**DOCUMENTATIE TEMA NR.4**

RESTAURANT MANAGEMENT SYSTEM

Student: Bumbu Iulia-Diana

An: II Semestrul: 2

Grupa: 30228

CUPRINS:

1. Obiectivul temei
2. Analiza problemei
3. Proiectare
4. Implementare
5. Rezultate
6. Concluzii
7. Bibliografie
8. Obiectivul temei:

1.1.Obiectiv principal:

Principalul scop al proiectului este de a realiza o aplicatie care sa faciliteze managementul unui restaurant, sistemul avand trei utilizatori: administratorul, chelnerul si bucatarul. Pentru fiecare tip de utilizator exista o fereastra care implementeaza functionalitai diferite in functie de specificul fiecarui angajat: administratorul gestioneaza meniul, chelnerul creeaza comenzile iar bucatarul este notificat de cate ori apare o noua comanda.

1.2.Obiective secundare:

Pentru o dezvoltare optima a unei aplicatii de acest tip trebuie realizati anumiti pasi pentru indeplinirea obiectivului principal.

Principalele **obiective secundare** sunt:

* Realizarea USE CASE-urilor si a scenariilor:

O diagrama a cazurilor de utilizare ofera o descriere generala a modului in care va fi utilizat sistemul, furnizand o privire de ansamblu asupra functionalitatilor care doresc a fi oferite de sistem. (capitolul 2)

* Identificarea structurilor de date:

Structura de date este o metoda sistematica de stocare a informatiilor intr-un calculator astfel incat sa poata fi folosite in mod eficient. (capitolul 3)

* Impartirea claselor:

Pentru a gestiona mult mai usor logica aplicatiei si logica realizarii functionalitatilor dorite proiectul se imparte in mai multe layere cu functionalitati diferite, programul avand astfel o arhitectura 3-tier (capitolul 3)

* Dezvoltarea algoritmilor:

Se va determina strategia necesara pentru a gestiona functionalitatile restaurantului(capitolul 3)

* Implementarea solutiei:

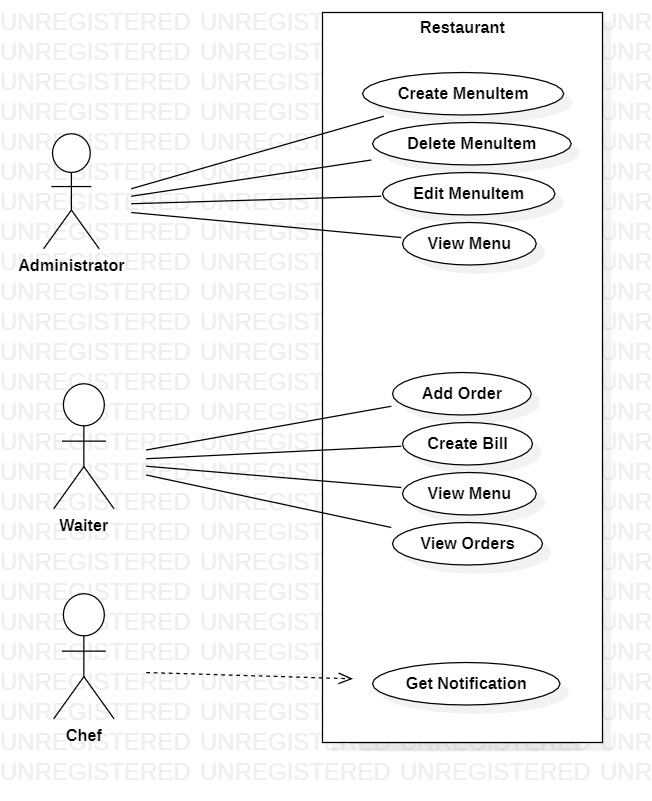
Este reprezentata de descrierea in cod a tuturor claselor anterior explicate, a metodelor necesare gestionarii meniului si comenzilor precum si procesarea de facturi sub forma de fisiere pentru a testa functionalitatea programului creat. (capitolul 4)

* Gestionarea rezultatelor:

Datele de intrare si de iesire se vor procesa folosind tehnica de serializare/deserializare (capitolul 5)

2.Analiza problemei:

2.1.USE CASE Diagram:



2.2. Scenarii:

Scenariile se diferentiaza in functie de tipul de utilizator, acestea fiind:

* Administrator:

O utilizare corecta a aplicatiei din ipostaza administratorului presupune posibilitatea adaugarii in meniu a unui nou produs, stergerii acestuia sau editarea sa. Asfel, pentru fiecare tip de operatie exista un text field aferent in care se introduc datele dorite, iar dupa apasarea butonului corespunzator acestea sunt procesate. Pentru verificarea realizarii operatiilor exista posibilitatea vizualizarii intregului meniu prin apasarea butonului “View All Menu”. Realizarea corecta a operatiilor presupune si introducerea datelor in casutele text intr-un format valid. Formatul valid se distinge in functie de tipul fiecarui produs deoarece in meniu pot exista atat produse de baza cat si produse compuse.

