SISTEM DE ADMINISTRARE A UNUI RESTAURANT

1. Obiectivul temei:

Obiectivul acestei teme este de a dezvolta si implementa un sistem de administrare a unui restaurant ce trebuie sa contina 3 interfete grafice cu utilizatorul, cei 3 utilizatori fiind: un administrator al restaurantului, un chelner si un bucatar. Prin realizarea acestei teme programatorul isi va imbunatati aptitudinile in ceea ce priveste programarea pe obiecte si va invata noi concepte POO precum lucrul cu Design Pattern-uri si utilizarea serializarii, deserializarii.

2. Analiza problemei:

**2.1 Analiza problemei, modelare:**

Se cere dezvoltarea si implementarea unui sistem de administrare a unui restaurant ce contine un meniu cu diverse produse. Sistemul trebuie sa fie capabil sa interactioneze cu 3 utilizatori diferiti, fiecare dintre acesti utilizatori avand accesul la operatii specifice in fuctie de staturul pe care il ocupa. Cei trei utilizatori sunt: administratorul restaurantului, chelnerul si bucatarul.

Administratorul restaurantului efectueaza operatii ce tin doar de meniu. Operatiile specifice sunt:

* adaugarea unui nou produs; Noul produs adaugat poate fi fie un produs de baza, fie un produs compus( ex.: cartofi cu crispy)
* stergerea unui produs din meniu
* editarea unui produs; Se poate modifica numele unui produs, pretul sau compozitia (in cazul in care produsul editat este un produs compus)
* vizualizarea tuturor produselor din meniu

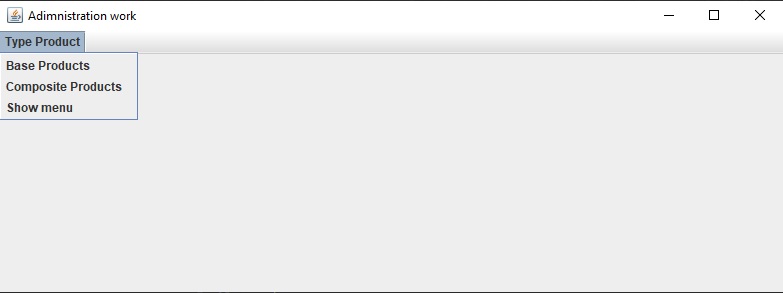
Chelnerul are rolul de a prelua comenzile de la clienti si de a anunta bucatarul ce trebuie sa pregateasca. Operatiile pe care le poate efectua sunt:

* crearea unei noi comenzi
* vizualizarea tuturor comenzilor
* crearea unui facturi pentru o comanda

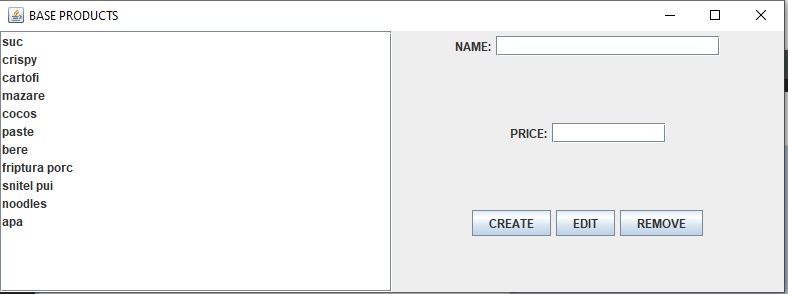
Bucatarul nu efectueaza direct nicio operatie, acesta trebuie sa fie notificat de fiecare data cand apare o noua comanda ce contine produse compuse pentru a le gati.

Pentru cei 3 utilizatori se va face cate o interfata grafica, ce va contine doar operatiile necesare si specifice.

**Interfata grafica a administratorului:**



Contine un meniu te tip barMenu, din care se selecteaza tipul de produs pe care se doreste sa se efectueze operatii. Daca se face click pe “Base Product”, atunci se va deschide o noua fereastra care va permine sa se efectueze operatii pe produse de baza, daca se da click pe “Composite Products” se va deschide o alta fereasta ce ii va permite utilizatorului sa efectueze operatii pe produsele compuse, iar daca se da click pe “Show menu” se va deschide o a 3 a fereastra ce va contine un tabel cu toate produsele din meniu si detaliile despre acestea.



