*Universitatea Tehnică Cluj-Napoca*

*Facultatea de Automatică și Calculatoare*

**RESTAURANT MANAGEMENT**

**SYSTEM**

*Tania Loreana Gabor*

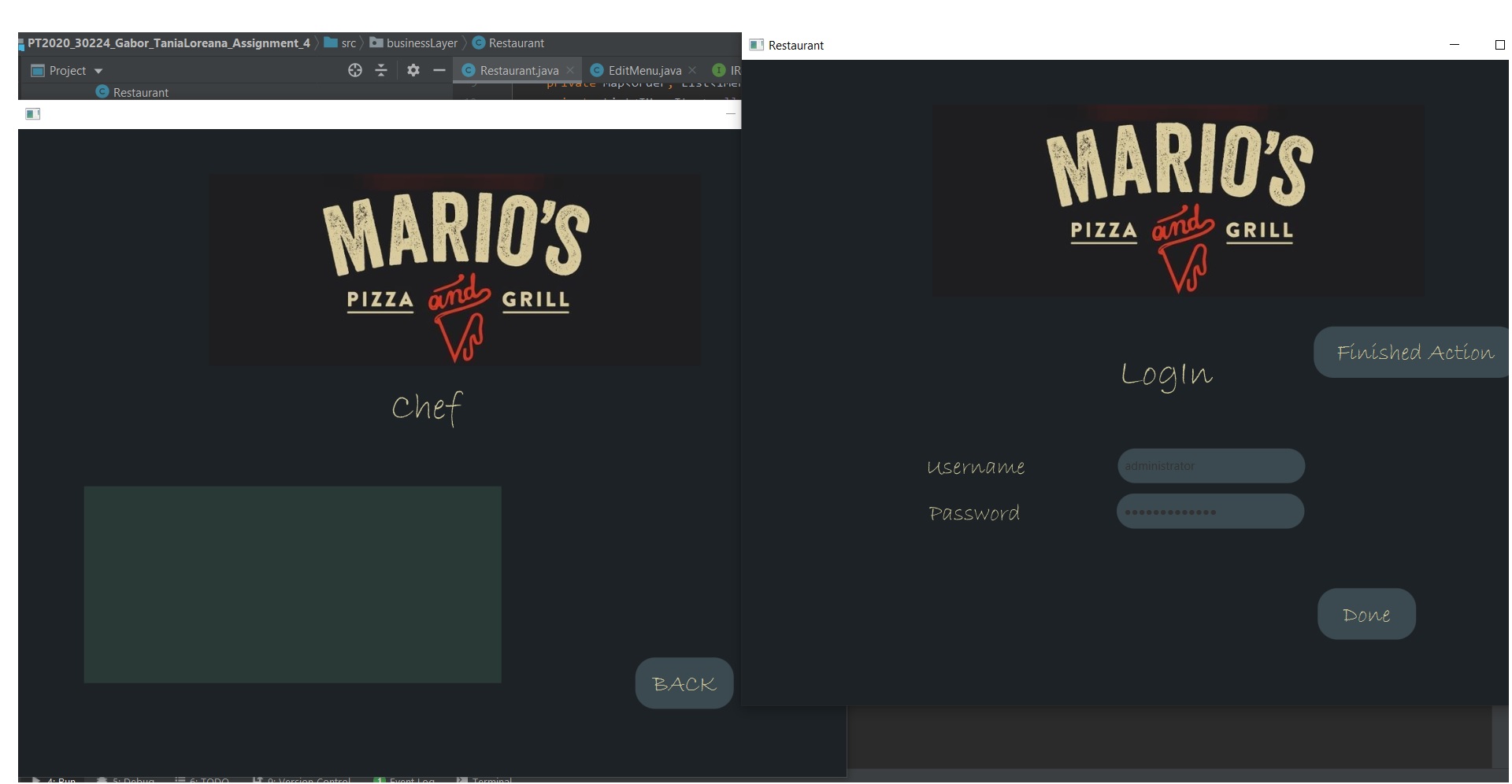
*Grupa:30224*

1. **Obiectivul principal**

Obiectivul principal al acestei teme este crearea unei aplicații pentru un sistem de administrare a unui restaurant. Sistemul are trei tipuri de utilizatori și fiecare dintre ei are o interfață grafică de utilizator diferită specifică pentru acțiunile pe care este autorizat să le efectueze. Administratorul poate insera, edita sau șterge elemente din meniu, precum și crea produse compuse sau de bază. Chelnerul poate vedea meniul și poate alege produse pentru a crea o nouă comandă. De asemenea, el poate genera factura pentru comanda respectivă. Bucătarul este înștiințat de fiecare dată când a fost făcută o nouă comandă.Atât administratorul cât și chelnerul pot vedea tablele corespunzătoare pentru meniu si comenzi.

1. **Analiza problemei**

Avem o anumită diagramă de clase pe care trebuie să o respectăm În primul rând, trebuie să definim ce este un Element din meniu <IMenuItem>și, de asemenea, trebuie să facem ca un produs să se distingă în produs de bază sau produs compus acesta fiind compus din mai multe produse de bază. Restaurantul va avea un meniu format din aceste produse. Apoi, trebuie să definim ce este o comandă și trebuie să asociem la fiecare comandă unele produse din meniu. Trebuie să creăm interfața grafică de utilizator pentru fiecare utilizator separat și, de asemenea, toate celelalte funcții pentru aceasta, cum ar fi,de exemplu, calcularea facturii într-un fișier txt.Am făcut un sistem de logare pentru diferitele tipuri de utilizator.Astfel pentru a te loga ca: administrator **username-ul** este