Formatul valid se introduce astfel in urmatoarele cazuri:

-produs de baza: **nume, pret** Exemplu: supa, 7

-produs compus care contine doar compuse de baza: **nume1, pret1;nume2, pret2** Exemplu: supa, 7;paine, 3

-produs compus care contine produs compus si produs de baza: **nume1, pret1;nume2, pret2/nume3, pret3** unde

1 si 2 formeaza un produs compus iar 3 este produs de baza Exemplu: supa, 7;paine, 3/garnitura, 8

-produs compus care contine doar produse compuse: **nume1, pret1;nume2, pret2/nume3, pret3;nume4, pret4** unde atat 1 si 2 cat si 3 si 4 sunt componente compuse ale produsului compus

Exemplu: supa, 7;paine, 3/garnitura, 8;friptura, 15

Se observa deci ca delimitatorul virgula “,” este folosit pentru a separa numele si pretul unui produs de baza, delimitatorul punct si virgula “;” este folosit pentru a separa componentele de baza ale unui produs compus,iar delimitatorul forward slash “/” este folosit pentru a delimita componentele compuse ale unui produs compus.

**ATENTIE!!!** Pentru evitarea confuziilor si introducerii datelor invalide nu se va lasa niciodata spatiu dupa caracterele punct si virgula, respectiv slash. Se poate lasa spatiu doar dupa virgula cand delimitam pretul de restul formatului. Aceasta regula este impusa deoarece daca am introduce de exemplu tiramisu, 12/ suc, 8 in meniu si ulterior am dori sa stergem produsul si in casuta de stergere scriem tiramisu, 12/suc, 8 fara spatiu dupa “/” produsul respectiv nu va fi gasit in meniu (sirul “ suc” este diferit de “suc”) si stergerea nu va realiza. In scopul de a evita astfel de situatii este interzisa lasarea spatiului dupa delimitatorii “;” si “/” pentru a ne asigura ca tot timpul denumirile sunt prelucrate in acelasi format.

Exemplu de scenariu VALID:

Pentru adaugarea elementelor in meniu se pot scrie comenzi de tipul tinand cont atat de codificarea corecta cat si de necesitatea existentei elementelor de baza pentru a crea un produs compus:

supa, 5

garnitura, 9.49

friptura, 15

gratar, 13

tiramisu, 12.5

inghetata, 10

apa, 7

suc, 8

tiramisu, 12.5;suc, 8

garnitura, 9.49;gratar, 13

supa, 5/inghetata, 10

garnitura, 9.49;gratar, 13/apa, 7

garnitura, 9.49;gratar, 13/tiramisu, 12.5;suc, 8

Pentru **a insera** un nou element in meniu se completeaza primul text field respectand formatul anterior prezentat si se apasa butonul “Add MenuItem”. Se tine cont ca un produs compus poate avea in componenta lui doar produse deja existente in meniu.

Pentru **a sterge** un element din meniu trebuie introdus in text field-ul corespunzator acelasi format in care produsul pe care vrem sa il stergem a fost introdus la inserare, dupa care se apasa butonul “Delete MenuItem”. Se presupune ca administratorul cunoaste de ce tip este fiecare item din meniul sau, insa daca exista confuzii legate de o anumita denumire sau pret al unui element al produsului pe care doreste sa il stearga se poate consulta tabelul cu toate elementele existente in meniu apasand “View All Menu” pentru a se asigura ca introduce datele intr-un format valid.

Pentru **a edita** un element din meniu trebuie introduse in text field-urile corespunzatoare acelasi format in care produsul pe care vrem sa il inlocuim a fost introdus la inserare, precum si produsul actualizat dupa care se apasa butonul “Edit MenuItem”. Daca noua valoare a produsului este de baza atunci nu sunt restrictii, insa daca este compus atunci se tine cont de regula ca un produs compus poate contine doar produse deja existente in tabel.

* Chelner:

Chelnerul are posibilitatea de a crea o noua comanda si de a genera o factura pentru comenzil create.

Pentru **a crea o noua comanda** chelnerul trebuie initial sa apese pe butonul “Refresh” pentru a actualiza meniul in cazul in care au intervenit intre timp modificari si ulterior sa introduce in casuta text numarul mesei la care se face comanda. Dupa apasarea butonului refresh intreg meniul este disponibil in fereastra iar pentru a selecta ce produse sunt adaugate la comanda se tine apasat tasta CTRL si se da click cu mouse-ul pe produsele dorite. Dupa ce acestea au fost selectate se poate apasa pe butonul “Add Order” pentru efectuarea comenzii.

Pentru **a vizualiza toate comenzile** facute se apasa butonul “View All Orders” iar acestea apar intr-un tabel.

Pentru **a genera o factura** pentru o comanda se introduce in text field-ul corespunzator id-ul comenzii pentru care se doreste sa se creeze o factura si se apasa butonul “Create Bill”. In cazul in care nu se stie exact id-ul comenzii se poate apasa butonul “View All Orders” pentru a inspecta campul IdOrder corespunzator fiecarei comenzi. Factura se va genera automat in format txt la apasarea butonului corespunzator.

