***DOCUMENTAȚIE***

TEMA 4 TEHNICI DE PROGRAMARE

***Sistem de gestiune a unui restaurant***

*Pogăcean Rahela- Alexandra,*

Calculatoare și Tehnologia Informației

Anul II, seria B, grupa 30226

**Cuprins**

1. **Obiective**
   1. **Obiectiv principal**
   2. **Obiective secundare**
2. **Analiza problemei**
   1. **Descriere**
   2. **Scenarii și cazuri de utilizare**
3. **Proiectare**
   1. **Decizii de proiectare**
   2. **Diagrame UML**
   3. **Structuri de date**
   4. **Proiectare clase și algoritmi**
4. **Implementare**
   1. **Metode**
   2. **Interfața utilizator**
5. **Rezultate și concluzii**
6. **Bibliografie**
7. **Obiective**
   1. **Obiectivul principal**

Obiectivul fundamental al acestei lucrări de laborator este proiectarea și implementarea unui sistem de gestionare a unui restaurant. Acest lucru se revendică a fi realizat prin dezvoltarea și aplicarea mai multor concepte esențiale. În acest fel, proiectarea se face prin tehnici de programare Design by Contract, polimorfism, Design Patterns, implementări JCF HashMap și HashSet, precum și prin serializare.

* 1. **Obiective secundare**

Un prim obiectiv secundar este constituit de respectarea paradigmelor programării orientate pe obiecte, structurarea codului in clase, ceea ce face ca acesta să fie mai lizibil și mai facil de urmărit. Realizarea diagramelor de clase și a use case-urilor reprezintă un alt obiectiv secundar al acestui proiect, precum și dezvoltarea ulterioară, propunându-se o serie de îmbunătățiri ale aplicației pentru o versiune mai complexă.

1. **Analiza problemei**

**2.1 Descriere**

Tema acestui proiect propune implementarea unui sistem de simulare a gestionării unui restaurant. Aplicația trebuie să aibă trei tipuri de utilizatori (administrator, chelner și bucătar). Administratorul poate adăuga, șterge sau modifica produsele existente în meniu. Chelnerul are abilitatea de a crea o nouă comandă pentru o masă, de a adăuga elemente din meniu, de a calcula factura pentru o comandă. Bucătarul este notificat de fiecare dată când trebuie să gătească mâncarea comandată prin intermediul chelnerului.

Implementarea claselor se realizează în funcție de o diagramă dată. Pentru simplificarea aplicației, sistemul se consideră a fi folosit de un singur administrator, un chelner și un bucătar. De asemenea, nu este nevoie de un proces de logare.

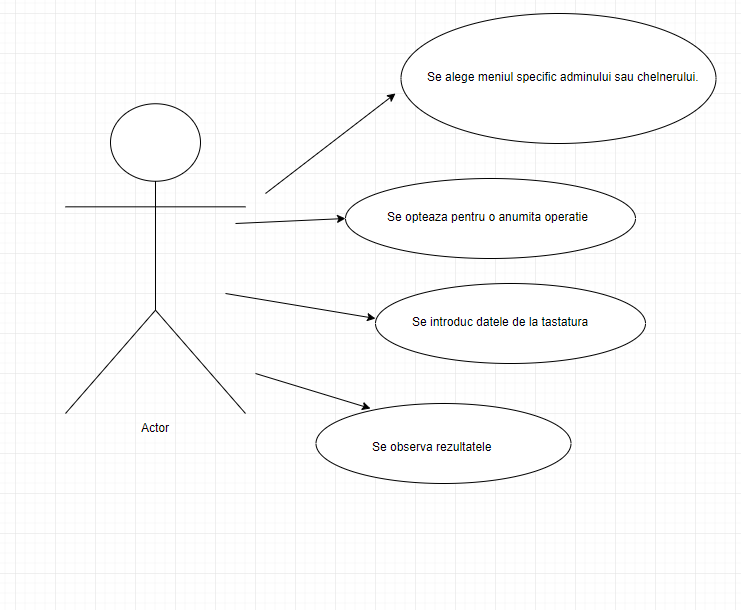
În cele ce urmează, voi prezenta mai detaliat cerințele care se cer a fi satisfăcute. În primul rând, se definește interfața RestaurantProcessing, ce conține principalele operații care pot fi executate de chelner sau de administrator, după cum urmează. Adminul are capacitatea de a crea noi elemente de meniu, de a le șterge sau de a le edita. Chelnerul poate crea o nouă comandă, calcula prețul unei comenzi și genera o factură în formatul .txt.