Mai sus este reprezentata fereastra de lucru pentru produsele de baza. Contine o lista cu toate produsele de baza din meniu in partea stanga, iar in partea dreapta avem 2 casute text si 3 butoane.

Pentru realizarea operatiei **CREATE,** se vor introduce datele specifice noului produs in cele 2 casute si anume numele si pretul. Numele trebuie sa fie un String, iar pretul un double. Dupa ce datele sunt introduse se apasa butonul CREATE, iar noul produs va fi introdus in meniu. Trebuie mentionat faptul ca nu pot exista 2 produse cu acelasi nume, astfel daca se incearca crearea unui astfel de produs se va afisa pe ecran un mesaj de eroare.

Pentru realizarea operatiei **EDIT**, se selecteaza produsul ce se doreste a fi modificat din lista, se introduce noul pret, iar apoi se apasa pe butonul de edit. Mentionz faptul ca se poate edita doar pretul pentru un produs de baza. In cazul in care se va incerca modificarea numelui, sau in cazul in care in casuta de price nu se introduce un double, pe ecran se va afisa un mesaj de eroare specific.

Pentru realizarea operatiei **REMOVE**, se selecteaza din lista produsul ce se doreste a fi sters, iar apoi se apasa pe butonul REMOVE.



In aceasta imagine este reprezentata fereastra de lucru pentru produsele compuse. Contine 2 liste: una cu produsele compuse si una cu produse de baza, 3 casute text, precum si 3 butoane.

Casutele text COMPOSITE si PRICE nu pot fi editate, adica nu se pot introduce date in interiorul acestora. Sunt folosite doar pentru vizualizare, fiind completate automat pe baza produselor de baza din compozitia unui produs compus.

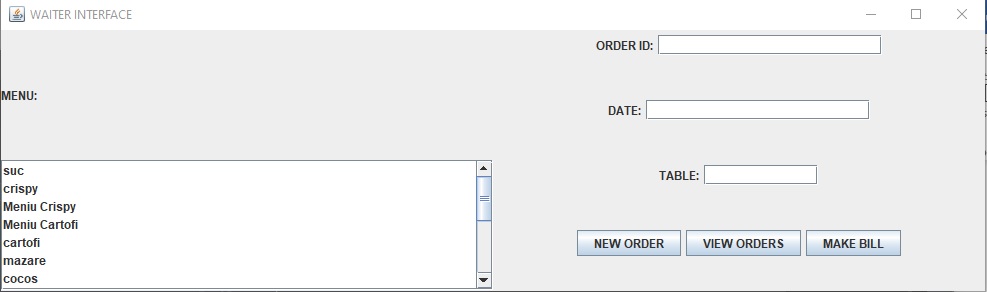
Pentru realizarea operatiei de **CREATE** se va introduce numele produsului compus in casuta text “NAME”, apoi se selecteaza produsele de baza care alcatuiesc noul produs, iar apoi se apasa pe butonul CREATE (pentru selectia mai multor produse din lista se foloseste combinatia ctrl+click stanga). In caz de succes noul produs se va afisa in lista Composite Products, iar in caz de esec se va afisa pe ecran un mesaj de eroare. Din nou, nu se permite folosirea aceluiasi nume cu a unui produs deja existent.

Pentru realizarea operatiei **EDIT**, in prima faza se selecteaza din lista Composite Products produsul ce se doreste a fi modificat. Apoi, daca se doreste modificarea numelui se introduce noul nume in casuta text, iar in cazul in care se doreste modificarea compozitiei se vor face urmatoarele:

* daca se doreste stergerea unui produs de baza din compozitie se selecteaza din lista Base Product acel produs
* daca se doreste adaugarea unui nou produs de baza se selecteaza din lista Base Product

Se pot selecta atat produse ce se doresc a fi sterse cat si produse ce se doresc a fi adaugate in acelasi timp. Dupa ce selectia este facuta se apasa pe butonul EDIT. Pentru deselectarea unui produs ctrl+click stanga.

Pentru realizarea operatiei **REMOVE** se selecteaza produsul ce sedoreste a fi sters , iar apoi se apasa butonul REMOVE.

