MINISTERUL EDUCAȚIEI, CERCETĂRII, TINERETULUI ȘI SPORTULUI
UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

Restaurant Management System

Gherghel Daniel-Andrei

Grupa 30224

Îndrumător de laborator: Claudia Pop

Cuprins

1. Cerințe funcționale..... 3
2. Obiective..... 3
2.1. Obiectiv principal..... 3
2.2. Obiective secundare..... 3
3. Analiza problemei..... 3
4. Proiectare..... 4
4.1. Diagrama de clase UML..... 4
4.2. Clase și algoritmi folosiți..... 4
5. Concluzii și dezvoltări ulterioare..... 10
6. Bibliografie..... 11

1. Cerințe funcționale

Să se implementeze un 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 meniu. Chelnerul poate crea o comandă nouă pentru o masă, poate adăuga elemente din meniu și poate calcula facture pentru o comandă. Bucătarul este informat de fiecare dată când trebuie să gătească alimente comandate prin intermediul unui chelner.

2. Obiective

2.1. Obiectiv principal

Obiectivul principal al proiectului este de a crea o aplicație prin intermediul căreia să se poata efectua operatii ata de catre administratorul unui restaurant cat si de catre chelner.

2.2. Obiective secundare

Design by Contract Programming Techniques

Polymorphism

Design Patterns: Observer, Composite

JCF HashMap and HashSet implementations

Serialization

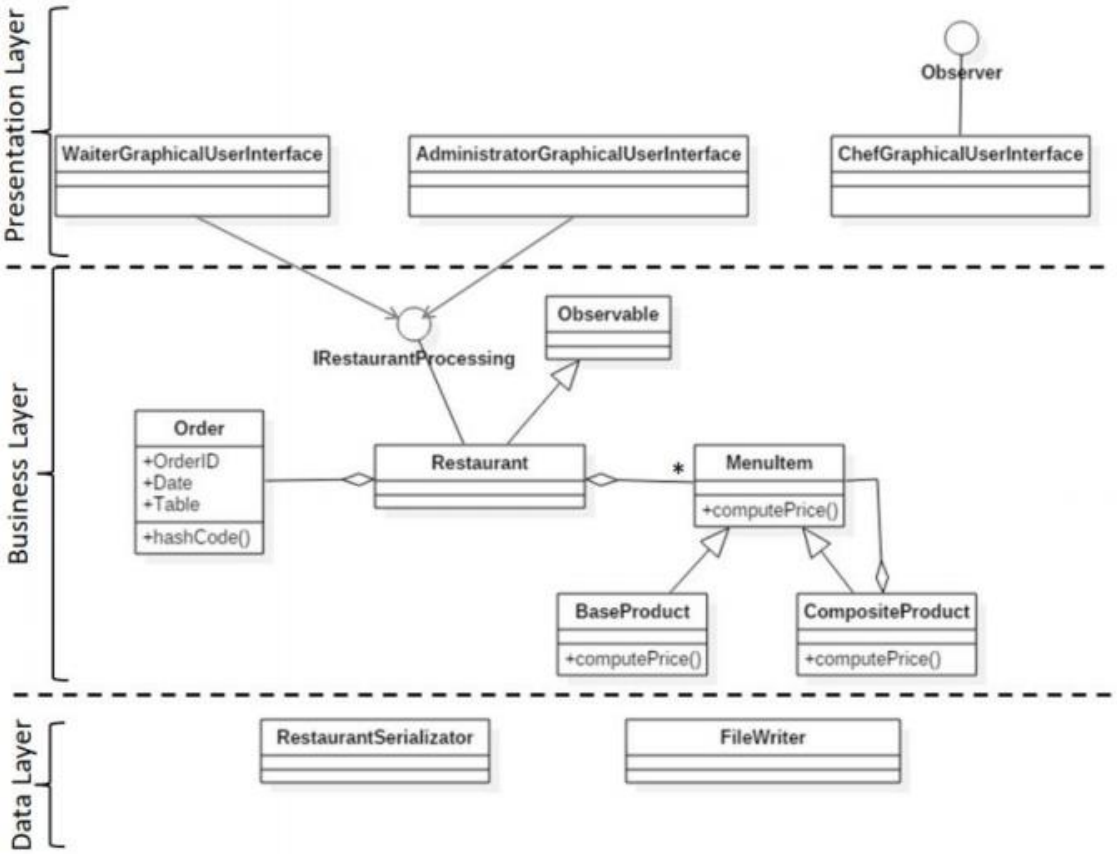
3. Analiza problemei

Un restaurant este un stabiliment, local public unde se pot consuma pe loc mâncăruri și băuturi, contra cost. De obicei mâncărurile sunt preparate în bucătăria proprie, de către o echipă specializată (bucătar, ajutor de bucătar), în coordonarea și supravegherea unui șef-bucătar.

Organizarea unui restaurant in cazul actual este realizata sub urmatoarea forma : administratorul care are dreptul de a efectua modificari asupra meniului, chelnerul care poate efectua preluarea de comenzi, de a face facturi si bcatarul care trebuie sa execute prepararea produselor preluate prin comenzi.

