Documentație TEMA 4 RESTAURANT MANAGEMENT

Nume: Câmpean Adelina Ioana

Grupa: 30227

Profesor Laborator: Pop Cristina

Cuprins

1. Cerințe Funcționale
2. Obiectivul temei
3. Analiza Problemei modelare, scenarii, cazuri de utilizare
   1. Diagrama USE CASE
4. Proiectare
   1. Diagrama de clase UML
   2. Diagrama de pachete
   3. Implementare
5. Rezultate
6. Concluzii
7. Bibliografie
8. Cerințe funcționale

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.

Pentru a simplifica aplicația, puteți presupune că sistemul este utilizat de un singur administrator, de un chelner și un bucătar-șef, și nu este nevoie de un proces de conectare.

Rezolvați următoarele:

1. Definiți interfața RestaurantProcessing care conține operațiile principale care pot fi executate de către

chelnerul sau administratorul, după cum urmează:

- Administrator: creați un element de meniu nou, ștergeți elementul de meniu, modificați elementul din meniu

- Ospătar: creați o comandă nouă; 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 compozit pentru a defini clasele MenuItem, BaseProduct și

CompositeProduct

- Utilizați modelul de design Observer pentru a notifica bucătar-șef de fiecare dată când o comandă nouă conține a

se adaugă un produs compozit.

3. Implementați restaurantul de clasă utilizând o colecție predefinită JCF care utilizează date hashtable

structura. Cheia hashtable va fi generată pe baza ordinului de clasă, care poate avea

asociate mai multe MenuItems. Utilizați JTable pentru a afișa informații legate de restaurant.

- Definiți o structură de tip Hartă <Ordine, Colecție <MenuItem >> pentru stocarea comenzii

informații din clasa Restaurant. Cheia hărții va fi formată din obiecte de tip

Ordine, pentru care metoda hashCode () va fi suprascrisă pentru a calcula valoarea hash

în cadrul Hărții din atributele Ordinului (Codul de comandă, data etc.)

- Definirea unei structuri de tip Collection <MenuItem> care va salva meniul din

restaurant. Alegeți tipul de colecție adecvat pentru implementarea dvs.

- Punerea în aplicare a clasei prin metoda "Design by Contract" (implicând pre, post,

invarianți și afirmații).

4. Elementele de meniu pentru popularea obiectului Restaurant vor fi încărcate / salvate din / într-un fișier folosind Serializarea.

1. Obiectivul temei

Această temă are ca scop simulare modului de funcționare a unui restaurant. În acest restaurant există un administrator care are abilitatea de a introduce un produs nou în meniu, de a crea meniul, de a edita un anumit produs, de a șterge un nou produs și de a vizualiza toate produsele introduse până în acel moment în meniul restaurantului. Există de asemenea și un chelner care are rolul de a prelua comenzi de la mesele din restaurant. Și există și un bucătar care este anunțat de fiecare dată când a fost preluata o nouă comandă.