* Bucatar:

Bucatarul nu are o interfata interactiva, insa acesta este notificat ori de cate ori o noua comanda este plasata, in fereastra sa aparand produsele pe care trebuie sa le gateasca!

In cazul in care formatul introducerii datelor in campurile text nu este respectat( lipseste punct si virgula “;” sau slash “/” la produse compuse / lipseste virgula “, “ la produse de baza / exista spatiu intre caractere dupa delimitatori / numele unui produs este scris diferit fata de meniu/ preturile introduse nu sunt numere intregi/reale

/ in cazul in care se comanda mai multe produse in acelasi order care nu exista individual in meniu etc. scenariul de functionare este gresit iar ca si consecinta executia programului se va incheia, fiind generate exceptii sau aparand mesaje de eroare in interfata in functie de tipul greselii. De aceea, se recomanda mare atentie la scrierea corecta a instructiunilor pentru o functionare fara impedimente.

* Proiectare:

3.1.Structuri de date:

Deoarece multipli produse/comenzi trebuie procesate in mai multe metode adesea este folosita ca structura de date ArrayList deoarece faciliteaza lucrul cu vectori prin metodele preimplementate. De exemplu:

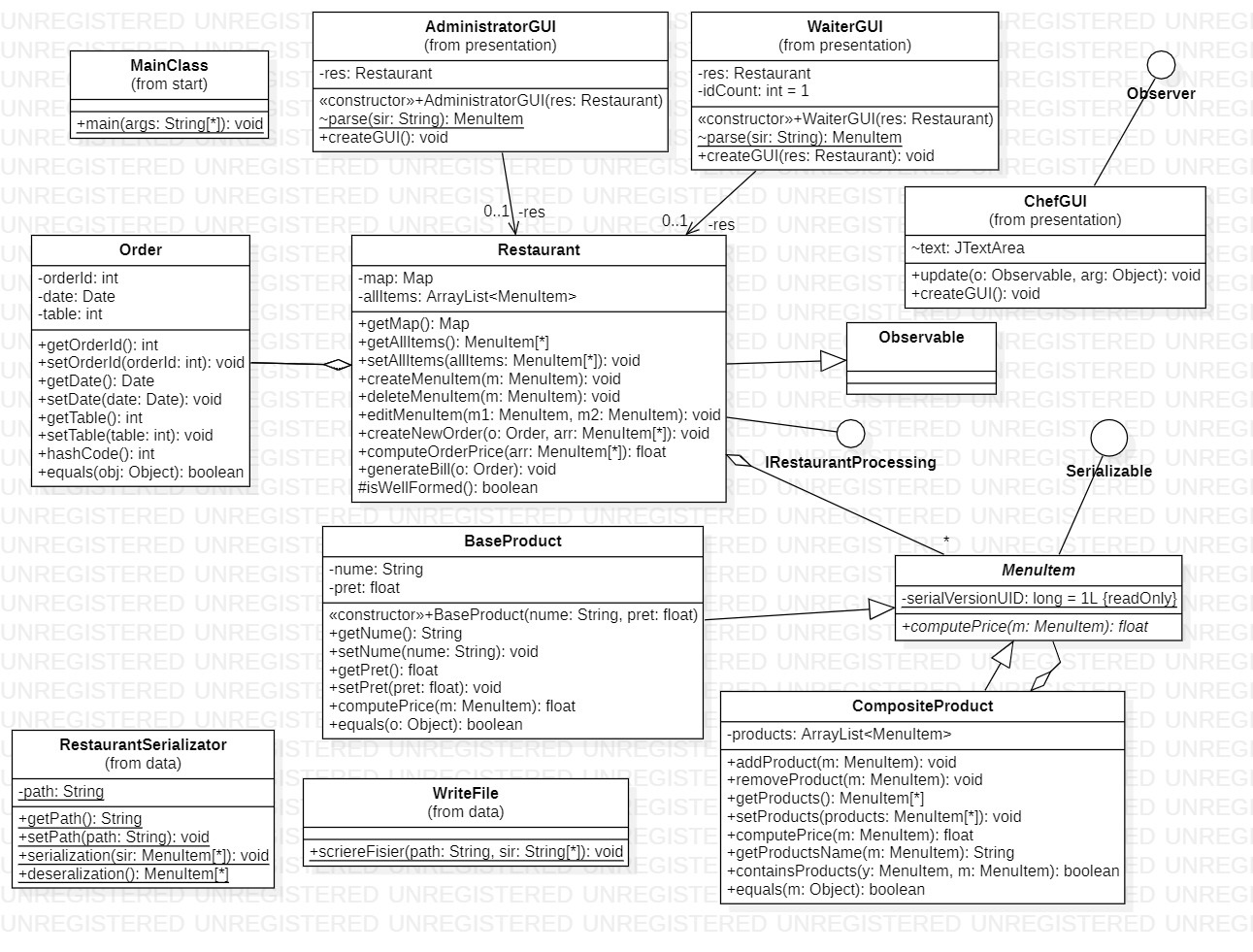
ArrayList<MenuItem> allItems= **new** ArrayList<MenuItem>();

