementarea solultiei utilizand HashMap si respecand anumite cerinte de proiectare | 5 |
| Testare | Testarea realizata pe parcursul proiectarii | 6 |

# Analiza Problemei

Diagram use case pentru inserarea unei comenzi

1. Se apasa pe butonul Waiter system interface
2. In partea stanga se introduc datele
3. Se apasa pe butonul Add new order

Diagram use case pentru inserarea unui nou produs in meniu

1. Se apasa pe butonul Administrator system interface
2. In partea stanga se introduc datele
3. Se apasa pe butonul Add new menu item

# Proiectare

## Structuri de date

Structurile de date folosite sunt List si HashMap.

Structura de date HashMap impelmenteaza interfata Map din Java. Aceasta structura stocheaza o pereche de tipul (cheie, valoare). Pentru a accesta o valoare trebuie sa cunoastem cheia. HashMap este cunoscut ca HashMap deoarece utlizeaza o tehnica numita Hashing(„ converteste un string lung la un string mai mic pentru a creste viteza de cautare).

HashMap face parte din pachetul java.util package, implementeaza interfetele Clonable si Serializable. HashMap nu permite chei duplicate dar permite valori duplicate.

Clasa HashMap are metode implementate care ne permit sa lucram mai usor, de exemplu metoda clear() care sterge toate perechile cheie,valoare din hashmap, put(k key, v value) prin care introducem in hashmap un nou set de cheie,valoare, get(Object key) care returneaza valoarea pentru cheia data, keySet() returneaza un set de chei aflate in hashMap, entrySet() care returneaza un setul de cheie,valoare aflate in hashmap, remove(Object key) prin care stergem setul de cheie, valoare cu cheia key din hasMap.

Structura de date folosita este List din pachetul java.util. List implementeaza interfata Collection. Structura de date List contine liste(secvente) de elemente indexate. Listele pot contine duplicate si permit un control precis asupra pozitiei unui element prin intermediul indexului acelui element. Clasa List are metode implementate care ne permite sa lucram mai usor, de exemplu pentru a adauga un element in lista folosim metoda add(Object a), pentru accesare indexata folosim metoda get(int index), pentru a afla numarul de elemente din lista folosim metoda size(), pentru stergere putem folosi atat metoda remove(Object a) cat si remove(int index) care sterge obiectul dat, respectiv sterge obiectul de pe pozitia data.

## Diagrama de clase

RestaurantProcessing

+addMenuItem(MenuItem): void

....

+WAGraphicalUserInterface()

...

WAGraphicalUserInterface

-pane3: JPanel

...

+ChefGraphicalUserInterfac()

...

ChefGraphicalUserInterface

Order

-orderId: int

-date: String

-table: int

+Order(int, String, int)

...

-meniu

Restaurant

-comenzi: Map<Order, List<MenuItem>

-name: String

+Restaurant(String)

...

0..\*

MenuItem

computePrice(): float

-menuList

0..\*

BaseProduct

-id: int

-denumire:String

-cantitate: float

-pret: float

+BaseProduct(int, float, float, String)

......

CompositeProduct

+computePrice(): float

....

Observable

## Algoritmi

In acest proiect nu s-au folosit algoritmi complicati, algoritmi simpli de calculare a pretului unei note de plata si calcularea unor conditii de valizare a datelor de intrare.

# Implementare

Pentru realizarea acestui proict s-au realizat doua pachete fiecare avand cateva clase. S-a ales distribuirea claselor in pachete pentru o mai buna organizare si portabilitate a proiectului.

Pachetul **business** contine contine sase clase. Aceste clase stau la baza logici aplicatiei.

