

**FACULTATEA DE AUTOMATICA SI CALCULATOARE**

**DEPARTAMENTUL CALCULATOARE**

**DISCIPLINA TEHNICI DE PROGRAMARE**

**Documentatie Tema 4**

**Aplicatie pentru managementul unui restaurant**

Pavel Madalina Adriana

Grupa 30224

**CUPRINS**

**1. OBIECTIVUL TEMEI ……………………………………………………………………..3**

**2. ANALIZA PROBLEMEI, MODELARE, SCENARII, CAZURI DE UTILIZARE …...4**

**3. PROIECTARE (DECIZII DE PROIECTARE, DIAGRAME UML, STRUCTURI DE DATE, PROIECTARE CLASE, INTERFETE, RELATII, PACKAGES, ALGORITMI, INTERFATA UTILIZATOR) …………………………..……………………………………5**

**4. IMPLEMENTARE …………………………………………………………………………9**

**5. REZULTATE …………………………………….………………………………………..11**

**6. CONCLUZII .……………………………………………………………………………....13**

**7. BIBLIOGRAFIE …………………………………………………………………………..13**

**1.OBIECTIVUL TEMEI**

Enuntul temei este urmatorul :

Se cere implementarea unui sistem de management al unui restaurant. Sistemul trebuie sa aiba 3 tipuri de utilizatori : administrator, chelner si bucatar. Administratorul poate adauga, sterge, modifica si vizualiza produsele existente in meniu. Chelnerul poate crea o noua comanda pentru o masa, poate adauga produse din meniu, poate vizualiza toate comenzile si poate genera factura pentru o comanda. Bucatarul este notificat de fiecare data cand trebuie sa gateasca mancarea comandata de clienti. Pentru simplificarea aplicatiei, se considera ca sistemul este folosit de un singur administrator, un singur chelner si un singur bucatar si nu este nevoie de un proces de login.

Obiectivul principal al aplicatiei este acela de a gestiona comenzile preluate de un chelner in cadrul unui restaurant, de a gestiona produsele pentru a usura munca administratorului si de a anunta bucatarul la fiecare aparitie a unei noi comenzi.

Obiectivele secundare sunt urmatoarele :

* Dezvoltarea de use-case-uri si scenarii (Capitolul 2)
* Alegerea structurilor de date (Capitolul 3)
* Impartirea pe clase, utilizarea interfetelor, evidentierea relatiilor (Capitolul 3)
* Dezvoltarea algoritmilor utilizati (Capitolul 4)
* Testarea solutiei (Capitolul 5)

**2.ANALIZA PROBLEMEI, MODELARE, SCENARII, CAZURI DE UTILIZARE**

Aplicatia incearca sa imbunatateasca gestionarea datelor cu privire la comenzile preluate de un chelner de la clienti si produsele inregistrate de un administrator. De asemenea, aplicatia trebuie sa notifice bucatarul de fiecare data cand o comanda este inregistrata, pentru a sti ce mancare sa gateasca.

O comanda este caracterizata de un id unic, data cand a fost inregistrata si numarul mesei de la care s-a preluat comanda. Aplicatia permite chelnerului sa inregistreze o noua comanda, sa genereze o factura pentru aceasta si sa afiseze pretul total al comenzii, sa vizualizeze toate comenzile inregistrate si sa stearga o comanda.

Un produs este caracterizat de un id unic, un nume unic, pret si eventual produsele din care este alcatuit (in cazul in care este vorba de un produs compus). Produsele inregistrate de administrator in meniu pot fi de 2 feluri : produse de baza si produse compuse (cele care sunt alcatuite din mai multe produse, fie de baza sau compuse la randul lor). Produsele de baza au nevoie de un pret introdus de administrator, in timp ce pentru produsele compuse pretul se calculeaza automat in functie de produsele din care sunt alcatuite. Aplicatia permite administratorului sa adauge un produs nou in meniu, sa modifice numele unui produs (daca numele nu coincide cu numele unui produs deja existent) si pretul acestuia (doar in cazul unui produs de baza deoarece in cazul unui produs compus aceasta operatie de modificare nu va avea efect), sa stearga un produs in functie de id-ul acestuia si sa vizualizeze toate produsele existente in meniu. Stergerea unui produs de baza din meniu nu implica stergerea acestuia din produsul compus in care a fost adaugat, deoarece restaurantul poate lua decizia de a servi acel produs doar daca acesta este cuprins intr-un produs compus.

