Documentatie Tema 4

Restaurant Management System

George Adam

Grupa 30225

Profesor Laborator: Dorin Moldovan

[1. Obiectivul temei 3](#_Toc36579405)

[2. Analiza problemei, modelare, scenarii, cazuri de utilizare 3](#_Toc36579406)

[3. Proiectare 4](#_Toc36579407)

[4. Implementare 8](#_Toc36579408)

[5. Rezultate 13](#_Toc36579409)

[6. Concluzii 13](#_Toc36579410)

[7. Bibliografie 14](#_Toc36579411)

Obiectivul temei

Scopul principal al acestei teme este de a implementa o aplicatie de management al unu restaurant. Astfel, aceasta include diferite operatii pentru a gestiona activitatea fiecarui tip de angajat. Fiecare dintre acestia are atribuite cateva operatii precum: editarea produselor din meniu, adaugarea produselor in meniu, crearea de noi comenzi, vizulizarea comenzilor existente sau vizualizarea meniului cu tot ce contine. Administratorul poate salva oricand situatia comenzilor si a meniului intr-un fisier, astfel se asigura persistenta datelor in cazul situatiilor neprevazute. Totodata, in interfata grafica sunt implementate mecanisme de vizualizare a comenzilor si a meniului pentru a usura munca angajatilor si a verifica corectitudinea.

|  |  |  |
| --- | --- | --- |
| Obiectiv secundar | Descriere | Capitol |
| Ease of use | Datorita implementarii unui GUI pentru utilizatorii aplicatiei, acestia o pot folosi in mod intuitiv, aplicatia indeplinindu-si rolul de a ajuta, nu a incurca. | 3 |
| Generare | Se pot genera bonuri pentru orice comanda existenta in momentul respectiv in restaurant, fiind nevoide doar de numarul acesteia. | 3 |
| Corectitudinea | Aplicatia permite vizualizarea informatiilor despre comenzi si meniu in interfata grafica. | 3 |

Analiza problemei, modelare, scenarii, cazuri de utilizare

* Cerinte functionale.
* Implementarea interfetei IRestaurantProcessing folosing Design by contract.
* Generarea unui model de stocare a produselor de baza si compuse din meniu.
* Crearea de comenzi folosind produse din meniu.
* Persistenta datelor, acestea fiind serializate in orice moment de timp la alegre.
* Vizualizarea datelor in GUI de catre angajatii cu acest drept.
* Generare de rapoarte pentru comenzi sub forma de fisier .txt.
* Implementarea unui GUI pentru a usura folosirea aplicatiei.
* Impartirea operatiilor permise pe pozitia fiecarui angajat.
* Use-cases

|  |  |
| --- | --- |
| Use-case | Solutie |
| Interfata IRestaurantProcessing | Interfata defineste functiile pe care clasa Restaurant trebuie sa le implementeze. Folosind Design by contract, fiecare functie are anumite conditii care trebuie indeplinite la inceperea executiei acesteia, si conditii pentru rezultatele sale. Toate acestea trebuie verificate de catre cel care implementeaza functiile. |
| Definirea de clase pentru produse. | Se creaza o clasa abstracta cu functii pentru urmatoarele doua clase: BaseProduct si CompositeProduct. Folosing polimorfismul si pattern-ul Composite Design se maximeaza nivelul de reutilizare a codului. |
| Alegerea structurilor de date | Se folosesc structuri de date precumm liste si map-uri pentru a stoca datele despre restaurant si cele necesare realizarii functiilor. |
| Afisarea in fisier | Rapoartele despre comenzi sunt generate din interfata grafica, apoi sunt salvate cu nume sugestiv in folderu unde se afla aplicatia. |
| Dezvoltarea algoritmilor | Se folosesc algoritmi de parsing. |
| Impartirea pe clase | Se creeaza clase pentru impartirea functionalitatii in pachete specifice. Se foloseste Composite Design Pattern, Design by Contract si Interfata Observer. |
| Implementarea solutiei | Se genereaza in faza initiala o diagrama UML pentru a avea structura proiectului, apoi se incepe implementarea propriu-zisa. |
| Testare | Se realizeaza prin verificarea informatiilor afisate in GUI sau a fisierelor generate pentru comenzi. |

Proiectare

* Decizii de proiectare

Proiectul este impartit in urmatoarele clase: MenuItem, BaseProduct, CompositeProduct, Order, Restaurant, FileWriting, Serializer, AdministratorGUI, ChefGUI, WaiterGUI, StartGUI si App. S-a ales aceasta impartirea pentru a separe generarea de fisiere, operatiile pe datele din restaurant si dezvolatea interfetei grafice.

