# DOCUMENTATIE TEMA 4: SISTEM DE MANAGEMENT

# AL UNUI RESTAURANT

# AGHENITEI BIANCA

CALCULATOARE ROMANA GRUPA 30225



# Obiectivul temei

**Tema vizeaza proiectarea și implementarea unei aplicații ce simuleaza magagementul unui restaurant.**

Sistemul ar trebui să aibă trei tipuri de utilizatori: administrator, ospătar și bucătar. Administratorul poate adăuga, șterge și modifica produsele existente din meniu. Chelnerul poate crea o nouă comandă pentru un tabel, adăugând elemente din meniu și calcula factura

Sistemul ar trebui să aibă trei tipuri de utilizatori: administrator, ospătar și bucătar.

**Obiective secundare:**

1. Definim interfața IRestaurantProcessing care conține principalele operații care pot fi executate de chelner / administrator, după cum urmează:

- Administrator: poate sa creeze un articol de meniu nou, sa stearga elementul de meniu, sa editeze elementul de meniu;

- Ospătar: poate sa creeze o nouă comandă sa calculeze prețul pentru o comandă, sa generze factura în format .txt

2. Definim și implementam clasele:

- Utilizam modelul de design compus pentru definirea claselor MenuItem, BaseProduct și CompositeProduct;

- Utilizam modelul de proiectare a observatorului pentru a anunța bucătarul de fiecare dată când se adaugă o nouă comandă care conține un produs compozit;

3. Implementam clasa Restaurant folosind o colecție JCF predefinită care se bazează pe o

structura de date hashtable. Cheia hashtable va fi generată pe baza comenzii clasei, care poate avea asociate mai multe meniuri de meniu. Folosim JTable pentru a afișa informațiile legate de restaurant;

- Definim o structură de tip Hartă <Comandă, Colecție <MenuItem>> pentru stocarea informațiilor referitoare la comandă în clasa Restaurant. Cheia hărții va fi formată din obiecte de tip Ordine, pentru care se va suprascrie metoda hashCode () pentru a calcula valoarea hash din hartă din atributele Comanda (OrderID, data etc.).

- Definim o colecție adecvată formată din obiecte MenuItem pentru a stoca meniul restaurantului;

- Implementam clasa Restaurant utilizând metoda Design după contract (implicând

condiții pre, post, invariante și asserturi);

4. Elementele de meniu pentru popularea obiectului Restaurant vor fi încărcate / salvate dintr-un fișier folosind Serialization;

5. Va fi generat un fisier jar;

6. Va fi generat un javadoc;

# Analiza problemei, modelare, scenarii, cazuri de utilizare

Problema impune realizarea unei interfete grafice pe care o vom crea cu ajutorul JavaSwing care ofera obiectele JTable, cu care va fi usor sa lucram pentru a afisa meniul si comenzi fiindca se pot stoca obiecte de orice fel si se pot selecta obiecte din tabel.

O alta problema o reprezinta impartirea permisiunilor fiecarui utilizator. Pentru aceasta fiecare tip de utilizator va avea propria fereastra, lucrul pe ele fiind facut in mod structurat cu arhitectura MVC. Operatiile specifice fiecraui utilizator vor fi definite in interfete specfice, MVC\_Modelul fiecaruia trebuind sa le implementeze.

**Cazuri de utilizare:**

La rularea aplicatie se vor deschide implicit cele 3 ferestre specifice fiecarui tip de utilizator. In fereastra ADMIN, un utilizator poate vedea toate itemele din meniul restaurantului. De asemenea, el are la dispozitie campuri in care poate sa intruduca noi elemente in meniu, atat simple cat si compuse. Pentru adaugarea unui produs compus nu se vor adauga initial componentele sale, acestea pund fi editate ulterior. Toate modificarile asupra meniului se reflecta instantaneu si in interfata grafic, atat in fereastra ADMIN, cat si in WAITER , tabelele trecand prin cicluri de improspatare.