Pentru bucatar, aplicatia pune la dispozitie o fereasta care contine un tabel in care vor fi inregistrate toate comenzile.

**3. PROIECTARE (DECIZII DE PROIECTARE, DIAGRAME UML, STRUCTURI DE DATE, PROIECTARE CLASE, INTERFETE, RELATII, PACKAGES, ALGORITMI, INTERFATA UTILIZATOR)**

Pentru a pastra datele odata introduse in aplicatie, aceasta nefiind conectata la o baza de date, am utilizat serializarea, datele fiind stocate intr-un fisier. La lansarea in executie a aplicatiei, aceasta initial realizeaza deserializarea datelor.

Pentru definirea produselor compuse am utilizat design pattern-ul Composite, iar pentru notificarea bucatarului de fiecare data cand chelnerul a adaugat o comanda am utilizat design pattern-ul Observer.

Am definit o structura de tipul Map<Order, Collection<MenuItem>> pentru stocarea informatiei cu privire la comenzi. Cheile utilizate la mapare sunt formate din obiecte de tipul Order, pentru care am suprascris metoda hashCode(), in care am utilizat atributele clasei Order pentru compunerea hashCode-ului.

Am definit o structura de tipul List<MenuItem> in care este salvat meniul restaurantului.

Am implementat clasa Restaurant folosind design pattern-ul Design by Contract, metodele din aceasta clasa fiind insotite de pre si post conditii, invarianti si assert-uri.

Aplicatia este structurata pe layere :

* Data Layer – are rolul de a realiza scrierea in fisier a facturii corespunzatoare unei comenzi si totodata se ocupa si cu serializarea informatiei pentru a salva starea aplicatiei in momentul in care aceasta este inchisa.
* Business Logic Layer – are rolul de a descrie logica aplicatiei, acesta face legatura cu interfata grafica din care se preiau datele si Data Layer; acest layer preia datele, se asigura ca acestea sunt valide si doar in acest caz efectueaza operatii pe ele.
* Presentation Layer – acest layer contine interfata grafica, care este alcatuita din mai multe ferestre, necesitate impusa de logica aplicatiei; prin intermediul interfetei grafice se introduc datele in aplicatie, date ce sunt mai intai validate si mai apoi transmise layer-ului Business Logic.

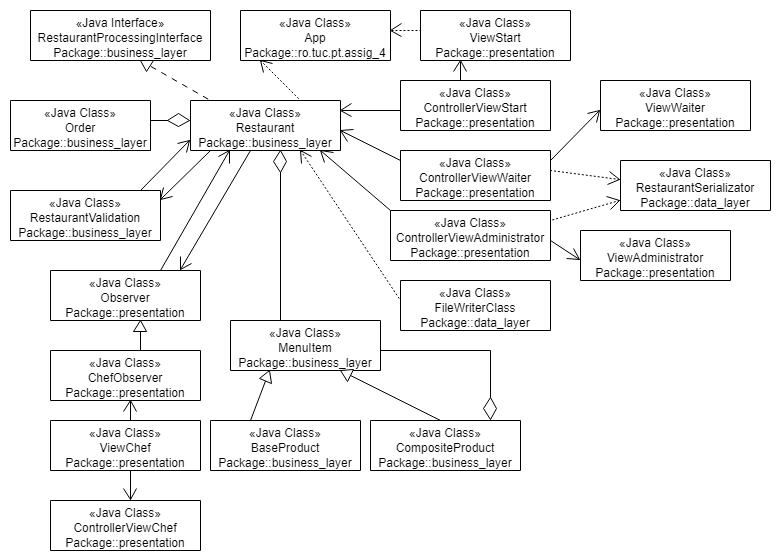
Clasele si interfetele utilizate in realizarea acestei aplicatii sunt :