1. Analiza Problemei, modelare, scenarii, cazuri de utilizare
   1. Diagrama USE-CASE

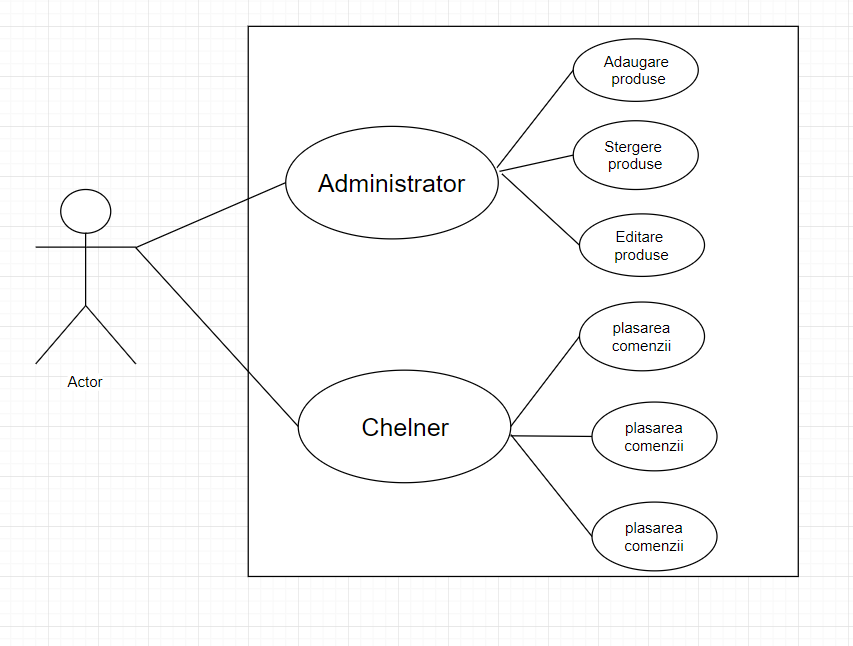


Diagrama USE CASE simulează scenariul de funcționare al aplicației.

De exemplu, un scenariu ar fi, cazul în care utilizatorul, accesează interfața administratorului în care, introduce un nou produs, sau editează un produs deja existent, creează un composite product, format din alte base products, sau composite products. Apoi, poate vizualiza tot ce există în meniu. După asta, intră în interfața waiter-ului, de unde poate să își aleagă produsele pe care dorește să le comande. Apoi se poate forma o factură pentru ultima comandă introdusă, care este afișată într-un fișier text.

1. Proiectare
   1. Diagrama de clase UML

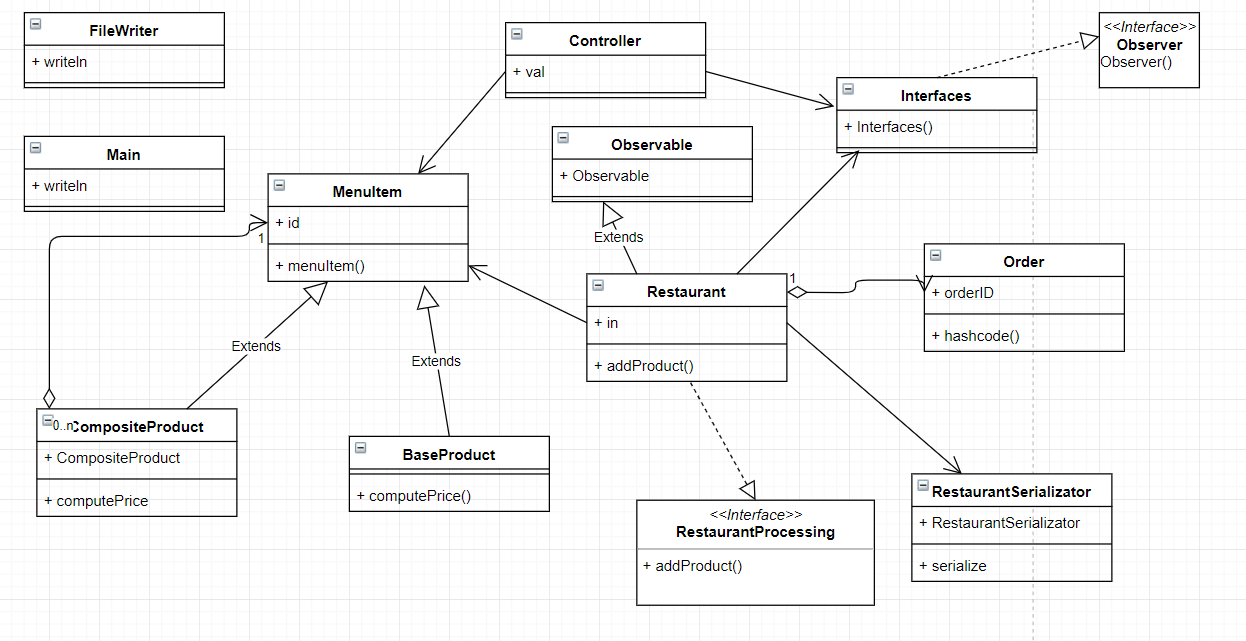
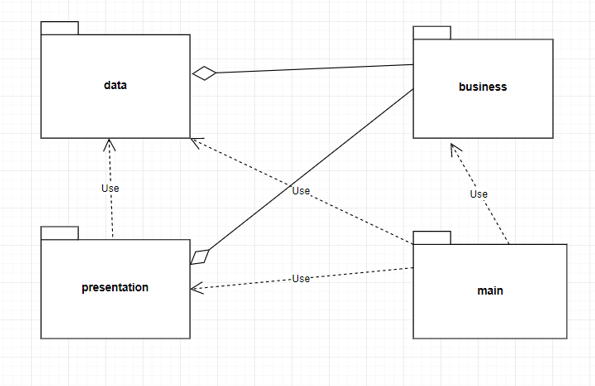