De asemenea se individualizeaza folosirea unei structuri de tipul Map<Order, ArrayList<MenuItem>> pentru a stoca informatiile legate de fiecare comanda in clasa Restaurant. Cheia map-ului este formata din obiecte de tipul Order:

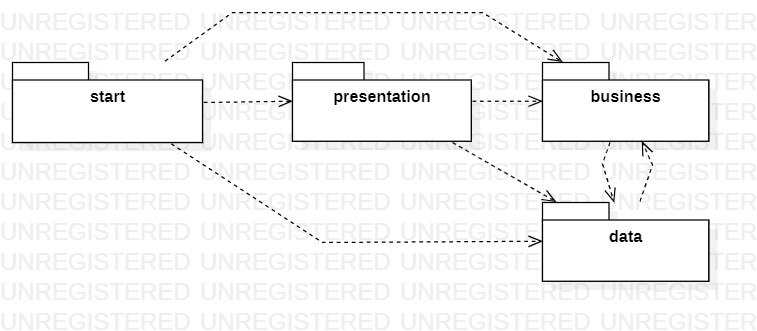
Map< Order, ArrayList<MenuItem>> map = **new** HashMap< Order, ArrayList<MenuItem>>();

3.2.Diagrama de clase

Diagrama de clase este folosita in modelarea orientata obiect pentru a descrie structura statica a sistemului si a modului in care este el structurat. De asemenea, aceasta ofera o notatie greafica pentru reprezentarea claselor (entitati care au caracteristici comune) si a relatiilor (legaturi dintre doua sau mai multe clase) dintre ele. Pentru lizibilitatea diagramei nu au fost adaugate relatiile de dependenta, insa acestea exista intre clasele: MainClass si AdministratorGUI, MainClass si WaiterGUI, MainClass si ChefGUI, MainClass si RestaurantSerializator, MainClass si Restaurant, MainClass si MenuItem, AdministratorGUI si RestaurantSerializator, AdministratorGUI si BaseProduct, AdministratorGUI si CompositeProduct, AdministratorGUI si MenuItem, WaiterGUI si Order, WaiterGUI si BaseProduct, WaiterGUI si CompositeProduct, WaiterGUI si MenuItem, Restaurant si WriteFile, Restaurant si Order, Restaurant si BaseProduct, Restaurant si CompositeProduct, RestaurantSerializator si MenuItem, IRestaurantProcessing si Order, IRestaurantProcessing si MenuItem, CompositeProduct si BaseProduct.



3.3.Diagrama de pachete:



UML furnizeaza mijloace de grupare a elementelor din cadrul diagramelor numite pachete. Pachetele constituie baza pentru controlul configuratiei si stocare, contribuind la reutilizarea codului.

3.4.Algoritmi:

In cazul acestui proiect nu au fost utilizati algoritmi preimplementati insa exista o anumita strategie in ceea ce priveste metoda HashCode din clasa Order pentru a asigura o dispersie corespunzatoare a datelor:

**function** hashCode () **is**

**//** fiecare camp din clasa se proceseaza conform regulii hashAnterior \* nrPrim + field

**hash = hash \* 13 + orderId;**

**hash = hash \* 13 + date;**

**hash = hash \* 13 + table;**

**return hash**

* Implementare:

4.1.Descrierea claselor din pachetul Business:

* Clasa MenuItem:

Clasa abstracta care implementeaza interfata Serializable pentru a putea serializa/deserializare obiectele instantiate din aceasta clasa. Este clasa parinte pentru BaseProduct si CompositeProduct structurate folosind Composite Design Pattern. Aceasta contine doar metoda abstracta

**public** **abstract** **float** computePrice(MenuItem m);

responsabila cu calculul pretului total al unui element din meniu.

* Clasa BaseProduct:

Extinde clasa MenuItem si are rolul de retine datele definitorii pentru un nou produs de baza pe care dorim sa il adaugam in meniu deoarece fiecare atribut este atribuit unui camp:

**private** String nume;

**private** **float** pret;

Se remarca existenta contructorului specific cu campuri,a metodelor get si set precum si a metodelor:

**public** **float** computePrice(MenuItem m) –implementeaza metoda abstracta din clasa extinsa returnand pretul produsului de baza

**public** **boolean** equals(Object o) -responsabila cu testarea egalitatii a doua obiecte de tip BaseProduct. Se testeaza daca obiectul trimis ca parametru este de acelasi tip si are campuri identice cu obiectul cu care il comparam

* Clasa CompositeProduct:

Clasa extinde MenuItem si are ca unic atribut sirul de produse:

**private** ArrayList<MenuItem> products = **new** ArrayList<MenuItem>();

responsabil cu pastrarea tuturor componentelor unui produs compus deoarece un produs compus poate fi format doar din produse de baza, cat si din produse de baza si alte produse compuse combinate. Se individualizeaza metodele de get si set cu ajutorul carora accesam acest atribut precum si:

**public** **void** addProduct(MenuItem m) –cu rol in adaugarea unui nou element in lista de produse a produsului compus

**public** **void** removeProduct(MenuItem m) –cu rol de stergere a unui element in lista de produse a produsului compus

**public** **float** computePrice(MenuItem m) –utilizata pentru returna pretul total al unui produs compus, prin insumarea tututor elemetelor componente

**public** String getProductsName(MenuItem m) –responsabila in generarea unui sir care contine numele respectiv cantitatea a fiecarui element constituent al listei de produse dintr-un produs compus

**public** **boolean** containsProducts(MenuItem y, MenuItem m) –verifica daca in lista de produse a elementului din meniu y exista elementul m

**public** **boolean** equals(Object m) –testeaza egalitatea a doua obiecte de tip CompositeProduct. Se testeaza daca obiectul trimis ca parametru este de acelasi tip si are aceeasi componenta ca si obiectul cu care il comparam

* Clasa Order:

Clasa contine constructorul cu campuri precum si metode get si set. Ca si atribute specifice clasei se disting:

**private** **int** orderId;

**private** Date date;

**private** **int** table;

Metodele existente in aceasta clasa sunt:

**public** **int** hashCode()–este necesara sa fie implementata deoarece Order este folosita drept cheie in cadrul map-ului din clasa Restaurant. Implementarea hashCode-ului se face conform regulii precizate in sectiunea 3.4.

**public** **boolean** equals(Object obj) -responsabila cu testarea egalitatii a doua obiecte de tip Order. Se testeaza daca obiectul trimis ca parametru este de acelasi tip si are campuri identice cu obiectul cu care il comparam.

* Interfata IRestaurantProcessing:

Interfata contine metodele createMenuItem, deleteMenuItem, editMenuItem, createNewOrder, computeOrderPrice si generateBill care urmeaza sa fie implementate in clasa Restaurant. Ca particularitate se remarca ca pentru fiecare metoda sunt specificate preconditii si postconditii conform tehnicii Design by Contract.

* Clasa Restaurant:

Clasa extinde Observable si implementeaza interfata IRestaurantProcessing iar atributele sale sunt allItems continand toate elementele existente in meniu si map responsabila cu stocarea informatiilor despre fiecare comanda si produsele asociate ei, atribute ce au metode de get si set corespunzatoare:

**private** Map< Order, ArrayList<MenuItem>> map = **new** HashMap< Order, ArrayList<MenuItem>>();

**private** ArrayList<MenuItem> allItems = **new** ArrayList<MenuItem>();

Deoarece implementeaza interfata IRestaurantProcessing clasa contine implementarea metodelor:

**public** **void** createMenuItem(MenuItem m) –adauga in lista allItems care reprezinta meniul un nou element

**public** **void** deleteMenuItem(MenuItem m) –sterge din meniul elementul trimis ca parametru, precum si toate elementele compuse care contin elementul m

**public** **void** editMenuItem(MenuItem m1, MenuItem m2 –inlocuieste elementul m1 din sir cu elementul m2 trimis ca parametru

**public** **void** createNewOrder(Order o, ArrayList<MenuItem> arr) –se adauga in map-ul restaurantului comanda o si produsele constituente trimise ca parametru arr. De asemenea, de cate ori o comanda este adaugata se notifica bucatarul apeland metodele setChanged si notifyObservers, acesta primind sirul de produse pe care trebuie sa le gateasca formatat ca si String.

**public** **float** computeOrderPrice(ArrayList<MenuItem> arr) –metoda primeste ca parametru un sir de produse al caror pret total trebuie sa il calculeze. Acesta se returneaza dupa insumarea tuturor preturilor itemelor din meniu care exista in comanda dorita

**public** **void** generateBill(Order o) –metoda este responsabila de formatarea comenzii primite ca parametru, generand un sir cu cele mai relevante date ale comenzii pe care il trimite metodei scriereFisier din WriteFile pentru a genera o factura in forma de fisier txt.

**protected** **boolean** isWellFormed()–metoda asociata invariantului clasei care atesta ca aceasta are date consistente in orice moment, indiferent de modificarea acestora prin metodele apelate

4.2. Descrierea claselor din pachetul Data:

* Clasa WriteFile:

Clasa este responsabila cu creerea unui nou fisier cu numele trimis ca si parametru “path” si scrierea in acesta a sirului de stringuri “sir”:

**public** **static** **void** scriereFisier( String path, ArrayList<String> sir)

* Clasa RestaurantSerializator:

Clasa contine ca atribut un string corespunzator fisierului de tip .ser care trebuie prelucrat:

**private** **static** String *path*;

Principala functionalitate a clasei este de a implementa metodele responsabile cu serializarea/deserializarea datelor:

**public** **static** **void** serialization(ArrayList<MenuItem> sir) –converteste obiectele din sirul trimis ca parametru intr-un byte stream si le introduce in fisierul cu numele path

**public** **static** ArrayList<MenuItem> deseralization()–byte streamul preluat din fisier este folosit pentru a se recrea in memorie obiectele anterior serializate

* 1. Descrierea claselor din pachetul Presentation:

Pentru a pastra cu fidelitate structura diagramei de clasa primite in assignment clasele responsabile cu creerea interfetei grafice au fost create unitar, fara a imparti in structura mvc. Astfel, s-au folosit ascultatori anonimi in cadrul metodelor corespunzatoare desenarii interfetei.

