

**TEHNICI DE PROGRAMARE FUNDAMENTALE
ASSIGNMENT 4**

**RESTAURANT MANAGEMENT
SYSTEM**

DOCUMENTATIE

Bakk Cosmin-Robert
Grupa 30227

Cuprins

1. Obiectivul temei
2. Analiza problemei, modelare, scenarii, cazuri de utilizare
3. Proiectare
 - 3.1. Diagrama UML
 - 3.2. Proiectare clase
4. Implementare si testare
 - 4.1. Metode
 - 4.2. Testare
5. Concluzii
6. Bibliografie

1. Obiectivul temei

Obiectivul acestei teme de laborator este de a implementa un sistem de management al unui restaurant. Restaurantul are un meniu, care contine produse de baza (base products) sau produse compuse (composite products), care sunt compuse din mai multe produse (acestea fiind la randul lor base products sau composite products). Fiecare produs de baza are un nume si un pret. Sistemul de management poate fi accesat de catre 3 tipuri de utilizatori: Administrator, Waiter si Chef. Fiecare dintre acesti utilizatori poate sa execute anumite operatii:

- Administrator – vizualizeaza lista de produse din meniu, adauga produse (base sau composite), sterge produse sau editeaza produse (pentru base – schimba numele sau pretul, pentru composite – schimba numele sau adauga/sterge elemente in compozitia acestuia).
- Waiter – vizualizeaza lista de comenzi, adauga produse la o comanda pentru o anumita masa, introducand numele produsului si cantitatea, confirma comanda pentru o anumita masa dupa ce a adaugat toate produsele, sau creeaza o nota de plata pentru o anumita masa
- Chef – vede fiecare comanda adaugata si din ce produse sunt alcatuite, iar lista acestuia de comenzi este actualizata la fiecare comanda noua

Obiectivele secundare sunt: respectarea paradigmei programarii orientate pe obiect, serializarea si deserializarea unui fisier .ser, folosirea pattern-ului arhitectural Model-View-Controller, folosirea design pattern-ului Observer, crearea unei documentatii Javadoc si crearea unei arhive .jar pentru portabilitate si testarea mai usoara a programului.

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

La executarea programului, datele despre meniu si despre comenzi vor fi actualizate prin serializare (se citesc la inceput din fisierul restaurant.ser si se rescriu la orice modificare a meniului sau a comenzilor, in acelasi fisier).

Utilizatorul are un meniu pentru identificare, constituit din butoanele Administrator, Waiter si Chef.

La apasarea butonului Administrator, se va deschide o fereastră nouă unde utilizatorul are următoarele opțiuni:

- afisarea produselor meniului – se va deschide o fereastră cu un tabel ce contine numele si pretul pentru fiecare produs din meniu
- crearea unui produs nou – dacă e selectat butonul “Base product” - se va deschide o fereastră nouă, unde utilizatorul trebuie sa introduca un nume si un pret pentru noul produs, iar dacă e selectat butonul “Composite product” – se va deschide o fereastră nouă, unde utilizatorul trebuie sa introduca un nume si numele produselor din compozitie (separate prin “;”) – pretul produsului de tip composite va fi calculat automat, insumand preturile produselor din compozitia acestuia
- editarea unui produs – dacă e selectat butonul “Base product” – se va deschide o fereastră nouă, unde utilizatorul trebuie sa introduca numele produsului, si poate sa introduca la alegere un nume nou sau un pret nou (sau ambele), iar dacă e selectat butonul “Composite product” – se va deschide o fereastră nouă, unde utilizatorul trebuie sa introduca numele produsului, si poate sa introduca la alegere un nume nou sau o lista de produse care sa fie sterse din compozitia produsului (separate prin “;”) sau o lista de produse care sa fie adaugate in compozitia produsului (separate prin “;”) (sau toate 3). Noul pret al produsului va fi calculat si actualizat automat.