administrator**,parola:** administrator,pentru a te loga ca și chelner **username-ul** waiter, **parola** waiter .Interfața de bucătar(chef) se deschide automat pentru a notifica o nouă comandă.De asemenea, utilizatorul se poate loga drept chef cu **username-ul** chef, **parola** chef.La final, pentru a salva în fișirul restaurant.ser modificările, trebuie apăsat butonul “Finished Action”.



* 1. **Diagrama cazurilor de utilizare**

O diagramă de caz este una dintre diagramele utilizate în UML pentru modelarea aspectelor dinamice ale unui program. Un caz de utilizare reprezintă cerințele unui tip de utilizator. Este o descriere a unui set de secvențe de acțiune pe care un program le execută atunci când interacționează cu entități din afara acestuia (actori) și duce la un rezultat care poate fi observat și util pentru actor.

**Pentru Administrator:**

• Titlul cazului de utilizare: Inserția Meniului

▪ Rezumat: acest caz de utilizare permite administratorului să insereze un nou produs în meniu

▪ Agenți: Administrator

▪ Scenariu principal de succes: ➢ Administratorul introduce datele pentru produsul care trebuie adăugate în câmpurile de text. . ➢ Se apasă butonul de introducere și se adaugă noul produs în funcție de tipul produsului,produs de bază sau compus.SE selectează tipul produsului de adăugat,dacă este de tip compus se va deschide o nouă interfață.

• Titlul cazului de utilizare: Actualizare Meniului

▪ Rezumat: Acest caz de utilizare permite utilizatorului să actualizeze un produs în tabel,pentru a-i schimba prețul.

▪ Actori: Administrator

▪ Principalul scenariu de succes: ➢ Administratorul introduce datele în câmpul text, precum și prețul articolului care trebuie schimbat➢ Butonul de actualizare este apăsat și produsul este actualizat la tabel.

