

**FACULTATEA DE AUTOMATICĂ ŞI CALCULATOARE**

**CATEDRA CALCULATOARE**

**RESTAURANT MANAGEMENT SYSTEM**

Documentatie Tema 4

Runcan Nicoleta

Grupa 30229

An 2019-2020

**Cuprins**

1.Obiectivul temei

2. Analiza problemei, modelare, scenarii, cazuri de utilizare

**Scenariu de utilizare**

**Cazuri de utilizare**

3.Proiectare

**Decizii de proiectare**

**Diagrama de clase**

**Structuri de date**

**Interfete**

**Interfata Utilizator**

4.Implementare

5.Rezultate

6.Concluzii

7.Bibliografie

1.Obiectivul temei

Dupa cum stim cu totii, orice restaurant detine 3 tipuri de angajati foarte importanti: administrator, ospatar si bucatari/bucatar-sef. Aceasta tema presupune efectiv o implementare a activitatilor pe care aceste trei tipuri de angajati ar putea sa le efectueze. In primul rand se considera ca restaurantul are un singur administrator, un singur bucatar-sef si un singur ospatar, iar obiectivul acestei teme este ca pentru fiecare dintre cele 3 personaje de mai sus sa fie implementata cate o interfata grafica, care reprezinta meniul pentru operatiile pe care le poate efectua fiecare: adiministratorul poate adauga elemente in meniul restaurantului, poate sterge elemente din acesta sau poate modifica elemente existente deja in meniu, ospatarul poate sa creeze o comanda noua pentru o masa, sa calculeze pretul total pentru acea comanda sau sa genereze o factura pentru comanda respectiva, iar bucatarul-sef trebuie sa fie anuntat de fiecare data cand a fost plasata o comanda care contine un produs compus. De asemenea, un alt obiectiv al acestei teme a fost familiarizarea studentilor cu diferite concepte de programare: Observer Design Pattern, Design by Contract method, Composite Design Pattern, si de asemenea conceptul de serializare.

2. Analiza problemei, modelare, scenarii, cazuri de utilizare:

Un restaurant este locul care ne face pe fiecare dintre noi sa petrecem cateva clipe minunate alaturi de persoane dragi, savurand delicatesele pe care le pune la dispozitia clientilor, si atmosfera placuta din interiorul acestuia. Pentru ca noi, ca si clienti, sa avem parte de o experienta asa cum ne dorim atunci cand mergem intr-un restaurant, este necesar ca acesta sa aiba sistemul sau de gestionare a management-ului correct implementat, care sa faca mai usoara viata angajatilor si mai placuta experienta clientilor. Exista o multitudine de modalitati prin care poate fi implementata o astfel de aplicatie prin care sa se gestioneze starea meniului unui restaurant de catre administrator, sau inregistrarea comenzilor si generarea facturilor pentru comenzi de catre ospatar si de asemenea sa fie capabila sa anunte bucatarul de fiecare data cand acesta trebuie sa inceapa prepararea unui produs compus.

**Scenariu de utilizare:**

Aceasta aplicatie ar trebui sa fie una foarte usor de folosit de fiecare dintre cele 3 tipuri de angajati. Astfel, atat interfata grafica a administratorului, cat si cea a ospatarului trebuie sa contina cateva JTextField-uri in care administratorul sau ospatarul poate sa introduca datele pe care le doreste atunci cand vrea sa efectueze o anumita operatie si de asemenea cateva butoane prin care sa poata alege ce operatie sa efectueze la un moment dat, iar interfata grafica a bucatarului este reprezentata doar de un AreaText, in care o sa apara cate un mesaj de fiecare data cand ospatarul a inregistrat o comanda care presupune preparea a cel putin un produs. In interfata grafica a administratorului ar trebui sa contina un buton, la apasarea caruia sa fie introdus un nou produs in meniu, un buton prin care sa se faca stergerea unui produs din meniu, un buton prin care sa se editeze elemente existente in meniu si unul prin care sa se poata viziona intreg meniul din restaurant intr-un tabel. In cea a ospatarului, urmatoarele butoane ar fi necesare: unul cu ajutorul caruia sa se inregistreze o noua comanda, unul prin care sa se calculee pretul acestei comenzi, unul prin care sa se gestioneze o factura pentru o anumita comanda, si de asemenea unul, prin apasarea caruia sa fie afisate toate comenzile intr-un tabel.