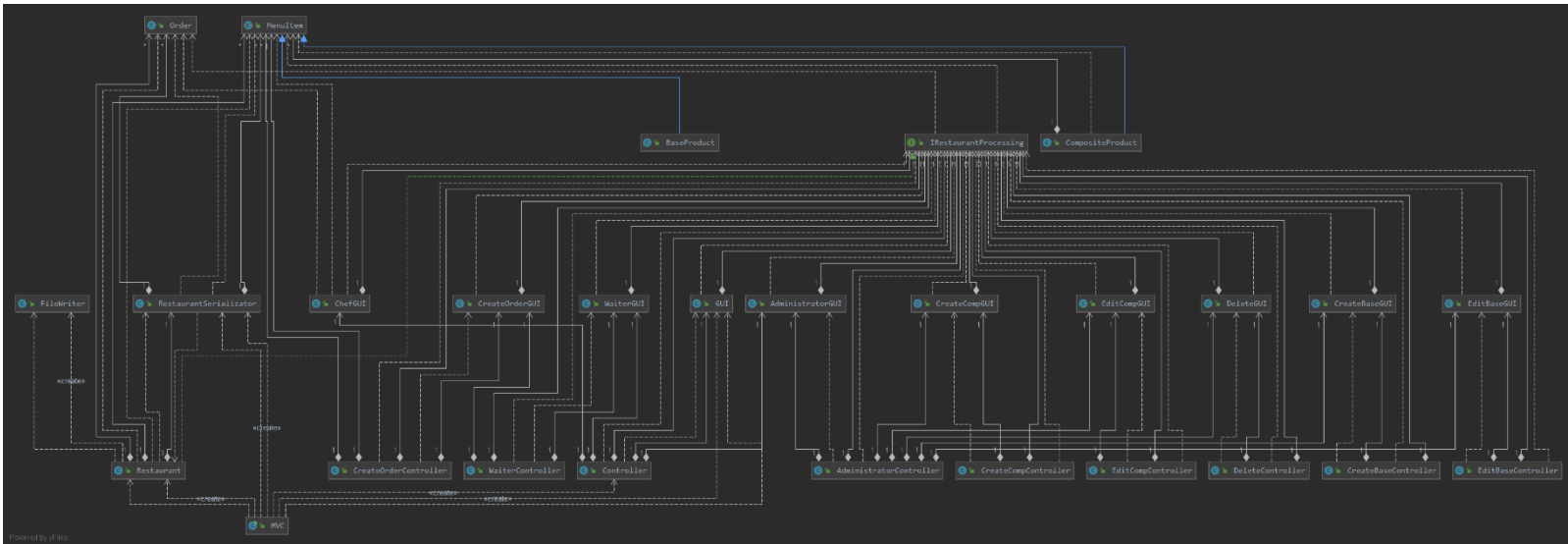
La apasarea butonului Waiter, se va deschide o fereastră nouă unde utilizatorul are următoarele opțiuni:

- afisarea comenzilor – se va deschide o fereastră cu un tabel ce contine id-ul, data si numarul mesei pentru fiecare comanda
- crearea unei comenzi noi – se completeaza inainte numarul mesei – se deschide o fereastră nouă unde trebuie completate numele produsului si cantitatea, după care se apasa un buton pentru a fi adaugate la comanda. Utilizatorul poate continua sa adauge produse la comanda, după care apasa butonul “OK” pentru a confirma comanda, care va apărea pe interfata grafica a Chef-ului
- crearea unei note de plata – se completeaza inainte numarul mesei – se creeaza un fisier .txt, care contine informatii despre comanda, impreuna cu suma totala de plata. Chiar dacă se creeaza mai multe comenzi pentru aceeasi masa, nota de plata pentru acea masa va fi creata pe baza informatiilor din ultima comanda.

La apasarea butonului Chef, se va deschide o fereastră nouă, unde utilizatorul va putea vedea dacă apare o nouă comandă de la un Waiter. La apariția unei comenzi, se va scrie în acea fereastră date despre comandă: ID-ul acesteia și produsele comandate, împreună cu cantitățile acestora.

3. Proiectare

3.1. Diagrama UML



(e încărcată, se poate vedea mai bine cu zoom)

3.2. Proiectare clase

1. Pachetul BusinessLayer

Clasa MenuItem: această clasă este abstractă și definește conceptul de produs din meniu.

Clasa BaseProduct: această clasă definește conceptul de bază produs, acesta fiind caracterizat printr-un nume și printr-un pret.

Clasa CompositeProduct: această clasă definește conceptul de produs compozit, acesta fiind caracterizat printr-un nume, un pret și o listă de produse din care este compus, de tipul MenuItem.

Interfața IRestaurantProcessing: această interfață conține metodele care se ocupă cu prelucrarea datelor din meniu și din comenzi, care vor fi implementate de clasa Restaurant.

Clasa Order: această clasă definește conceptul de comandă, aceasta fiind caracterizată printr-un ID, o dată și numărul unei mese.

Clasa Restaurant: această clasă implementează metodele specifice meniului și comenzilor restaurantului.