În al doilea rând, se definește și se implementează clasele din diagrama oferită drept reper, folosindu-se două modele de design. Composite Design Pattern se folosește pentru a defini clasele MenuItem, BaseProduct și CompositeProduct. Observer Design Pattern este utilizat pentru a notifica ospătarul de fiecare dată când trebuie procesată o nouă comandă ce conține un produs te tip composite.

În al treilea rând, se implementează clasa Restaurant folosindu-se o colecție JCF predefinită, care utilizează o structură de date hashtable. Cheia din hashtable va fi generată pe baza clasei Order, care poate avea asociate mai multe MenuItems. Se folosește JTable pentru a afișa informațiile relaționate cu restaurantul. Voi prezenta acum pașii ce necesită a fi urmați în vederea realizării aspectelor menționate. Se definește o structură de tipul Map<Order>, Collection<MenuItem> pentru a stoca în clasa Restaurant datele corelate cu comanda. Cheia din Map va fi formată din obiecte de tipul Order, pentru care metoda hashCode() va fi suprascrisă pentru a calcula valoarea hash în Map din atributele comenzii (OrderID, date etc). Se definește o structură de tipul Collection<MenuItem> în care se va salva meniul restaurantului. Se cere a se alege cel mai potrivit tip de colecție pentru propria implementare. Se definește o metodă de acest tip pentru clasa Restaurant și se implementează clasa folosind metoda Design by Contract (implicând pre, post condiții, aserțiuni).

În ultimul rând, elementele din meniu pentru popularea obiectului Restaurant vor fi salvate folosind serializarea.

* 1. **Scenarii și cazuri de utilizare**



1. **Proiectare**
   1. **. Decizii de proiectare**

Am optat pentru divizarea acestei teme în mai multe task-uri. Pentru început, am decis să proiectez interfața grafică. Am testat funcționalitatea acesteia prin introducerea datelor de intrare (câmpurilor specificate) și apoi am afișat acele valori în spațiile destinate răspunsului. Proiectarea interfeței grafice, asigurându-mă de corectitudinea datelor introduse și de afișarea corectă a lor a fost urmată de implementarea sistemului de gestiune al unui restaurant cu ajutorul tehnicilor de programare Design by Contract, a Design Patterns- urilor, a Serializarii. Pe aceste concepte s-a fundamentat întreaga temă, în rândurile următoare fiind prezentate. Design by Contract implică anumite condiții ale creatorilor software, în sensul în care specificațiile interfeței ar trebui să fie formale, precise, exacte, verificabile, inteligibile, extinzând definiția comună a tipurilor de date abstracte , implementând precondiții, postcondiții și invarianți. Specificațiile se referă la contracte, în concordanță cu metafora conceptuală cu condiții și obligații ale contractelor business. Un alt concept implementat este polimorfismul, care se referă la abilitatea de a lua mai multe forme, la furnizarea unei singure interfețe entităților de tipuri diferite sau folosirea unui singur simbol pentru o reprezentare multiplă. Composite Pattern este un model de proiectare partițională și descrie un grup de obiecte care este tratat în același mod ca și o unică instanță a aceluiași tip de obiect. Intenția unui composite este de a compune obiecte în structuri arborescente și de a furniza fiecărui nod al structurii să realizeze un anumit task. O altă structură importantă este HashMapul, care face parte din colecțiile din Java și stochează datele în perechi (Key, Value). Pentru a avea posibilitatea de a accesa o anumită cheie, este nevoie neapărat să îi cunoaștem cheia corespunzătoare. HashMap este cunoscut ca și HashMap deoarece acesta folosește o tehnică numită Hashing. Dispersia/Hashing este o tehnică de conversie a unui String de dimensiune mare într-un String de o dimensiune mai redusă, care reprezintă actualmente același șir. O valoare mai mică ar facilita indexarea și ar îmbunătăți rapiditatea căutărilor. HashSet utilizează, de asemenea, HashMap în mod intern.