1. **Clasa BaseProduct** – extinde clasa MenuItem. Clasa contine un constructor si suprascrie metoda toString().
2. **Clasa CompositeProduct** - extinde clasa MenuItem. Clasa contine in plus ca variabila instanta un obiect de tipul List<MenuItem> pentru stocarea produselor din care este alcatuit produsul compus (design pattern-ul Composite). Pe langa constructor, clasa contine un getter si un setter pentru lista de produse si suprascrie metoda computePrice() din superclasa si metoda toString().
3. **Clasa MenuItem** – implementeaza interfata Serializable; contine 3 variabile instanta: id-ul produsului (int), numele produsului (String) si pretul (float). Clasa contine 2 constructori, unul care contine toti parametrii si celelalt contine doar primii 2 parametri. Pe langa constructori, contine metode de get si set pentru variabilele instanta si metoda computePrice().
4. **Clasa Order** – implementeaza interfata Serializable; contine 3 variabile instanta: id-ul comenzii (int), data (un obiect de tipul Date) si numarul mesei (int). Pe langa constructor, clasa contine metode de get si set pentru variabilele instanta si suprascrie metodele toString(), hashCode() si equals(Object).
5. **Clasa Restaurant** – extinde Observable si implementeaza interfetele Serializable si RestaurantProcessingInterface. Clasa contine o constanta serialVersionUID (static final long), necesara in procesele de serializare/deserializare, o lista de observeri (List<Observer>), un obiect de tipul Map<Order, List<MenuItem>> pentru maparea comenzilor, o lista de produse pentru meniu (List<MenuItem>) si un obiect de tipul RestaurantValidation pentru validarea datelor. Pe langa constructor, clasa contine o metoda care notifica toti observerii, cate o metoda pentru generarea liniilor din tabelele pentru comenzi, respectiv meniu, metoda isWellFormed() si metode de get pentru variabilele instanta. In plus, clasa descrie corpul metodelor declarate in interfata RestaurantProcessingInterface.
6. **Interfata RestaurantProcessingInterface** – in aceasta interfata sunt declarate metodele de adaugare, stergere, editare a numelui/pretului unui produs, de creare si stergere a unei comenzi, de calculare a pretului total al unei comenzi si de generare a facturii.
7. **Clasa RestaurantValidation** - implementeaza interfata Serializable; contine o variabila instanta de tipul Restaurant. Pe langa constructor, clasa contine metode care au rolul de a valida datele de intrare, inainte ca aceastea sa fie procesate in Restaurant.
8. **Clasa FileWriterClass** – contine o metoda care are rolul de a crea un fisier si de a scrie informatiile necesare facturii in acesta.
9. **Clasa RestaurantSerializator** – contine o metoda care are rolul de a serializa datele cu privire la un obiect de tipul Restaurant, starea obiectului fiind stocata in fisierul “restaurant.ser”.
10. **Clasa ChefObserver** – extinde clasa Observer; contine o variabila instanta de tipul ViewChef, care reprezinta fereastra pentru bucatar. Pe langa constructor, clasa descrie corpul metodei update() din superclasa, care are rolul de a afisa informatiile cu privire la comenzile introduse de chelner in fereastra bucatarului, pentru ca acesta sa stie ce trebuie sa gateasca.
11. **Clasa ControllerViewAdministrator** – contine 3 variabile instanta : un obiect de tipul ViewAdministrator, care reprezinta fereastra din UI, un obiect de tipul Restaurant si un obiect de tipul RestaurantSerializator, care se ocupa cu serializarea datelor din restaurant. Clasa contine un constructor si clase interne care implementeaza interfata ActionListener, clase care descriu comportamentul fiecarui buton din fereastra pentru administrator : **Clasa ActionListenerAdd, Clasa ActionListenerEdit, Clasa ActionListenerDelete, Clasa ActionListenerViewAll si Clasa ActionListenerBack**.
12. **Clasa ControllerViewChef** – contine 3 variabile instanta : un obiect de tipul ViewChef, care reprezinta fereastra din UI, un obiect de tipul Restaurant si un obiect de tipul ChefObserver, care este notificat de fiecare data cand trebuie sa se afiseze in interfata comenzile introduse de chelner. Clasa contine un constructor.