Clasa MenuItem este o clasa abstracta care defineste functiile care trebuie implementate de BaseProduct si CompositeProduct. Acestea trei au fost realizaet utilizand Composite Design Pattern.

Clasa BaseProduct reprezinta un produs simplu al restaurantului, care poate intra in componenta unui CompositeProduct.

Clasa CompositeProduct reprezinta un produs compus, cum ar fi “Supa de pui” in lumea reala. Acestea compun meniul restaurantului.

Clasa Order reprezinta o comanda. Aceasta contine masa la care se face comanda, ID-ul comenzii si data si ora la care se creeaza.

Clasa Restaurant implementeaza interfata IRestaurantProcessing care este creata sub standardul Design by contract. Aceasta defineste functii pe care clasa Restaurant trebuie sa le implementeze si sa verifice conditiile specificate de interfata. Aceasta clasa contine un obiect de tipul Map pentru a stoca comenzile si lista de produse continute de comanda pe baza hashcode-ului obiectului de tip order, dar si o lista care reprezinta meniul restaurantului.

Clasa FileWriting implementeaza o metoda care preia informatiile de la obiectele din clasa Restaurant si printeaza intr-un fisier informatii despre o comanda specificata in interfata grafica.

Clasa Serializer serializeaza clasa Restaurant intr-un fisier .ser pentru a asigura persistenta datelor. Totodata, cand este rulata aplicatia, informatiile din acest fisier sunt deserializate.

Clasa AdministratorGUI implementeaza interfata grafica pentru administratorul restaurantului.

Clasa StartGUI implementeaza interfata grafica pentru a le porni pe celelalte.

Clasa ChefGUI implementeaza interfata grafica pentru bucatarul restaurantului si foloseste interfata Observer pentru a fi notificat de fiecare data

Clasa WaiterGUI implementeaza interfata grafica pentru osapatarii restaurantului.

Clasa App este clasa care contine functia Main a aplicatiei.

* Structuri de date

Structurile de date folosite sunt:

* List: Stocare informatii pentru parsat sau produse.
* ArrayList: Stocare de informatii despre produse.
* String: folosit la informatiile provenite din GUI si detalii despre produse.
* Integer/Double: pentru a salva preturile si stocurile produselor din baza de date.
* Map: pentru a salva comenzile cu lista de produse din ele.
* Proiectare clase

Clasele sunt:

* Order: clasa care defineste informatiile despre o comanda.
* Restaurant: clasa care contine comenzile si meniul si implementeaza functii pentru acestea.
* FileWriting: clasa pentru a genera fisiere cu informatii despre comenzi.
* Serializer: clasa pentru serializarea datelor, asigurand persistenta acestora.
* AdministratorGUI: clasa pentru dezvolarea GUI-ului pentru administrator.
* ChefGUI: clasa pentru dezvoltarea GUI-ului pentru Chef.
* WaiterGUI: clasa pentru dezvoltarea GUI-ului pentru Waiter.
* StartGUI: clasa pentru a porni celelalte GUI-uri.
* App: Clasa care contine functia main.
* Algoritmi
* Parsing:

Parseaza informatiile introduse pentru a crea, edita, sterge comenzi sau produse.

* Apeluri:

Se apeleqaza metode cu informatii transmise prin intermediul GUI.

* Verificarea operatiilor:

Se pot afisa in interfata grafica informatiile despre Restaurant.

* Scrierea in fisier:

Se genereaza un fisier .txt care contine informatii despre o anumita comanda.

* Interfata Grafica

Interfata grafica este impartita 4 componente: Start, Waiter, Administrator si Chef.