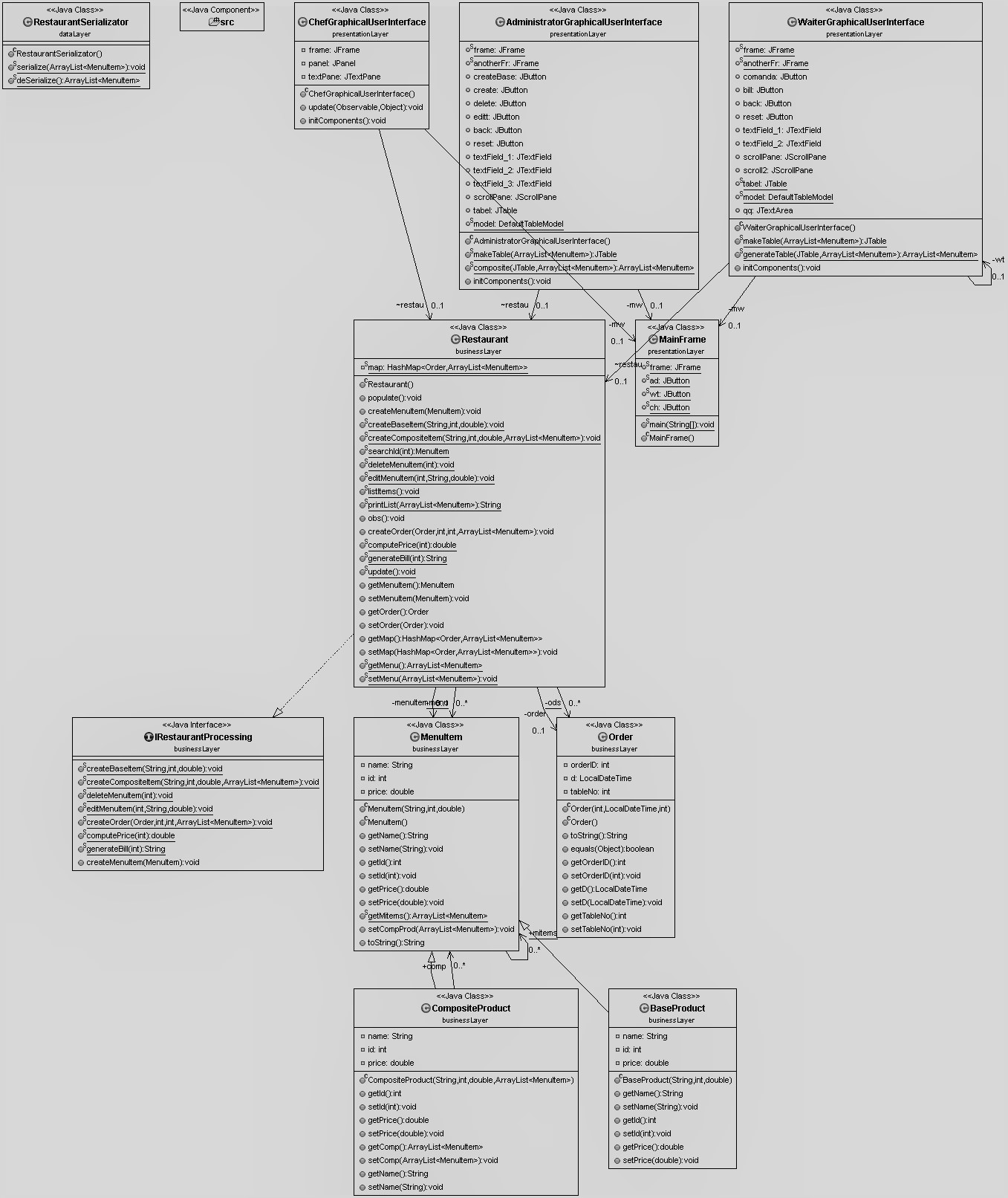
Un alt mecanism implementat este serializarea, un mecanism de conversie a stării unui obiect într-un byte stream. Deserializarea este procesul reversibil unde bytre stream-ul este folosit pentru a recrea obiectul actual în memorie. Acest mecanism este folosit pentru a persista obiectul. Byte stream-ul creat este o platformă independentă. În acest fel, obiectul serializat pe o platformă poate fi deserializat pe o platformă diferită. Pentru a face un obiect serializabil implementăm interfața java.io.Serializable. Clasa ObjectOutputStream conține metoda writeObject() pentru a serializa un obiect. ObjectInputStream conține readObject() pentru deserializarea unui obiect.

**3.2. Diagrame UML**

Unified Modeling Language(UML) este un limbaj standard pentru descrierea de modele si specificatii software. UML oferă o largă gamă de diagrame pentru modelarea diferitelor situații în cadrul unui proiect de dezvoltare software.