2. Pachetul DataLayer

Clasa FileWriter: această clasă se ocupă cu crearea unei note de plată, scriind detaliile acesteia într-un fișier .txt

Clasa RestaurantSerializator: această clasă se ocupă cu preluarea datelor despre comenzi și despre meniu dintr-un fișier .ser și cu scrierea datelor modificate înapoi în fișier

3. Pachetul Main

Clasa MVC: aceasta clasa este clasa main a proiectului, se ocupa cu crearea unui nou serializator si a unui nou obiect de tipul Restaurant, obtinut pe baza informatiilor din serializator. De asemenea, aceasta clasa porneste interfata grafica principala.

4. Pachetul PresentationLayer

Clasa GUI: Aceasta clasa este formata din componentele interfetei grafice principale si asezarea lor in JPanel-uri. Metodele acestei clase sunt cele pentru adaugare de ascultatori pe butoane.

Clasa Controller: Aceasta clasa contine clasele de ascultatori pe butoane.

Clasa AdministratorGUI: Aceasta clasa este formata din componentele interfetei grafice accesate de catre Administrator si asezarea lor in JPanel-uri. Metodele acestei clase sunt cele pentru adaugare de ascultatori pe butoane.

Clasa AdministratorController: Aceasta clasa contine clasele de ascultatori pe butoane.

Clasa WaiterGUI: Aceasta clasa este formata din componentele interfetei grafice accesate de catre Waiter si asezarea lor in JPanel-uri. Metodele acestei clase sunt cele pentru adaugare de ascultatori pe butoane.

Clasa WaiterController: Aceasta clasa contine clasele de ascultatori pe butoane.

Clasa ChefGUI: Aceasta clasa este formata din componentele interfetei grafice accesate de catre Chef si asezarea lor in JPanel-uri. Metoda aceste clase este cea specifica interfetei Observer, si anume update(), prin care se primeste modificarea listei de comenzi de la clasa Restaurant.

Clasa CreateBaseGUI: Aceasta clasa este formata din componentele interfetei grafice accesate de catre Administrator atunci cand se doreste crearea unui base product si asezarea lor in JPanel-uri. Metodele acestei clase sunt cele pentru adaugare de ascultatori pe butoane si pentru a returna valorile din textfield-uri.

Clasa CreateBaseController: Aceasta clasa contine clasele de ascultatori pe butoane.

Clasa CreateCompGUI: Aceasta clasa este formata din componentele interfetei grafice accesate de catre Administrator atunci cand se doreste crearea unui composite product si asezarea lor in JPanel-uri. Metodele acestei clase sunt cele pentru adaugare de ascultatori pe butoane si pentru a returna valorile din textfield-uri.

Clasa CreateCompController: Aceasta clasa contine clasele de ascultatori pe butoane.

Clasa DeleteGUI: Aceasta clasa este formata din componentele interfetei grafice accesate de catre Administrator atunci cand se doreste crearea unui base product si asezarea lor in JPanel-uri. Metodele acestei clase sunt cele pentru adaugare de ascultatori pe butoane si pentru a returna valorile din textfield-uri.

Clasa DeleteController: Aceasta clasa contine clasele de ascultatori pe butoane.

Clasa EditBaseGUI: Aceasta clasa este formata din componentele interfetei grafice accesate de catre Administrator atunci cand se doreste crearea unui base product si asezarea lor in JPanel-uri. Metodele acestei clase sunt cele pentru adaugare de ascultatori pe butoane si pentru a returna valorile din textfield-uri.

Clasa EditBaseController: Aceasta clasa contine clasele de ascultatori pe butoane.

Clasa EditCompGUI: Aceasta clasa este formata din componentele interfetei grafice accesate de catre Administrator atunci cand se doreste crearea unui base product si asezarea lor in JPanel-uri. Metodele acestei clase sunt cele pentru adaugare de ascultatori pe butoane si pentru a returna valorile din textfield-uri.

Clasa EditCompController: Aceasta clasa contine clasele de ascultatori pe butoane.

Clasa CreateOrderGUI: Aceasta clasa este formata din componentele interfetei grafice accesate de catre Administrator atunci cand se doreste crearea unui base product si asezarea lor in JPanel-uri. Metodele acestei clase sunt cele pentru adaugare de ascultatori pe butoane si pentru a returna valorile din textfield-uri.

Clasa CreateOrderController: Aceasta clasa contine clasele de ascultatori pe butoane.