• Titlul cazului de utilizare: Ștergere Meniu

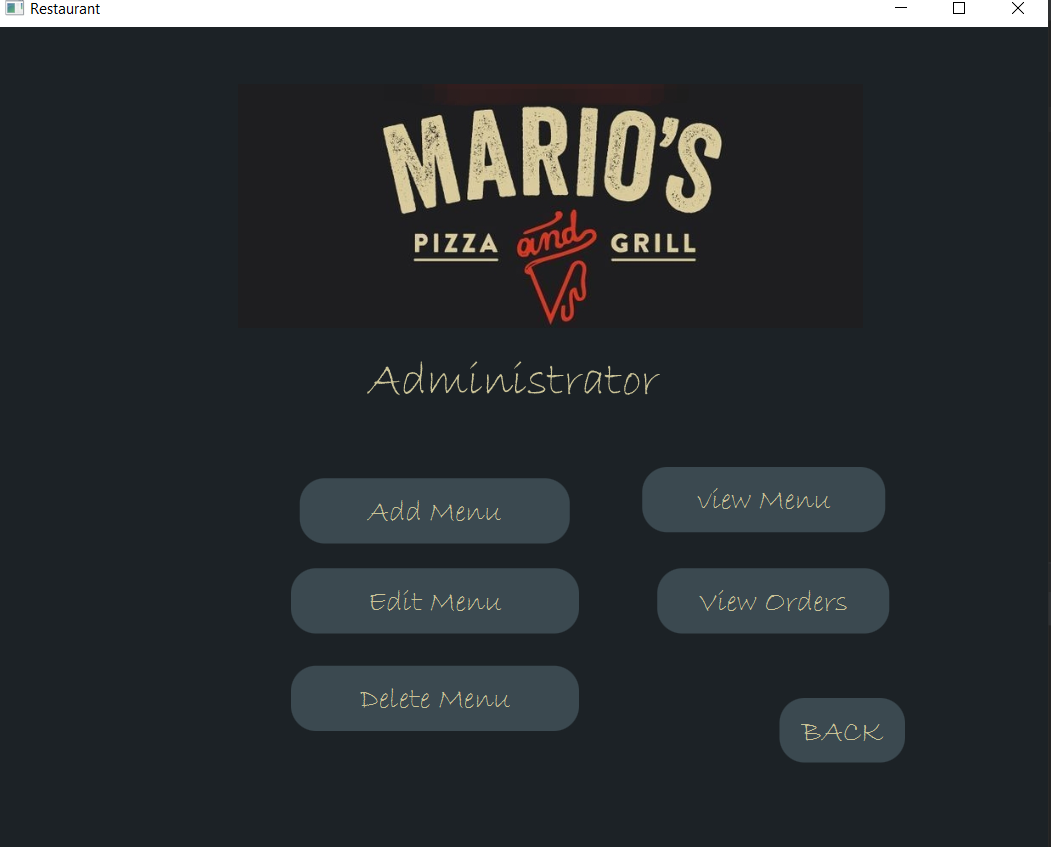
▪ Rezumat: acest caz de utilizare permite utilizatorului să șteargă un produs din tabel. este apăsat și produsul este șters din tabel

• Titlul cazului de utilizare: Vezi Comenzile

▪ Rezumat: acest caz de utilizare permite utilizatorului să vizualizeze comenzile într-un tabel.

• Titlul cazului de utilizare: Vezi Meniul

▪ Rezumat: acest caz de utilizare permite utilizatorului să vizualizeze meniul într-un tabel.



**Pentru Chelner**

• Titlul cazului de utilizare: Noua comandă

▪ Rezumat: acest caz de utilizare permite chelnerului să facă o nouă comandă

▪ Actorii: Chelnerul(waiter)

▪ Scenariul principal de succes: ➢ Chelnerul introduce data,id-ul comenzii,masa .Pentru a adăuga meniu corespunzător comenzii ,va apasa pe butonul “Add Menu”, iar la final, pentru a notifica cheful si a introduce comanda apasă butonul “Create Order” .

• Titlul cazului de utilizare: Vezi Comenzile

▪ Rezumat: acest caz de utilizare permite utilizatorului să vizualizeze comenzile într-un tabel.

• Titlul cazului de utilizare: Vezi Meniul

▪ Rezumat: acest caz de utilizare permite utilizatorului să vizualizeze meniul într-un tabel.

• Titlul cazului de utilizare: Generează Prețul

▪ Rezumat: acest caz de utilizare permite chelnerului să genereze prețul unei

comenzi.Trebuie să introducă datele corespunzătoare comenzii și să apese pe butonul “Compute Price”.

▪ Actorii: Chelnerul(waiter)

▪ Scenariul principal de succes: ➢ Chelnerul introduce data,id-ul comenzii,masa .Prețul este vizibil în caseta de text corespunzătoare.