**Interfata grafica a chelnerelui:**

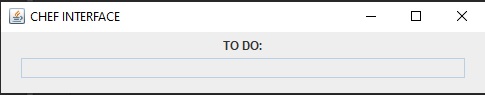
Dupa cum se vede contine o lista cu toate produsele din meniu, 3 casute text in care se introduc date despre comanda si 3 butoane.

Pentru realizarea operatiei **NEW ORDER** in prima faza se trec datele necesare in casutele text, order id si table fiind int, iar date String. Apoi se selecteaza din meniu produsele comandate, iar apoi se apasa butonul NEW ORDER. In cazul in care datele introduse in casutele text sunt identice cu datele altei comenzi, atunci se va afisa un mesaj de eroare pe ecran, iar comanda respectiva nu va putea fi procesata.

Pentru operatia **VIEW ORDERS** se apasa doar butonul respectiv, iar apoi se va deschide o fereastra cu toate comenzile.

Pentru operatia **MAKE BILL** se vor introduce datele specifice comenzii careia se doreste a se emite un bonul. In cazul in care comanda nu exista se va afisa un mesaj pe ecran, iar in cazul in care exista se va genera un bon in format .txt.

**Interfata grafica a bucatarului:**

****

Dupa cum am spus mai sus, bucatarul nu efectueaza nicio operatie, el doar este anuntat la crearea unei noi comenzi ce trebuie sa pregateasca. Astfel, daca o noua comanda contine produse compuse asestea vor fi afisate in casuta text din interfata. In cazul in care comanda nu contine produse compuse se va afisa “NOTHING NEW”.

**2.2 Scenarii/ Use case-uri:**

**Use case:** se ruleaza fisierul .jar file, pentru ca mai apoi se genereze interfetele grafice

**Primary actor**: administrator, chelner, bucatar

**Cazul de succes:**

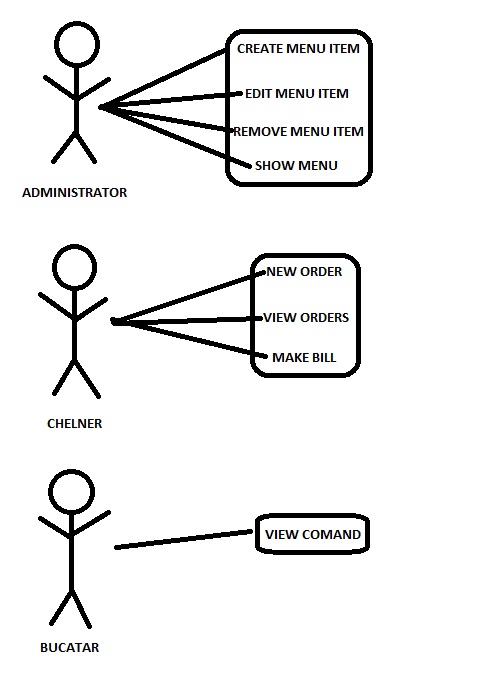
1. Se ruleaza aplicatia din linia de comanda, dandu-se ca argument un fisier .ser
2. Se vor genera interfetele grafice
3. Fiecare utilizator acceseaza interfata sa grafica
4. Se efectueaza operatiile dorite de fiecare utilizator
5. Dupa fiecare operatie efectuata de administrator, datele se vor serializa si se vor sa lva in fisierul dat ca si argument
6. La final e inchid interfetele grafice

Se poat vizualiza in final datele prin intermediul tabelelor ce pot fi accesate din interfata grafica, atat pentru meniu cat si pentru comenzi.

Pentru fiecare utilizator in parte am detaliat in **2.1** fiecare pas pentru fiecare operatie

**Scenarii de esec:**

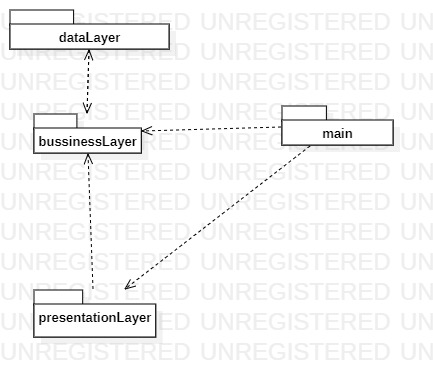
* Fisierul dat ca si argument nu exista, se va genera o eroare
* Datele introduse intr-una dintre interfetele grafice sunt gresite, se va afisa un mesaj de eroare