4. Implementare si testare

4.1. Metode

1. Pachetul BusinessLayer

Clasa MenuItem (metodele vor fi implementate in BaseProduct si CompositeProduct)

```
public abstract double computePrice();
public abstract String getName();
public abstract void setName(String newName);
public abstract void setPrice(double newPrice);
```

Clasa BaseProduct

```
public BaseProduct(String nume, double pret)
    -constructorul clasei, primeste doua variabile de instanta – numele si pretul
public String getName()
    -getter pentru numele produsului
public void setName(String newName)
    -setter pentru numele produsului
public void setPrice(double newPrice)
    -setter pentru pretul produsului
public double computePrice()
    -getter pentru pretul produsului
```

Clasa CompositeProduct

```
public CompositeProduct(String nume)
public double computePrice()
    -calculeaza pretul produsului pe baza produselor din lista si il returneaza
public String getName()
    -getter pentru numele produsului
public void setName(String newName)
    -setter pentru numele produsului
public void setPrice(double newPrice)
    -setter pentru pretul produsului
public void addProduct(MenuItem p)
    -adauga un produs la lista de produse
public ArrayList<MenuItem> getProductList()
    -getter pentru lista de produse
```

Interfata IRestaurantProcessing (metodele vor fi implementate in Restaurant)

```
void createMenuItem(MenuItem x);
void deleteMenuItem(MenuItem x);
void editMenuItem(MenuItem a, MenuItem b);
void createOrder(Order x, ArrayList<MenuItem> items, int[] quantities);
void generateBill(int table);
ArrayList<MenuItem> getMenuItems();
HashMap<Order, ArrayList<MenuItem>> getOrders();
int[][] getQuantities();
```

Clasa Order

```
public Order(Date date, int table)
    -constructorul clasei, primeste doua variabile de instanta – data si numarul mesei; in constructor se
    incrementeaza numarul total de comenzi si ID-ul este setat cu numarul total de comenzi in acel moment
public int getOrderID()
    -getter pentru ID-ul comenzii
```

```

public Date getDate()
    -getter pentru data comenzii
public int getTable()
    -getter pentru numarul mesei
public static void setOrderNo(int orderNo)
    -setter pentru numarul total de comenzi
public int hashCode()
    -returneaza hashcode-ul asociat comenzii

```

Clasa Restaurant

```

public Restaurant(RestaurantSerializator ser)
    -constructorul clasei, seteaza campurile menuItems, orders si quantities cu cele primite de la serializator, iar
    referinta spre serializator e salvata intr-o variabila de instanta
void createMenuItem(MenuItem x)
    -adauga MenuItem-ul x la menuItems si se apeleaza functia serialize() a serializatorului
void deleteMenuItem(MenuItem x)
    -sterge MenuItem-ul x din menuItems si se apeleaza functia serialize() a serializatorului
void editMenuItem(MenuItem a, MenuItem b)
    -MenuItem-ul a din menuItems este inlocuit cu MenuItem-ul b si se apeleaza functia serialize() a
    serializatorului
void createOrder(Order x, ArrayList<MenuItem> items, int[] quantities)
    -se adauga perechea x,items in HashMap-ul orders si se pastreaza cantitatile quantities in tabloul quantities[][]
    -se apeleaza functia serialize() a serializatorului
    -este notificat observer-ul
void generateBill(int table)
    -se genereaza String-ul pentru nota de plata pentru masa cu numarul primit ca parametru; acest String este
    trimis apoi functiei createBill() al unui obiect de tipul FileWriter
ArrayList<MenuItem> getMenuItems()
    -getter pentru menuItems
HashMap<Order, ArrayList<MenuItem>> getOrders()
    -getter pentru orders
int[][] getQuantities()
    -getter pentru quantities

```

2. Pachetul DataLayer

Clasa FileWriter

```

public void createBill(String s, int i, int o)
    -se scrie String-ul s intr-un fisier .txt cu numele "Bill table <i> for order <o>"

```

Clasa RestaurantSerializator

```

public RestaurantSerializator (String path)
    -constructorul clasei, primeste numele fisierului din care trebuie sa citeasca, pe care il retine in variabila de
    instanta path si pe care il trimite apoi functiei deserialize()
public void deserialize(String path)
    -se citesc din fisier date si se salveaza corespunzator in variabilele de instanta menuItems, orders si quantities
public void serialize(Restaurant restaurant)
    -se scriu in fisier datele preluate de la restaurantul primit ca parametru
public ArrayList<MenuItem> getMenuItems()
    -getter pentru menuItems
public HashMap<Order, ArrayList<MenuItem>> getOrders()
    -getter pentru orders
public int[][] getQuantities()
    -getter pentru quantities