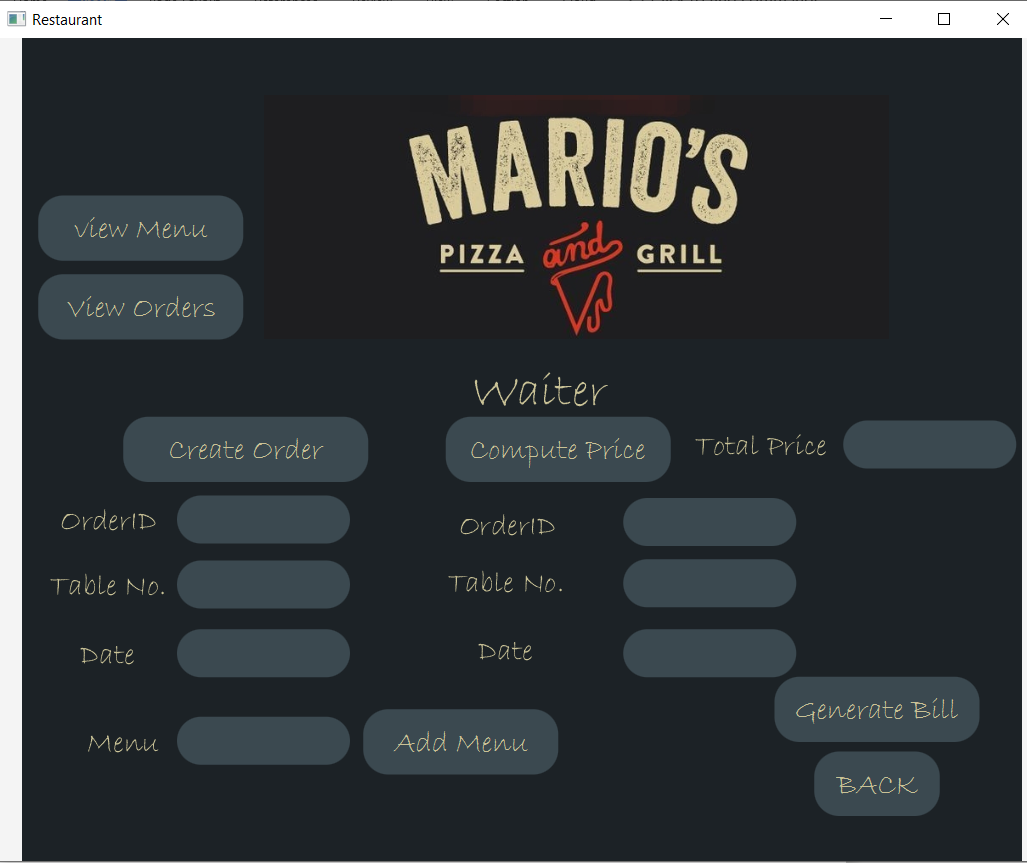
• Titlul cazului de utilizare: Generează Factură

▪ Rezumat: acest caz de utilizare permite chelnerului să genereze factura unei

comenzi.Trebuie să introducă datele corespunzătoare comenzii și să apese pe butonul “Generate Bill”.

