Assignment 4

**Restaurant Management System**

Alexuc Tudor  
Grupa 30221/1 - An2 CTI

**1.Obiectivul temei**

**Cerinta :**

Obiectivul acestui proiect este de a modela si implementa o aplicatie care realizeaza operatiile de baza pentru un restaurant. Aplicatie are ca functionalitati adaugarea, editarea, stergerea si vizualizarea meniului si posibilitatea chelnerului de a lua comenzi.

Spre indeplinirea obiectivului principal trebuie respectate si urmate anumite obiective secundare, precum folosirea unor design patterns pentru functionarea corecta a aplicatiei si folosirea HashMap-urilor.

Aplicatia va avea 2 tipuri de utilizatori (Administrator si Waiter) :

* Administratorul are urmatoarele functii
  + crearea de noi obiecte pentru meniu
  + stergerea obiectelor din meniu
  + editarea obiectelor din meniu
* Chelnerul are urmatoarele functii
  + crearea unei noi comenzi
  + calcularea pretului unei comenzi
  + generarea chitantei sub forma .txt

De asemenea se cere ca aplicatia sa poata serializa si deserializa datele stocate in clasa Restaurant.

**2. Analiza problemei**

Ordinea rezolvarii a fost:

1. Realizarea claselor pentru MenuItem
2. Realizarea clasei Restaurant cu functiile necesare claselor de control pentru Administrator si Waiter
3. Realizarea claselor pentru interfetele grafice pentru userii Admin si Chelner
4. Relizarea serializarii

Aplicatia proiectata trebuie sa implementeze functionarea unui restaurant. Astfel, exista 2 tipuri de utilizatori si anume administrator si chelner.

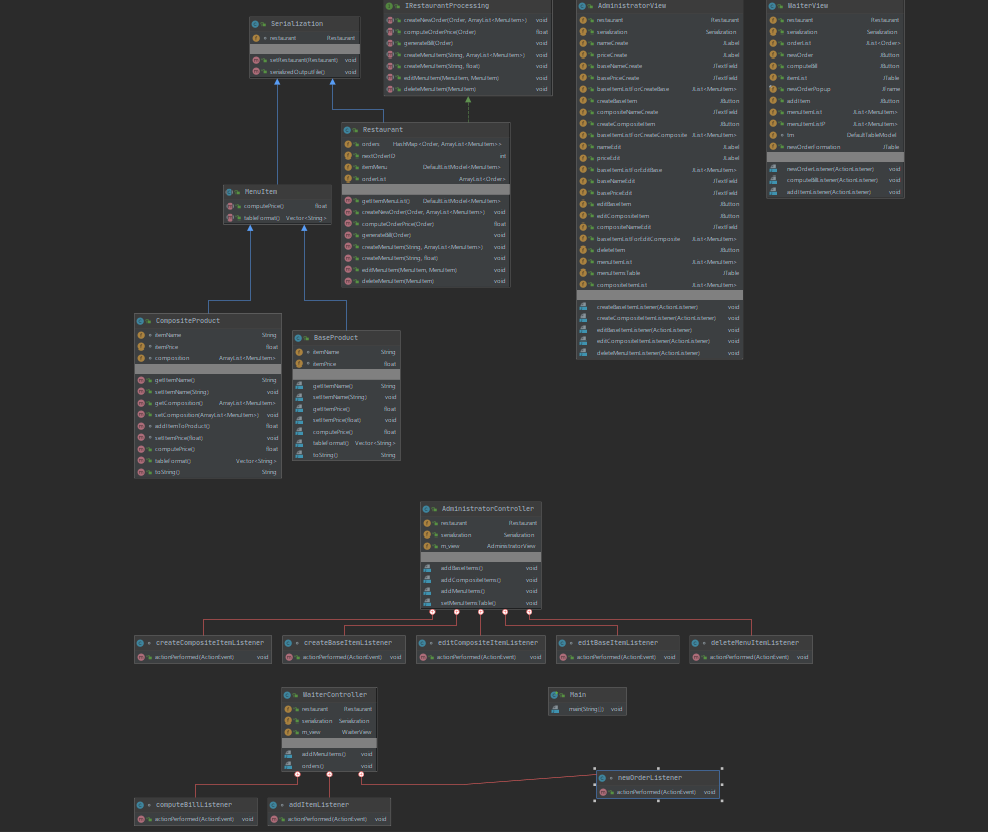
Fiecare utilizator are de realizat operatiile mentionate mai sus.