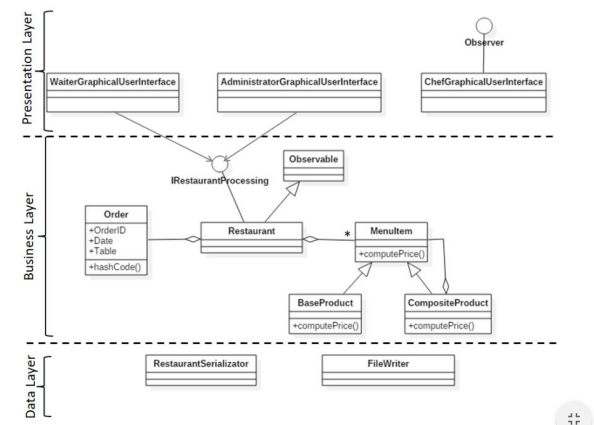
In fereastra WAITER, un utilizator poate crea o comanda noua si genera o factura. Utilizatorul selecteaza itemii doriti afisati intr-un tabel dand click pe ele. La click, elementele dorite sunt copiate intr-un nou tabel, iar cand ospatarul este satisfacut apasa pe butonul Place Order. Daca se doreste generarea unei fcaturi in format txt utilizatorul bifeaza casuta „Bill”.

Fereastra CHEF se deschide initial cu mesjaul „Waiting for order”. Ea soloseste tehnica Observable pentru a fi informata de crearea unei noi comezi. Cand un utilizatorul apasa butonul Create Order din WAITER, in interfata CHEF apare un mesaj cu produsele ce fac parte din comanda, schimbarea fcaandu-se instantaneu.

In cazul in care calea fisierului de intrare este invalida, programul va arunca o exceptie care va fi semnalata in terminal. De asemenea, el nu va rula daca argumentele nu sunt trimise corect. Formatul apelului corect este:

*java -jar PT2020\_Group\_FirstName\_LastName\_Assignment\_4.jar restaurant.ser*

# 3.Proiectare (decizii de proiectare, diagrame UML, structuri de date, proiectare clase, interfete, relatii, packages, algoritmi, interfata utilizator)

****

Am incercat sa respect cat mai intocmai diagrama UML prezentata in cerinta. Cu toate acestea, am considerat necesare anumite modificari:

- pentru a diferentia intre operatiile pe care le poate executa Waiterul/Adminul am creat doua interefete IRestaurantProcessing, cate una specifica fiecaruia. Astfel, IRestaurantProcessingWaiter permite crearea unei facturi, iar IRestaurantProcessing permite modificarea meniului.

-intrucat interfetele Observable si Observer erau depricated, am mimat folosirea lor, implementand prectic metodele cu acealasi functionalitati;

-am implementat clase MVC atat pentru Waiter, cat si pentru Admin, pentru o mai buna gestiune a interfetei grafice;

-nu am mai implementat clasa Restaurant Serializator fiindca am gasit foarte facila realizarea serializarii in Main;

Am considerat utila folosirea arhitecturii Layered intrucat eram familiara cu ea din proiectele anterioare. De aceea, am impartit proiectul in 3 pachete principale:

-Presentation Layer: se ocupa de parte vizuala, interfata cu utilizatorul;

-Bussiness layer: se ocupa cu gestiunea operatiilor pentru a obtine rezultatul dorit;

-Data Layer: se ocupa cu manipularea datelor „raw”;

Pentru stocarea datelor importante si voluminoase, precum Orderurile si meniul am optat pentru folosirea unor colectii predefinite JFC, dupa caz.

Comenzile le-am stocat intr-o tabela de dispersie, hash code-ul fiind generat dintr-o combinatie a caracteristicilor comenzii. Astefel, in clasa Order am suprascris metoda hashCode.

„ A HashMap  store items in "key/value" pairs, and you can access them by an index of another type (e.g. a String).

One object is used as a key (index) to another object (value). It can store different types: String keys and Integer values, or the same type, like: String keys and String values”

Pentru a stoca elementele din meniu am ales un ArrayList fiindca elementele sunt usor de accesat dupa index. Pentru aceasta am avut grija sa folosesc metoda „Design by contract”, care presupune verificarea unor conditii de validitate la inceputul, sfarsitul si in timpul operarii pe ArrayList. Astfel, am definit metoda boolean wellDefined() care este apelata inainte si dupa apelul unei metode care ar putea modifica ArrayList. In cazul in care ArrayListul a fost modificat in mod ilegal, acest lucru se va semnala cu ajutorul unui assert.