Diagrama de clase este formată din pachetele Presentation, Business și data. Pachetul Business conține clasele Base Product, Composite Product, MenuItem, Order și Restaurant. Pachetul Presentation conține clasele cu interfețele grafice. Pachetul Data conține FileWriter și RestaurantProcessing.

4.2. Diagrama de pachete

Sunt prezente 3 pachete, care conțin clasele care se ocupă de interfețe, de exemplu în Presentation, în pachetul business, sunt clasele care realizează programul în sine iar în pachetul data, se află clasa de scriere în fișier și o interfață care conține metodele importante.

Pachetul Business

Conține clasa Base product care extinde clasa meniu Item. Clasa Composite product extinde de asemenea clasa meniu item și conțin metoda compute price, care este suprascrisă din meniu item. În composite product se află un array list de meniu item, care are rolul de a salva produsele din care este alcătuit. Această metodă parcurge lista de produse, de meniu items a fiecarui produs de tip composite product si calculeaza pretul produsului composite prin realizarea sumei prețurilor produselor din care este alcătuit acesta.

Clasa Meniu item reține un id pentru fiecare produs, un nume și un preț.

Clasa Order conține un id, o dată la care s-a realizat comanda și masa de la care se ia comanda. Se suprascrie o metoda de tip hashCode care are rolul de a seta id-ul comenzii, prin intermediul caruia introducem elemente în hashmap.

Clasa Restaurant extinde clasa Observable, pentru a putea notifica bucătarul pentru primirea noilor comenzi și implementează interfața restaurant processing în care au fost declarate antetele metodelor folosite. Clasa conține un hashmap în interiorul căruia se salveaza, ca și cheie un order și un array list de meniu items, astfel se pot adăuga în hash map valori la anumite chei, adică la valoarea id-ului comenzii, datei sau mesei. Se instanțiază și un obiect de tipul restaurant serializator, pentru realizarea metodei de serializare si deserializare.

În această clasă se implementează metodele de adaugare produs, în care se adaugă în array listul de produse al restaurantului, preparatele adăugate de utilizator prin intermediul interfeței grafice. Metoda de stergere a unui anumit produs, funcționează pe baza găsirii unui anumit produs în funcție de id-ul acestuia, si prin parcurgerea array listului de preparate din meniu. La găsirea acestuia, produsul respectiv este eliminat. Editarea unui produs, este asemănătoare cu ștergerea, doar că în momentul găsirii produsului respectiv, acestuia i se modifică, fie numele fie prețul. Există o metodă care afișează conținutul listei.