13. **Clasa ControllerViewStart** – contine 3 variabile instanta : un obiect de tipul ViewStart, care reprezinta fereastra din UI, un obiect de tipul Restaurant si un obiect de tipul RestaurantSerializator, care se ocupa cu serializarea datelor din restaurant. Clasa contine un constructor si o clasa interna care implementeaza interfata ActionListener, clasa ce descrie comportamentul combo box-ului din fereastra de start : **Clasa ActionListenerComboBox**.
14. **Clasa ControllerViewWaiter** - contine 3 variabile instanta : un obiect de tipul ViewWaiter, care reprezinta fereastra din UI, un obiect de tipul Restaurant si un obiect de tipul RestaurantSerializator, care se ocupa cu serializarea datelor din restaurant. Clasa contine un constructor si clase interne care implementeaza interfata ActionListener, clase care descriu comportamentul fiecarui buton din fereastra pentru chelner : **Clasa ActionListenerAdd, Clasa ActionListenerComputeBill, Clasa ActionListenerViewAll si Clasa ActionListenerBack**.
15. **Clasa Observer** – este o clasa abstracta, ce implementeaza interfata Serializable; contine o variabila instanta de tipul Restaurant. Pe langa constructor, clasa contine metoda abstracta update().
16. **Clasa ViewAdministrator** - extinde clasa JFrame; contine ca variabile instanta obiecte de tipul JTextField pentru id-ul, numele si pretul produsului, dar si pentru produsele din care este alcatuit produsul compus (pentru adaugarea unui produs de baza sau produs compus), noul nume sau noul pret (pentru editarea produsului); obiecte de tipul JButton pentru butoanele corespunzatoare operatiilor de adaugare, editare, stergere a unui produs, afisarea tabelului, si butonul back; si 4 obiecte de tipul JTable, DefaultTableModel, JScrollPane si ListSelectionModel pentru afisarea tabelului in interfata grafica, plus o variabila de tipul int in care este stocat id-ul selectat din tabel. Clasa contine un constructor in care este creata fereastra, metode de get pentru variabilele instanta si metode utilizate pentru a adauga cate un ActionListener fiecarui buton. In cazul operatiilor de editare sau stergere a unui produs, este necesara selectarea id-ului corespunzator produsului existent in meniu.
17. **Clasa ViewChef** - extinde clasa JFrame; contine ca variabile instanta 3 obiecte de tipul JTable, DefaultTableModel si JScrollPane pentru afisarea tabelului in interfata grafica. Clasa contine un constructor in care este creata fereastra si metode de get pentru variabilele instanta.
18. **Clasa ViewStart** - extinde clasa JFrame; contine ca variabila instanta un obiect de tipul JComboBox<String>, care permite selectarea urmatoarei ferestre ce se doreste a fi deschisa. Clasa contine un constructor in care este creata fereastra, o metoda de get pentru variabila instanta si o metoda utilizata pentru a adauga un ActionListener combo box-ului.
19. **Clasa ViewWaiter** - extinde clasa JFrame; contine ca variabile instanta obiecte de tipul JTextField pentru id-ul comenzii, numarul mesei, produsele ce compun comanda (pentru adaugarea unei comenzi), valoarea totala a comenzii (se afiseaza cand se apasa butonul pentru generarea facturii); obiecte de tipul JButton pentru butoanele corespunzatoare operatiilor de adaugare, stergere a unei comenzi, afisarea tabelului, generarea facturii si butonul back; si 4 obiecte de tipul JTable, DefaultTableModel, JScrollPane si ListSelectionModel pentru afisarea tabelului in interfata grafica, plus o variabila de tipul int in care este stocat id-ul selectat din tabel. Clasa contine un constructor in care este creata fereastra, metode de get pentru variabilele instanta si metode utilizate pentru a adauga cate un ActionListener fiecarui buton. In cazul in care se doreste generarea facturii sau stergerea unei comenzi, este necesara selectarea id-ului corespunzator comenzii.
20. **Clasa App** – nu contine variabile instanta si nici constructori. Metoda main a acestei clase lanseaza in executie aplicatia si deserializeaza datele pentru restaurant.