**Cazuri de utilizare:**

Aplicatia este foarte usor de folosit de catre utilizatori, datele introduse trebuie doar sa fie correct introduse si sa aiba un format valid, pentru ca operatiile pe care le poate face sa fie executate cu success.

3.Proiectare

**Decizii de proiectare:**

Pentru proiectarea acestei aplicatii am ales modelul MVC, care presupune izolarea logicii interne fata de considerentele interfetei cu utilizatorul. In clasa Model(care in aplicatia implementata de mine ar fi clasa Restaurant) sunt manipulare operatiile.De asemenea exista si cate o clasa controller pentru administrator si ospatar care controleaza accesul la aplicatie, aici facandu-se legatura intre partea vizibila utilizatorului si programul care ruleaza in spatelele acestor interfete; iar clasa View corespund reprezentarii grafice, sau mai bine zis exprimarea ultimei forme a datelor.



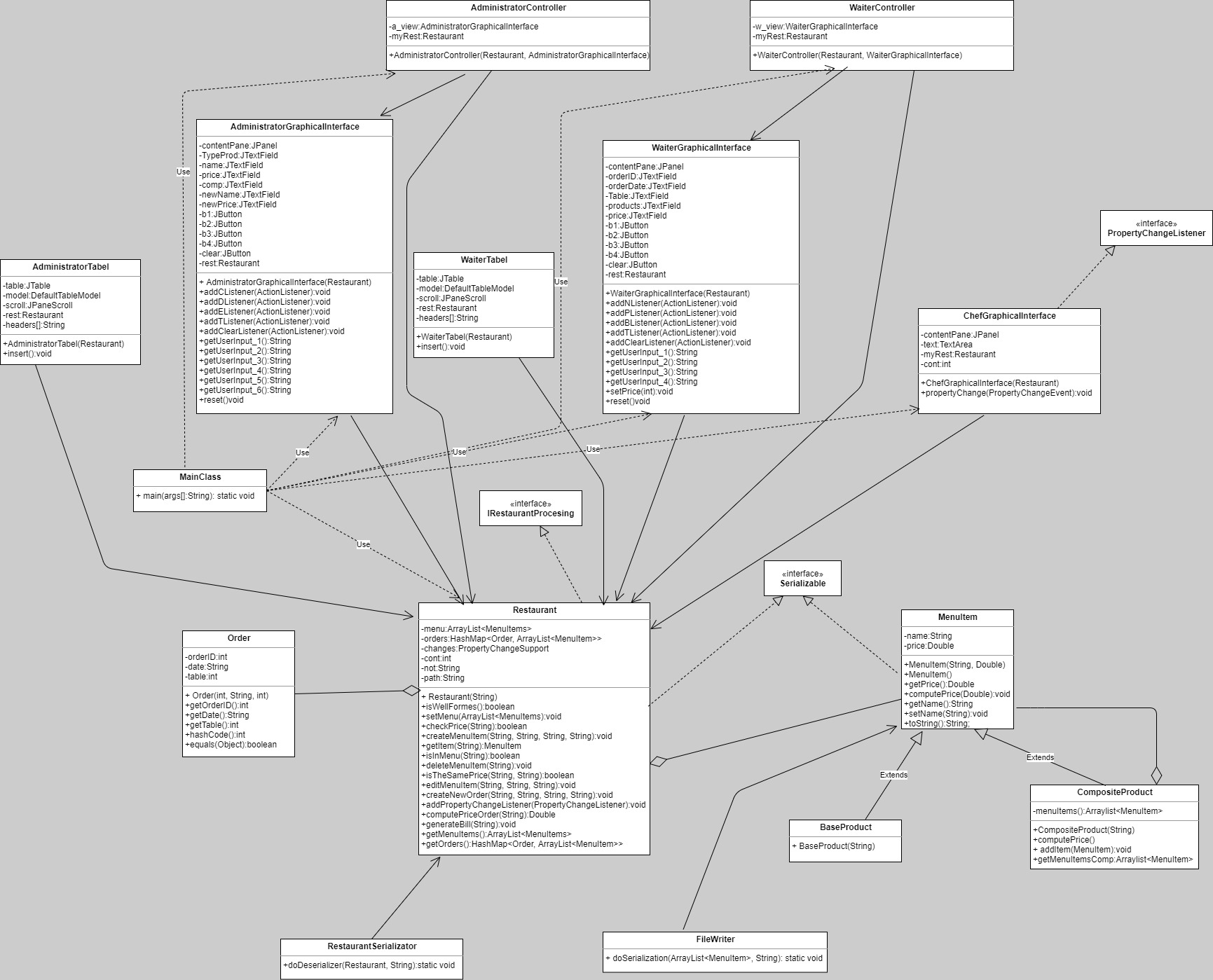
O alta decizitie de proiectare a fost folosirea serializarii. Serializarea obiectelor inseamna salvarea si restaurarea starii obiectelor. Obiectele oricarei clase care implementeaza interfata *Serializable*, pot fi salvate  intr-un stream(fluxde date) si restaurate  din acesta. In aceasta aplicatie, colectia meniului este cea care este serializata. Astfel, prin serializare-deserializare, modificarile efectuate asupra meniului intr-o anumita rulare a aplicatiei vor fi salvate intr-un fisier de tip .ser, iar la o rulare urmatoare acesteia, prin deserializare, aceste modificari ii vor fi restituite meniului.