Au fost create 2 metode de Create table, una pentru vizualizarea elementelor din meniul restaurantului și una pentru vizualizarea istoricului de comenzi, realizate într-o rulare. Prima metoda are un String pentru a defini antetul tabelului și o matrice de tipul object, prin intermediul careia se populează tabelul. Se parcurge lista de produse, din restaurant, apoi se verifică dacă produsul respectiv este composite product, atunci pentru acesta este necesar sa afisam si produsele din care este alcatuit sau base product, care nu are alte ingrediente si atunci se completeaza doar id-ul, numele, pretul respectivului produs si tipul acestuia. La composite Product pretul este 0, deoarece se calculeaza pe baza preturilor produselor din care este alcatuit. Dacă produsul este base product, atunci în rubrica de ingrediente se va afișa o - .

String column[]={"ID","NAME","PRICE","TYPE","INGREDIENTS"};

Object data[][]=**new** Object[menu.size()][5];

**for**(MenuItem m1 : menu)

{

**if**(m1 **instanceof** CompositeProduct) {

**for**(MenuItem m2: ((CompositeProduct) m1).getMenu()) {

in =in+ m2.getName()+",";

}

m1.computePrice();

data[i][0] = m1.getId();

data[i][1] = m1.getName();

data[i][2] = m1.getPrice();

data[i][3] = "Composite Product";

data[i][4] = in;

i++;

in="";

}

A doua metodă de create order table are același scop ca și cea de dinainte, diferă antetul acestuia, și faptul că aici se parcurge un hashmap de order si un array list de produse. Apoi se parcurge lista de produse introduse pentru o anumita comandă (produsele comandate de utilizator, la o anumită masă), acestea se salvează într-un string, care apoi se atribuie matricii, pe poziția corespunzătoare. Valorile comenzii și anume, id-ul, data și masa de la care s-a preluat comanda, se iau prin intermediul getterului getKey() specific hash map-ului. Apoi se creeaza un jtable, în care se vor salva antetul și matricea care populează apoi acesta se va returna.

**for** (Entry<Order, ArrayList<MenuItem>> m1 : productorder.entrySet())

{

String str = "";

**for**(MenuItem w: m1.getValue())

{

str = str + w.getName()+", ";

}

data1[j][0] = m1.getKey().orderID;

data1[j][1] = m1.getKey().date;

data1[j][2] = m1.getKey().table;

data1[j][3] = str;

j++;

str=**new** String();

}

Pachetul Data

Conține clasele file writer si restaurant serializator. Prin intermediul clasei file writer, se realizează scrierea facturii unei anumite comenzi într-un fișier text. Clasa Restaurant Serializator conține clasele serialize și deserialize, prin intermediul cărora se salvează de la o rulare la alta produsele introduse în meniu. Metoda de serializare codeaza informația în secvențe de 0 și 1, pentru a fi salvată, la momentul închiderii programului, iar metoda de deserializare, realizează decodificarea respectivelor secvențe, și repunerea acestora în tabela, pentru a fi vizualizate.

FileWriter fw=**new** FileWriter("bill.txt");

fw.write("FACTURA! "+"\r\n");

fw.write("\r\n");

fw.write("ID COMANDA: ");

fw.write(idor+"\r\n");

**for**(MenuItem mp : m2)

{

fw.write("Produsul: "+ mp.getName()+ " avand pretul: "+ mp.getPrice()+"\r\n");

mp.computePrice();

pricetot+=mp.getPrice();