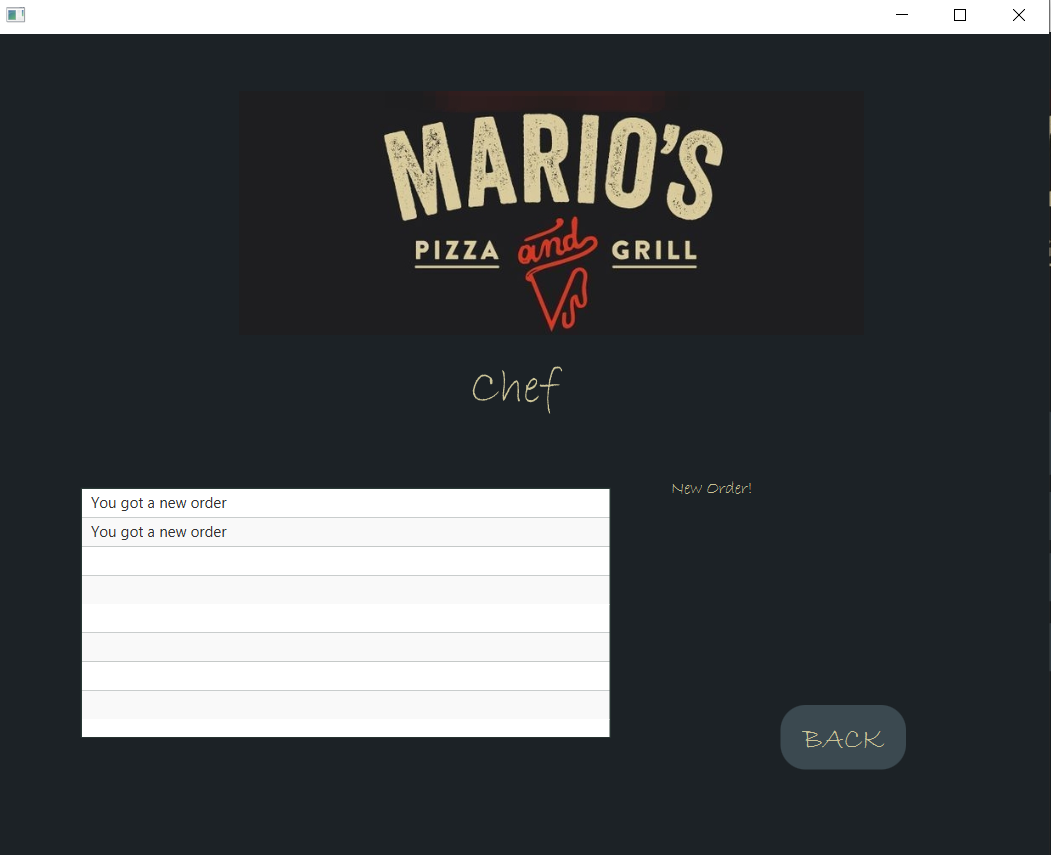
▪ Actorii: Chelnerul(waiter)

▪ Scenariul principal de succes: ➢ Chelnerul introduce data,id-ul comenzii,masa .Se generează factura pentru acea comandă sub formă de fișier txt.



**Pentru Bucătar(Chef)**

Acesta va primi notificare de fiecare dată când o comandă este făcută de către chelner.



Toți utilizatorii au un buton de “Back” care îl va duce pe utilizator în pagina anterioară.

**3. Proiectarea problemei**

**3.1 Proiectarea deciziilor**

După ce cerințele problemei au fost clar descrise, următorul pas este proiectarea problemei și luarea deciziilor despre modul în care fiecare pas va fi implementat.

Vom folosi o arhitectură stratificată. La nivel de bussiness avem logica aplicației, la nivel de prezentare avem interfețele grafice de utilizator pentru administrator, chelner și bucătar, iar în stratul de date vom avea clasele pentru scrierea într-un fișier pentru a crea txt factură și, de asemenea, o clasă pentru serializare de care vom avea nevoie pentru a nu pierde produsele deja adăugate înainte în meniu.

În primul rând, vom crea elementul meniu, produsul de bază și produsul compus pe baza modelului de Composite Design Pattern. Fiecare produs compus poate fi fabricat din alte produse de bază.

Apoi vom crea o comandă care are doar informațiile unei comenzi și în clasa Restaurant voi crea o structură de tipul hashmap, mapând comanda cu meniul asociat.

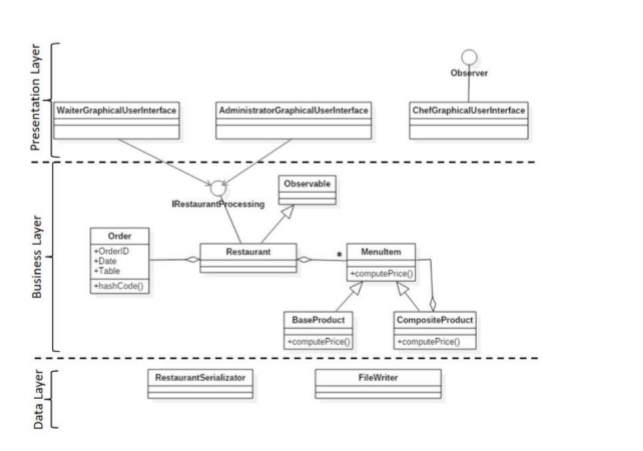
În clasa Restaurant vom avea, de asemenea, metodele necesare fiecărui utilizator pentru a-și efectua operațiunile permise.