De asemenea pentru pastrarea comenzilor restaurantului am decis sa folosesc o colectie predefinita bazata pe structua de date: tabela de dispersie, aceasta fiinf un HashMap. Cheia acestui HashMap este reprezentata dintr-un obiect de tip Order, iar valoarea va fi reprezentata dintr-un ArrayList de MenuItems, in care se pastreaza produsele comandate.

**Structuri de date:**

Dupa cum am mentionat si mai sus, una dintre structurile de date folosite a fost un HashMap de comenzi, in care sunt stocate comenzile restaurantului. Cheia acestui HashMap este un obiect de tip Order, pentru care in clasa Order, va fi suprascrisa metoda hashCode(), pentru a calcula valoarea hash din Map din atributele acestei clase Order(orderId, date si table), iar valoarea este reprezentata tot dintr-o structura de date: un ArrayList de MenuItems, care pastreaza produsele continute in comanda respective(din data aceea si de la msa respective). De asemenea si pentru pastrarea produselor in meniu se foloseste tot un ArrayList de MenuItems, si tot un ArrayList de MenuItems se foloseste si pentru pastrarea produselor care intre in component unui produs compus(un produs compus poate sa contina in component sa atat produse de baza, cat si alte produse compuse existente deja in meniu.

**Diagrama de clase:**

****

**Interfete:**

In implementarea acestei aplicatii a fost definite o interfata numita IRestaurantProcessing, care contine definitia operatiilor principale care pot fi efectuate de administrator si de ospatar. De asemenea pe langa aceasta interfata, programul mai foloseste si alte interfete predefinite: Clasa Restaurant si MenuItem implementeaza interfata Serializable prin fara de care nu s-ar putea efectua operatiile de serializare-deserializare a meniului. Si de asemenea clasa ChefGraphicalInterface implementeaza interfata PropertyChangeListener, aici fiind implementata si metoda specifica acestei interfete: metoda propertyChange() cu ajutorul carora se face anuntarea bucatarului in momentul in care este plasata o comanda care contine un produs compus.

**Interfata Utilizator(Utilizarea acesteia) :**

Interfața grafică sau Graphical User Interface(GUI) este o [interfață cu utilizatorul](https://ro.wikipedia.org/wiki/Interfa%C8%9B%C4%83_cu_utilizatorul) bazată pe un sistem de afișaj ce utilizează elemente grafice. Interfața grafică este situata funcțional între utilizator și implementarea care se afla in spatele acestei interfete. In cazul de fata sunt implementate 3 interfete grafice: una pentru administrator, una pentru ospatar si una pentru bucatar care contine doar un AreaText in care este afisat urmatorul mesaj: “A fost introdusa o comanda care contine un produs compus”.