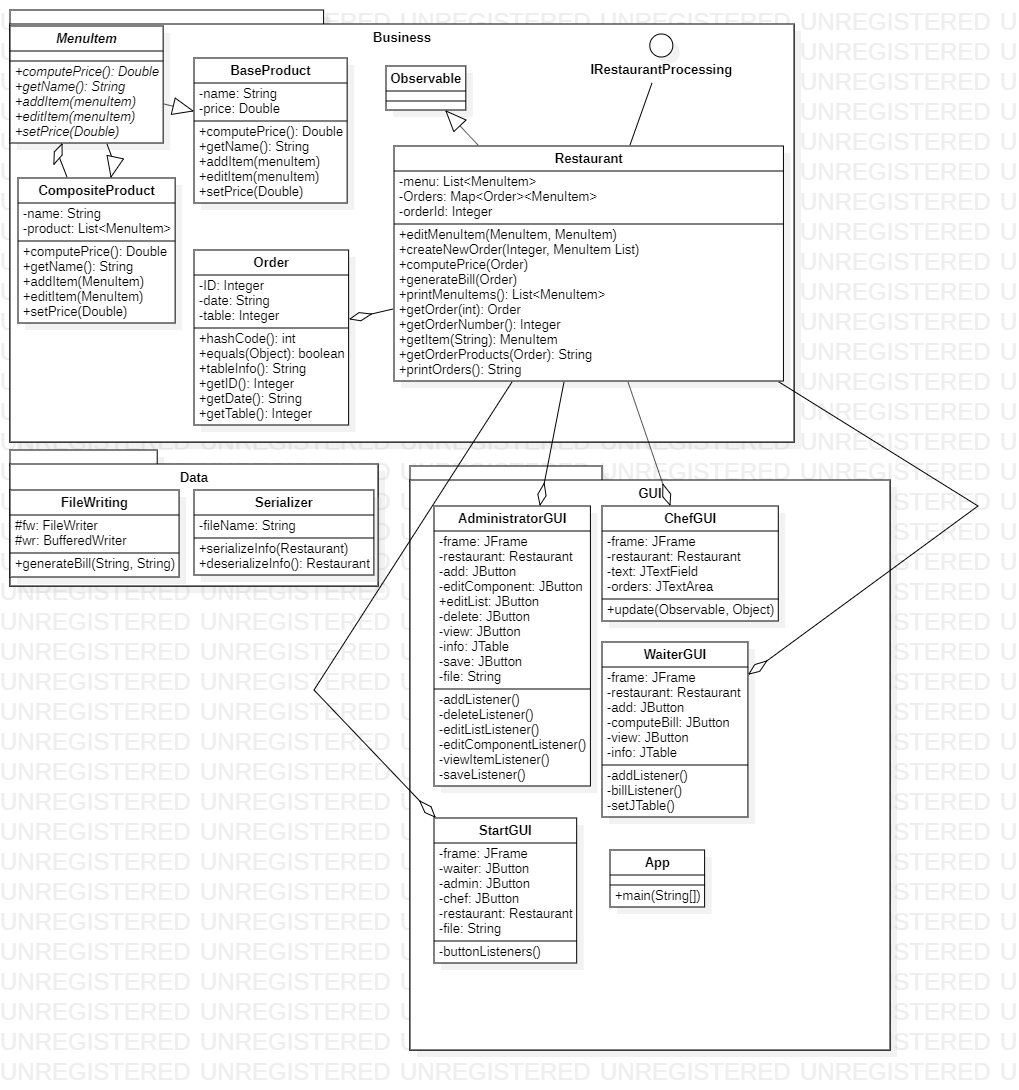
Aplicatia porneste cu interfata Chef si Start, din ultima se poate alege cea corespunzatoare angajatului respectiv.

Interfata pentru Chef se updateaza automat cand este introdusa o noua comanda de catre Waiter.

Waiter-ul poate introduce noi comenzi, vizualiza comenzile existente si sa genereze fisiere .txt ca si bonuri pentru o anumita comanda.

Administratorul poate adauga noi produse, poate edita produsele, poate vizualiza meniul, poate sterge produse si poate salva datele din restaurant in acel moment.

* Diagrama UML



Clasele AdministratorGUI, ChefGUI, WaiterGUI, StartGUI au o relatie de compozitie cu clasa Restaurant.

Clasa Restaurant implementeaza interfata IRestaurantProcessing si extinde clasa Observable.

Clasele BaseProduct si CompositeProduct mostenesc clasa MenuItem.

Clasa MenuItem are o relatie de compozitie cu clasa CompositeProduct.

* Packages

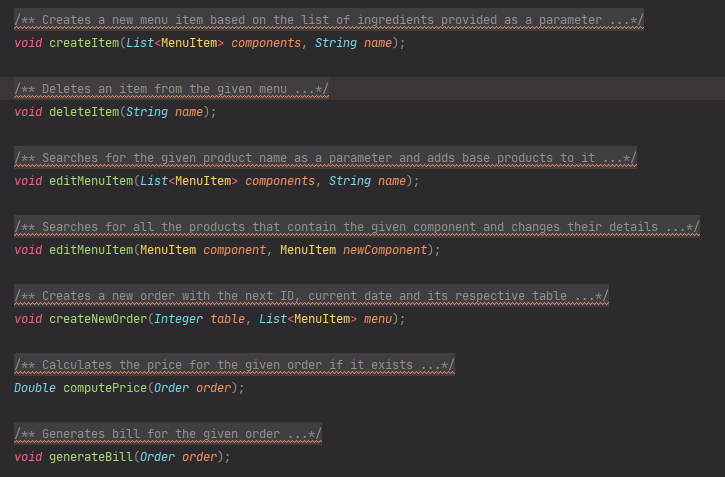
Pachetele prezente sunt Business, Data si GUI.