}

Pachetul Presentation

Conține clasa Controller, care face legătura dintre interfața grafică și restul programului. Această clasă conține action listenerele butoanelor din interfata. La apelul acestei clase, în momentul realizării constructorului, se apelează și metoda de deserialze, din Restaurant serializator, care salvează informațiile din tabel. Conține clase interne, care deschid frameurile corespunzătoare butoanelor. În prima interfată, butonul end are rolul de a realiza serializarea datelor. În momentul apăsării butonului de delete product, se verifică mai întâi in care rubrici au fost introduse datele, si anume daca este un base product sau un composite product, apoi se elimină. Pentru adăugarea unui produs, se verifică din nou ce fel de produs dorim să introducem, dacă este base product, se creează un nou obiect de tip base product, și se adaugă în meniul restaurantului. Dacă este un composite product, atunci se creează un nou obiect de tip composite product si se seteaza meniul acestuia cu un nou arraylist. În momentul apăsării butonului de edit product, se verifică mai întâi in care rubrici au fost introduse datele, si anume daca este un base product sau un composite product, apoi se introduc noile date, pentru a actualiza produsul respectiv. Butonul de done, are rolul de a anunța când s-a încheiat introducerea de ingrediente pentru un anumit composite product. Butonul de addd, este situat în interfața grafică a waiterului si are rolul de a adăuga mai multe produse într-o comandă, în cazul în care id-ul introdus, există deja și în meniul restaurantului, astfel încât să poată fi comandat. Apoi se apasă butonul de place order (newo) care plasează comanda, generate bill, scrie în fișier factura iar compute price calculează pretul final al comenzii.

Clasa Interfaces, conține clasele care alcătuiesc programul și anume o interfață pentru administrator, care adaugă, șterge sau editează produsele din meniu, o interfață pentru chelner, care preia comenzi și o interfață pentru bucătar, care are un text area în care se scrie, de fiecare dată când se plasează o nouă comandă.

1. Rezultate

Rezultatele sunt vizibile în fiecare tabelă, se pot verifica operațiile de inserare, actualizare și ștergere pentru produsele din meniul restaurantului. Pentru fiecare comandă se poate observa, în tabel, masa de la care a fost preluată, conținutul acesteia, adică ce produse au comandat, și de asemenea se pot observa informațiile detaliate ale comenzii procesate, în fișierul text bill.txt.

Se poate observa și eficiența design pattern-ului observer, prin vizualizarea conținutului frame-ului chef, în care apare un mesaj în momentul în care s-a plasat o comandă cu succes, astfel bucătarul fiind notificat, pentru a putea executa comanda corespunzătoare.

1. Concluzii

În concluzie, această aplicație simulează modul în care funcționează un restaurant, cum administratorul, sau patronul, are dreptul să modifice meniul, iar chelnerul preia comenzi pe care i le înmânează bucătarului pentru a realiza preparatele respective.

Din această temă am învățat cum să folosesc serializarea, să păstrez informațiile de la o rulare la alta, de asemenea, serializarea este un mecanism de transformare a stării unui obiect într-un flux de octeți. Deserializarea este procesul inversat în care fluxul de octeți este utilizat pentru a recrea obiectul Java real în memorie. Acest mecanism este folosit pentru a persista obiectul. Fluxul de octeți creat este independent de platformă. Deci, obiectul serializat pe o singură platformă poate fi deserializat pe o altă platformă. Pentru a face ca un obiect Java să fie serializabil, implementăm interfața java.io.Serializable. Clasa ObjectOutputStream conține metoda writeObject () pentru serializarea unui obiect.

Am mai invatat cum functioneaza clasa Observer si Observable, modelul de observator este folosit atunci când există o relație unu-la-multe dintre obiecte, cum ar fi modificarea unui obiect, obiectele descendente ale acestuia fiind notificate automat. Modelul de observator se încadrează în categoria modelului comportamental.

1. Bibliografie

* <https://www.geeksforgeeks.org/iterate-map-java/>
* <http://www.tutorialspoint.com/java/java_serialization.htm>
* <http://javarevisited.blogspot.ro/2011/02/how-hashmap-works-in-java.html>
* <https://dzone.com/articles/how-copy-static-hashmap-jtable>
* <https://javarevisited.blogspot.com/2012/01/what-is-assertion-in-java-java.html>
* <https://www.javatpoint.com/java-filewriter-class>
* <https://www.geeksforgeeks.org/java-util-hashmap-in-java/>