Manager de restaurant

Facultatea de Automatică și Calculatoare

Student: *Petrea Irina-Alexandra*

An 2, grupa 30226

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

Cuprins:

1. **Obiectivul temei**
   1. **Obiectivul principal**
   2. **Obiective secundare**
2. **Analiza problemei**
3. **Proiectarea soluției**
4. **Implementare**
5. **Rezultate**
6. **Concluzii**
7. **Bibliografie**

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

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

1. Obiectivul temei

**Obiectivul principal** al temei este cel de a realiza o aplicație desktop cu interfață grafică pentru gestionarea unui restaurant. Această aplicație permite efectuarea de operații de către un administrator de restaurant, precum și de un chelner. Aplicația este proiectată conform paradigmelor programării orientate pe obiecte folosind limbajul de programare Java. Pentru organizarea pe pachete și clase s-a folosit modelul arhitectural **Layers**. Punem în evidență mai multe obiective secundare, părți constituente ale obiectivului principal sus-menționat:

* **Divizarea responsabilităților pe clase și metode**, ca bună practică în proiectarea aplicației, prin structurarea pe pachete de clase conform modelului arhitectural **Layers**, pentru a facilita înțelegerea programului și dezvoltarea ulterioară – va fi detaliată în capitolele 3 și 4
* **Realizarea diagramelor UML** (diagrame de clase, de pachete, etc), pentru a facilita proiectarea ulterioară și transpunerea funcționalităților în cod
* **Realizarea unei interfețe grafice ușor de utilizat**
* **Lucrul cu fișiere și realizarea unui executabil .jar**
* **Generarea de fișiere txt**
* **Implementarea modelului arhitectural Composite pentru unele clase**
* **Implementarea modelului arhitectural Observable pentru unele clase**
* **Definirea de tag-uri javadoc particulare pentru respectarea principiului de implementare Design by Contract si generarea de javadoc in acest sens**
* **Folosirea colecției HashMap pentru a eficientiza operațiile de căutare care se efectuează des**
* **Lucrul cu serializarea/deserializarea obiectelor în Java**

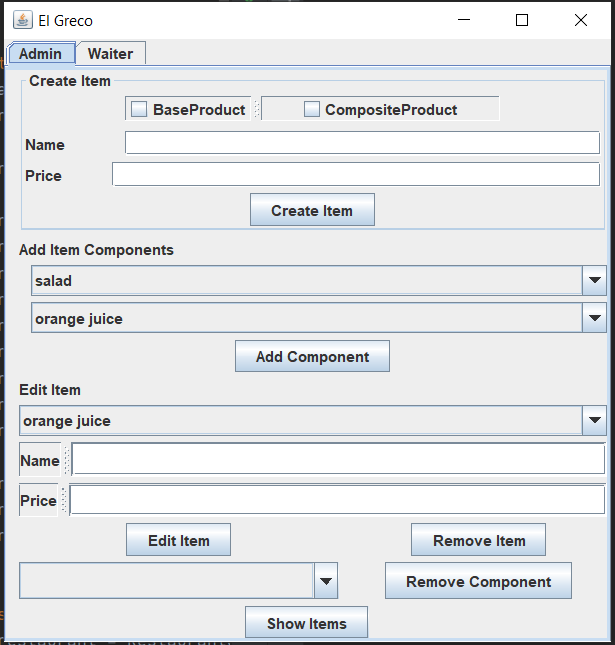
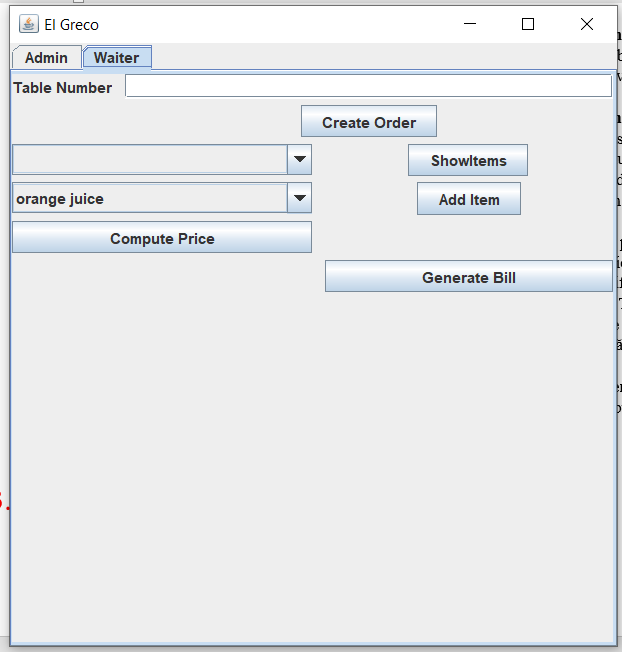
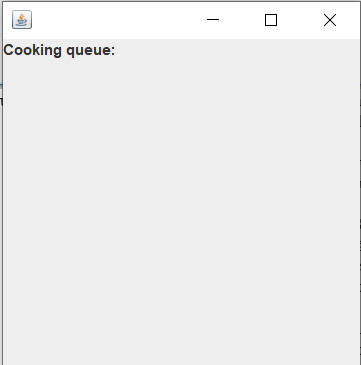
1. Analiza problemei

