**GESTIONARE DE RESTAURANT**

**--Tema 4--**

Student: Mătieși Darius-Andrei

Serie/Grupa: A/30225

Profesor laborator: Moldovan Dorin

An universitar: 2019-2020

Contents

[1.Obiectivul Temei 3](#_Toc39610298)

[2.FOLOSIRE PROGRAM! 4](#_Toc39610299)

[3.Proiectare 5](#_Toc39610300)

[4.Implementare 11](#_Toc39610301)

[5. Rezultate 12](#_Toc39610302)

[6. Posibilitati de imbunatatiri ulteriore si concluzii 15](#_Toc39610303)

[7.Bibliografie 15](#_Toc39610304)

# 1.Obiectivul Temei

1.1 Obiectivul principal

**Cerinta proiectului**

Se cere o aplicație pentru procesarea comenzilor si gestiunea unui restaurant.Sistemul trebuie să aibă tipuri de utilizatori: administrator, chelner si bucatar. De asemenea, aplicația trebuie să urmeze modelul layered arhitecture și să folosească minim următoarele clase :

•**Model classes** - the data models of the application;

•**Business Logic classes** – implement the application logic;

•**Presentation classes** – implement the user input/output;

•**Data access classes** - implement the access to the database;

Pentru simplificarea aplicatiei se vor considera un singur bucatar, un singur chelner si un singur administrator.

Obiective secundare:

|  |  |  |
| --- | --- | --- |
| Obiectiv secundar | Descriere | Capitol |
| Dezvoltare de use case-uri și scenarii | Printr-un *use-case* se înțelege o listă de acțiuni care definesc interacțiunile dintre un rol (actor) și un sistem cu scopul atingerii unui obiectiv. | 2 |
| Proiectare schemă UML | Proiectarea diagramei de clase. | 3 |
| Alegerea structurilor de date | Prezentarea structurilor de date pentru atingerea obiectivului principal. | 3 |
| Proiectare clase | Utilizarea claselor Client, Client\_to\_comanda, Comanda , Produs , Produs\_to\_comanda, pentru a genera un sistem de gestiune a comenzilor pe baza tabelelor din relațional | 3 |
| Devoltarea algoritmilor | Descrierea algoritmilor folosiți pentru realizarea gestiunii comezilor | 3 |
| Implementarea soluției | Descriere clase și metode și  conexiune relațional-program-utilizator | 4 |
| Rezultate | Prezentare a câtorva scenarii de testare | 5 |

# 2.FOLOSIRE PROGRAM!

Pentru a se face corect citirea și interpretarea datelor de intrare, pentru fiecare panel va trebui scrisa informatia in minim unul din TextBox- urile disponibile pentru preluarea datelor din Frame-ul respectiv.

Este important de precizat ca unde este precizat pretul, se asteapta o valoare de tip double, unde este precizat un nume este asteptat un String, iar unde este precizata masa la care se face comanda, cat si ID-ul comenzii pentru sters se asteapta o valoare intreaga in TextBox;

Pentru introducerea unui item composite si pentru adaugarea unui nou order, la fiecare produse se adauga intai produsele de baza, **pe rand ,** apoi se inscrie in text box numele produsului compus, respectiv numarul mesei la care se efectueaza comanda, dupa care se apasa pe butonul ADD Composite, respectiv ADD Order.

Interfetele sunt unite intr-o interfata principala, din care se ajunge pe rand la utilizatorul dorit apasand butonul, iar fiecare dintre butoanele viitoare actioneaza una din abilitatile utizilatorului curent.

Pentru generarea unei chitante, se merge in „ Vizualizare comenzi actuale ” dupa care se apasa pe Generate Bill, acest buton din urma creeand o nou fereastra in care trebuie inscris numarul comenzii pentru care sa se genereze chitanta. Id-ul comenzii este identic cu cel precizat in tabel.

Pentru adaugarea unui composite item este necesar ca ingredientele lui sa se gaseasca deja in meniu.

