***Documentație Tema 4***

**Realizator: Stroia Lucian Dorin**

*● Cerința temei*

**Cuprins**

*● Scopul realizării aplicației*

*● Abordarea problemei*

*● Structura interna a programului*

*● Diagrame UML*

*● Concluzii*

***Cerința temei***

*Cerința acestei teme este constituita de implementarea unui sistem de management al unui restaurant, care surprinde activitatea a 3 persoane importante din cadrul unui restaurant. Unul dintre aceștia este administratorul, care se ocupa cu alimentarea restaurantului, modificarea sau ștergerea de produse, altul este bucătarul, care primește comenzile ordonate de către ultimul pion important din acest sistem, adică chelnerul, prin intermediul căruia se realizează comenzile, se generează o chitanță a unei comenzi.*

*Operațiile dorite pentru implementare sunt:*

* 1. *Introducerea de produse alimentare*
  2. *Modificarea unui anumit produs*
  3. *Ștergerea unui anumit produs*
  4. *Vizualizarea tabelului cu produse alimentare*
  5. *Introducerea unei comenzi*
  6. *Generarea unei facturi a comenzii curente*
  7. *Obținerea sumei totale realizată din prețul fiecărui aliment din comanda respectivă*
  8. *Vizualizarea tabelului cu comenzi*

***Scopul realizării aplicației***

*Se dorește stocarea unor alimente într-un fișier, aceste alimente fiind inserate de către administrator, acesta fiind singurul care poate să insereze, modifice, șteargă sau vizualizeze produsele întregului restaurant, adică modificări interioare ale restaurantului, asupra stocului.*

*Chelnerul are posibilitatea de a genera comenzi, fiind presupus faptul că aceste comenzi sunt ordonate de către clienții restaurantului. Poate vizualiza tabelul care conține comenzile generate, poate obține suma de bani pe care trebuie să o plătească o persoană după ce efectuează o comandă, poate crea o factură, in format .txt, care conține informații despre produsele achiziționate dar și suma pe care o are de plată persoana careia îi aparține comanda respectivă.*

***Abordarea problemei***

*Pentru o bună gestionare a datelor inserate de către utilizator, comunicarea dintre acesta si program a fost realizată prin intermediul a 3 ferestre principale, fiecare reprezentând unul dintre cele 3 piese importante din cadrul unui restaurant, adică administratorul, bucătarul si chelnerul. Datele referitoare la stocul restaurantului sunt salvate într-un fișier după închiderea programului, la o pornire ulterioară a acestuia, fiind posibil accesul la acest fișier de unde sunt extrase si prin intermediul lor, se realizează popularea ArrayList-ulu de produse.*

***Structura internă a programului***

*Programul este structurat in 3 pachete, fiecărui pachet fiindu-i atribuit mai multe clase care în care sunt descrise metodele aplicate asupra datelor introduse de către utilizator astfel încât să se obține rezultatele așteptate de către utilizator.*

*♦Primul pachet, numit ”BussinesLogic”, are ca și componente următoarele clase:*

1. *MenuItem*

*Această clasă este de tipul abstract, fiind definite caracteristicile esențiale ale unui produs,*

*mai exact numele acestuia, id-ul, si prețul. Această clasă implementează Serializable, din cauza faptului că se dorește stocarea obiectelor de tipul MenuItem într-un fișier odată cu închiderea programului, iar la deschiderea acestuia, se dorește extragerea informațiilor din fișier, acest lucru fiind posibil prin intermediul procedeului de SERIALIZARE.*

*Este definită și metoda computePrice, specifică fiecărui obiect de tipul MenuItem, dar și metoda de toString pentru afișarea informațiilor referitoare la obiectul respectiv.*

1. *BaseProduct*

*Această metodă extinde clasa abstractă MenuItem, fiind un obiect care are anumite*

*proprietăți spre deosebire de cealaltă clasă care extinde tot MenuItem. Este generată metoda de toString pentr a da informații în legătură cu atributele obiectului respectv.*

1. *CompositeProduct*

*Această clasă extinde MenuItem, fiind caracterizată tot prin intermediul unui id, a unui*

*nume și a unui preț, însă diferența primordială și esențială în comparație cu un BaseProduct este pusă în evidență prin intermediul unui ArrayList de MenuItems, deoarece, se dorește ca un CompositeProduct să fie constituit din mai multe obiecte de tipul MenuItems, adică este posibil să existe un CompositeProduct realizat prin intermediul mai multor BaseProduct-uri, sau prin intermediul mai multor CompositeProduct-uri, sau se pot amesteca cele doua tipuri de produse, realizându-se un mix de produse care constituie un singur produs.*