***Administrator***

Interfata grafica a adimistratorului contine mai multe JLabel-uri care reprezinta mai exact niste etichete, care pot fi titlul, informații, indicații etc. care pot ajuta utilizatorul să folosească progamul; 6 JTextField-uri, in care acest utilizator va introduce datele de intrare pe care vrea sa le foloseseasca pentru efectuarea operatiilor disponibile si de asemenea inca 5 butoane:

* **Create Menu Item**
* **Delete Menu Item**
* **Edit Menu Item**
* **View MenuItems**
* **Clear**

La apasarea butonului de **Create Menu Item**, un nou produs va fi adaugat in meniu. Dar inainte de apasarea acestui buton trebuie sa ne asiguram ca datele pe care noi vrem sa le folosim pentru introducerea unui nou produs sunt corecte si valide. Astfel, in JTextField-ul marcat de eticheta: **Product Type** trebuie introdus tipul produsului “Base” sau “Composite”; in urmatorul JTextFiled marcat de **“Name”** trebuie introdus numele produsului care va fi adaugat in meniu, dupa care,in cazul in care in prima casuta am introdus variant **“Base”,** in JTextField-ul marcat de eticheta **“Price”** trebuie introdus pretul pe care il atribuim acestui produs, in schimb, daca in prima casuta variant aleasa a fost **“Composite”,** casuta cu pretul nu va fi luata in considerar, dar in acest caz, va trebui ca in JTextField-ul marcat de eticheta **“Component Products”** trebuie introduse numele produselor, separate prin virgula si spatiu, care se doresc a intra in componenta acelui produs compus, iar bineinteles daca cel putin unul dintre produsele puse in acest camp, nu se gaseste in meniu, atunci noul produs nu va fi adaugat.

Butonul **Delete Menu Item** efectueaza stergerea unui produs din meniu. Inainte ca butonul sa fie apasat, este necesara introducerea numelui produsului in casuta etichetata cu **“Name”** despre care am mai vorbit si anterior. Daca in meniu va fi gasit un produs cu acest nume, el va fi sters si de asemnenea vor fi sterse si toate produsele compuse in ca caror componenta se regaseste prosului cu acest nume.

Prin apasarea butonului **Edit Menu Item,** un produs existent in meniu va fi modificat. Aplicatia implementata de mine, permite ca prin acest buton, fie numele unui produs sa poata fi schimbat, fie pretul acestuia, fie atat pretul cat si numele. Inainte de apasarea acestui buton trebuie introdus in casuta **“Name”,** numele produsului asupra caruia dorim sa efectuam modificari, si de asemenea, fie in JTextField-ul **“New Name”,** noul nume pe care dorim sa-l atribuim produsului, fie in JTextField-uol **“New Price”,** noul pret pe care dorim sa-l atribuim produsului, fie in ambele casute daca dorim sa efectuam ambele modificari simultan.

Prin butonul **“View MenuItems”,** se va deschide o noua fereastra cu un tabel cu meniul restaurantului. Acest tabel contine trei coloane: numele produsului, pretul si produse componente ale unui produs; pe aceasta coloana vor aparea date doar daca este vorba despre un produs compus.

Butonul **“Clear”** are rolul de “curate” casutele in care se introduc date.

***Ospatar***

Asemanator, meniului administratorului, si aceasta interfata contine cateva JLabel-uri care functioneaza fie pe post de etichete si mai multe JTextField-uri in care se introduce datele necesare efectuarii operatiilor puse la dispozitie. Si in aceasta interfata, sunt puse la dispozitie 5 butoane:

* **Add New Order**
* **Compute Price**
* **Generate Bill**
* **View Orders**
* **Clear**

Prin apasarea butonului **“Add New Order”,** o noua comanda va fi adaugata in HashMap-ul orders. Dar inainte de apasarea butonului, trebuie introduse datele care sunt necesare in adaugarea unei comenzi. In JTextField-ul cu eticheta **“Order Id”** trebuie introdus id-ul comenzii, in urmatoare casuta etichetata cu **“Order Date”,** trebuie introdusa data la care s-a facut comanda, iar in urmatoarea casuta de **“Table”** trebuie introdus numarul mesei de la care a fost facuta comanda. Aceste trei date sunt necesare pentru generarea cheii noului element care va fi adaugat in HashMap. Dupa introducerea acestor date, trebuie introduse in JTextField-ul **“Order Products”,** numele produselor comandate, separate prin virgule si spatiu. Daca niciunul dintre produsele comandate nu se gaseste in meniu, atunci aceasta comanda nu va fi adaugata.