Aceste diagrame sunt de mai multe tipuri: de activitate, de componente, de clasă, package, de secvență, use-case și deployment.

În continuare, va fi prezentată diagrama de clase și de pachete pentru sistemul de gestiune a restaurantului implementat.



* 1. **Proiectare clase și algoritmi**

Proiectul este structurat în trei pachete (businessLayer, dataLayer și presentationLayer) , fiecare conținând mai multe clase grupate în funcție de funcționalitatea lor. Astfel, businessLayer cuprinde clasele BaseProduct, CompositeProduct, IrestaurantProcessing, MenuItem, Order, Restaurant. BaseProduct moștenește clasa MenuItem și se referă la produsele de bază ce pot exista în meniul unui restaurant (produse primare, de exemplu apa, pâine). CompositeProduct extinde și ea MenuItem, conținând produsele compuse din meniu, care se formează din produsele de bază (de exemplu, supă de teiței, cafea cu lapte). Aceste două clase demonstrează complexitatea aplicației, existând o gamă variată de produse și fiind permisă selectarea oricarei variante dorite. MenuItem descrie un obiect din meniu, în sensul în care acesta poate fi primar sau derivat. MenuItem implementează Serializable. Order este clasa ce descrie o comandă efectuată, conținând detalii precum numărul acesteia, numărul mesei la care trebuie onorată, precum și momentul curent al procesării comenzii. IRestaurantProcessing este o interfață în care sunt declarate (fără implementare) metodele principale din Restaurant, metode care contribuie la funcționarea aplicației.

1. **Implementare**
   1. **Metode**

O clasă de o importanță aparte, care ilustrează cel mai în detaliu aplicația de gestiune este Restaurant, clasă ce extinde Observable și implementează IRestaurantProcessing. Aceasta conține un obiect de tipul MenuItem, unul de tipul Order, un HashMap<Order, ArrayList<MenuItem> , un ArrayList<MenuItem> și un ArrayList<Order>. HasMap-ul facilitează corelarea produselor comandate cu numărul comenzii. În constructorul din Restaurant se inițializează aceste structuri și se face deserializarea (menu = RestaurantSerializator.deSerialize() ). În cele ce urmeză, sunt implementate metodele necesare funcționării normale a aplicației. Metoda statică createBaseItem permite adăugarea unui produs basic nou în meniu de către administrator. Este implementată într-un mod rapid și simplist, se introduc atributele necesare de către utilizator în interfață (nume, id și preț), se creează un obiect de tipul MenuItem care se adaugă în lista de elemente din meniu. createCompositeProduct se realizează în mod similar cu metoda precedentă, cu precizarea ca mai primește încă un parametru, și anume, un ArrayList<MenuItem> ce conține lista de produse BaseProduct din care se va alcătui produsul composite. Pentru o implementare mai eficientă a ștergerii unui element din meniu, am ales să creez metoda searchId care returnează obiectul al cărui id se cere și se elimină efectiv produsul identificat din lista de itemi. Pentru editMenuItem se folosește din nou searchId, localizându-se obiectul ce se dorește a fi actualizat, urmând ca acesta să fie modificat prin setarea numelui și a prețului. Urmează metode de afișare a elementelor unei liste, ce vor permite vizualizarea produselor basic din care este format un produs compus.