*Este definită clasa toString prin intermediul căreia aflăm informații despre atributele acestui tip de obiect.*

1. *Order*

*Această clasă este pentru a stoca un obiect care pune în evidență atributele unei comenzi,*

*mai exact orderId prin intermediul căreia fiecare comandă se identifică în mod unic, un String date, care reprezintă data în care s-a procesat comanda respectivă, si un întreg care reprezintă numărul mesei de la care s-a efectuat comanda respectivă.*

*Tot aici, este definită și metoda hashCode prin intermediul căreia este returnat id-ul fiecărui Order, necesar pentru introducerea unei comenzi în structura în care vor fi stocate fiecare comandă la care sunt asociate un anumit număr de MenuItems. Această structură este numită HashMap.*

1. *IRestaurantProcessin*

*Această clasă este de fapt o interfață în care sunt definite principalele metode pentru*

*realizarea operațiilor importante menționate anterior. Fiind o interfață, aici sunt definite doar antetele metodelor care urmează să fie implementate în interiorul clase care implementează această interfața*

1. *Restaurant*

*Această clasă reprezintă creierul întregului program, deoarece aici sunt coordonate toata*

*acțiunile utilizatorului care dorește efectuarea operațiilor de introducere, ștergere sau update-area unui produs, dar și generarea unei chitanțe, aflarea notei de plată, etc.*

*Aici sunt definite 2 ArrayList-uri importante unul în care se stochează produsele introduse de către administrator, atât produse de tipul BaseProduct cât și produse de tipul CompositeProduct, iar celălalt ArrayList este utilizat pentru a stoca multitudinea de produse alimentare care pot constitui o comandă.*

*Este definită și o structură de tipul HashMap, care are ca și cheie un obiect de tipul Order , obiect care conține metoda hashCode prin intermediul căreia este returnat codul la care se dorește stocarea Order-ului respectiv în HashMap. Acesta HashMap conține ca și al doilea parametru un ArrayList de MenuItems specifice fiecărei comenzi.*

*La apelarea constructorului din această clasă, este creat frame-ul principal, este creat si un obiect de tipul RestaurantSerializator prin intermediul căruia se realizează serializarea si desereazilarea datelor dintr-un fișier. Aici, este apelată metoda de deserializare pentru a popula ArrayList-ul de produse din restaurant.*

*Interfata Serializable nu declara nici metode, nici atribute, serveste doar pentru identificarea obiectelor serializabile. Clasele care nu implementeaza aceasta interfata nu vor putea fi serializate sau deserializate. Operatia de deserializare presupune ca clasa obievtului poseda si un constructor cu vizibilitate publica si fara argumente. Atributele obiectului vor fi initializate dintr-un flux de date. In cazul serializarii unui obiect care apartine unui graf de obiecte daca se ajunge la un obiect care nu implementeaza interfata Serializable, atunci se genereaza exceptia NoSerializableException . Clasele care necesita o tratare speciala pe parcursul serializarii si a deserializarii trebuie sa implementeze urmatoarele doua metode cu urmatoarele signaturi:*

*private void writeObject(java.io.ObjectOutputStream out)*

*throws IOException*

*private void readObject(java.io.ObjectInputStream in)*

*throws IOException, ClassNotFoundException;*

*Sunt definite metodele principale de adăugare produse, ștergere produse, update produse, creare comenzi, creare Items pentru o comandă, generarea unei chitanțe si aflarea prețului pentru o comanda curentă.*

*Pentru listarea datelor au fost folosite doua metode, care returnează un DefaultTableModel, specific atât tabelului de Produse cât și tabelului de comenzi.*

*Crearea de produse se realizează după completarea datelor necesare de către utilizator, și după bifarea tipului de produs pe care utilizatorul dorește să îl introducă. Sunt afișate mesaje de eroare în cazul în care spațiile necesare pentru crearea unui obiect nu sunt completate.*

*În cazul în care administratorul introduce un BaseProduct, operațiunea decurge normal, acest obiect de tipul BaseProduct fiinid adăugat în stocul restaurantului.*

*În cazul în care administratorul dorește să introducă un obiect de tipul CompositeProduct, se deschide un frame suplimentar unde utilizatorul trebuie să specifice din ce este format acel CompositeProduct. Amintesc faptul că un Composite Product poate fi alcătui atât din BaseProduct-uri cât și din CompositeProduct-uri.*