* Clasa AdministratorGUI:

Clasa are rolul de a implementa interfata grafica asociata utilizatorului de tip administrator si are ca parametru restaurantul al carui meniu il manipuleaza:

**private** Restaurant res;

Acesta este setat in constructorul clasei. Se individualizeaza metodele:

**static** MenuItem parse(String sir) –metoda prelucreaza sirul primit ca parametru astfel incat la final poate returna un element de tip MenuItem creat conform regulilor prezentate in sectiunea 2.2 in partea in care se specifica formatul valid. Astfel, in functie de ce delimitatori contine sirul primit ca parametru se poate instantia un produs de baza/ un produs compus care contine doar produse de baza/ un produs compus care contine atat produse de baza cat si compuse/ un produs compus care contine doar produse compuse.

**public** **void** createGUI()–in cadrul aceste metode se dezvolta intreaga logica a conceperii interfetei grafice:

Contine elemente din Java Swing folosite pentru a customiza o interfata grafica cat mai user-friendly cu scop de a face aceasta aplicatie mult mai usor de folosit. Astfel, s-au folosit urmatoarele elemente:

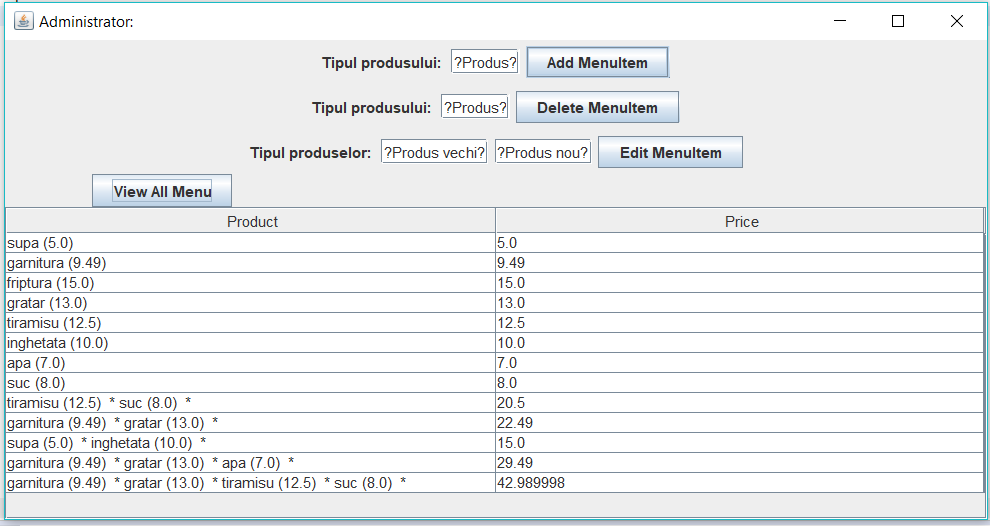
* JFrame: -rama in care adaugam toate elementele si care se inchide la comanda dispose\_on\_close. Pentru aceasta rama s-a adaugat un ascultator anonim care la inchiderea ferestrei apeleaza metoda responsabila cu serializarea datelor. Daca fereastra nu e destul de mare pentru a se vedea intreg tabelul in cazul in care s-au adaugat multe produse se poate mari tragand de marginile acesteia.
* JPanel: -s-au instantiat patru panouri care au servit la gruparea mult mai usoara a componentelor, precum si a unui panou principal care le include pe celelalte folosind modul de organizare BoxLayout
* JLabel: -exista trei etichete pentru a indica locul in care se introduc datele despre produse
* JTextField: -cele patru spatii in care se introduc de la tastatura produsele in formatul valid anterior specificat
* JTable, JScrollPane: -folosita pentru a afisa datele despre meniu in forma de tabel
* JButton: -exista patru butoane pentru fiecare operatie specifica administratului, la apasarea acestora meniul fiind manipulat. Procesarea evenimentelor aparute la apasara acestora se face in cadrul ascultatorilor anonimi:

b1.addActionListener–buton responsabil cu procesarea evenimentului la adaugarea unui nou obiect in meniu. Se preia textul din casuta text corespunzatoare si se apeleaza metoda parse generand un MenuItem. Se verifica daca acest obiect nu exista deja in meniu si ca are in componenta lui doar produse deja existente in cazul in care este produs compus. Daca aceste conditii sunt indeplinite se poate realiza inserarea asa ca se apeleaza metoda createMenuItem din Restaurant, iar in caz contrar sunt generate niste mesaje corespunzatoare