Clasele si interfetele prezentate la numerele 1.-7. fac parte din pachetul

business\_layer, cele prezentate la numerele 8.-9. fac parte din pachetul data\_layer, cele prezentate la numerele 10.-19. fac parte din pachetul presentation, iar cea prezentata la numarul 20. face parte din pachetul ro.tuc.pt.assig\_4.

Diagrama UML este urmatoarea :



**4. IMPLEMENTARE**

Metodele implementate in fiecare clasa sunt :

1. Clasa CompositeProduct

* public float computePrice() – calculeaza pretul produsului ca suma preturilor produselor din care este compus

1. Clasa MenuItem

* public float computePrice() – returneaza pretul produsului

1. Clasa Restaurant

* public void addNewMenuItem(MenuItem newMenuItem) – adauga un produs in meniu
* public void deleteMenuItem(int id) – sterge produsul cu id-ul id din meniu
* public void editMenuItemName(int id, String newName) – editeaza numele produsului cu id-ul id
* public void editMenuItemPrice(int id, float newPrice) – editeaza pretul produsului cu id-ul id (doar pentru produse de baza)
* public void createNewOrder(int idOrder, Date date, int tableNumber, List<MenuItem> items) – creeaza o comanda
* public void deleteOrder(int idToDelete) – sterge o comanda
* public float computePriceForOrder(int orderID) – returneaza pretul total al comenzii cu id-ul orderID
* public void generateBill(String fileName, int orderID) – genereaza factura corespunzatoare comenzii cu id-ul orderID in fisierul fileName
* public void notifyAllObs() – notifica toti observerii din lista de observeri, apeland metoda update() pt fiecare
* public Object[][] viewOrders() – genereaza liniile tabelului pentru comenzi
* public Object[][] viewMenu() – genereaza liniile tabelului pentru meniu
* protected boolean isWellFormed() – valideaza datele din restaurant

1. Clasa RestaurantValidation

* public boolean canAddNewMenuItem(MenuItem newItem) – returneaza true daca obiectul newItem poate fi adaugat in meniu, in caz contrar returneaza false
* public boolean canDeleteMenuItem(int idToDelete) - returneaza true daca produsul cu id-ul idToDelete poate fi sters, in caz contrar returneaza false
* public boolean canEditMenuItemName(int id,String newName) - returneaza true daca produsul cu id-ul id se gaseste in meniu si newName nu se mai regaseste in produsele din meniu, in caz contrar returneaza false
* public boolean canEditMenuItemPrice(int id, float newPrice) – returneaza true daca produsul cu id-ul id se gaseste in meniu si newPrice > 0, in caz contrar returneaza false
* public boolean canCreateNewOrder(int idOrder, Date date, int tableNumber, List<MenuItem> items) – returneaza true daca parametrii pot caracteriza o comanda, in caz contrar returneaza false
* public boolean canDeleteOrder(int idToDelete) – returneaza true daca comanda cu id-ul idToDelete poate fi stearsa, in caz contrar returneaza false
* public boolean canComputePriceForOrder(int orderID) – returneaza true daca comanda cu id-ul orderID se gaseste in mapare, in caz contrar returneaza false
* public boolean canGenerateBill(String fileName, int orderID) – returneaza true daca numele fisierului nu e null si daca comanda cu id-ul orderID se gaseste in mapare, in caz contrar returneaza false

1. Clasa FileWriterClass

* public void write(String fileName, String data) – scrie in fisierul cu numele fileName string-ul data

1. Clasa RestaurantSerializator

* public void serialize(Restaurant restaurant) – realizeaza serializarea obiectului de tipul Restaurant primit ca parametru