Se vor prezenta funcționalitățile oferite de aplicație, precum și detalierea cazurilor de utilizare a acesteia.

**Aplicația** facilitează gestiunea unui restaurant din 3 puncte de vedere:

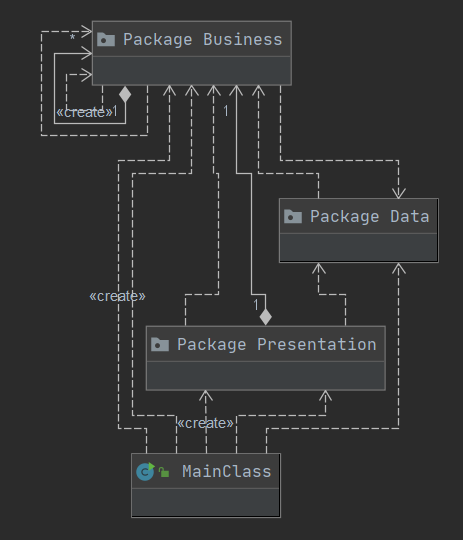
* **Administrator** – poate adăuga, șterge și modifica elemente din meniul restaurantului prin intermediul zonei aferente din interfața grafică. De asemenea, poate să adauge produse compozite și să adauge la acestea drept componente alte elemente din meniu. Poate să modifice prețul produselor de bază
* **Chelner** – poate adăuga noi comenzi și produse asociate comenzilor prin intermediul zonei aferente a interfeței grafice. Comenzile sunt identificate printr-un identificator unic. Poate sa adauge produse la o anumita comanda din lista de produse a restaurantului. Poate sa afișeze prețul comenzii selectate, produsele din comandă și poate genera o factură în format txt.
* **Chef/bucătar** – dispune de o interfață grafică prin care poate vedea ce produse compuse au fost adăugate în comenzi, pentru a le putea pregăti.

Interacțiunea cu utilizatorul se realizează astfel: această aplicație prezintă interfață grafică, precum și posibilitatea de a lucra cu un argument dat la execuție, ce reprezintă un fișier binar care conține pe forma serializată a obiectului instanță a clasei Restaurant. Se vor prezenta pe rând aceste cazuri de utilizare:

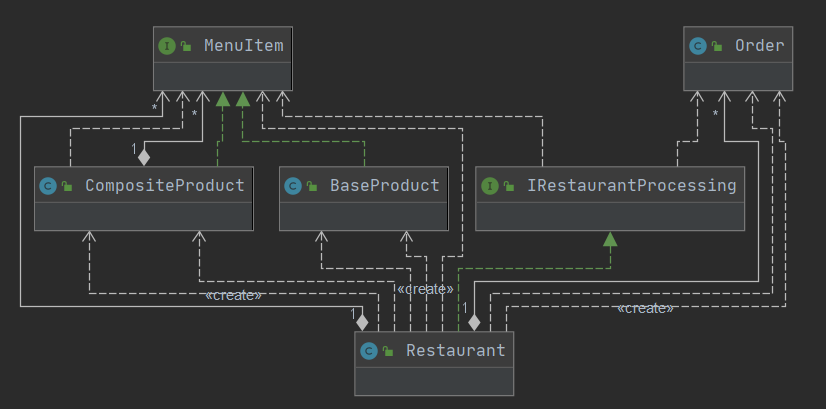
* **Interfață grafică: Administrator de restaurant**   
    