**3. PROIECTARE:**

In acest proiect au fost luate cateva decizii importante de proiectare precum: structurarea aplicatiei pe layere, folosirea design pattern-urilor Composite si Observer, si folosirea serializarii si deserializarii pentru pastrarea informatiilor necesare functionarii optime a aplicatiei. De asemenea am decis ca aplicatia sa fie rulata prin intermediul executabilului .jar, din linia de comanda insotit fiind de un fisier .ser in care se va face serializarea.

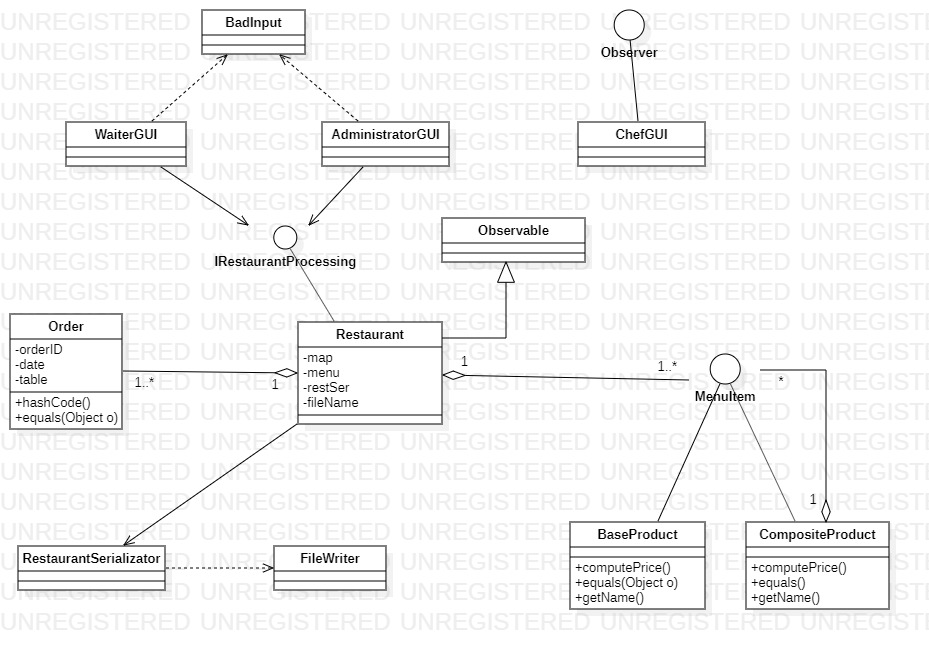
Principalele layere din acest proiect sunt:

* bussinessLayer- contine clasa principala a aplicatiei, care se ocupa cu logica
* dataLayer- acest layer se ocupa de stocarea si reutilizarea datelor (prin serializare si deserializare)
* presentationLayer- layer ce se ocupa de partea de interfata grafica cu utilizatorul, mediu prin intermediul caruia utilizatorul interactioneaza cu aplicatia

Pachetele aplicatiei sunt urmatoarele:

* businessLayer- contine clase ale caror scop principal este sa se ocupe cu logica aplicatiei
  + clasa Order
  + clasa Restaurant
  + clasa BaseProduct
  + clasa CompositeProduct
  + interfata MenuItem
  + interfata IRestaurantProcessing
* dataLayer- contine clase care se ocupa cu manipularea datelor
  + clasa RestaurantSerializator
  + FileWriter
* control- contine clase care se ocupa de partea de control a interfetei grafice cu utilizatorul
  + ControlAdmin
  + ControlAdminBp
  + ControlAdminCp
  + ControlWaiter
* view- contine clasere ce se ocupa cu afisarea interfetelor grafice
  + ViewAdmin
  + ViewAdminCp
  + ViewAdminBp
  + ViewChef
  + ViewWaiter
* exceptionLayer- contine o clasa ce extinde Exception; arunga un nou tip de exceptie
  + BadInput
* main- contine main-ul aplicatiei
  + MainClass

Mai departe voi prezenta diagrama de clase si obiecte corespunzatoare acestei aplicatii:



Cea mai semnificativa **Clasa** a acestui proiect este clasa **Restaurant**. Un restaurant are un meniu (mai multe obiecte de tip MenuItem), iar la un restaurant se pot da mai multe comenzi, fiecare comanda avand mai multe produse comandate (o structura de date ce poate face legatura intre comenzi si produsele din meniu). In final, restaurantul trebuie sa mentina produsele din meniu cu modificarile facute, acest lucru facandu-se cu un serializator. De asemenea, in momentul realizarii unei noi comenzi, bucatarul trebuie sa fie anuntat, astfel clasa extinde Observable. Restaurant implementeaza interfata IRestaurantProcessing, deci, toate metodele aferente acestei interfete.

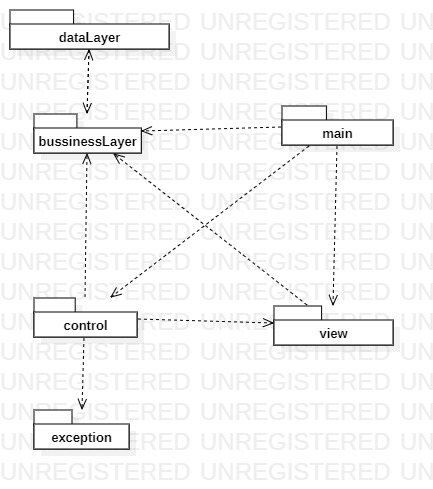
**Clasele BaseProduct si CompositeProduct**, implementeaza ambele interfata MenuItem. Astfel un meniu poate avea atat doar produse de baza cat si produse compuse, alcatuite din produse de baza. Aici se utilizeaza Design Pattern-ul Composite pe care l-am amintit la inceputul acestui capitol. Interfata MenuItem joaca rolul de componenta, BaseProduct de frunza, iar CompositeProduct de compus. Drept urmare un obiect de tip CompositeProducts contine o lista de elemente de tipul MenuItem, care alcatuiesc practic produsul compus.

**Interfetelor grafice WaiterGUI si AdministratorGUI** (mai concret claselor de tip Control din aceste 2 interfete) le este asociat cate un obiect de tipul IRestaurantProcessing, pentru a putea avea acces la operatiile care se pot efectua intr-un restaurant.

**Interfata grafica ChefGUI** implementeaza interfata Observer pentru a putea observa ce produse compuse trebuie sa gateasca din meniu. Deci, prin urmare, celalalt Desing Pattern prezent in acest proiect se utilizeaza ca bucatarul sa fie notificat de fiecare data cand apare o noua comanda cu produse copuse. Acest Design Pattern consta intr-un obiect Observable care este restaurantul si un obiect de tip Observer care este interfata grafica pentru bucatar. In momentul in care se creeaza comanda restaurantul notifica, in timp ce bucatarul primeste sau nu notificarea (in functie de cazul in care ne aflam).

**Interfetele** acestui proiect sunt:

* Observer- care este implementata de interfata grafica a bucatarului, pentru a implementa metoda de update care face posibila afisarea produselor ce trebuiesc a fi gatite
* IRestaurantProcessing- este implementata de clasa Restaurant si contine operatiile care pot fi efectuate de catre administrator si chelner:
  + Administrator: crearea unui nou produs, stergerea unui produs, editarea unui produs;
  + Chelner: crearea unei noi comenzi, calculeaza pretul unei comenzi, generarea unei facturi in format .txt
* MenuItem- este implementata de clasele BaseProduct si CompositeProduct. Aceasta interfata favorizeaza utilizarea Design Pattern-ului Composite. Metodele definite sunt:
  + computePrice();
  + equals(Object o);
  + getName();



In imaginea de mai sus este prezentata diagrama de pachete si relatiile stabilite intre ele. Am prezentat mai sus scopul fiecarui pachet, precum si clasele continute si scopul acestora.

**Structurile de date** utilizate sunt:

* HashMap- utilizata in clasa Restaurant pentru a se face legatura intre comenzi si produsele comandate. Pe post de cheie se foloseste on obiect de tip Order, iar pe post de valoare un ArrayList de obiecte de tip MenuItem;
* ArrayList- se utilizeaza in mai multe clase, dar in mod special in clasa Restaurant pentru meniu acesta continand o lista de obiecte d tip MenuItem

**4. Implementarea:**

In acest capitol se va prezenta in detaliu fiecare clasa cu metodele ei cele mai importante.