* Clasa **Order** contine trei atribure: orderId de tipul int in care se stocheaza id comenzii, date de tipul string in care se stocheaza data comenzii si atributul table de tipul int in care se stocheaza numarul mesei de la care s-a efectuat comanda. Constructorul clasei este Order(int id, String date, int table) si initializeaza aceste atribute. In acesta clasa avem doua metode mai speciale si trei metode de tipul getter-lor getId(), getDate() si getTable. Cele doua metode mai speciale care sunt suprascriese sunt hashCode() si equals(Object obj) care ajuta la implementarea corecta a logicii structurii de date HashMap utilizand obiectele Order si ArrayList<MenuItem>. Metoda hashCode() returneaza indexul unde trebuie stocata setul de cheie, valoare in HashMap. Implementarea proprie se bazeaza pe id-ul comenzii modul 40. Metoda equals(Object obj) a fost suprascrisa pentru a putea compara daca doua obicte au aceleasi valori ale atributelor(implementarea standard verifica daca obiectele au aceeasi referinta), astfel daca doua obiect au aceleasi valori ale atributelor atunci metoda returneaza true si obiectele se considera egale.
* Clasa **MenuItem** este o clasa abstracta care implementeaza interfata Serializable(pentru serializarea acestor clase) care specifica metodele abstracte computePrice(), addMenuItem(), getNume(), getPret(), getCantitate(), getId() pentru a putea fi implementate de clasele care mostenesc clasa abstracta MenuItem. Clasa MenuItem impreuna cu clasele BaseProduct si CompositeProduct respecta designul Composite Design Pattern. Acest design perimite lucrul mai usor in implementarea sistemului de management al restaurantului.
* Clasa **BaseProduct** extinde clasa abstracta MenuItem si implementeaza interfata Serializable. Aceasta clasa face parte din designul Composite Design Pattern. Atributele clasei sunt id de tipul int(in care se stocheaza id-ul produsului), denumire de tipul String(in care se stocheaza denumirea produsului), cantitate de tipul float(in care se stocheaza cantitatea produsului) si pret de tipul float(in care se stocheaza pretul produsului). Constructorul clasei este BaseProduct(int id, float p, float c, String d) prin care se initializeaza atributele specificate mai sus, iar metodele sunt cele specificate de clasa abstracta MenuItem suprascrise in functie de cerinte.
* Clasa **CompositeProduct** extinde clasa abstracta MenuItem si implementeaza interfata Serializable. Aceast clasa impreuna cu celelate doua clasa, prezentate mai sus, fac parte din designul Composite Design Pattern. Aceasta clasa are un singur atribut numit menuList si care este un ArrayList<MenuItem> , adica aceasta clasa retine o compozitie de clase de baza din acest design. (Pentru o mai buna intelegere, clasa CompositeProduct stocheaza o lista de elemente de tipul BaseProduct). Metodele din aceasta clasa sunt cele specificate de clasa abstracta MenuItem suprascrise in functie de modul de lucru cu aceasta clasa.
* Interfata **RestaurantProcessing** defineste cateva metode care trebuiesc implementate de clasa care va implementa aceasta interfata si anume clasa Restaurant. Interfata defineste metodele addMenuItem(MenuItem m), removeMenuItem(int id), editMenuItem(MenuItem m), createOrder(int id, String data, int table, ArrayList<MenuItem> lista) , computePrice(Order a), generateBill(Order a).
* Clasa **Restaurant** implementeaza in spate spate logica sistemului de management, din aceasta cauza este mai complexa si cu mai multe metode. Atributele clasei Restaurant sunt comenzi de tipul HashMap<Order,ArrayList<MenuItem>> in care se socheaza perechile chei,valoare adica Order si ArrayList<MenuItem>, atributul meniu de tipul ArrayList<MenuItem> in care se stocheaza o lista cu produse care se pot servi in restaurant, si atributul name de tipul String in care se salveaza numele restaurantului.

Constructorul clasei initializeaza atributul name, precum si atributul meniu prin apelarea metodei deserializeMenu()(urmeaza a fi descrisa mai jos).

Clasa Restaurant extinde Observable si implementeaza RestaurantProcessing. Observable este o clasa folosita pentru crearea altor subclase prin care alte parti ale programului por observa. Cand un obiect din acest tip de subclasa a suferit o modificare, clasa observatoare( clasa care implementeaza Observer) este notificata. Notificarea acestei clase se face prin metodele setChanged() si notifyObservers(Object obj) (prin care putem transmite obiect pentru a putea fi prelucrate).