*Stergerea unui anumit produs se realizează prin căutarea acestuia în lista de produse ale restaurantului. Odată ce este găsit, este înlăturat din listă.*

*Update-ul este asemănător, însă, odată cu identificare obiectului respectiv, sunt modificate numele si prețul obiectului cu cele indicate de către utilizator.*

*Pentru crearea unei comenzi, se instanțiază un obiect de tipul Order cu atributele introduse de către utilizator prin intermediul interfeței grafice. Prin intermediul metodei createItemsForOrder sunt adăugate produse dintr-un comboBOX în arrayList-ul de produse specifice comenzii curente.*

**public** **void** createItemsForOrder() {

MenuItem m\_aux=**null**;

**int** ok=0;

**for**(MenuItem m:listOfProducts)

{ **if**(m.getName().equalsIgnoreCase(w1.getCombo().getSelectedItem().toString()))

{

m\_aux=m;

ok=1;

**break**;

}

}

**if**(ok==0) {

**for**(MenuItem m:listOfProducts)

{

String aux="";

**int** index = w1.getCombo().getSelectedItem().toString().indexOf("-");

aux=w1.getCombo().getSelectedItem().toString().substring(0, index);

**if**(aux.equalsIgnoreCase(m.getName()))

{

m\_aux=m;

**break**;

}

}

}

**if**(m\_aux!=**null**)

listOfProductsPerOrder.add(m\_aux);

}

*Prin metoda computePriceForOrder, se parcurge ArrayList-ul de MenuItems specific unui order și se realizează o suma a prețurilor specifice fiecărui produs. Acest preț este folosit atât pentru afișarea în interfață cât și pentru crearea chitanței.*

*Metoda generateBill creează un fișier cu extensia .txt prin intermediul căruia utilizatorul este informat de atributele obiectelor pe care le-a comandat dar și despre suma pe care trebuie să o plătească în urma procesării comenzii respective.*

*Sunt definite si listenere specifice fiecărui frame, prin intermediul cărora utilizatorul interacționează cu programul.*

*♦ Cel de-al doilea pachet ”dataAcces” cuprinde următoarele clase:*

1. *RestaurantSerializator*

*Această clasă este creată pentru realizarea serializării si deserializării, aici fiind definite*

*2 metode care se ocupă cu aceste acțiuni. Una dintre aceste acțiuni, se numește serialization și primește ca și parametru un ArrayList de MenuItems.*

**public** **void** serialization(ArrayList<MenuItem> sir)

{

**try** {

FileOutputStream out1 = **new** FileOutputStream("informatii.ser");

ObjectOutputStream out2= **new** ObjectOutputStream(out1);

out2.writeObject(sir);

out2.close();

out1.close();

}

**catch** (IOException i) {

i.printStackTrace();

}

}

*Deserializarea se ocupă cu extragerea datelor din fișier, odată cu pornirea programului.*

*Deserializarea s-a realizat în interiorul constructorului de la Restaurant, pentru a fi extrase datele necesare odată cu instanțierea obiectului de tip RESTAURANT. Acestă metodă returnează un ArrayList de MenuItem.*

**public** ArrayList<MenuItem> deserialization()

{

**try**

{

FileInputStream in1 = **new** FileInputStream("informatii.ser");

ObjectInputStream in2 = **new** ObjectInputStream(in1);

ArrayList<MenuItem> deserializationArray = (ArrayList<MenuItem>) in2.readObject();

in2.close();

in1.close();

**return** deserializationArray;

}

**catch** (IOException i) {

i.printStackTrace();

**return** **null**;

} **catch** (ClassNotFoundException c) {

c.printStackTrace();

**return** **null**;

}

1. *Clasa FileWriterClass*

*Această clasă este responsabilă cu scrierea informațiilor legate de produsele achiziționate*

*de către utilizator în cadrul unei comenzi. Aici este definită metoda chitanță care primește ca și parametri un ArrayList de MenuItems care urmează să fie scrise în fisierul .txt dar și un int care reprezintă prețul total al comezii.*

*♦ Ultimul pachet ”gui” cuprinde doar clasa test:*

1. *Clasa Administrator*

*Această clasă deschide frame-ul pentru Administrator. Aici avem butoanele care ne permit*

*introducerea, editarea sau ștergerea unui produs dar și butonul pentru vizualizarea tabelului de produse.*

1. *Clasa Chef*

*Această clasă deschide frame-ul pentru Chef, unde se poate observa daca bucătarul*