**Clasa BaseProduct:**

Contine 2 variabile instanta: name (numele produsului) si price (pretul produsului). Clasa contine metode de get si set, precum si metodele definite in interfata MenuItem pe care o mosteneste:

* computePrice()- returneaza pretul produsului
* equals()- suprascrie metoda din clasa Object
* getName()- returneaza numele produsului

**Clasa CompositeProduct:**

Contine 2 variabile instanta: name(numele produsului) si itemList (un ArrayList ce contine obiecte de tipul MenuItem, obiecte ce alcatuies produsul). Cele mai importante metode ale clasei sunt:

* computePrice()- returneaza pretul produsului compus, adunand pretul tuturor produselor ce compun produsul compus;
* equals()- suprascrie metoda din clasa Object
* getName()- returneaza numele produsului
* addProduct()- adauga un produs in lista de produse
* removeProduct()- elimina un produs din lista de produse;

**Clasa Order:**

Contine 3 variabile instanta: orderId, date (data la care s-a realizat comanda) si table (numarul mesei). Cele mai importante metode ale clasei sunt:

* hashCode()- suprascrie metoda din clasa Objects; returneaza un hashCode care se formeaza cu ajutorul variabilelor instanta. Metoda este utila in clasa Restaurant cand se face mapare dupa comanda
* equals()- suprascrie metoda din clasa Object

**Clasa Restaurant:**

Probabil cea mai importanta clasa a proiectului, deoarece implementeaza operatiile care pot fi facute de chelner si administrator. Variabilele instanta sunt: map (HashMap ce are ca si cheie un obiect de tip Order, iar ca valoare un ArrayList cu biecte MenuItem, folosita pentru gestionarea comenzilor), menu (ArrayList cu obiecte de tip MenuItem, reprezinta meniul restaurantului), restSer (obiect prin intermediul caruia se efectueaza serializarea si deserializarea) si fileName (numele fisierului din care se face deserializarea si in care se face serializarea).

Metodele clasei sunt:

* createMenuItem()- operatie specifica pentru administrator, creeaza un nou obiect de tip MenuItem si il introduce in meniu, in cazul in care acesta nu exista;
* deleteMenuItem()- operatie specifica pentru administrator, sterge un obiect de tip MenuItem dat ca si parametru din meniu, in cazul in care acesta exista;
* editMenuItem()- operatie specifica pentru administrator, editeaza un obiect de tip MenuItem din meniu, in cazul in care acesta exista; Are 2 parametri, primul parametru reprezinta obiectul care urmeaza sa fie editat, al doilea este obiectul cu modificarile efectuate;
* createOrder()- operatie specifica pentru chelner, creeaza o noua comanda. Are 2 parametri: primul este un obiect de tip Order, al doilea un ArrayList<MenuItem>.Se comanda se insereaza in map cu cheia Order si valoarea ArrayList;
* computePriceOrder()- operatie specifica pentru chelner, returneaza pretul total pentru o comanda. Parametrul este un obiect de tipul Order dupa care se face cautarea in HashMap;
* generateBill()- operatie specifica pentru chelner, genereaza o factura in format .txt cu toate datele despre o comanda. Comanda este selectata prin intermediul parametrului metodei care este un obiect de tip Order;
* isWellFormed()- metoda ce ne asigura ca dupa executarea unei operatii structura clasei ramane aceeasi;

**Clasa FileWork:**

Se ocupa cu generarea facturilor. Nu contine variabile instanta, contine doar 2 metode:

* createFile()- creeaza un fisier de tip .txt;
* makeBill()- creeaza o factura cu datele specifice;

**Clasa RestaurantSerializator:**

Are rolul de a pastra datele necesare ale acestui proiect. Nu are variabile instanta, contine 2 metode:

* serialization()- serializeaza datele din meniu intr-un fisier dat ca si parametru;
* deserialization()- deserializeaza datele dintr-un fisier dat ca si parametru;

**Clasele ControlAdmin, ControlAdminBp, ControlAdminCp:**

Toate aceste clase reprezinta partea de control a interfetei grafice cu administratorul. Acestea au rolul de a crea ascultatori pentru butoanele sau listele identificate in interfata. Metodele definite in interfata IRestaurantProcessing care reprezinta operatiile specifice unui administrator sunt aici folosite pentru generarea ascultatorilor.