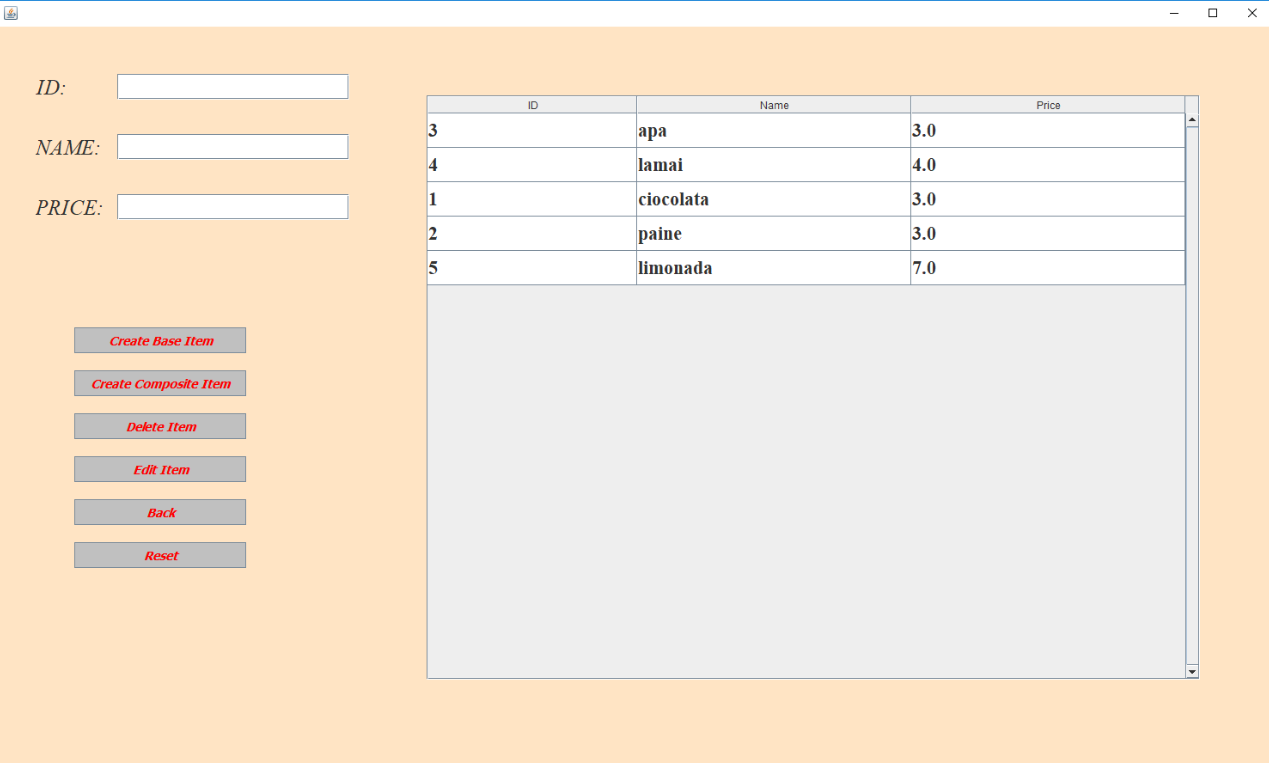
De asemenea, am proiectat metodele specifice chelnerului. createOrder permite efectuarea unei comenzi cunoscând id-ul order-ului și numărul mesei. În acest mod, se creează o comandă cu id-ul dat, care se adaugă în lista de order-e și în HashMap alături de lista de produse comandate. computePrice are rolul de a returna prețul total al comenzii, parcurgându-se lista de comenzi pentru identificarea orderului cu id-ul dorit, precum și a HashMapului, cu scopul de a parcurge lista de MenuItem specifică respectivei comenzi pentru a putea însuma prețul fiecărui item comandat din listă. Metoda generateBill returnează un String cu detaliile facturii (numărul comenzii, data și ora la care a fost procesată, precum și prețul total ce trebuie achitat de client. Metoda se folosește de computePrice pentru a putea genera suma totală ce trebuie plătită.

În dataLayer există o singură clasă, și anume, RestaurantSerializator, unde sunt simplementate metodele de serializare și deserializare. Acestea permit păstrarea, salvarea datelor introduse sau modificate, chiar după ieșirea din program.