4. Proiectare

4.1. Diagrama de clase UML



4.2. Clase și algoritmi folosiți

Pachetul **Business** conține clasele de baza ale aplicației. În pachet se găsesc clasele: *MenuItem, Order, Restaurant si interfetele Obsrver1 si Observable*. Interfetele defines metodele utilizate pentru a crea legatura dintre interfata ChefInterface si WaiterInterface.

Clasa MenuItem defineste proprietatile unui produs aflat in restaurant: id, nume, gramaj si pret, precum si suprascrierea metodei equals.

Clasa Order defineste proprietatile unei comenzi: id order, data si masa la care se efectueaza comanda. De asemenea clasa implementeaza si o metoda de generare a unui hashCode unic utilizat pentru popularea hashMap-ului unde sunt stocate comezile in clasa restaurant.

```
public int hashCode() {
        int hash = 7;
        hash = 31 * hash + (int) this.orderId;
        hash = 31 * hash + (this.data == null ? 0 :
this.data.hashCode());
        hash = 31 * hash + (int) this.table;
        return hash;

    }
```

Clasa Restaurant implementeaza metodele de adaugare de MenuItem-uri cat si de comenzi, stergerea si update-ul unui MenuItem. Clasa mai implemeneteaza si metoda de calculare a pretului unue comenz. Tot aici sunt implemetate si metodele declarate in clasa Observable: addChef (observer), dleteChef (delete observer) si notifyChef, utilizata pentru a trimite notificare Chef-lui.

```
public void addOrder(Order o,ArrayList<MenuItem> list)
	{
		isWellFormed();
		orders.put(o,list);
	}

	public double getPrice(Order o)
	{
		double pret;
		pret=0;
		for(MenuItem c: orders.get(o))
			{
				pret+=c.getPret();
			}
		return pret;
```

MenuItem-urile sunt retinute intr-un array list, pe cand comenzile impreuna cu lista de MenuItem-uri sunt retinute intr-un HashMap pe baza unui hashCode generat in functie de detaliile comenzii.

`Pachetul` **`Data`** contine clasa: RestaurantSerialization. Clasa implementeaza metodele de serializare si deserializare pentru listele de MenuItem-uri si comenzi. Serializarea efctueaza practic salvarea informatiilor intr-un fisier text, informatii care la apelarea deserializarii sunt preluate si reintroduse in “liste”.

```
public void serItems( ArrayList<MenuItem> mItems){
      this.mItems = mItems;
      try {
         FileOutputStream fileOut =
         new FileOutputStream("item.ser");
```

```
            ObjectOutputStream out = new ObjectOutputStream(fileOut);
            out.writeObject(mItems);
            out.close();
            fileOut.close();
            System.out.printf("Serialized data is saved in item.ser");
         } catch (IOException i) {
            i.printStackTrace();
         }

      }
```

Pachetul **PT2019** contine clasa App care contine functia principala de unde se deschide interfata cu utilizatorul, care este reprezentata printr-o fereastra de comada de unde se poate deschide fereastra pentru administrator, chlner sau chef.

Pachetul **Presentation** contine 5 clase unde se modeleaza interfata.

Clasa Control creeaza Frame-ul principal care contine 3 butoane de unde se selecteaza operatorul care doreste sa utilizeze aplicatia.

Clasa AdministratorInterface are rolul de a realiza Frame-ul unde sunt disponibile operatiile asupra listei de MenuItem-uri. Frame-ul are 4 textField-uri unde se introduce datele legate de un MenuItem si 4 butoane de unde se pot executa operatiile: add, delete, update si showall. Clasa contine, de asemenea, si un JTable care este populat cu lista de MenuItem-uri. Dupa deschiderea aplicatiei se recomanda apasarea butonului Shoe All care va afisa in JTable toate articolele din meniu existente. Toate celelalte operatii sunt insotite de serializare si deserializare, apasarea oricarui buton echivalent unei operatii este vizibil imediat asupra tabelului. Popularea JTable-ului se face cu ajutorul functiei:

```
public void getColumns() {

      table=new JTable();
    String[] columns = {"Item Id","Name","Gramaj","Pret"};

    DefaultTableModel defm=new DefaultTableModel(columns,0);
    getTableData(defm);
    //return columns;
      contentPane.add(table);
      JScrollPane scrollPane = new JScrollPane();
      scrollPane.setBounds(23, 148, 371, 125);
      this.getContentPane().add(scrollPane);
      table = new JTable(defm);
      scrollPane.setViewportView(table);
```

```
}
```

Clasa Bill are rolul de a realiza Frame-ul unde se deschide o factura sub forma unui Jtable care contine toate produsele comandate . Frame-ul are si un textField care contine pretul total al comenzii. Populatea JTable-ului se face cu ajutorul acelorasi functii.

Clasa ChefInterface are exact aceeasi structura ca si clasa Bill cu exceptia faptului ca are doar Jtable-lul. Tabelul se populeaza cu produsele continute la adaugarea fiecarei comenzi, produse care trebuie pregatite de catre Chef. Transmiterea informatiilor se face prin intermediul metodei notifyChef din clasa Restaurant.