```

3. Pachetul Main

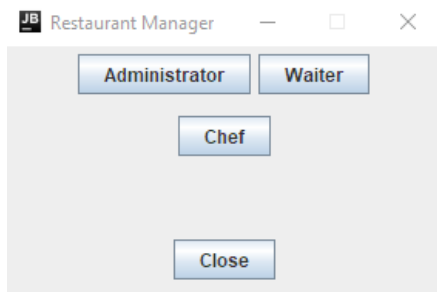
Clasa MVC

```
public static void main(String[] args)
    -metoda main a programului, primeste in args[0] numele fisierului .ser
    -se instantiaza un obiect de tipul serializator
    -se instantiaza un obiect de tipul Restaurant
    -se creeaza interfata grafica principala
    -printr-un controller se leaga obiectul de tip Restaurant la interfata grafica
    -se face vizibila interfata grafica
```

3. Pachetul PresentationLayer

Clasele GUI – plaseaza componente pe JPanel-uri, iar metodele pun ascultatori pe butoane sau pe TextField-uri
Clasele Controller – contin clasele de ascultatori pe butoane

Clasa GUI

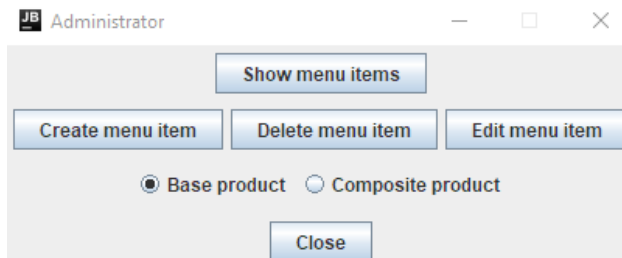


```
public GUI(IRestaurantProcessing model)
void addAdministratorListener(ActionListener a)
void addWaiterListener(ActionListener a)
void addChefListener(ActionListener a)
void addCloseListener(ActionListener a)
```

Clasa Controller

```
public Controller(IRestaurantProcessing model, GUI view)
class AdministratorListener
class WaiterListener
class ChefListener
class CloseListener
```

Clasa AdministratorGUI



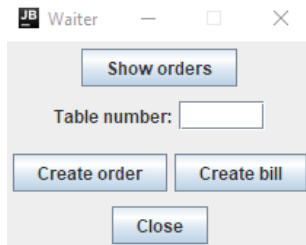
```
AdministratorGUI(IRestaurantProcessing model)
void addCreateListener(ActionListener a)
void addDeleteListener(ActionListener a)
void addEditListener(ActionListener a)
void addBaseListener(ActionListener a)
void addCompositeListener(ActionListener a)
void addCloseListener(ActionListener a)
```


Clasa AdministratorController

AdministratorController(IRestaurantProcessing model, AdministratorGUI view)

```
class ShowListener
class CreateListener
class DeleteListener
class EditListener
class BaseListener
class CompositeListener
class CloseListener
```

Clasa WaiterGUI



WaiterGUI(IRestaurantProcessing model)

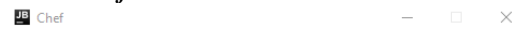
```
void addShowListener(ActionListener a)
void addCreateListener(ActionListener a)
void addBillListener(ActionListener a)
void addCloseListener(ActionListener a)
```

Clasa WaiterController

WaiterController(IRestaurantProcessing model, WaiterGUI view)

```
class ShowListener
class CreateListener
class BillListener
class CloseListener
```

Clasa ChefGUI

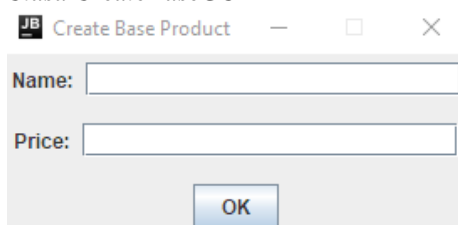


ChefGUI(IRestaurantProcessing model)

```
public void update(Observable observable, Object o)
```

-obiectul o este tip ArrayList<Object>, pe prima pozitie aflandu-se HashMap-ul orders, iar pe a doua array-ul quantities

Clasa CreateBaseGUI



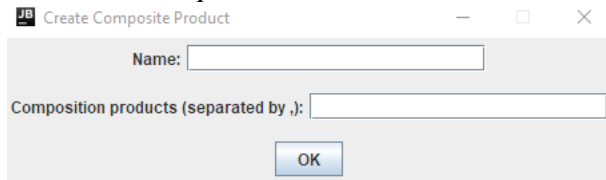
CreateBaseGUI(IRestaurantProcessing model)

```
public void addOkListener(ActionListener a)
public String getNum()
public String getPret()
```

Clasa CreateBaseController

CreateBaseController(IRestaurantProcessing model, CreateBaseGUI view)
class OkListener