Administratorul trebuie sa vizualizeze meniul restaurantului si poate sa adauge produse in acest meniu, sa le stearga, sau sa le modifice. Aceste produse care se afla in meniu pot fi de 2 tipuri. Ele pot fi produse simple, pe care un client le poate comanda in orice combinatie cu alte produse, sau produse compuse din alte produse aflate in meniu. Aici intervine composite desgin pattern-ul pentru clasa MenuItem.

Chelnerul poate sa creeze o noua comanda, sa calculeze totalul pentru o comanda si sa genereze nota de plata.

**3.Proiectare**

**Diagrama UML:**



**Decizii de proiectare:**

O decizie importanta de implementare este realizarea functiilor principale pentru Chelner si Admin in clasa Restaurant care implementeaza interfata IRestaurantProcessing.

Astfel, clasa Restaurant va lua locul de model din structura MVC atat pentru Admin cat si pentru Chelner , de aceea ea este si clasa serializata.

O alta decizie de implementare este realizarea serializarii la inchiderea ferestrei principale a administratorului sau a chelnerului.

Pentru manipularea produselor introduse clasele View vor folosi JList pentru modul lor intuitiv de utilizare.

**Pachete:**

Aplicatia foloseste structura Buisiness Data Presentation si are cate un package pentru fiecare strat.

**4.Implementare**

Clasa Restaurant va contine functiile principale de manipulare si creare a datelor pentru orders si menu items.

Structurile MVC pentru Admin si Chelner se vor ocupa de interfetele grafice si folosirea claselor din Restaurant plus cateva clase aditionale pentru introducerea datelor in JLists.

**Clasa Restaurant :**

public HashMap<Order, ArrayList<MenuItem>> orders = new HashMap<>();  
public int nextOrderID = 1;  
public DefaultListModel<MenuItem> itemMenu = new DefaultListModel<>();  
public ArrayList<Order> orderList = new ArrayList<>();  
  
  
public DefaultListModel<MenuItem> getItemMenuList()  
{  
 return itemMenu;  
}  
  
@Override  
public void createNewOrder(Order newOrder, ArrayList<MenuItem> orderedItems)  
{  
 LocalDate currentDate = LocalDate.*now*();  
 newOrder.setDate(currentDate.toString());  
 newOrder.setOrderID(nextOrderID);  
 this.nextOrderID++;  
  
 orders.put(newOrder,orderedItems);  
 orderList.add(newOrder);  
}  
  
@Override  
public float computeOrderPrice(Order order)  
{  
 float sum = 0.0f;  
 for(MenuItem mi : orders.get(order))  
 {  
 sum = sum + mi.computePrice();  
 }  
 return sum;  
}  
  
@Override  
public void generateBill(Order order)  
{  
 File file = new File("Bill for order" + order.getOrderID() +".txt");  
  
 try {  
 file.createNewFile();  
 FileWriter writer = new FileWriter(file);  
 writer.write("Bill No."+ order.getBillID() + " :\n");  
 orders.get(order).forEach( (menuItem) ->  
 {  
 try  
 {  
 writer.write(menuItem + "\n");  
 } catch (IOException e)  
 {  
 e.printStackTrace();  
 }  
 });  
 writer.write("For the price of " + computeOrderPrice(order) +".");  
 writer.flush();  
 writer.close();  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
}  
  
@Override  
public void createMenuItem(String name, ArrayList<MenuItem> productComposition)  
{  
 CompositeProduct item = new CompositeProduct();  
 item.setItemName(name);  
 item.setComposition(productComposition);  
 item.setItemPrice(item.computePrice());  
 itemMenu.addElement(item);  
}  
  
@Override  
public void createMenuItem(String name, float price)  
{  
 BaseProduct item = new BaseProduct();  
 item.setItemName(name);  
 item.setItemPrice(price);  
 itemMenu.addElement(item);  
}  
  
@Override  
public void editMenuItem(MenuItem item, MenuItem editedItem)  
{  
 itemMenu.set(itemMenu.indexOf(item),editedItem);  
}  
  
@Override  
public void deleteMenuItem(MenuItem item)  
{  
 itemMenu.removeElement(item);  
}