  Administratorul restaurantului poate să efectueze următoarele operații:
  + Creează un nou entry în meniul restaurantului, în secțiunea **Create Item**. Acesta poate fi produs de bază sau produs compus, lucru care se specifică prin bifarea căsuței corespunzătoare din partea de sus. Dacă nu se bifează nimic, se va crea un obiect compus. Pentru obiectele de bază se poate introduce și un preț.
  + Adaugă componente la un produs compus, în secțiunea **Add Item Components**. În prima listă apar toate obiectele compuse, din care administratorul va selecta pe care dorește să îl modifice. În lista de jos, apar toate produsele din meniul restaurantului, mai puțin cel selectat. Prin butonul Add Component se adaugă componenta selectată din al doilea meniu la produsul compus selectat în primul meniu.
  + Modifică produse din meniu, în secțiunea **Edit Item**. În funcție de tipul produsului selectat, se va afișa fie interfața de produs de bază, unde se poate modifica atât prețul cât și numele, fie interfața de produs compus, unde se poate modifica numele produsului și se pot elimina elemente din componența acestuia. Tot în această secțiune se pot șterge produse din meniul restaurantului. Se face o ștergere în adâncime, astfel că dacă se șterge un produs ce este parte componentă a altor produse, acele produse se vor șterge la rândul lor.
  + Vizualizează conținutul meniului prin interacțiunea cu butonul **ShowItems**. Acesta va deschide o nouă fereastă în care se pot vedea, organizate în formă tabulară, toate produsele din meniul restaurantului alături de prețurile lor.
* **Interfață grafică – Chelner**Chelnerul care interacționează cu aplicația poate efectua următoarele operații:
  + **Adaugă o nouă comandă** prin introducerea numărului mesei și apăsarea butonului **Create Order**.
  + **Adaugă produse la o comandă** prin selectarea id-ului comenzii (ultima comandă adăugată are id-ul cel mai mare), selectarea produsului ce se dorește a se adăuga și apăsarea butonului **Add Item**.
  + **Vizualiza toate comenzile** prin apăsarea butonului **ShowItems**. Astfel, se va deschide o nouă fereastră în care vor apărea, în formă tabulară, toate comenzile înregistrate la restaurant, alături de totalul pentru fiecare comandă.
  + **Genera prețul comenzii selectate** prin apăsarea butonului **Compute Price**. Prețul comenzii va fi calculat și va apărea imediat dedesubt butonului.
  + **Genera factura pentru comanda selectată** prin apăsarea butonului **Generate Bill**. Factura se va genera sub formă de fișier text în directorul corespunzător aplicației.
* **Interfață grafică – Chef/Bucătar**Aici, bucătarul care interacționează cu aplicația va putea vedea ce produse compuse trebuie pregătite, pe măsură ce acestea sunt adăugate într-o comandă.
* **Argument la execuție – opțional** – se poate da la execuție un argument care reprezintă calea către un fișier ce conține reprezentarea binară a obiectului instanță a clasei **Restaurant** rezultat prin efectuarea operației de serializare asupra acestuia. Dacă se specifică acest parametru, aplicația deserializează conținutul acestui fișier și pornește aplicația folosind obiectul rezultat prin deserializare. Dacă nu se specifică acest parametru la execuție, aplicația creează un obiect instanță a clasei **Restaurant** având o specificație default.
* **Generarea fișierului restaurant**.**ser –** la închiderea ferestrei de interfață grafică pentru Administrator și Chelner, aplicația serializează conținutul obiectului instanță a clasei **Restaurant** care a fost folosit în cadrul aplicației și îl stochează într-un fișier binar cu numele restaurant.ser, aflat în directorul curent al aplicației.

1. Proiectarea soluției

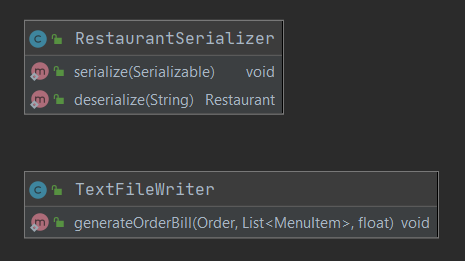
Se vor prezenta diagramele de pachete și de clase ce pun în evidență aspectele de proiectare OOP ale aplicației, precum și justificări legate de alegerile făcute în contextul proiectării.