Clasa CreateCompGUI

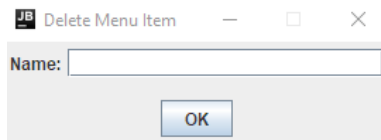


CreateCompGUI(IRestaurantProcessing model)
public void addOkListener(ActionListener a)
public String getNume()
public String getCompozitie()

Clasa CreateCompController

CreateCompController(IRestaurantProcessing model, CreateCompGUI view)
class OkListener

Clasa DeleteGUI

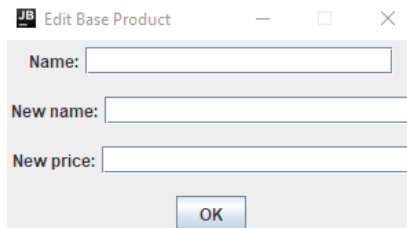


DeleteGUI(IRestaurantProcessing model)
public void addOkListener(ActionListener a)
public String getNume()

Clasa DeleteController

DeleteController(IRestaurantProcessing model, DeleteGUI view)
class OkListener

Clasa EditBaseGUI

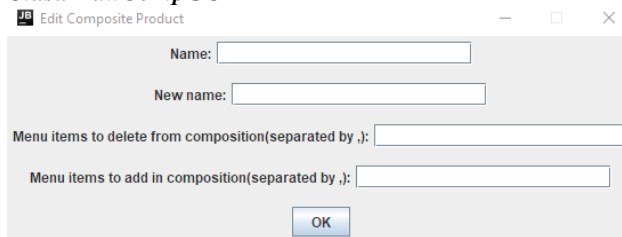


EditBaseGUI(IRestaurantProcessing model)
public void addOkListener(ActionListener a)
public String getNume()
public String getNumeNou()
public String getPretNou()

Clasa EditBaseController

EditBaseController(IRestaurantProcessing model, EditBaseGUI view)
class OkListener

Clasa EditCompGUI

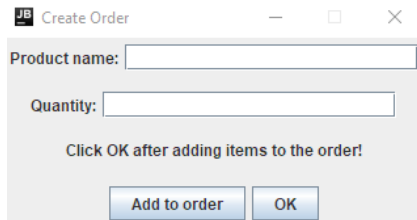


```
EditCompGUI(IRestaurantProcessing model)
public void addOkListener(ActionListener a)
public String getNume()
public String getNumeNou()
public String getNumeAdaugat()
public String getNumeSters()
```

Clasa EditCompController

```
EditCompController(IRestaurantProcessing model, EditCompGUI view)
class OkListener
```

Clasa CreateOrderGUI



```
CreateOrderGUI(IRestaurantProcessing model)
public String getCantitate()
public void reset()
public void addAddListener(ActionListener a)
public void addOkListener(ActionListener a)
```

Clasa CreateOrderController

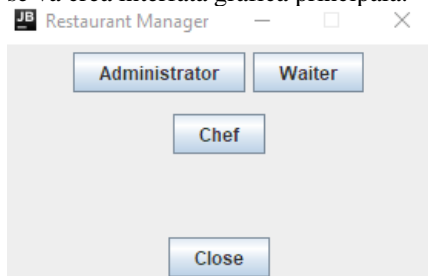
```
CreateOrderController(IRestaurantProcessing model, CreateOrderGUI view, int table)
class AddListener
class OkListener
```

4.2. Testare

In urma executiei comenzii:

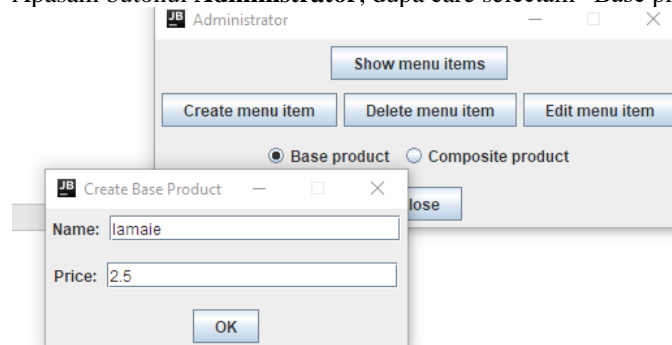
```
java -jar PT2020_30227_Bakk_Cosmin-Robert_Assignment_4.jar restaurant.ser
```

se va crea interfata grafica principala:

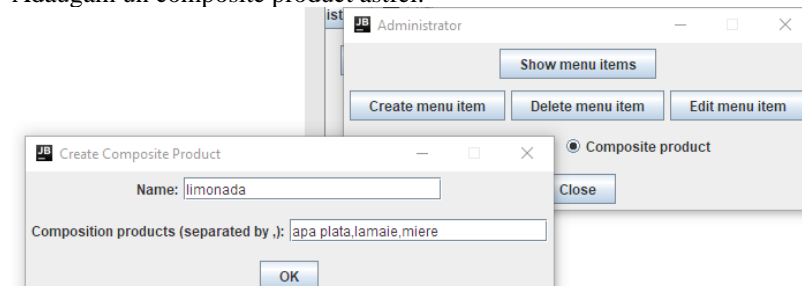


Dupa apasarea butonului “Chef” se va deschide fereastra corespunzatoare utilizatorului de tip Chef, unde vom putea urmari comenzile.