b2.addActionListener– buton responsabil cu procesarea evenimentului la stergerea unui obiect din meniu. Se preia textul din casuta text corespunzatoare si se apeleaza metoda parse generand un MenuItem. Obiectul rezultat este trimis ca parametru metodei deleteMenuItem din Restaurant

b3.addActionListener– buton responsabil cu procesarea evenimentului la editarea unui obiect din meniu. Se preia textul din casutele text corespunzatoare si se apeleaza metoda parse generand doua MenuItem: unul vechi pe care dorim sa il inlocuim si unul nou pe care il dorim in meniu. Inainte de a se apela metoda editMenuItem din Restaurant se verifica daca produsul vechi inserat exista in meniu, daca produsul nou are cumva duplicat in meniu sau daca produsul nou este compus si nu contine doar elemente existente in meniu. In aceste cazuri se va genera un mesaj de eroare, altfel se va realiza editarea. Se observa ca aceasta functionalitate este extrem de permisiva deoarece putem inlocui si produse de baza cu produse compuse si viceversa, eliminand restrictii de tip.

b4.addActionListener–buton responsabil cu procesarea evenimentului de afisare a intregului meniu in format de tabel prin adaugarea unei noi linii pentru fiecare element din sirul de MenuItem din Restaurant



* Clasa WaiterGUI:

Clasa are rolul de a implementa interfata grafica asociata utilizatorului de tip chelner si are ca parametru restaurantul al carui meniu si map de comenzi il manipuleaza precum si un idCount care se incrementeaza la fiecare noua comanda efectuata:

**private** Restaurant res;

**private** **int** idCount = 1;

Acesta este setat in constructorul clasei. Se individualizeaza metodele:

**static** MenuItem parse(String sir) –metoda prelucreaza sirul primit ca parametru astfel incat la final poate returna un element de tip MenuItem creat conform regulilor prezentate in sectiunea 2.2 in partea in care se specifica formatul valid. Astfel, in functie de ce delimitatori contine sirul primit ca parametru se poate instantia un produs de baza/ un produs compus care contine doar produse de baza/ un produs compus care contine atat produse de baza cat si compuse/ un produs compus care contine doar produse compuse.

**public** **void** createGUI()–in cadrul aceste metode se dezvolta intreaga logica a conceperii interfetei grafice:

Contine elemente din Java Swing folosite pentru a customiza o interfata grafica cat mai user-friendly cu scop de a face aceasta aplicatie mult mai usor de folosit. Astfel, s-au folosit urmatoarele elemente:

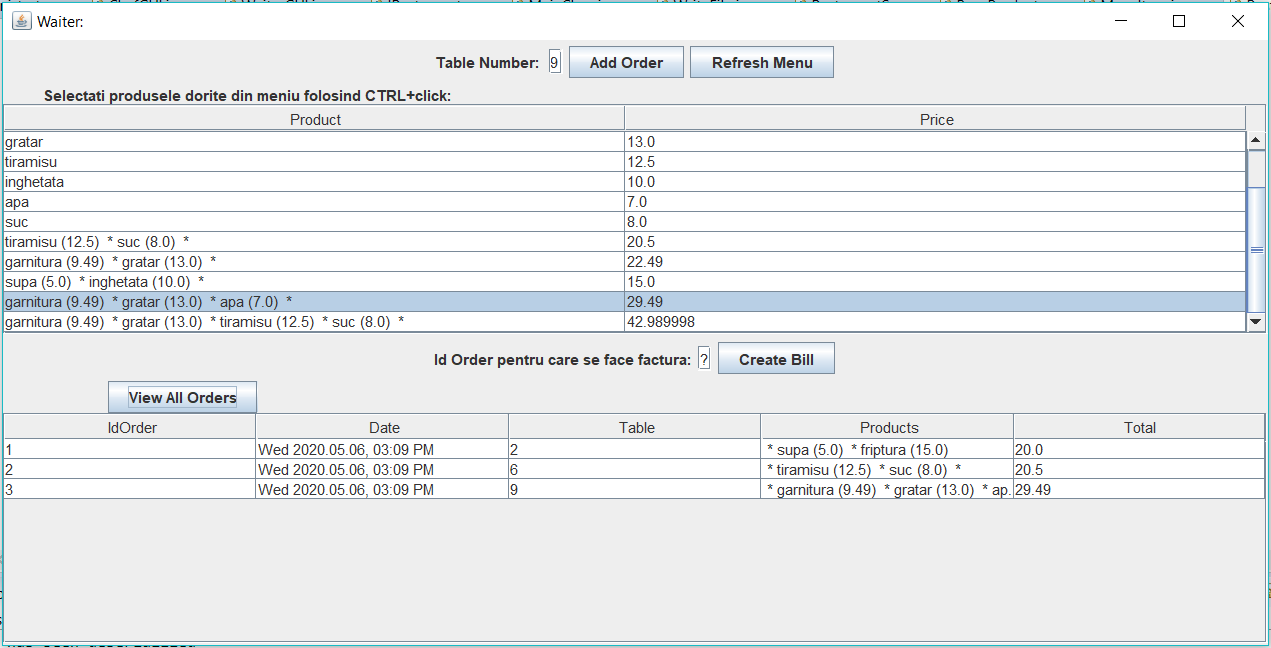
* JFrame: -rama in care adaugam toate elementele si care se inchide la comanda dispose\_on\_close. . Daca fereastra nu e destul de mare pentru a se vedea intreg tabelul in cazul in care s-au adaugat multe comenzi se poate mari tragand de marginile acesteia.
* JPanel: -s-au instantiat patru panouri care au servit la gruparea mult mai usoara a componentelor, precum si a unui panou principal care le include pe celelalte folosind modul de organizare BoxLayout
* JLabel: -exista trei etichete pentru a indica care e campul pentru numarul mesei la care se face comanda, cum se realizeaza selectia elementelor din meniu si unde este campul pentru id-ul comenzii pentru care generam factura
* JTextField: -cele doua casute text in care se introduce de la tastatura numarul mesei si id-ul comenzii
* JTable, JScrollPane: -folosite pentru a afisa datele despre meniu si toate comenzile efectuate in forma de tabel
* JButton: -exista patru butoane pentru fiecare operatie specifica chelnerului, la apasarea acestora meniul fiind manipulat. Procesarea evenimentelor aparute la apasara acestora se face in cadrul ascultatorilor anonimi:

b1.addActionListener– buton responsabil cu procesarea evenimentului de creare de comanda. Fiecare element selectat din meniul in format de tabel afisat dupa apasarea butonului de Refresh este adaugat in lista de produse a noii comenzi. Masa acesteia este preluat din text field-ul corespunzator, dupa care se apeleaza metoda createNewOrder din Restaurant. Id-ul unei comenzi este unic deoarece dupa fiecare comanda efectuata acesta este incrementat

bt1.addActionListener– buton responsabil cu procesarea evenimentului de refresh al meniului, facand ca tabelul sa apara in format de tabel cu date actualizate in cazul in care intre timp administratorul a efectuat modificari

bt2.addActionListener– buton responsabil cu procesarea evenimentului de generare a facturii pentru comanda cu id introdus in casuta text corespunzatoare prin apelarea metodei generateBill din Restaurant

bt3.addActionListener– buton responsabil cu procesarea evenimentului de afisare a tuturor comenzilor efectuate in forma de tabel



* Clasa ChefGUI:

Clasa are rolul de a implementa interfata grafica asociata utilizatorului de tip bucatar. Acesta este notificat ori de cate ori o comanda este plasata, de aceea clasa implementeaza interfata Observer. Metodele existente in cadrul acestei clase sunt:

**public** **void** update(~~Observable~~ o, Object arg)–metoda asigura notificarea bucatarului prin afisarea in textArea a produselor pe care trebuie sa le gateasca

**public** **void** createGUI()–in cadrul aceste metode se dezvolta intreaga logica a conceperii interfetei grafice:

Contine elemente din Java Swing folosite pentru a customiza o interfata grafica cat mai user-friendly cu scop de a face aceasta aplicatie mult mai usor de folosit. Astfel, s-au folosit urmatoarele elemente:

* JFrame: -rama in care adaugam toate elementele si care se inchide la comanda dispose\_on\_close
* JPanel: -s-au instantiat un panou care a servit la gruparea mult mai usoara a componentelor folosind modul de organizare BoxLayout
* JLabel: -exista o eticheta care indica ca bucatarul este notificat
* JTextArea: - folosita pentru a scrie produsele in spatiul text existent in cadrul ferestrei



* 1. Descrierea claselor din pachetul Start:
* Clasa MainClass:

**public** **static** **void** main(String[] args) –primeste ca argument calea fisierului care trebuie deserializat. Se apeleaza de asemenea metodele responsabile cu generearea interfetei grafice pentru fiecare tip de utilizator

1. Rezultate:

Pentru realizarea verificarii rezultatelor programul este configurat sa ruleze din terminal si sa deserializeze un fisier primit ca parametru. In acest sens s-a configurat un fisier .jar. La inchiderea aplicatiei noile date din meniu sunt serializate. Corectitudinea rezultatelor se poate verifica prin analizarea facturilor generate (in cazul in care utilizatorul de tip chelner a facut acest lucru).

.

1. Concluzii:

In concluzie, aceasta tema mi-a oferit oportunitatea de a invata cum sa folosesc serializarea/deserializarea si de a folosi noi patternuri. Astfel, am sedimentat o parte din conceptele de baza ale lucrului cu acestea si am aprofundat alte cunostinte in legatura cu POO . Proiectul poate avea alte dezvoltari ulterioare prin adaugarea de noi functionalitati pentru fiecare tip de utilizator sau prin imbunatatirea aspectului interfetei grafice.

7. Bibliografie:

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

2. https://www.geeksforgeeks.org/serialization-in-java/

3.https://docs.oracle.com/javase/7/docs/technotes/tools/windows/javadoc.html#tag