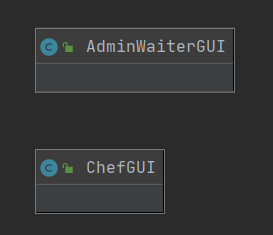
Pachetul default



Pachetul Business



Pachetul Data



Pachetul Presentation

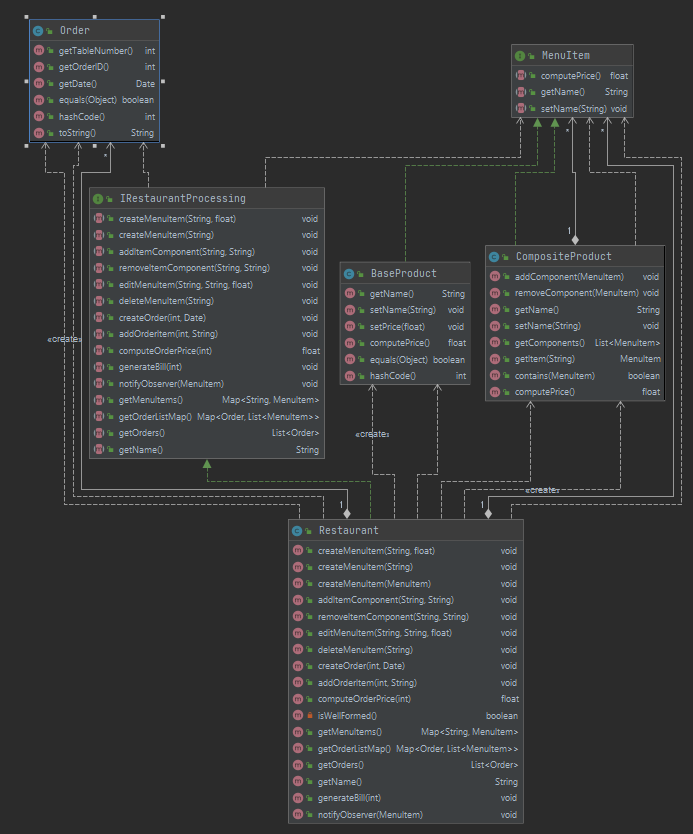
Diagramele de pachete de mai sus înfățișează structura pachetelor importante din implementarea aplicației. Această aplicație este structurată conform modelului arhitectural Layers, ce presupune împărțirea responsabilităților pe mai multe nivele, lucru care facilitează înțelegerea și dezvoltarea ulterioară a aplicației. Fiecare pachet este descris împreună cu toate clasele și responsabilitățile acestora după cum urmează:

* **Pachetul Business –** Conține clasele ce descriu modelul de date al aplicației și interfețele prin care exteriorul interacționează cu acest model.
* **Pachetul Presentation –** conține clase destinate interacțiunii cu utilizatorul – clasele AdminWaiterGUI și ChefGUI modelează ferestrele interfeței grafice cu care utilizatorul interacționează pe parcursul execuției aplicației.
* **Pachetul Data –** conține clase destinate lucrului cu fișiere text și fișiere în format binar pe care le generează prin serializare sau le citește și interpretează prin deserializare.
* **Pachetul default –** conține clasa MainClass, cu metoda Main, de unde începe execuția aplicației.

1. Implementare

Se va analiza fiecare clasă importantă în parte, argumentând alegerile de implementare acolo unde este cazul.

* + - 1. **Pachetul Data**
         * **Clasa RestaurantSerializer -** furnizează metode statice pentru interpretarea fișierului dat ca parametru opțional la execuția programului. Metoda **serialize** primește ca parametru un obiect instanță a clasei **Restaurant** și generează fișierul **restaurant.ser** prin procesul de serializare. Metoda **deserialize** primește ca parametru un șir de caractere ce reprezintă calea către un fișier cu conținut binar obținut prin serializarea unui obiect instanță a clasei **Restaurant**, deserializează conținutul acestui fișier și returnează obiectul instanță a clasei **Restaurant** obținut prin deserializare.
         * **Clasa TextFileWriter –** furnizează metoda statică **generateOrderBill** care primește ca parametrii un obiect instanță a clasei **Order**, o colecție de **MenuItem** ce reprezintă conținutul comenzii, precum și un număr reprezentând totalul comenzii. Această metodă generează un fișier text reprezentând factura pentru comanda respectivă.
      2. **Pachetul Business**