În clasele GUI este implementată în principal ceea ce utilizatorul vede și conținutul cu care acesta poate interacționa.

Controllerul este prima pagină care se deschide alături de pagina de ChefGUI. Acesta controlează fluxul de date în model și actualizează fișierul.ser ori de câte ori se schimbă datele,prin apăsarea butonului “Finished Action”.

**3.2 Diagrama UML - diagrama clasei**

Diagrama de clase este o diagramă scare descrie structura unui sistem, arătând clasele sistemului, atributele, operațiile sau metodele acestora și relația dintre obiecte.Ni s-a oferit o anumită diagramă de urmat.



**3.3 Pachetele**

Au fost create trei pachete pe baza arhitecturii stratificate.

Stratul de date este locul în care avem clasele care interacționează cu alte fișiere externe, stratul bussinessLayer conține logica programului și presentationLayer este pachetul pentru interfața grafică cu utilizatorul.DataLayer reprezintă pachetul în care s-au făcut clasele FileWriter, pentru scrierea in fisierele txt care reprezintă facturile, cât și RestaurantSerialization unde se face serializarea si deserializarea fisierului de input restaurant.ser.

1. **Implementare**

În această secțiune voi descrie fiecare clasă cu metodele sale principalele

**4.1 Clasa IMenuItem(Interfață)**

Această interfață este cea care creează un element meniu. Va fi oclasă care a fost implementată alături de CompositeProduct si BaseProduct pe modelul Composite Design Pattern.Aceasta conține metoda compositePrice care va trebui implementată

În clasele BaseProduct și CompositeProduct.

**4.2 Clasa BaseProduct**

 Această clasă este o clasă care implementează interfața IMenuItem. Acesta conține metoda de compositePrice , dar și getteri și setteri.

**4.3 Clasa CompositeProduct**

 Această clasă este o clasă care implementează interfața IMenuItem. Acesta conține în comparație cu BaseProduct, o listă ArrayList de tipul IMenuItem, deoarece un CompositeProduct poate fi compus din multe alte IMenuItem.

Această clasă conține metoda de compositePrice dar și getteri și setteri.

**4.4 Clasa Order**

Această clasă este cea care conține toate atributele unei comenzi, precum și getters și setters pentru acestea.

**4.5 Clasa Restaurant**

Această clasă este cea în care avem principala a aplicației. Conține o listă ArrayList care păstrează toate elementele din meniu, o listă de matrice care conține toate elementele din meniu pentru comanda curentă, un HashMap pentru maparea IMenuItems la comenzi .

Metoda addMenu adaugă un meniu la lista de meniuri.

Metoda createComposite preia din lista de elemente din meniu un element din indexul specificat și îl transformă într-un produs compozit.

Metoda deleteMenu este metoda prin care administratorul poate șterge un meniu.În fiecare din acestea am adăugat postcondiții și precondiții,folosindu-ma de metoda isWellFormed() ca invariant, care imi spune daca un obiect este null.

Clasa are, de asemenea, getters și setters.

**4.6 Clasa FileWriter**

Această clasă are doar metoda write. Obține o comandă ca parametriu și apoi trimite textul comenzii specificate pentru a fi tipărit într-un fișier txt reprezentând factura.

**4.7 Clasa RestaurantSerialization**

Aceasta este clasa serializează restaurantul. Conține 2 metode, serializează care serializează restaurantul și deserializează ceea ce va deserializa restaurantul și va obține lista elementelor din meniu.

**4.7 Clasa IRestaurantProcessing**

Această clasă este interfața care descroe principalele funcții care trebuie implementate de clasa Restaurant care o implementează.

**4.7 Clasele Observer și Observable**

Observable este un model de design comportamental. Precizează comunicarea între obiecte: cei care sunt observați și observatori. Un observabil este un obiect care notifică observatorii despre modificările stării sale.Modelul de observare este un model de proiectare software în care un obiect, numit subiect, menține o listă a dependenților săi, numiți observatori și îi notifică automat cu privire la orice modificări de stare, de obicei apelând la una dintre metodele lor.