Clasa WaiterInterface deschide Frame-ul contine toate comenzile pe care le poate efectua chelnerul restaurantului.

In primul rand contine 3 textField-uri unde se introduce informatiile cu privire la comanda: idul comenzii, data comenzii si masa de la care se face comanda.

Frame-ul contine si doua JTable-uri, primul contine toate produsele care se gasesc in meniu si se autopopuleaza la apasarea butonului waiter din interfata Control, al doilea contine comenzile plasate cu detaliile fiecarei comenzi si se actualizeaza la fiecare adaugare de comanda, instantaneu.

Frame-ul contine un buton numit “Price” care calculeaza pretul comenzii cu informatiile introduce in textField0urile corespunzatoare si il afiseaza in textField-ul din partea de jos a ferestrei ca un “double”, reprezentand suma in RON.

In fereastra se mai gaseste si un buton “Bill” care genereaza o factura pe baza informatiilor extrase din HashMap-ul din clasa Restaurant. Intr-o fereastra noua, dar si o factura sub forma unui fisier text.

```
btnBill.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent arg0) {
                  Order o=new Order();
                  setOrder(o);
                  Bill b=new Bill(o,r);
                  b.setVisible(true);
                  double pret;
```

```
                    pret=r.getPrice(o);
                    String s;
                    s="Item Id   Name    Gramaj    Pret\n";

                    c=r.getOrders().get(o);
                  for (int i = 0; i < c.size(); i++) {

                  s+="   "+c.get(i).getItemId()+"    "+c.get(i).getName()+"
"+c.get(i).getGramaj()+"        "+c.get(i).getPret()+"\n";


                  }
                  s+="TOTAL: "+pret;
                  generateBill(s);


              }

                    });

public void generateBill(String data)
      {
              BufferedWriter writer = null;
              try {
                  //create a temporary file
                  String timeLog = new
SimpleDateFormat("yyyyMMdd_HHmmss").format(Calendar.getInstance().getTime());
                  File logFile = new File(timeLog);

                  // This will output the full path where the file will be
written to...
                  System.out.println(logFile.getCanonicalPath());

                  writer = new BufferedWriter(new FileWriter(logFile));
                  writer.write(data);
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            try {
                // Close the writer regardless of what happens...
                writer.close();
            } catch (Exception e) {
            }
        }

             }
```

Si in final o camanda se plaseaza urmand urmatorii pasi: se introduce informatiile comenzii pe textField-urile corespunzatoare si pe urma se incepe introducerea produselor care se comanda prin introducerea numelui produselor in textField-ului de deasupra butonului “Add Item” urmata de apasarea butonului. La fiecare apasare a butonului se introduce in comanda

un produsul specificat, de retinut ca o comanda trebuie sa contina minimum 2 produse. Fiecare produs introdus se introduce intr-o lista de produse pe baza careia se face comanda si se introduce in HashMap-ul de comenzi.

La finalizarea introducerii de produse se apasa butonul “Add” moment in care comanda este plasata. Apasarea butonului are ca si effect si serializarea comenzii si actualizarea tabelului unde sunt afisate comenzile.

```
btnAdd.addActionListener(new ActionListener() {
		public void actionPerformed(ActionEvent arg0) {
			Order o=new Order();
			setOrder(o);
			for(MenuItem c: list)
				System.out.println(c.getName()+ " "+c.getGramaj());
			if(list.size()>=2) {
			r.addOrder(o, list);

			r.addChef(chef);
			r.notifyChef(list);
			System.out.println(r.getOrders().toString());
			list=new ArrayList<MenuItem>();
			//r.setOrders(null);
			rs.serOrders(r.getOrders());
			getColumns1();
			}
			else {
				System.out.println("Trebuie introduse mai multe
	produse la comanda!");
			}

		}
```

5. Concluzii și dezvoltări ulterioare

Aplicația implementată este ușor de folosit și realizează toate operațiile necesare pentru activitatea intr-un restaurant. Aplicația poate fi dezvoltată adăugând mai multe comenzi pentru utilizatori, cum ar fi anularea unei comenzi sau calcularea salariilor angajatilor sau totalul incasrilor efectuate de fiecare chelner sau chiar incasarile totale pe o zi a restaurantului. De asemenea in aplicatie se poate introduce si efectuarea de plati si tipurile de plati care se pot efectua. Este ideala pentru utilizare in cadrul oricarui tip de restaurant.

Bibliografie

https://docs.oracle.com/cd/E19683-01/806-7930/assert-13/index.html

https://stackoverflow.com/questions/15754523/how-to-write-text-file-java

https://ro.wikipedia.org/wiki/Restaurant

https://www.geeksforgeeks.org/hashmap-get-method-in-java/

https://www.geeksforgeeks.org/overriding-equals-method-in-java/

https://www.baeldung.com/java-hashcode

https://www.geeksforgeeks.org/hashmap-put-method-in-java/

http://www.tutorialspoint.com/java/java_serialization.htm

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