1. Clasa ChefObserver

* public void update() – metoda abstracta din clasa Observer; realizeaza afisarea in tabelul bucatarului

1. Clasa ActionListenerAdd

* void actionPerformed(ActionEvent e) – in functie de controller-ul in care se afla clasa interna, metoda preia datele din interfata si apeleaza metoda de adaugare

1. Clasa ActionListenerEdit

* void actionPerformed(ActionEvent e) – metoda preia datele din interfata si apeleaza metodele de editare

1. Clasa ActionListenerDelete

* void actionPerformed(ActionEvent e) – in functie de controller-ul in care se afla clasa interna, metoda preia datele din interfata si apeleaza metoda de stergere

1. Clasa ActionListenerViewAll

* void actionPerformed(ActionEvent e) – in functie de controller-ul in care se afla clasa interna, metoda preia datele din interfata si apeleaza metoda de afisare

1. Clasa ActionListenerBack

* void actionPerformed(ActionEvent e) – realizeaza revenirea in fereastra principala

1. Clasa ActionListenerComboBox

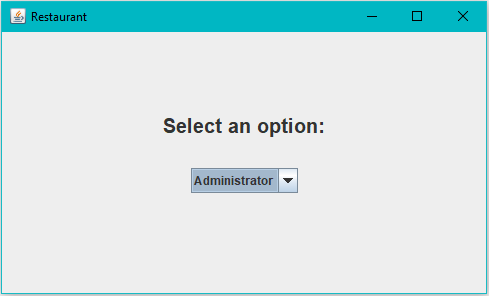
* void actionPerformed(ActionEvent e) – realizeaza trecerea in fereastra urmatoare in functie de String-ul ales din combo box