Prin butonul **“Compute Price”,** se va calcula pretul comenzii adaugate actual, daca aceasta comanda nu poate fi adaugata din motivul specificat mai sus, dar si din alte motive, pretul total va fi zero. Pentru vizualizarea acestui pret total, exista un JTextField etichetat **“Total price”,** in care este afisat acest pret.

Butonul **“Generate Bill”** se ocupa de generarea unei facturi intr-un fisier text(ex:”Bill1.txt) pentru coamanda tocmai plasata. Daca aceasta comanda nu poate fi adaugata, nu va fi generata nicio facture.

Butonul **“Clear”** are rolul de “curate” casutele in care se introduc date.

***Chef***

Interfata bucatarului contine doar un **JTextArea**, in care este afisat urmatorul mesaj in cazul in care a fost adaugata o noua coamnda care contine un produs compus: “A fost introdusa o comanda care contine un produs compus”.

4 Implementare:

In acest capitol se vor descrie clasele si metodele importante, componente a acestor clase, cu logica din spatele lor.

**Pachetul BusinessLayer:**

Acest pachet, pe langa interfata **IRestaurantProcessing** despre care am amintitmai sus, mai contine inca 5 clase.

Clasa **MenuItem** este absolut necesara acestui program. Ea are doua variabile instanta, una de tip String: name care reprezinta numele, si una de tip double, care reprezinta pretul produsului. Tot aici regasim un constructor, cu un sigur parametru de tip String, care initializeaza variabila instanta name; dar si un constructor gol. Ca si metode importante in aceasta clasa, regasim doar metoda computePrice(), care este mai degraba un setter pentru variabila instanta price. Prin aceasta metoda setam o valoare pentru variabila instanta price, in rest regasindu-se doar metode accesoare si mutatoare.

Doua clase care mostenesc aceasta clasa **MenuItem,** sunt: clasa **BaseProduct**, care nu contine nicio alta variabila instanta sau metoda in plus fata de clasa parinte, si clasa **compositeProduct,** care contine in plus ca si variabila instanta un ArrayList de MenuItems, deoarece un produs compus poate sa contina in component sa atat produse de baza, dar si produse compuse care exista in meniu; folosindu-se pentru implementarea si definirea acestor trei clase conceptual de Composite Design Pattern. De asemenea in aceasta clasa este suprascrisa si metoda de computePrice(), deoarece pretul unui produs compus se seteaza ca si suma tuturor produselor care intra in component sa.

Clasa **Order** are trei variabile instanta, doua de tip int: orderId, table, care reprezinta id-ul comenzii si numar mesei care a facuyt comanda; si una de timp String:date, care reprezinta data la care a fost pusa comanda. Aceasta clasa are un singur constructor care initializeaza aceste trei campuri. In aceasta clasa se regasesc doua metode importante: hashCode(), despre care am mai tot amintit, are rolul de a genera o cheie pentru HashMap-ul in care se tin comenzile, pe baza celor trei atribute ale clasei. Tot in aceasta clasa trebuie suprascrisa metoda equals(), pentru ca generarea cheilor sa se faca corect.

Clasa **Restaurant** reprezinta creierul acestei aplicatii, in ea fiind implementate metodele de baza, descrise si in interfata IRestaurantProccesing. De asemenea aceasta clasa este implementata folosind conceptual de Design by Contract, adica in implementarea metodelor sunt folosite preconditii si postconditii, pentru a documenta schimbarea starii cauzata de o bucata din program. Metodele din aceasta clasa specifica conditiile lor pre si post, ce trebuie sa fie adevarat inainte de executarea lor si ce trebuie sa fie adevarat dupa, ele fiind specificate prin asertiuni. Aceasta clasa are ca si variabile instanta importante un ArrayList de MenuItems in care este pastrat meniul restaurantului si un HashMap in care sunt pastrate comenzile. De asemenea tot o variabila instanta a cestei clase este un obiect de timp PropertyChangeSupport, care gestioneaza o lista de ascultatori care trebuie notificati atunci cand o proprietate se schimba, acest obiect ajuta la anuntarea bucatarului in momentul in care un produs compus a fost comandat.