Mesele sunt in numar de 25 .

Maximul de elemente de adaugat in meniu sunt 25.

Pentru meniul de Edit items, se vor introduce noile valori dorite, iar pretul se va updata automat pentru un composite Item, in functie si in cooncordanta cu valoarea pretului actual al fiecarui produs adaugat in lista de ingrediente al produsului updatat. Acestea trebuie separate prin virgula si spatiu pentru a se putea face citirea in mod corect . Exemplu: „Ceapa, Sunca, Branza” .

Daca se adauga un order care contine un produs compus, atunci seful va fi alertat si va aparea pe ecran atata interfata bucatarului, cat si o notificare ca a fost facuta o comanda. La aceasta se poate raspunde cu „Yes” , iar in consola se va afisa un mesaj cum ca bucatarul s-a apucat de gatit comanda respectiva sau „No” in cazul in care s-a hotarat ca bucatarul deja gateste altceva si in acest caz se afseaza mesajul „Bucatarul este ocupat cu alta comanda” .

Pentru ca informatiile sa fie serializate cu succes, trebuie sa se iasa folosind unul dintre butoanele „Exit” puse la dispozitie, deoarece apasarea acestora vor declansa serializarea datelor cu succes in fisierul restaurant.ser. La rularea programului se face deserializarea automat.

Comanda necesară este: java –jar PT2020\_30225\_Matiesi \_Darius\_Assignment\_4.jar restaurant.ser

unde restaurant.ser-> fișierul de intrare serializat care contine informatiile din restaurant

# 3.Proiectare

**3.1 Decizii de proiectare**

Pentru aceasta aplicatie am ales sa implementez o interfata mare care se imparte mai apoi in 3 interfete mai micute si separate si anume TopGui care se separa in AdminGUI, WaiterGUI si ultimul dar nu cel din urma ChefGUI care implementeaza fiecare functionalitatile cuvenite.

A fost respectata cu desavarsire diagrama de clase prezentata in cerintele temei .

Pentru notificarea bucatarului am ales sa folosesc java. util. Observer si java. util. Observerable

Pentru prelucarea comenzilor am ales sa folosesc HashMap, iar hashCode-ul este calculate folosind variabilele corespunzatoare unei comenzi.

Pentru mentinerea comenzilor si a meniului s-au implementat in paralel si ArrayList-uri.

**3.2 Diagrama UML**

****

**3.3 Structuri de date**

Structurile de date pe care am decis să le folosesc sunt din pachetul **java.util.Array.List** folosite pentru a stoca în paralel informația din tabele mai util, fapt care a ușurat și munca la efectuarea operațiilor pe tabele. Pachetul ne permite să folosim funcții care ne ajută să accesăm și să modelăm foarte ușor termenii unei structuri de genul, precum:

* add(Object): adăugarea unui element în listă(insertLast());
* remove(Obiect): ștergerea unui element din listă;
* isEmpty(): verificare dacă lista e goală;
* getFirst(): returnează primul element din listă;
* get(int): returnează elementul de pe poziția int-ului respectiv și altele.

Pe langa acestea am mai folosit si hashMap din biblioteca **java.util.HashMap**; aceasta a fost utila pentru pastrarea comenzilor si accesarea mai rapida la acestea. Am folosit functiile:

* put(Object) pentru a adauga un element din map;
* get(Object) pentru a scoate un element din map;

**3.4. Proiectare de clase**

Acestă aplicație folosește următoarele clase:

În pachetele:

1. bll
   1. BaseProduct
      1. Clasa model abstarcta echivalentă produselor formate doar dintr-un ingredient ( produse simple )
   2. CompositeProduct
      1. Clasa model abstracta echivalenta produselor formate din mai multe ingrediente ( produse complexe sau compuse )
   3. iRestaurantProccesing
      1. Interfata echivalentă atributelor tuturor utilizatorilor din Restaurant
   4. MenuItem
      1. Clasa model echivalentă unui item obisnuit din meniu care poate ori ori simplu ori compuus
   5. Order
      1. Clasa model echivalentă pentru comanda
   6. Restaurant
      1. Clasa model echivalenta pentru simularea unui restaurant
2. dao
   1. RestaurantSer
      1. Clasa responsabilă pentru serializare si deserializarea restaurantului .
3. presentation
   1. AdminGUI
      1. Reprezinta clasa corespunzatoare simularii administratorului care poate sa modifice produselee din meniu, sa adauge altele sau sa stearga, dupa bunul plac;
   2. TopGUI
      1. Reprezintă clasa corespunzatoare interfetei care leaga cei trei utilizatori ai restaurantului
   3. ChefGUI
      1. Reprezinta clasa corespunzatoare bucatarului. Acesta este alertat de fiecare data cand se da o comanda noua la un composite Item.
   4. WaiterGUI
      1. Reprezinta clasa corespunzatoare chelnerului . Acesta poate sa preia comenzi, sa vizualizeze comenzile actuale si sa genereze bonul de masa;
4. start
   1. MainClass
      1. Clasa principală a programului, clasa MainClass din pachetul start este și clasa în care se generează triajul textului și separarea operațiilor, dar și executarea acestora prin apel cu ajutorul obiectelor de tip DAO și BLL

**3.5 Interfețe**

Proiectul a fost conceput pe baza utilizării unei anumite interfețe și doar a acesteia, și anume pe baza interfeței List care mă ajută la implementarea ArrayList-urilor necesare pentru realizarea operațiilor pe tabelele bazei de date în corcodanță .

Pe langa aceasta folosim iRestaurantProccesing , Observable si Map.

**3.6 Relații**

Aplicația conține următoarele relații:

* de agregare: Clasa „MainClass”( intreg) conține un obict de tip TopGUI(parte)
* de dependență: Clasa ChefGUI suprascrie metoda “update” corespunzătoare doar clasei respective
* de generalizare(interfete) : de exemplu Restaurant implements iRestaurantProcessing

**3.7 Algoritmi**

Pentru realizarea gestiunii comenzilor am optat pentru implementarea următorilor algoritmi:

1. **Separarea cuvintelor**- pentru acest algoritm am dorit să fac separarea folosind metodele puse la dispoziție de Regex si folosind metoda .split după un anumit tipar. Am ales sa separ String-urile după ”:” în primul rând si dupa ”, ” pentru cea de-a doua despartire, în cazul în care comanda este Delete , Insert sau Order pentru a separa corect argumente și a se efectua corect operațiile asupra tabelelor bazei de date.

2. **Printarea informatilor in fisier**- pentru acest algoritm,am declarant doua obiecte, unul de tip FileWriter si apoi unul de tip PrintWriter care se foloseste de cel din urma. Scrierea se face paragraph cu paragraph si ofera informatii despre numarul mesei, id-ul comenzii, produsele comandate si nu in ultimul rand lasa si un mesaj calduros astefel incat sa mai serveasca clientii masa la noi .

3. **Stergerea din tabel-** pentru acest algoritm am ales să folosesc Iterator pentru parcurgerea listelor tabelului de legatură cu lista de obiecte de tip Client\_to\_comanda, respective a tabelului client cu lista de obiecte de tip Client pentru a putea să o traversez și să șterg de pee a în acelali timp. Stergerea se realizează appi și din tabel, prima data din tabelul de legătura și apoi din tabelul client, datorită constrngerilor de cheie strină din baza de date.

4.**Efecturarea comenzii**- pentru acest algoritm, am ales să parcurg lista de persoane, să verific numele persoanei și pentru aceasta, dacă există, să parcurg lista de produse și pentru produsul comandat, daca există, se face o actualizare atât în tabel cît și in lista de produse a cantității acestuia, dacă cantitatea cerută este mai mica decât cea deja existentă în depozit. În acest caz, se actualizează cantitatea și după se face actualizarea prețului comenzii, se introduce în tabela order, respective in lista orderObj , apoi în cele două tabele de legătura și în final se realizează facturarea într-un document PDF.

