A picture containing drawing

Description automatically generated

# Tema 4

# RESTAURANT MANAGEMENT SYSTEM

Indrumator de laborator: Moldovan Dorin

Student: Manici Valentin-Dan

Grupa: 30226

# Contents

Tema 4 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .. . . . . . . 1

Restaurant management system. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .1

1.Obiectivul temei. . . .. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 3

Obiective secundare. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .3

2.Analiza problemei, modele, scenarii, cazuri de utilizare . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 3

[3. Proiectare. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .](#_Toc36769940)3

[4.Implementare. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .](#_Toc36769945)7

[5.Concluzii. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 11](#_Toc36769946)

[6.Bibliografie. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 11](#_Toc36769947)

# 1.Obiectivul temei

Implementarea unui proiect de tipul Restaurant management system.

## Obiective secundare:

1. Systemul ar trebui sa aiba trei tipuri de utilizatori: administrator, waiter si chef
2. Serializarea restaurantului
3. Generarea de facturi in format txt

# 2.Analiza problemei, modelare, scenarii, cazuri de utilizare

Problema cere respectarea unei diagrame. Aceasta contine trei tipuri de utilizatori: administratorul care trebuie sa aiba posibilitatea sa adauge un produs, sa stearga un produs, sau sa modifice un produs din meniul restaurantului; waiter-ul care trebuie sa aiba posibilitatea de a crea un order, sa adauge elemente din meniu si sa calculeze factura pentru comanda, de asemenea chef-ul trebuie notificat prin waiter cand trebuie sa gateasca.

Trebuie definita o structura pentru a stoca informatiile despre orders si o structura pentru a stoca elementele din meniul restaurantului.

Meniul restaurantului trebuie salvat si incarcat utilizand serializarea.

# 3. Proiectare

## Decizii de implementare

Am decis sa stochez datele din meniul restaurantului intr-un ArrayList, iar informatiile despre order le voi stoca intr-un HashMap.

## Pachete

Programul este organizat in trei pachete:

* “DataLayer” : Acest pachet contine clasele FileWriter si RestaurantSerializator, care sunt necesare pentru salvarea meniului prin serializare si generarea notei de plata in format txt.
* “BusinessLayer” : Acest pachet contine clasele BaseProduct, CompositeProduct, MenuItem, Observer, Order, Restaurant, RestaurantProcessing, aceste clase au rol in managementul restaurantului, modul in care sunt vazute produsele, ingredientele pentru produsele compuse, si modul in care se formeaza si se modifica meniul
* “PresentationLayer” : Acest pachet contine clasele AdministratorGui, ChefGui, WaiterGui, Main, clase care formeaza interfata cu utilizatorii, respectiv Clasa Main din care se controleaza programul.

## Proiectare clase

1. Clasa MenuItem : contine campurile id, name

Constructorul public MenuItem(int id, String name) {}

Metodele:

public int getId() { }

public void setId(int id) { }

public String getName() { }

public void setName(String name) { }

public abstract double computePrice()

1. Clasa BaseProduct extinde clasa MenuItem: contine campul price

Constructorul public BaseProduct(int id, String name, double price) { }

Metodele:

public double getPrice() { }

public void setPrice(double price) { }

public double computePrice() { }

1. Clasa CompositeProdus extinde clasa MenuItem: contine campurile price si ingredients care reprezinta un ArrayList de MenuItem

Constructorul public CompositeProduct( int id, String name, double price, List<MenuItem> ingredients) { }

Metodele:

public double computePrice() { }

public void addItem(MenuItem item) { }

public void deleteItem(String name) { }

public List<MenuItem> getIngredients() { }

public void setIngredients(List<MenuItem> ingredients) { }

1. Clasa Order contine campurile orderID, date, table

Constructorul public Order(int orderID, String date, int table) { }

Metodele:

public int hashCode() { }

public boolean equals(Object obj) { }

public int getId() { }

public void setId(int id) { }

public int getTable() { }

public void setTable(int table) { }

1. Clasa Observer contine campul subject de tipul Subject

Metoda public abstract void update(String text)

1. Clasa Subject contine campul observers reprezentat de o lista de observatori si campul state

Metode:

public String getState() { }

public void setState(String state) { }

public void attach(Observer observer){ }

1. Clasa Restaurant care implementeaza interfata RestaurantProcessing contine campurile menu care este reprezentat de un ArrayList de MenuItem si de campul orderInfo reprezentat de un HashMap care utilizeaza orders pentru cheie