* + - * + **Clasa BaseProduct –** este clasa ce modelează produsele de bază. Un produs de bază este descris complet de numele și prețul acestuia.
        + **Clasa CompositeProduct –** este clasa ce modelează produsele compuse. Un produs compus este descris complet de numele acestuia și lista de componente (care pot fi la rândul lor produse compuse sau produse de bază).
        + **Interfața MenuItem** – este interfața pe care o implementează clasele **BaseProduct** și **CompositeProduct**, de asemenea clasa cu care interacționează clasa **Restaurant** și este folosită pentru implementarea modelului arhitectural Composite.
        + **Clasa Order –** este clasa ce modelează comenzile. O comandă este caracterizată de ID-ul comenzii, numărul mesei și data la care comanda a fost generată.
        + **Clasa Restaurant –** este clasa ce modelează restaurantul, obiectul principal al acestei aplicații. Este important să menționăm unele variabile instanță de bază ale acestei clase:

**orders** – este lista tuturor comenzilor înregistrate în restaurant. S-a folosit o colecție de tip ArrayList.

**menuItems**  - conține toate produsele din meniul restaurantului. Am folosit o colecție de tip HashMap care are drept cheie un String ce reprezintă numele produsului, iar valoarea este un MenuItem ce reprezintă produsul. Am făcut această alegere întrucât am făcut presupunerea că nu pot exista două produse care să aibă același nume. Mai mult, la interacțiunea cu utilizatorul, numele produselor sunt preluate sub formă de obiecte String ce reprezintă numele acestora, iar colecția HashMap este foarte eficientă pentru căutarea obiectelor.

**orderListMap –** este colecția ce mapează fiecare comandă la conținutul ei, adică la lista de MenuItems asociată comenzii respective. Utilizarea unui HashMap s-a dovedit a fi și aici potrivită, întrucât adăugarea și căutarea unei comenzi sunt operații foarte des efectuate într-un restaurant, iar colecția HashMap furnizează implementări foarte eficinte pentru astfel de metode.

* + - * + **Interfața IrestaurantProcessing –** este interfața prin care exteriorul poate interacționa cu obiectul instanță a clasei **Restaurant**. Definește toate operațiile importante ce se pot efectua de către un Administrator sau Chelner asupra restaurantului.
      1. **Pachetul Presentation**
         * **Clasa AdminWaiterGUI –**  este clasa ce implementează interfețele grafice pentru administrator și pentru chelner. Acestea se prezintă ca două sub-ferestre ale aceleiași ferestre. Utilizarea acestei interfețe a fost detaliată la capitolul 2.
         * **Clasa ChefGUI -** este clasa ce implementează interfața grafică a bucătarului. Această clasă implementează interfața Observer, întrucât bucătarul trebuie să știe când la o comandă s-a adăugat un produs compus, pentru a-l putea pregăti.
      2. **Pachetul default**
         * **Clasa MainClass**

1. Concluzii

În urma realizării acestei teme, am dobândit cunoștințe cu privire la utilizarea modelelor ahitecturale Layers, Observable și Composite, folosirea serializării obiectelor pentru a aduce aplicația într-o stare precedentă cunoscută, precum și generarea de fișiere text și javadoc. Am dobândit cunoștințe valoroase în legătură cu principiile modelului Design by Contract. De asemenea, am aprofundat cunoștințe cu privire la citirea și scrierea din fișier, realizarea unui executabil .jar care să fie rulat cu argumente în linia de comandă, realizarea unei interfețe grafice complexe, realizarea diagramelor UML ca ajutor în proiectarea aplicației, precum și realizarea unei documentații cuprinzătoare.

1. Bibliografie

[1] <http://coned.utcluj.ro/~salomie/PT_Lic/4_Lab/Assignment_4/Assignment_4_Indications.pdf>

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

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

[4] <https://www.jetbrains.com/help/>

[5] <https://www.udemy.com/>

[6] <https://www.geeksforgeeks.org/>

[7] <https://www.w3schools.com/>

[8] <https://stackoverflow.com/>

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