Apasam butonul **Administrator**, dupa care selectam “Base product” si adaugam primul base product:



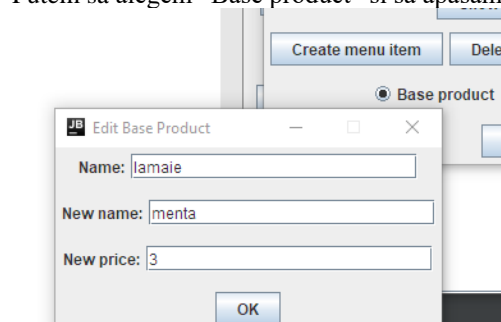
Adaugam astfel mai multe base products: apa plata, miere, cola.
Adaugam un composite product astfel:



Daca apasam butonul “Show menu items”, vom putea vedea produsele din meniu:

NUME	PRET
lamaie	2.5
apa plata	3.5
miere	2.0
cola	5.5
limonada	8.0

Putem sa alegem “Base product” si sa apasam butonul “Edit menu item”:

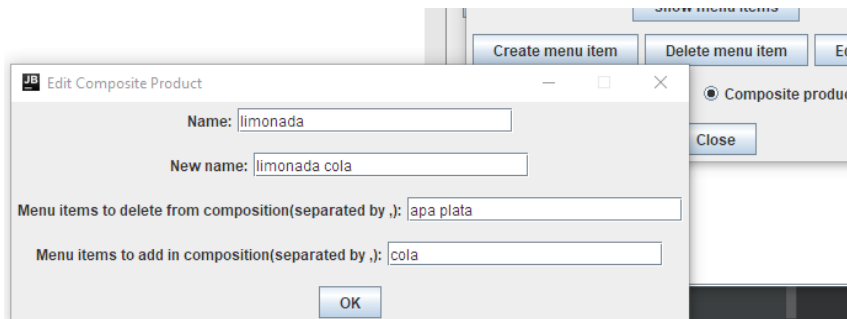


Verificam din nou produsele din meniu:

NUME	PRET
menta	3.0
apa plata	3.5
miere	2.0
cola	5.5
limonada	8.5

Numele produsului “lamaie” a fost modificat in “menta”, iar pretul a fost marit cu 0.5, modificare care a aparut si pentru composite product-ul “limonada”.

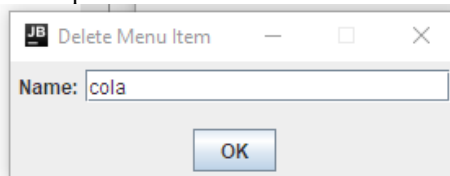
Putem sa alegem “Composite product” si sa apasam butonul “Edit menu item”:



Deschidem din nou meniul pentru a vedea schimbarea:

Lista Menu Items	
NUME	PRET
menta	3.0
apa plata	3.5
miere	2.0
cola	5.5
limonada cola	10.5

Daca apasam “Delete menu item”:



Produsul “cola” va fi sters, impreuna cu toate composite product-urile care il contineau:

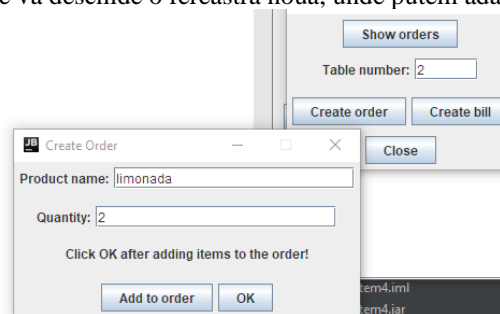
Lista Menu Items	
NUME	PRET
menta	3.0
apa plata	3.5
miere	2.0

Adaugam din nou cola si composite product-ul limonada pentru a testa sistemul de comenzi.