Constructori:

public Restaurant() { }

public Restaurant(List<MenuItem> menu, Map<Order, ArrayList<MenuItem>> orderInfo) { }

Metode:

public void createNewMenuItem( int id, String name, double price, ArrayList<MenuItem> ingredients) { }

public void deleteMenuItem(String name) { }

public void editMenuItem(int id, String name, double price, ArrayList<MenuItem> ingredients) { }

public Order createNewOrder(int id, String date, int table, ArrayList<MenuItem> meniu) { }

public double computePrice(Order o) { }

public boolean isWellFormed() { }

public ArrayList<MenuItem> getItemList(){ }

public void generateBill(int order, String date, int tableNb, double price)

public void serializeRestaurant() { }

public List<MenuItem> getMenu() { }

public void setMenu(List<MenuItem> menu) { }

public Map<Order, ArrayList<MenuItem>> getOrderInfo() { }

public void setOrderInfo(Map<Order, ArrayList<MenuItem>> orderInfo) { }

1. Interfata RestaurantProcessing contine metodele:

public void createNewMenuItem(int id, String name, double price, ArrayList<MenuItem> ingredients);

public void deleteMenuItem(String name);

public void editMenuItem(int id, String name, double price, ArrayList<MenuItem> lst);

public Order createNewOrder(int id, String date, int table, ArrayList<MenuItem> menu);

public double computePrice(Order o);

public void generateBill(int order, String date, int tableNb, double price);

public void serializeRestaurant();

public ArrayList<MenuItem> getItemList();

1. Clasa FileWriter contine metoda

public static void createBill(int order, String date, int tableNb, double price) { }

1. Clasa RestaurantSerializator contine metodele

public static void serializeRestaurantData(Restaurant restaurant) { }

public static Restaurant deserializeRestaurantData() { }

1. Clasa AdministratorGui reprezinta interfata pentru administrator
2. Clasa ChefGui reprezinta interfata pentru chef
3. Clasa WaiterGui reprezinta interfata pentru waiter
4. Clasa Main de aici se ruleaza programul

# Diagrame UML

A screenshot of a cell phone

Description automatically generated

1. Implementare

A screenshot of a social media post

Description automatically generated

Interfata administratorului contine butoanele create, edit, delete care daca sunt apasate vor executa actiunile permise pentru acest tip de utilizator. Pentru a fi posibila crearea unui produs este nevoie sa se populeze campurile din drepta: id, name, si price. Pentru a se crea un produs compus in campul destinat ingredientelor se vor adauga aceste ingrediente, sub urmatoarea forma: id,name,price,id,name,price. Acesta forma va fi preluata de program si se vor crea ingredientele pentru produsele compuse, care vor aparea alaturi de produsul principal in tabelul din partea dreapta a interfetei.

Stergerea unui produs se face cu ajutorul numelui acestuia, se va putea urmari vizual in tabelul din dreapta care reprezinta meniul ca produsul selectat pentru a fi sters va fi eliminat din meniu. Codul prin care se implementeaza aceasta actiune este urmatorul:

butonDelete.addActionListener(**new** ActionListener() {

**public** **void** actionPerformed(ActionEvent e) {

**if** (e.getSource() == butonDelete) {

String name = nameT.getText();

**for** (**int** i = 0; i < restaurant.getMenu().size(); i++) {

**if** (name.equals(restaurant.getMenu().get(i).getName())) {

restaurant.deleteMenuItem(name);

}

}

**int** row=model.getRowCount();

**for** (**int** k = row - 1; k >= 0; k--) {

model.removeRow(k);

}

**for** (**int** j = 0; j < restaurant.getMenu().size(); j++) {

Vector v = **new** Vector();

v.add(restaurant.getMenu().get(j).getId());

v.add(restaurant.getMenu().get(j).getName());

v.add(restaurant.getMenu().get(j).computePrice());

model.addRow(v);

}

}

nameT.setText("");

}

});

Prin apasarea butonului Edit si prin introducerea numelui unui produs se poate modifica id-ul si / sau pretul acelui produs. Codul care realizeaza aceasta actiune este urmatorul:

butonEdit.addActionListener(**new** ActionListener() {

**public** **void** actionPerformed(ActionEvent e) {

**if** (e.getSource() == butonEdit) {

String name = nameT.getText();