*primește o noua comandă.*

*Clasa Chef implementează interfața ijava.util.Observer, o interfață care trebuie implementată de clasă care ar trebui să fie informată prin schimbări în clasa observabilă. Clasa observabilă ar trebui extinsă de clasa care este observată. În interfața Observer, există metoda update () . Metoda observabilă de clasă a determinat metoda setChanged () la schimbarea adevărată. notifyObservers () metoda notifică observatorul despre schimbare.*

1. *Clasa Waiter*

*Această clasă deschide frame-ul pentru Waiter unde se găssec butoane pentru*

*introducerea unei noi comenzi, produsele fiind selectate din comboBOX, butoane pentru generarea chitanței, pentru vizualizarea tabelului de comenzi, pentru afișarea sumei totale de plată pentru comanda curentă.*

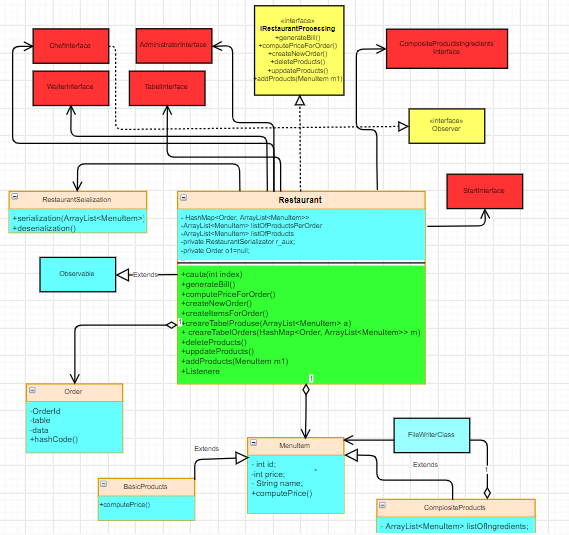
1. *Clasa CompositeProductIngredients*

*Aceasta clasă deschide un frame suplimentar în momentul în care se dorește introducerea*

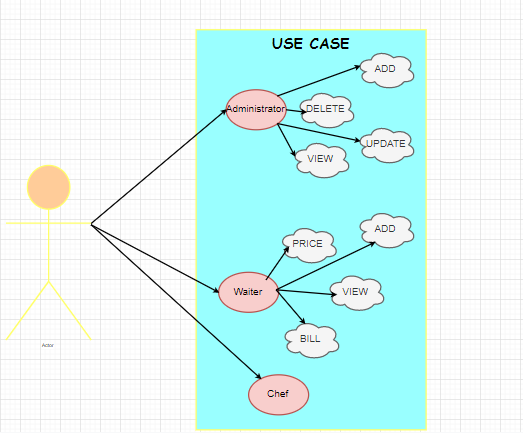
*unui Composite Product, utilizatorul fiind nevoit să aleagă o opțiune dintre Base Product sau Composite Product pentru a indica din ce este format Composite Product-ul pe care dorim să îl creeăm.*

***Diagrame UML***

***●Diagrama de clase***

******

***●Diagrama USE CASE***

******

***Pașii de execuție a programului:***

*Inițial, se trebuie create obiectele de tip Meniu Items pentru a fi utilizate de către chelner care prin intermediul lor, creează o comandă care ulterior urmează să fie procesată de către bucătar, stocată în Hashmap prin intermediul metodei de ”put” specifica acestui tip de date, fiecare comandă identificându-se în mod unic prin intermediul id-ului introdus de către chelner.*

*Se pot aduce modificări asupra produselor și afla informații legate de comenzi, interfața grafică fiind destul de flexibilă si prietenoasă cu utilizatorul, în sensul în care te avertizează daca a fost uitat un spațiu necompletat.*

***Concluzi***

*În concluzie, această tema a fost destul de benefică pentru mine, deoarece au fost repetate unele concepte pe care le-am învățat, precum serializarea datelor într-un fișier .ser, aceste date fiind salvate odată cu închiderea programului și extrase la pornirea programului, fiind necesare pentru popularea stocului restaurantului. Ideea de utilizare a interfeței Observable, care ajută prin intermediul metodei notifyObservers să indice faptul că s-au realizat niște modificări de care trebuie ținut cont. Realizând acest proiect, am putu să îmi dau seama cum funcționează întreaga acțiune din cadrul unui restaurant, acest sistem fiind structurat in 3 parți importante, chef, waiter și administrator, fiecare având anumite atribuții si responsabilități de care depinde întreaga funcționare a acelui centru comercial.*

***BIBLIOGRAFIE***

<http://users.utcluj.ro/~igiosan/teaching_poo.html>