# 4.Implementare

Clasa BaseProduct:

* descriere:clasă simulând un consumator real, componentă agestiunii comenzilor, fiind caracterizată de Id(idClient), nume(nume), adresa(adresa) ;
* variabile de instanță: idClient(int), nume(String), adresa(String);
* constructori: unul cu trei paramentri: nume, adresa, idClient;
* getters : getName () , getProducts (), getPrice() ;
* setters : setIdClient(), setIdComanda() ,setPrice(double);

Clasa CompositeProduct:

* descriere: clasă simulând un depozit al produselor fiind caracterizată de id(idProdus), nume( nume), cantitate(quantity), pret(pret);
* variabile instanță: idProdus (int), nume(String), quantity (int), pret(double);
* constructori: unul cu patru paramentri: nume, quantity, pret, idClient;
* getters : getName () , getProducts (), getPrice() ;
* setters : setIdClient(), setIdComanda() ,setPrice(double);

Clasa Order:

* descriere: clasă simulând o gestiune a restaurantului fiind caracterizată de id(idOrder), nume\_client( numeClient), numeProdus(numeProdus), cantitate(quantity), pret(pret);
* variabile instanță: id(idOrder), nume\_client( numeClient), numeProdus(numeProdus), cantitate(quantity), pret(pret);
* constructori: unul cu cinci paramentri: numeClient , numeProdus, quantity, pret, idOrder;
* getters: getIdOrder (), getPret(), getDate (), getNrMasa ();
* setters: setIdOrder (int), setPret(float), setDate (Date), setNrMasa();

Clasa MenuItem:

* descriere: clasă simulând o legătură între tabelele client și comandă fiind caracterizată de idClient și idComanda;
* variabile instanță: idClient, idComandă;
* contructori: unul cu doi paramentri: idComanda și idClient
* getters : getName () , getProducts (), getPrice() ;
* setters : setIdClient(), setIdComanda() ,setPrice(double);

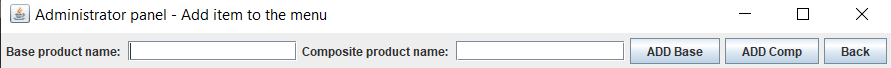
Clasa Restaurant:

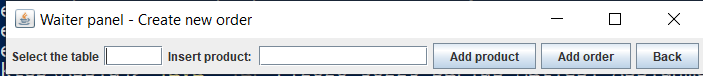
* descriere: clasă simulând o legătură între tabelele produs și comandă fiind caracterizată de idProdus și idComanda;
* variabile instanță: idProdus, idComandă;
* contructori: unul cu doi paramentri: idComanda și idProdus()
* getters : getMenu () , getMap () ;
* setters : setMenu (), setMap () ;

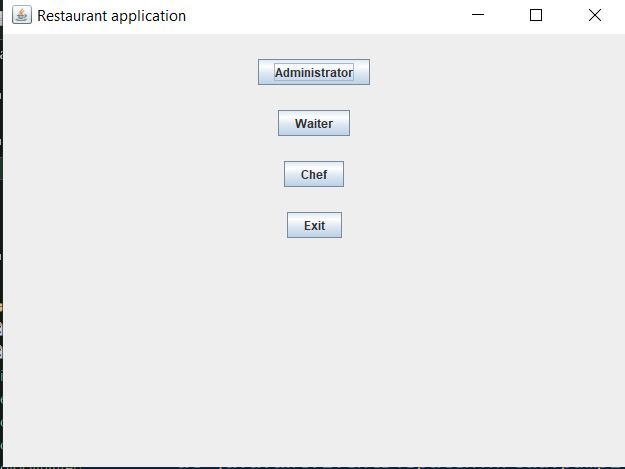
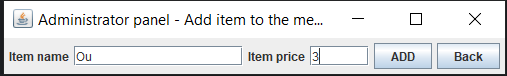
# 5. Rezultate