**for** (**int** i = 0; i < restaurant.getMenu().size(); i++) {

**if** (name.equals(restaurant.getMenu().get(i).getName())) {

restaurant.editMenuItem(Integer.*parseInt*(idT.getText()), nameT.getText(),

Double.*parseDouble*(priceT.getText()),menu);

}

}

**int** row=model.getRowCount();

**for** (**int** k = row - 1; k >= 0; k--) {

model.removeRow(k);

}

**for** (**int** j = 0; j < restaurant.getMenu().size(); j++) {

Vector v = **new** Vector();

v.add(restaurant.getMenu().get(j).getId());

v.add(restaurant.getMenu().get(j).getName());

v.add(restaurant.getMenu().get(j).computePrice());

model.addRow(v);

}

}

idT.setText("");

nameT.setText("");

priceT.setText("");

ingredientsT.setText("");

}

});

A screenshot of a social media post

Description automatically generated

Interfata waiter contine tabelul in care se vor inregistra comenzile, butoanele Create si Compute si campurile in care se vor aduga id-ul comenzii, numarul mesei si numele produsului comandat. . . . . . . . . . . . . . . . . . . . . . . . . . . . .

La apasarea butonului create se va crea orderul daca toate cele trei campuri sunt completate: id, table, produs.

La apasarea butonui Compute se va crea factura in format txt pentru comanda respectiva.

butonCompute.addActionListener(**new** ActionListener() {

**public** **void** actionPerformed(ActionEvent e) {

**if** (e.getSource() == butonCompute) {

**if** (!produsT.getText().isEmpty()) {

**double** price=0;

**for** (**int** j = 0; j < restaurant.getMenu().size(); j++) {

**if**(produsT.getText().equals(restaurant.getMenu().get(j).getName())) {

price=restaurant.getMenu().get(j).computePrice();

}

}

restaurant.generateBill(Integer.*parseInt*(idT.getText()), df.format(dateobj),

Integer.*parseInt*(tableT.getText()),price);

o.update("idle");

}

}

}

});

A screenshot of a cell phone

Description automatically generated

In interfata chef-ului va aparea starea idle cand nu gateste si cooking for table tableNumber cand acesta este notificat de waiter, cand waiter-ul creaza o comanda, iar in momentul cand acesta calculeaza suma de plata pentru comanda respectiva chef-ul va reveni la starea idle. Observatorul este transmis ca parametru constructorului waiter.

. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

Un exemplu de afisarea al produselor, comenzilor si de afisare a starii sefului poate fi observat in urmatoarea imagine, in care sunt toate trei interfetele:

A screenshot of a computer

Description automatically generated

In clasa Restaurant in metoda createNewMenuItem se poate observa utilizarea aserturilor:

**public** **void** createNewMenuItem( **int** id, String name, **double** price, ArrayList<MenuItem> ingredients) {

**assert** name != **null**;

**assert** price != 0;

**assert** id != 0;

MenuItem product;

**int** size = menu.size();

**if**(ingredients.size() == 0)

{

product = **new** BaseProduct(id, name, price);

}

**else**

{

product = **new** CompositeProduct(id, name, price,ingredients);

}

menu.add(product);

**assert** size == menu.size() - 1;

}

In metoda se alege ce fel de produs este primit ca argument deoarece ambele produse extind clasa MenuItem, dar Composite produse contine in plus ingrediente reprezentate de catre alte produse.

De asemenea in metoda deleteMenuItem: pentru a se sterge un produs din meniu numele acestuia nu trebuie sa fie null.

**public** **void** deleteMenuItem(String name) {

**assert** name != **null**;

**for**(**int** i=0; i<menu.size(); i++) {

**if**(menu.get(i).name.equals(name))

{

menu.remove(i);

}

}

}

1. Concluzii

Am invatat cum sa implementez ca un meniu sa fie accesat de mai multe tipuri de utilizatori. De asemenea sa creez factura in format txt si sa serializez informatiile dintr-un obiect.

1. Bibliografie

<https://www.geeksforgeeks.org/internal-working-of-hashmap-java/>

<https://www.sitepoint.com/implement-javas-equals-method-correctly/>

<https://www.tutorialspoint.com/java/java_serialization.htm>

<https://www.geeksforgeeks.org/serialization-in-java/>

<https://www.tutorialspoint.com/design_pattern/observer_pattern.htm>

<https://www.guru99.com/java-swing-gui.html>

<https://stackoverflow.com/questions/11415160/how-to-enable-the-java-keyword-assert-in-eclipse-program-wise>