Metodele clasei Restaurant sunt **addMenuItem(MenuItem m)** prin care se adauga in lista meniu un nou produs, apoi se cheama metoda searilizeMenu(meniu) pentru a serializa si salva datele in memorie. Metoda **removeMenuItem(int id)** sterge din lista meniu obiectul care are atributul id de valoarea id. Dupa acestea se chema metoda searilizeMenu(meniu) pentru a actualiza datele salvate prin serializare. Metoda **getMeniu()** returneaza lista meniu,metoda **creatOrder**(int id, String data, int table, ArrayList<MenuItem> lista) prin care se creaza o noua comanda si se introduce in HashMap prin metoda put(key, value), apoi se apeleaza metoda setChanged si metoda notifyObservers(Object obj) pentru a notifica obsevatorul de schimbarea facuta, se trimite lista de MenuItem prin metoda notifyObservers(obj) pentru a putea fi prelucrata de clasa ChefGraphicalUserInterface. Metoda **computePriceOrder(Order a)** calculeaza pretul produselor unei comenzi si creeaza o factura. Metoda primeste ca argument o comanda apoi prin metoda get(key) se obtine valoarea de la aceea cheie aflata in HashMap si anume o lista cu produsele comandate. Factura se realizeaza prin crearea unui obiect de tipul PrintWriter care creaza un fisier .txt extern; prin metoda println putem scrie stringuri in acel fisier extern astfel s-a realizat factura cu produsele comandate si pretul acestora. Metoda **searilizeMenu**(**ArrayList<MenuItem> m1)** primeste ca argument o lista de MenuItem pentru a putea fi serializate. Java furnizeaza un mecanism numit **serializare** prin care un obiect poate fi reprezentat ca o secventa de bytes care includ datele obiectului precum si informatii despre tipul obiectului si tipurile de date stocate in obiect. Dupa serializare obiectul este scris intr-un fisier cu extensia .ser. Acest fisier poate fi citit prin procedeul numit deserializare. Clasele **ObjectInputStream** si **ObjectOutputStream** sunt clase de nivel inalt care contin metodele pentru serializare si deserializare. Pentru metoda searilizeMenu se creeaza un obiect de tipul FileOutStream in care se va serializa obiectul, se creeaza un obiect ObjectOutputStram cu acel fisier cu extensia .ser deschis, si prin metoda writeObject(Object obj) se serializeaza obiectul dat ca si argument. Metoda **deserializeMenu()** returneaza o lista de MenuItem, obtinuta in urma deserializarii fisierului( se creeaza un obiect de tipul ObjectInputStream, se intializeaza cu o noua instanta de tipul ObjectInputStream(FileInputStream(„calea catre fisierul nostru .ser)), se apeleaza metoda readObject() care returneaza obiectul citit (in cazul nostru o lista cu obiecte de tipul MenItem.

Pachetul **presentation** contine doua clase **ChefGraphicalUserInterface** si **WAGraphicalUserInterface.**

* Clasa **WAGraphicalUserInterface** extinde clasa Jframe si implementeaza ActionListener. In acesta clasa s-au creat interfata grafica pentru chelner si administrator.
* Clasa **ChefGraphicalUserInterface** extinde Jframe si implementeaza interfata Observer. Interfata Observer contine metoda update(Observable arg0, Object arg1) prin care aceasta clasa urmareste clasa care implementeaza Observable.

# Testare

Testarea s-a realizat pe parcursul dezvoltarii aplicatiei, urmarind comportamentul si rezultatele( comparate cu cele asteptate de noi) astfel s-au corectat unele erori si s-a optimizat programul.

# Concluzii si Dezvoltari Ulterioare

Realizand acest proiect am invatat sa folosesc HashMap-ul implementat in Java, am inteles modul de functionare am metodelor equals si hashCode, modul in care trebuie sa le suprascriu pentru a da functionalitatea pe care o doresc in functie de obiectele folosite(metoda equals implementata de java compara daca doua referinte daca sunt egale, iar metoda hashCode returneaza un numar random diferit pentru fiecare instantade obiect).

Am invatat despre designul Composite Desgin Pattern din java care descrie un grup de obiecte care sunt tratate ca o singura instanta pentru acelasi tip de obiect, adica putem descrie o interfata si alte obiect care pot implementa acesta interfata, aceste obiect sunt obiecte de baza si obiecte compuse din aceste obiecte de baza).

Am invata despre java Observable si Observer prin care o clasa poate fi observata de o alta clasa observatoare( adica la o modificare in clasa care extinde Observable, prin metodele setChanged() si notifyObservers(Object obj) se apeleaza metoda update din clasa care extinde Observers)

Am invatat sa folosesc mecanismul numit serializare pentru a salva obiect in memorie. In forma cea mai simpla serializarea obiectelor inseamna salvarea si restaurarea starii obiectelor.

Dezvoltarie ulterioare. Proiectul mai are nevoie de cateva imbunatatiri precum ar fi:

* O interfata mult mai interactiva cu elemente noi, cu functionalitati noi, mult mai aranjata si ordonata care poate oferi utlizatorului o experienta cat mai usoara de utilizare
* Un sistem de logarea prin care putem restrictiona acestul in sistem a altor persoane.
* Putem adauga alte mecanisme de verificare a datelor introduse de utilizator si putem afisa mesaje de eroare in acest caz.

# Bibliografie

https://www.geeksforgeeks.org/observer-pattern-set-1-introduction/

https://www.tutorialspoint.com/design\_pattern/observer\_pattern.htm

https://www.geeksforgeeks.org/java-util-observable-class-java/

https://dzone.com/articles/working-with-hashcode-and-equals-in-java

https://www.geeksforgeeks.org/composite-design-pattern/

https://www.geeksforgeeks.org/java-util-observable-class-java/