Functiile de introducere a datelor in liste si tabel din AdministratorController

public void addBaseItems()  
{  
 DefaultListModel<MenuItem> baseItemListModel = new DefaultListModel<>();  
 for(int i = 0;i < restaurant.getItemMenuList().getSize();i++)  
 {  
 if(restaurant.getItemMenuList().get(i) instanceof BaseProduct)  
 {  
 baseItemListModel.addElement(restaurant.getItemMenuList().get(i));  
 }  
 }  
  
 m\_view.baseItemListForCreateBase.setModel(baseItemListModel);  
 m\_view.baseItemListForCreateComposite.setModel(baseItemListModel);  
 m\_view.baseItemListForEditBase.setModel(baseItemListModel);  
 m\_view.baseItemListForEditComposite.setModel(baseItemListModel);  
 serialization.setRestaurant(restaurant);  
 serialization.serializedOutputFile();  
}  
  
public void addCompositeItems()  
{  
 DefaultListModel<MenuItem> compositeItemListModel = new DefaultListModel<>();  
 for(int i = 0;i < restaurant.getItemMenuList().getSize();i++)  
 {  
 if(restaurant.getItemMenuList().get(i) instanceof CompositeProduct)  
 {  
 compositeItemListModel.addElement(restaurant.getItemMenuList().get(i));  
 }  
 }  
 m\_view.compositeItemList.setModel(compositeItemListModel);  
 serialization.setRestaurant(restaurant);  
 serialization.serializedOutputFile();  
}  
  
public void addMenuItems()  
{  
 DefaultListModel<MenuItem> menuItemListModel = new DefaultListModel<>();  
 for(int i = 0;i < restaurant.getItemMenuList().getSize();i++)  
 {  
 menuItemListModel.addElement(restaurant.getItemMenuList().get(i));  
 }  
 m\_view.menuItemList.setModel(menuItemListModel);  
 setMenuItemsTable();  
 serialization.setRestaurant(restaurant);  
 serialization.serializedOutputFile();  
}  
  
public void setMenuItemsTable()  
{  
 DefaultTableModel model = new DefaultTableModel();  
 model.setColumnCount(7);  
 for(int i = 0;i < restaurant.getItemMenuList().getSize();i++)  
 {  
 model.addRow(restaurant.getItemMenuList().get(i).tableFormat());  
 }  
 m\_view.menuItemsTable.setModel(model);  
}

Evenimentul de inchidere a ferestrelor care declanseaza **serializarea** :

view.addWindowListener(new java.awt.event.WindowAdapter()  
{  
 @Override  
 public void windowClosing(java.awt.event.WindowEvent windowEvent)  
 {  
 {  
 serialization.setRestaurant(restaurant);  
 serialization.serializedOutputFile();  
 System.*exit*(0);  
 }  
 }  
});

Clasa de implementare a **serializarii**

public class Serialization implements Serializable  
{  
 Restaurant restaurant;  
  
 public void setRestaurant(Restaurant restaurant) {  
 this.restaurant = restaurant;  
 }  
  
 public void serializedOutputFile()  
 {  
 try {  
 FileOutputStream fout = new FileOutputStream("restaurant.ser");  
 ObjectOutputStream out = new ObjectOutputStream(fout);  
 out.flush();  
 } catch (FileNotFoundException e) {  
 e.printStackTrace();  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
}

Clasa **Order** contine settere si gettere si functiile necesare HashMap-ului

private int orderID;  
 private String date;  
 private int table;  
 private ArrayList<Integer> tableStatus;  
 private float price;  
 private int billID;  
  
 public Order()  
 {  
 this.setPrice(0.0f);  
 tableStatus = new ArrayList<>();  
 orderID = 1;  
 billID = 1;  
 }  
  
 public Order(int orderID, String date, int table)  
 {  
 super();  
 this.tableStatus = new ArrayList<>();  
 this.orderID = orderID;  
 this.date = date;  
 this.table = table;  
 this.setPrice(0.0f);  
 }  
  
 public int getBillID() {  
 return billID;  
}  
 public void setBillID(int billID) {  
 this.billID = billID;  
}  
 public int getOrderID() {  
 return orderID;  
 }  
 public void setOrderID(int orderId) {  
 this.orderID = orderId;  
 }  
 public String getDate() {  
 return date;  
 }  
 public void setDate(String date) {  
 this.date = date;  
 }  
 public int getTable() {  
 return table;  
 }  
 public void setTable(int table) {  
 tableStatus.set(table,1);  
 this.table = table;  
 }  
 public int getTableStatus()  
 {  
 return tableStatus.get(this.table);  
 }  
 public float getPrice() {  
 return price;  
 }  
 public void setPrice(float price) {  
 this.price = price;  
 }  
  