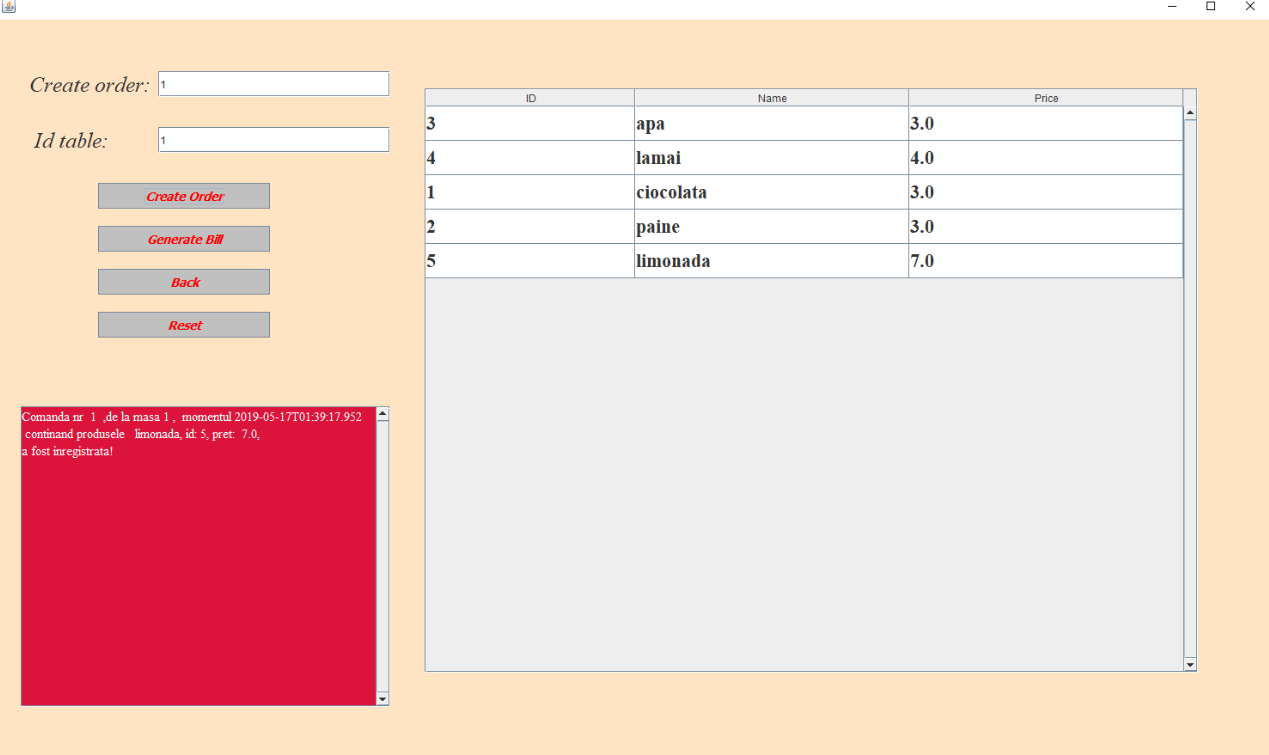
* 1. **Interfața utilizator**



Pachetul presentationLayer este destinat interfeței grafice a aplicației și cuprinde 4 clase. MainFrame este pagina principală a programului, care oferă o perspectivă inedită asupra restaurantului și permite accesul spre taskurile administratorului, respectiv ale chelnerului. Astfel, prin apăsarea butonului Administrator, se va genera deschiderea celei de a doua ferestre a aplicației. Acest Frame este prevăzut cu trei zone de textField, pentru introducerea de către utilizator a numelui produsului, a id-ului și a prețului. În funcție de operația e se dorește a fi efectuată, se alege unul din butoane (Create Base Item, Create Composite Item, Delete Item, Edit Item), operațiile fiind prezentate anterior. De asemenea, există un buton Back ce permite revenirea la pagină principală a aplicației, precum și un buton de Reset, care face curățenie în zonele de textField, facilitând introducerea unor date noi. Un aspect important al acestui frame îl constituie tabelul (Jtable), care permite vizualizarea tuturor produselor din meniu, precum și a schimbărilor care le afectează, dar care și oferă user-ului posibilitatea de a selecta produsele dorite în vederea formării unui item composite.



Din meniul principal se poate accesa și pagina destinată waiter-ului, care are rolul de a efectua o comandă și de a o onora. Astfel, frame-ul este prevăzut cu două secțiuni de JTextField, unde se introduc id-ul comenzii și numărul mesei. Comanda se efctuează prin apăsarea butonului Create Order, afișându-se în zona de JtextArea detaliile comenzii. Generate Bill are rolul de a obține detaliile facturii, acestea fiind, la rândul lor, vizualizate tot cu ajutorul JtextArea- ului. Butoanele de Back si Reset au acelasi scop ca și la frame-ul dedicat adminului, iar tabelul permite și aici selectarea produselor dorite în vederea comenzii.



1. **Rezultate și concluzii**

Proiectul a fost elaborat într-o manieră simplistă și minimalistă, având numeroase posibilități de dezvoltare ulterioară. Astfel, operațiile de editare și ștergere s-ar putea implementa prin selectarea directă a produselor dorite din tabel, fără introducerea câmpurilor pentru identificarea acestora. Aceasta ar fi o metodă mult mai eficientă și mai facilă, din perspectiva interacțiunii user-ului cu interfața aplicației.

1. **Bibliografie**

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

<https://docs.oracle.com/javase/8/docs/technotes/guides/language/assert.html>

<https://www.geeksforgeeks.org/composite-design-pattern/>