Prima metoda importanta a acestei clasa este metoda de adaugare a unui produs nou. Aceasta metoda are ca si parametri 4 String-uri, unul care reprezinta tipul produsului, unul care reprezinta numele, unul care reprezinta pretul unul care reprezinta produsele component si este folosit doar daca este vorba despre un compositeProduct. Aceasta metoda a fost implementata astfel; se verifica daca primul parametru este “Base”, daca este, inseamna ca se doreste adaugarea unui produs de tip Base, ii este setat pretul si dupa care este adaugat in meniu, dar daca primul parametru nu este “Base”, si este “Composite”, atunci fac un split pentru string-ul de produse component dupa regex-ul “, “, pentru a lua fiecare nume de produs scris in acel String, dupa care verific daca in meniu se gasesc produse care sa aiba toate aceste nume si daca da, aoelez metoda de compuPrice() pentru un obiect de tip compositeProduct si dupa care il adaug in meniu. Ca si preconditie a acestei metode verific daca primul parametru are doar aceste doua forme “Base” sau “Composite”, daca pretul introdus are o forma valida sau daca este vorba despre un produs compus, string-ul de produse component sa nu fie null, iar ca si postconditie, ca size-ul arrayList-ului sa fi crescut cu 1.

A doua metoda este metoda de stergere a unui produs din meniu, aceasta metode are ca si parametru un string care reprezinta numele produsului care trebuie sters. Aceasta metoda este o metode recursive deoarece in momentul in care se doreste stergerea unui produs, trebuie sterse si toate produsele compuse care contin acest produs. Ca si preconditii pentru aceasta metoda am ales sa verific ca parametrul metodei sa nu fie nul, iar in meniu sa se regaseasca un produs care sa aiba acest nume. Iar ca si postconditii, verific ca size-ul ArrayList-ului de dupa stergere trebuie sa fie mai mic decat size-ul de dinainte de stergere.

A treia metoda importanta este cea de editare a produselor, care are trei String-uri ca si parametrii. Aceasta metode permite modificarea numelui unui produs, modificarea pretului unui produs sau atat modificarea numelui cat si a pretului. Ca si preconditii pentru aceasta metoda am ales sa verific ca in meniu sa existe un produs cu numele pe care il dam noi ca si prim paramtru al metodei, el reprezentand produsul pe care dorim sa il editam. Si de asemenea o alta preconditie este ca String-ul newPrice dat ca si parametru sa fie un pret valid(doar cu cifre si sa nu fie negatic), iar ca si postconditii, trebuie ca in meniu sa nu se mai regaseasca un produs cu numele dat ca si prim parametru functiei si sau sa existe in meniu un produs care sa aiba pretul egal cu newPrice-ul dat ca si parametru.

A patra metoda importanta este cea prin care se adauga o noua comanda. Aceasta metoda are ca si parametrii 4 String-uri: primul reprezentand id-ul comenzii, al doilea data comenzii, al treilea masa de la care s-a comandat si al patrulea un string cu produsele comandate. La fel si aici se efectueaza un split dupa virgula si spatiu acestor produse comandatek, dupa care se verifica existenta in meniu a produselor cu aceste nume, daca niciun produs nu a fost gasit in meniu, atunci nu se va mai adauga comanda in hashMap-ul de orders. Tot aici se verifica si daca printer produsele comandate se gaseste un compositeProduct si daca se gaseste se apeleaza metoda de firePropertyChange() pentru a anunta bucatarul ca a fost introdusa o comanda cu un CompositeProduct. Ca si pre si postconditii la fel se verifica validitatea datelor de intrare si daca size-ul Arraylist-ului este mai mare sau egal cu valoarea de dinainte de adaugare

Iar ultimele doua metode sunt cele de computePrice(), si generateBill(), care la fel au ca si parametrii cate un string de produse, si care calculeaza pretul produselor comandate, si dupa efectuarea unei facturii intr-un fisier de tip text. Am gandit aceste trei metode legate de comenzi astfel: se introduce o comanda, i se face pretul acesteia, dupa care facture si asa mai departe pentru fiecare comanda. Si aici ca si pre si post contidii se verifica validitatea datelor de intrare si a rezultatelor.