 @Override  
 public int hashCode() {  
 return this.orderID;  
 }  
  
 @Override  
 public boolean equals(Object o) {  
 if( o == null) {  
 return false;  
 }  
 if(!(o instanceof Order)) {  
 return false;  
 }  
  
 if(o == this) {  
 return true;  
 }  
 return this.getOrderID() == ((Order) o).getOrderID();  
 }  
  
@Override  
public String toString() {  
 return "Order{" +  
 "orderID=" + orderID +  
 '}';  
}

Composite design pattern-ul pentru **MenuItem**:

public abstract class MenuItem extends Serialization  
{  
 public abstract float computePrice();  
 public abstract Vector<String> tableFormat();  
}

public class BaseProduct extends MenuItem  
{  
 String itemName;  
 float itemPrice;  
  
 public String getItemName() {  
 return itemName;  
 }  
  
 public void setItemName(String itemName) {  
 this.itemName = itemName;  
 }  
 public float getItemPrice()  
 {  
 return itemPrice;  
 }  
 public void setItemPrice(float price)  
 {  
 this.itemPrice = price;  
 }  
 @Override  
 public float computePrice()  
 {  
 return itemPrice;  
 }  
  
 @Override  
 public Vector<String> tableFormat()  
 {  
 Vector<String> result = new Vector<>();  
 result.add(0,this.itemName);  
 result.add(1,String.*valueOf*(this.computePrice()));  
 return result;  
 }  
  
 @Override  
 public String toString()  
 {  
 return this.getItemName();  
 }  
}

public class CompositeProduct extends MenuItem  
{  
 String itemName;  
 float itemPrice;  
 ArrayList<MenuItem> composition = new ArrayList<>();  
  
 public String getItemName() {  
 return itemName;  
 }  
 public void setItemName(String itemName) {  
 this.itemName = itemName;  
 }  
 public ArrayList<MenuItem> getComposition() {  
 return composition;  
 }  
 public void setComposition(ArrayList<MenuItem> composition) {  
 this.composition = composition;  
 }  
 float addItemToProduct()  
 {  
 return itemPrice;  
 }  
 void setItemPrice(float price)  
 {  
 this.itemPrice = price;  
 }  
  
 @Override  
 public float computePrice()  
 {  
 float sum = 0.0f;  
 for(MenuItem product : this.composition)  
 {  
 sum = sum + product.computePrice();  
 }  
 return sum;  
 }  
  
 @Override  
 public Vector<String> tableFormat() {  
 Vector<String> result = new Vector<>();  
 int index = 2;  
 result.add(0,this.itemName);  
 result.add(1,String.*valueOf*(this.computePrice()));  
  
 for(MenuItem b : this.composition)  
 {  
 result.add(index,b.toString());  
 }  
 return result;  
 }  
  
 @Override  
 public String toString()  
 {  
 return this.getItemName();  
 }  
}

**5.Testare**

**Clasa main + .jar file**

Citirea datelor din fisier se face cu ajutorul java.io.FileInputStream; java.io.IOException si java.io.ObjectInputStream pentru citirea din fisierul serializat.

Programul a fost testat pe cazuri adaugate din interfata grafica a Administratorului si folosite apoi cu interfata grafica a Chelnerului.

**6.Concluzii**

Programul este un exercitiu bun pentru intelegerea si utilizarea HashMap-urilor si a sistemelor complexe cu mai multe interfete grafice . Astfel , am realizat importanta crearii unei legaturi clare intre grupurile de clasa pentru fiecare user pentru a asigura transmiterea corecta a claselor, si modul in care serializarea datelor creaza o salvare usor de folosit a datelor pentru utilizari multiple asupra aceluiasi set de date.

Aplicatia poate fi extinsa prin adaugarea altor functii de creare sau procesare a produselor din meniu si poate de astemenea sa fie modificata pentru o legatura cu o baza de date pentru salvarea meniului si a comenzilor.

**7.Bibliografie**

Materialul ajutator: Hashing in Java

https://dzone.com/articles/layers-standard-enterprise

https://stackoverflow.comhttps://docs.oracle.com/javase/7/docs/api/javax/swing/JScrollPane.html

https://www.tutorialspoint.com/design\_pattern/composite\_pattern.html

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