Este utilizat în principal pentru implementarea sistemelor distribuite de gestionare a evenimentelor, în software-ul „condus de evenimente”. În aceste sisteme, subiectul este de obicei denumit „flux de evenimente” sau „flux de evenimente”, în timp ce observatorii sunt numiți „scufundarea evenimentelor”.

La acest proiect, interfața ChefGraphicalUserInterface este notificată de fiecare dată când este încărcată o nouă comandă.Restaurant extinde clasa Observable , iar

ChefGraphicalUserInterface implementează interfața Observer.Astfel, prin funcția notifyAll() trimite un semnal de fiecare dată când este adăugată o comandă.În Main, am adăugat prin metoda addObserver, clasa de Chef.

**4.7 Clasele care definesc Interfețele Grafice-GUI**

Clasele care definesc interfața grafică a acestui proiect, sunt incluse în pachetul presentationLayer.

Acesteasunt :ViewOrders,ViewMenu,ChefGaphicalUserInterface,WaiterGraphicalUserInterface,Controller,EditMenu,AddMenu,AddComposite,DeleteMenu.Pentru realizarea interfeței grafice, am folosit JavaFx.A fost nevoie să includ librariile folosite de JavaFX 11 și în .jar.În interfața Controller am realizat sistemul de logare.Această interfață este cea care se deschide prima, deodată cu cea de ChefGraphicalUserInterface ,pentru a notifica o nouă comandă.Interfața AddComposite a fost necesară pentru a face distincția între adăugarea unui produs de bază și adăugarea unui produs compus.Tot clasa Controller conține un buton Finished Action care atunci când este apăsat produce serializarea în restaurant.ser.A fost nevoie de o clasă Wrapper pentru Main , pentru a adăuga în fișierul .jar librăria specifică JavaFX.

**5. Concluzii**

În această temă a trebuit să folosim și să învățăm câteva concepte și tehnici noi. În primul rând am avut deja diagrama clasei și a trebuit să putem proiecta aplicația noastră pe baza ei. Am învățat cum să utilizăm serializarea și deserializarea, cum să implementăm modelul de design observabil și compozit, precum și cum să lucrezi cu hashing în java.

Pe parcursul dezvoltării acestui proiect am învățat și mi-am îmbunătățit câteva

aptitudini:

- O mai bună utilizare a Intelliji.

- O mai bună utilizare a limbajului de codare Java

- O mai bună înțelegere și utilizare a programării orientate pe obiect

- Cod mai organizat și clar

- O mai bună înțelegere a elementelor grafice ale interfeței

- Aflarea despre observatorul modelului de proiectare.

- Aflarea proiectării prin contract.

- Aflarea despre serializare.

**6.Posibile îmbunătățiri**

-Adăgarea mai multor restaurante.

- Adăugarea mai multor detalii despre Comenzi

- Adăugarea mai multor detalii la meniuri

- Tipărirea stărilor fiecărei comenzi

- Construirea unei părți client a aplicației.

- Adăugarea către partea de manager a aplicației a unei zone destinate companiilor, nu numai clienților personali.

- Aplicația poate fi dezvoltată prin adăugarea opțiunii de angajați multiplii,adică mai mulți chelneri, mai mulți bucătari sau poate chiar mai mulți administratori.

- Aplicația ar putea calcula nu doar prețul unor comenzi pentru a fi adăugate facturii, ci ar putea calcula și venitul zilnic al restaurantului.

- Adăugarea feedbackului din partea bucătarului spre chelner.

- În urma calculului venitului zilnic s-ar putea realiza o statistică de vinit lunar.

- Dacă se va cunoaște venitul restaurantului se pot adăuga opțiuni de salarizare.

- S-ar putea lua în calcul ordinea comenzilor și timpul de servire.

- E posibilă introducerea unui feedback de la clienți.

-Și în final i s-ar putea adăuga o parte de rezervări care a putea rula online.

**6. Bibliografie**

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

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

<https://beginnersbook.com/2014/07/super-keyword-in-java-with-example/>

https://beginnersbook.com/java-io-tutorial-with-examples/