Tot o metoda importanta este cea wellFormed() care este de tip Boolean, in ea facandu-se o verificare a meniului sa fie diferit de nul, si o verificare a listei de produse associate unei comenzi, ca si aceasta sa fie diferita de nul. Aceasta metoda trebuie sa returneze true indiferent de ce metoda se apeleaza, ea fiind apelata cu assert la inceputul si sfarsitul fiecareia dintre cele 6 metode importante ale programului.,

**Pachetul DataLayer**

In acest pachet se gasesc doua clase:FileWriter si RestaurantSerializator, care contin fiecare cate o metoda statica, in fileWriter metoda doSerialization(), in care se efectueaza serializarea meniului, are ca si parametrii un ArrayList de MenuItems si un string care reprezinta fisierul in care sa se faca scrierea. Aceasta metoda de serializare este apelata in clasa Restaurant la sfarsitul fiecarei metode de adaugare in meniu de stergere a unui produs si de editare, pentru e fi pastrate toate modificarile efectuate asupra meniului. In clasa RestaurantSerializator se gaseste metoda statica doDeserializer() prin care se face deszerializarea, aceasta metoda fiind apelata la inceput in clasa MainClass().

**Pachetul PresentationLayer**

In acest pachet se afla implementarea interfetelor grafice, si legatura dintre aceste clase si Restaurant prin clasele de Controller. Detaliile despre aceste clase din pachetul amintit mai sus au fost descries in capitol anterioare(Interfata grafica, si la Decizii de proiectare)

5.Rezultate:

S-ar putea afirma ca rezultatele pe care la obtinem in urma implementarii unei aplicatii sunt cele mai importante, dar mai important este ca aceste rezultate sa fie cele asteptate. In aceasta aplicatie rezultatele pot fi observate in tabelul care contine tot meniul, dar si in tabelul in care se gasesc comenzile. Dupa fiecare operatie aleasa de a se efectua, rezultatul poate fi vizualizat in tabelele respective. De exemplu, daca dorim sa stergem un produs din meniu, din meniu trebuie sa fie sterse toate produsele compuse care contin in componenta sa acel produs care trebuie sters. De asemenea, daca introducem un produs compus, pretul acestui produs trebuie sa fie suma preturilor produselor care intra in component acestui produs compus. De asemenea rezultatele mai pot fi vazute si prin generarea corecta a facturilor, prin calcularea corecta a pretului total alo comenzii si asa mai departe. In concluzie, este o aplicatie compleza cu foarte multe rezultate.

6.Concluzii:

As putea concluziona aceasta documentatie prin sustinerea faptului ca ar putea exista o multitudine de modalitati de implementare a acestei aplicatii. Interfetele grafice ar putea fi gandite intr-o multime de moduri, poate mai usor de folosit, poate mai greu de folosit sau cu functii suplimentare. Dar cert este ca prin aceasta tema am trecut prin multe concepte de programare in Java si nu numai, am lucrat cu diferite colectii, am aprofundat notiunea de serializare, dar de asemenea si a lucrului cu design-ul de proiectare MVC. A fost o tema compleza, prin care am acumulat un bagaj imens de notiunin noi legate de programarea in Java.

7.Bibliografie

* <https://docs.oracle.com/javase/7/docs/technotes/tools/windows/javadoc.html#tag>
* <https://stackoverflow.com/questions/11415160/how-to-enable-the-java-keyword-assert-in-eclipse-program-wise>
* <https://www.tutorialspoint.com/java/java_serialization.htm>
* <https://docs.oracle.com/javase/tutorial/uiswing/events/propertychangelistener.html>