Triunghiul MenuItem-BaseProduct-CompositeProduct presupune folosirea design patternului „Composite”, conform caruia un tip de obiect poate fi de baza sau compus din mai multe obiecte de baza;

*„In*[*software engineering*](https://en.wikipedia.org/wiki/Software_engineering)*, the****composite pattern****is a partitioning*[*design pattern*](https://en.wikipedia.org/wiki/Design_pattern_(computer_science))*. The composite pattern describes a group of objects that are treated the same way as a single instance of the same type of object. The intent of a composite is to "compose" objects into tree structures to represent part-whole hierarchies. Implementing the composite pattern lets clients treat individual objects and compositions uniformly.*[*[*](https://en.wikipedia.org/wiki/Composite_pattern#cite_note-GangOfFour-1)*„*

# Rezultate

Am resuti sa implementeez schema prezentata in specificatii cu mici modificaari e care le-am considerat necesare. .Rezultatele sunt cele asteptate, lucru valabil pentru toate cele 3 interfete. Astfel, un Administrator poate sa modifice elementele din meniu, iar consecintele apar instantaneu in toate interfetele. De asemenea un Waiter poate genera un order nou, avand posibilitatea de a alege elementele dorite dintr-un tabel printr-un simplu click. Orice schimbare facuta in interfata administartorului este facuta si in cea a ospatarului, astfel ca un ospatar poate adauga la comanda un element proaspat introdus in meniu. La apasarea butonul de plasare a comenzii, Cheful este atentionat si o factura este generata.

De asemenea, am avut succes cu serializarea datelor intr-un fisier .ser. Am realizat in pralabil acest lucru in main, codul fiind comentat. Intrucat aveam o eroare a fost nevoie sa precizez serialVersionUID la inceputul fiecarei clase care a fost serializata. In main a ramas doar codul pentru recuperarea datelor din fisierul ser, care se transpun in interfata grafica la rulare.

# Concluzii

In urma finalizarii acestui proiect consider ca am fixat unele dintre notiunile programarii orientate pe obiect, m-am familiarizat cu lucrul cu obiecte si interfete si am fost introdusa unor notiuni cu totul noi pentru mine, precum serializarea, Design Patternul Observer, Tehnica Design By Contract, Desing Paternul Composite. Cu singuranta a fost cea mai voluminoasa tema din punctul de vedere al informatiilor noi, dar ma bucur ca am reusit sa o duc la capat, abordand toate notiunile.

Desi la crearea claselor si a metodelor nu am intalnit probleme, lucrul cu Design Patternuri mi-a dat de furca, fiind o notiune cu totul noua. Din fericire am avut prezentarea powerpoint de pe site-ul domnului profesor pentru informatii necesare si indicii referitoare la crearea claselor, astefel ca am reusit sa rezolv treptat toate problemele si nelamuririle pe care le aveam initial.

Din pacate timpul nu mi-a permis sa adaug mai mult proiectului, insa m-am gandit la mai multe posibilitati de imbunatatire ulterioara a lui:

* folosirea unui regex pentru a verifica datele introduse in TextFields;
* gasirea unui mod mai elegant de a introduce elemente in meniu. Poate o solutie sa nu trebuiasca introduse separat BaseProduct si CompositeProduct;
* adaugarea unei ferestre de login;

- tratarea explicita a cazurilor in care datele de intrare sunt incorecte si notificarea utilizatorului;

-tratarea mai multor exceptii, atat la calcul, cat si la introducerea datelor;

-permiterea introducerii datelor cu coeficienti reali;

-imbunatatiri la interfata grafica: imagini, culori;

-adaugarea unei melodii pe fundal;

In concluzie, pot spune ca m-am distrat realizand acest proiect si sunt multumita cu rezultatul final. Sunt fericita ca am fixat niste notiuni importante de OOP, care ma vor ajuta la dezvoltarea in viitor a unor proiecte mai complexe. Lucrul cu design Patternuri mi se pare foarte facil, mai ales cu Composite, deoarece permite realizarea unei mici recursivitati pentru a afisa sau calcula anumite caracteristici ale obiectelor. Cu siguranta m-a intrugat aceasta parte si voi cauta sa o mai folosesc ulterior.