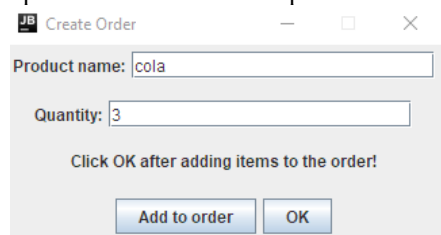
Se apasa butonul **Waiter**

Putem sa scriem numarul mesei 2 si apasam butonul “Create order”.

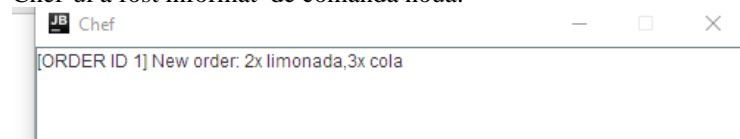
Se va deschide o fereastră nouă, unde putem adauga un anumit numar de produse la comanda:



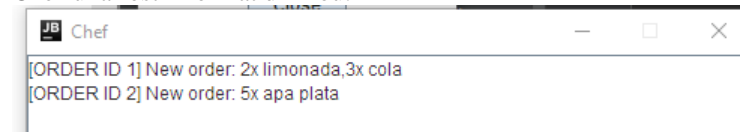
Apasam “Add to order” si putem sa mai adaugam si alte produse:



Dupa ce apasam din nou “Add to order, putem apasa “OK” si fereastra se va inchide.
Chef-ul a fost informat de comanda noua:



Mai facem inca o comanda, pentru masa 3, care contine doar 5x apa plata.
Chef-ul a fost informat din nou:



Daca se apasa pe “Show orders”, se pot vedea comenzile de la mese:

ID	DATE	TABLE
1	Thu May 07 07:57:32 EE...	2
2	Thu May 07 07:59:22 EE...	3

Pentru a face nota de plata, scriem numarul meselor si apasam “Create bill”.
Dupa crearea notelor de plata pentru mesele 2 si 3 vom avea fisierele .txt:

- Bill table 2 for order 1
- Bill table 3 for order 2

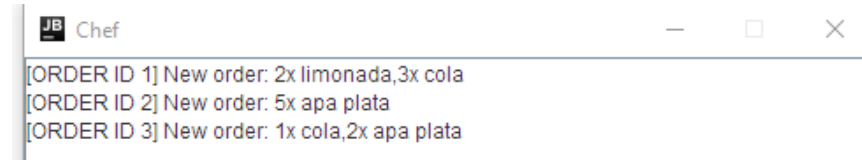
Continutul primului fisier:

```
Bill table 2 for order 1 - Notepad
File Edit Format View Help
Order number 1
Table 2
2x limonada 17.0(unit price: 8.5)
3x cola 16.5(unit price: 5.5)
Order date: Thu May 07 07:57:32 EEST 2020
Bill created on: Thu May 07 08:00:53 EEST 2020
Total price: 33.5
```

Continutul celui de-al doilea fisier:

```
Bill table 3 for order 2 - Notepad
File Edit Format View Help
Order number 2
Table 3
5x apa plata 17.5(unit price: 3.5)
Order date: Thu May 07 07:59:22 EEST 2020
Bill created on: Thu May 07 08:00:55 EEST 2020
Total price: 17.5
```

Putem face de exemplu o comanda noua pentru masa 2:
Chef-ul a fost informat:



Lista de comenzi:

ID	DATE	TABLE
1	Thu May 07 07:57:32 EE...	2
2	Thu May 07 07:59:22 EE...	3
3	Thu May 07 08:04:14 EE...	2

Generam acum nota de plata pentru masa 2:

```
Bill table 2 for order 3 - Notepad
File Edit Format View Help
Order number 3
Table 2
1x cola 5.5(unit price: 5.5)
2x apa plata 7.0(unit price: 3.5)
Order date: Thu May 07 08:04:14 EEST 2020
Bill created on: Thu May 07 08:05:25 EEST 2020
Total price: 12.5
```

In urma inchiderii si redeschiderii programului, datele despre comenzi si despre meniu au fost pastrate.

5. Concluzii

Acest proiect poate fi util pentru un simplu management al unui restaurant. Proiectul m-a ajutat sa invat mai multe despre lucrul cu interfete grafice, despre serializare si despre design pattern-ul Observer. De asemenea, am invatat sa comentez adecvat codul pentru a putea face o documentatie de tip Javadoc.

6. Bibliografie

1. <https://www.baeldung.com/java-observer-pattern>
2. <https://www.baeldung.com/java-serialization>
3. <https://www.geeksforgeeks.org/composite-design-pattern/>
4. http://coned.utcluj.ro/~salomie/PT_Lic/4_Lab/Assignment_4/Assignment_4_Indications.pdf