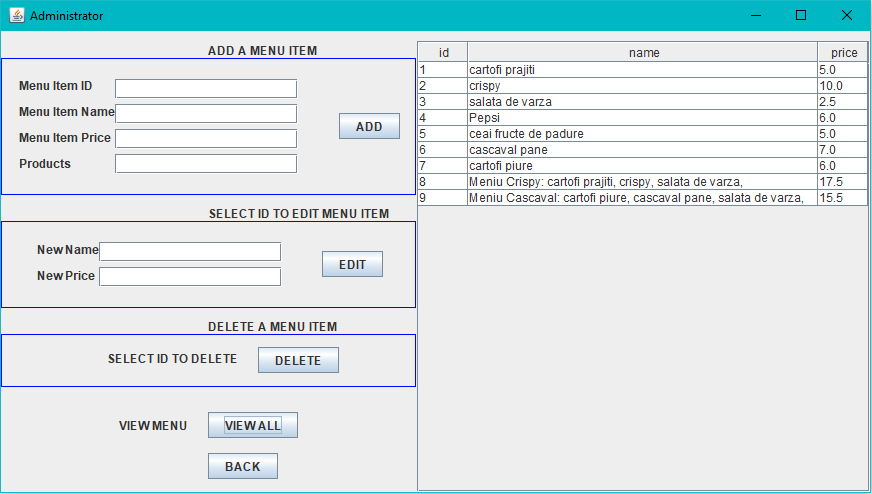
1. Clasa ActionListenerComputeBill

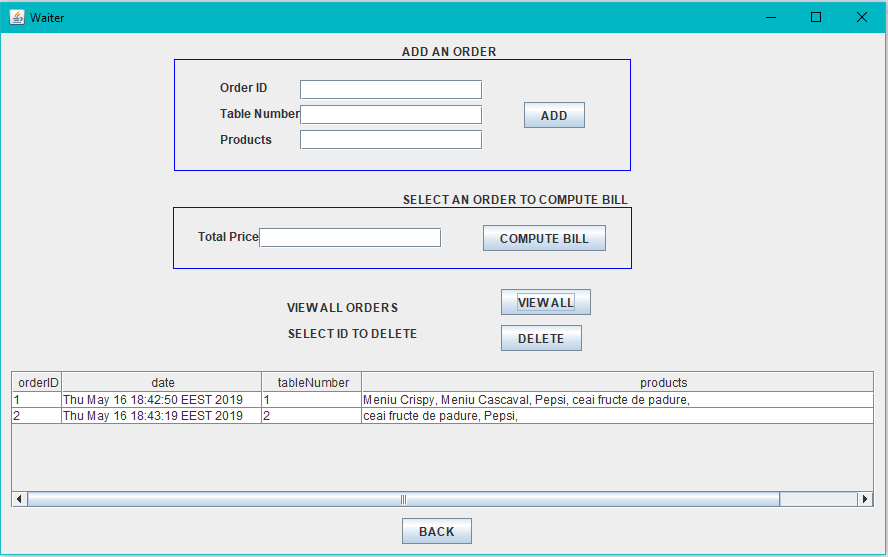
* void actionPerformed(ActionEvent e) – apeleaza metodele de generare a facturii si de calculare a pretului total al comenzii

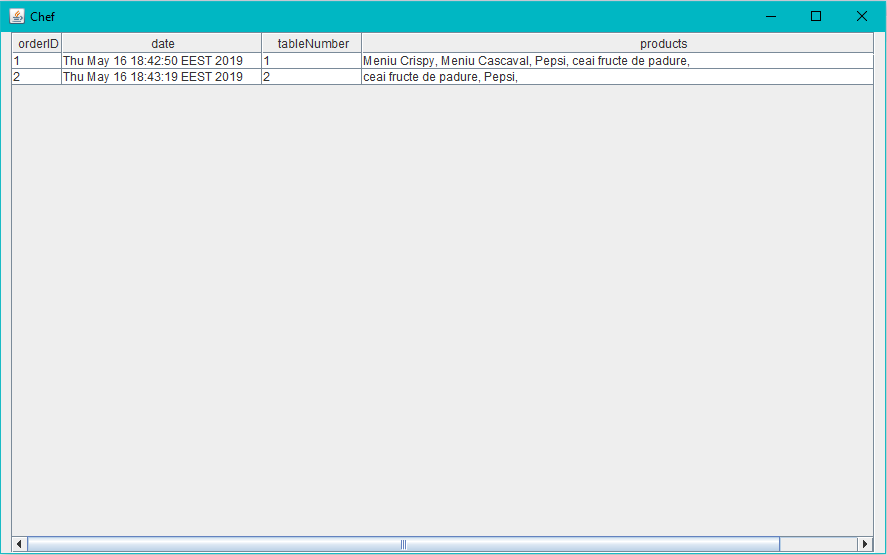
**5. REZULTATE**

Interfata utilizator este urmatoarea :









**6. CONCLUZII**

In concluzie, aceasta tema m-a ajutat sa imi imbunatatesc cunostiintele legate de design pattern-uri, serializare si utilizarea maparilor.

Printre posibilitatile de dezvoltare ulterioara se numara : extinderea aplicatiei astfel incat sa suporte mai mult de 3 utilizatori si adaugarea unei ferestre de login.

**7. BIBLIOGRAFIE**

* cursuri PT
* <https://www.tutorialspoint.com/design_pattern/composite_pattern.htm?fbclid=IwAR1fb4qIxIPACUQW0lLLvLlBXltpYzHopNHurnQuqXjJhQjKJhgpELxfPvc>
* <https://www.tutorialspoint.com/design_pattern/observer_pattern.htm?fbclid=IwAR2Ih0_lPOv4oe3qWwvftuEZzy9CYkDVChbDvI7fX2E7m4wcazTDm8JfERY>
* <https://www.geeksforgeeks.org/serialization-in-java/?fbclid=IwAR2I3ZI2Yrq7I6jXhu6nLSDuuM1qORiuUlkPsd7JSD3S9s6EtXetGN3m-Eg>
* <https://www.javaworld.com/article/2074956/icontract--design-by-contract-in-java.html?fbclid=IwAR2wyf2YF7QUmYKV3ZffHswHjtss4HWHe7IsSpvyCl6UyQibrtHmUQmQh8o>
* <https://www.baeldung.com/java-write-to-file>