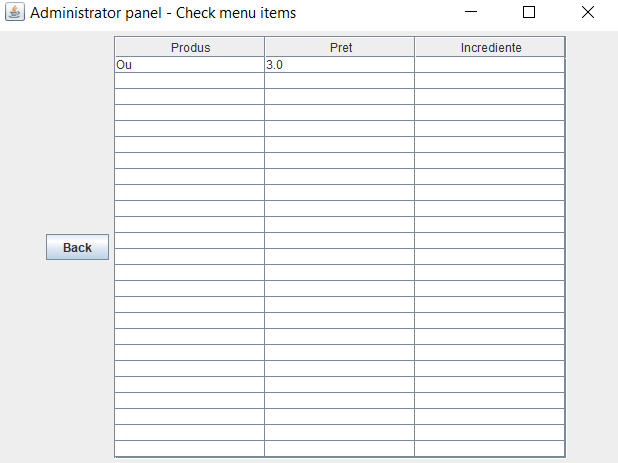
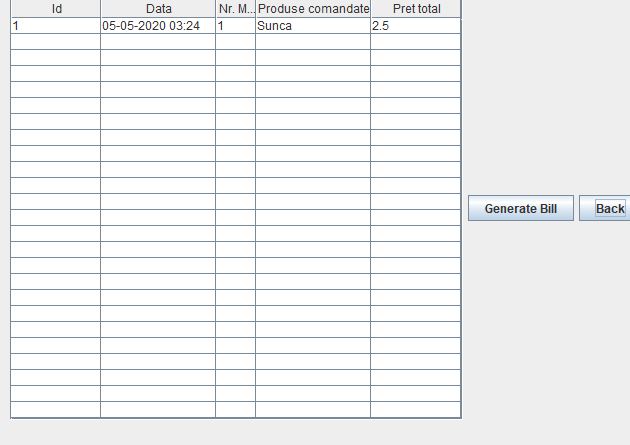
In cele ce urmeaza am oferit cateva exemple privind rezulatele

Interfetele pentru adaugarea produselor si pentru editare, dar si pentru vizulaizarea comenzilor si al meniului









# 6. Posibilitati de imbunatatiri ulteriore si concluzii

In ceea ce priveste Programare Orientata pe Obiect, aceasta tema m-a ajutat in mod desavarsit. In primul rand am inteles mai multe despre conceptele in care am fost initiat semestrul trecut si am reusit sat rec niste bariere in ceea ce inseamna pentru mine munca. Pot spune ca am invatat noi design pattern-uri, Design by Contract si Observer Design Pattern, dar si Composite Design Pattern pentru produse. A fost o munca desavarsita pentru intelegerea tuturor conceptelor si aprofundarea pe invariant si pe post si pre conditii. A fost o tema care m-a provocat la creatie, mi-a sporit imaginatia si m-a lasat sa mi exprim dorintele de fauritor in orice maniera am dorit, dar la final a iesit un proiect de care sunt mandru.

In ceea ce priveste modalitati de imbunatatire ulterioare, ar trebui precizate urmatoarele aspecte:

* Existenta a mai multor bucatari si chelneri
* Existenta unui sistem de login in sistem in functie de atributele fiecaruia
* Eventualitatea clientului de a vedea meniul cand face o comanda
* Introducerea unui mod automat in care daca bucatarul a primit o comanda si o pregateste, sa se puna intr-o lista de asteptare viitoarele comenzi, pe scurt, un sistem de monitorizare al timpului de introducere in coada a comenzilor

# 7.Bibliografie

[1] Cursul de Programare Orientată pe Obiecte 2019-2020(PPT)- Conf. Dr. Ing. T. Cioara

[2] https://stackoverflow.com/questions/2540548/how-do-i-get-eclipse-to-use-a-different-compiler-version-for-java

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

[4] <https://www.baeldung.com/reading-file-in-java>

[5] <https://www.lucidchart.com/documents#/templates?folder_id=home&browser=icon>