Clasele contin 2 variabile instanta principale: view (partea de vizualizare a interfetei grafice) si restaurant (care reprezinta modelul interfetei grafice- contine metodele principale care actioneaza asupra datelor din Restaurant).

**Clasa ControlWaiter:**

Aceasta clasa reprezinta partea de control a interfetei grafice cu chelnerul. Clasa generaza ascultatori pentru butoanele si lista din interfata grafica. Acesti ascultatori sunt implementati folosind metodele care sunt specifice chelnerului din interfata IRestaurantProcessing, metode ce le-am specificat mai sus.

Contine 2 variabile instanta principale: view (partea de vizualizare a interfetei grafice) si restaurant (care reprezinta modelul interfetei grafice- contine metodele principale care actioneaza asupra datelor din Restaurant).

**Clasele ViewAdmin, ViewAdminBp, ViewAdminCp:**

Aceaste clase sunt responsabile cu afisarea interfetei grafice a administratorului, cu partea de vizualizare. Contin toate componentele necesare afisarii ferestrei cu care administratorul interactioneaza. Metodele acestor clase sunt utilizate tot pentru ascultatori, mai specific pentru a atribui ascultatori componentelor vizate.

**Clasa ViewWaiter:**

Aceasta clasa este responsabila cu afisarea interfetei grafice a chelnerelui, cu partea de vizualizare. Contine toate componentele necesare afisarii ferestrei cu care chelnerul interactioneaza. Metodele acestor clase sunt utilizate tot pentru ascultatori, mai specific pentru a atribui ascultatori componentelor vizate.

**Clasa ViewChefWaiter:**

Aceasta clasa este responsabila cu afisarea interfetei grafice a bucatarului, cu partea de vizualizare. Contine toate componentele necesare afisarii ferestrei cu care bucatarul interactioneaza. Aceasta clasa nu are metode de tip ascultatori, deoarece nicio componenta a acestei interfete nu necesita ascultatori. Dupa cum se stie, aceasta clasa implementeaza interfata Observer, drept urmare impelenteaza si metoda update.

Metoda de update este folosita pentru afisarea produselor ce trebuiesc a fi gatite. In momentul in care metoda createOrder() a clasei Restaurant notifica o noua comanda se apeleaza automat clasa update.

**Clasa BadInput:**

Clasa este utilizata pentru aruncarea unui nou tip de exceptie. Extinde clasa Exception.

**5. Rezultate:**

Rezultatele pot fi vizualizate in diverse moduri:prin vizualizarea tabelelor ce pot fi accesate din interfetele grafice sau direct din interfata grafica dupa efectuarea fiecarei operatii. Dupa fiecare operatie se afiseaza un mesaj de eroare in cazul in care aceasta nu a decurs conform planului. De exemplu daca se adauga un produs nou in meniu, dupa efectuarea operatiei se vor putea vedea modificarile in lista cu produse din meniu. In cazul in care adaugarea nu are succes se afiseaza un mesaj de eroare pe ecran.

**6.Concluzii:**

In sfarsit putem spune ca am reusit sa dezvoltam si sa implementam un sistem de administrare a unui restaurant,prin intermediul caruia am invatat sa utilizam un Design Pattern in functie de necesitate pentru o eficienta mai buna, precum si cum sa serializam si deserializam date.

Ca si dezvoltari ulterioare am putea adauga mai multi utilizatori, iar fiecare interfata grafica sa necesite autentificare.

**7. Bibliografie:**

1. <https://www.journaldev.com/1535/composite-design-pattern-in-java>
2. <https://www.journaldev.com/1739/observer-design-pattern-in-java>
3. <http://coned.utcluj.ro/~salomie/PT_Lic/4_Lab/Assignment_4/Assignment_4_Indications.pdf>
4. <https://stackoverflow.com/questions/11415160/how-to-enable-the-java-keyword-assert-in-eclipse-program-wise>
5. <https://www.geeksforgeeks.org/serialization-in-java/>