In Business sunt implementate clasele MenuItem, BaseProduct si CompositeProduct conform Composite Design Pattern, acestea ingloband produsele simple si compuse din meniu.

In Data sunt implementate clasele FileWriting si Serializer care se ocupa cu generarea de fisiere .txt si serializarea datelor pentru persistenta.

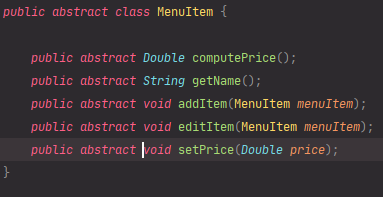
Implementare

* IRestaurantProcessing



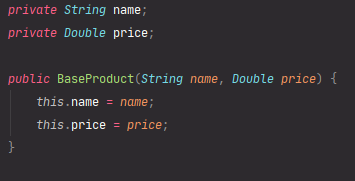
Interfata IRestaurantProcessing implementeaza metodele specifice clasei Restaurant. Interfata este definita cu ajutorul Design by contract, astfel fiecare metoda are pre sau post conditii care trebuie verificate in implementarea ei propriu-zisa.

* MenuItem



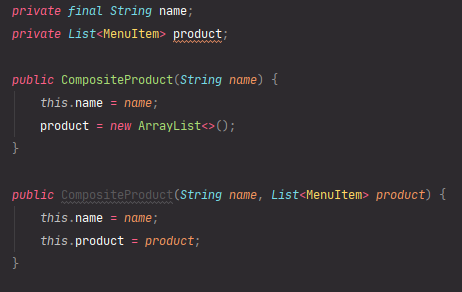
Conform Composite Design Pattern, clasa MenuItem defineste metode care trebuie implementate de clasele care o extind, mai exact BaseProduct si CompositeProduct.

* BaseProduct



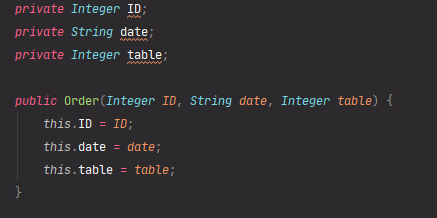
Clasa BaseProduct reprezinta un produs simplu care intra in componenta altora complexe. Aceasta este definita de nume si pret, si implementeaza toate functile din clasa MenuItem, mai putin addItem la care arunca o exceptie de tip UnsupportedOperationException.

* CompositeProduct

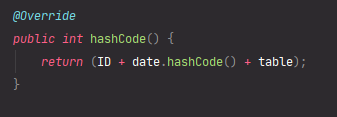


Clasa CompositeProduct reprezinta produsul final din meniul restaurantului. Acesta contine o lista de BaseProduct-uri si un nume. Implementeaza toate metodele din clasa MenuItem, mai putin setPrice pentru care atunci cand va fi folosita se va arunca o exceptie de tip UnsupportedOperationException.

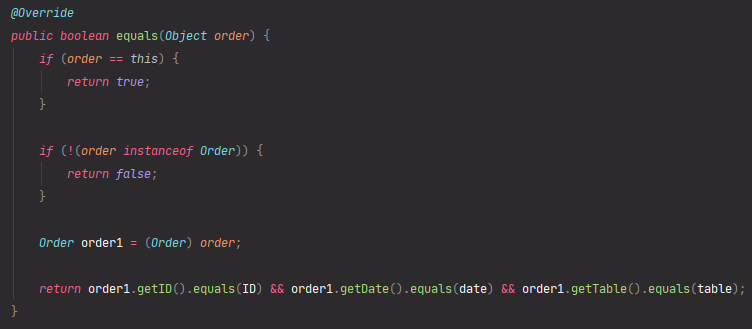
* Order



Clasa Order reprezinta comanda pe care clientul o va da. Aceasta este caracterizate de ID, nume si masa careia ii este asignata.



Functia implicita de hash a clasei este suprascrisa pentru ca va fi folosita in stocarea instantelor acestei clase intr-o structura de tip Map.



Metoda equals a clasei este suprascrisa pentru a fi corecta comparatia dintre doua obiecte de acest tip.

* FileWriting



Constructorul acestei clase seteaza atributele acesteia.

Metoda generateBill seteaza numele fisierului .txt ce urmeaza sa fie generat, apoi scrie in el continutul trimis ca parametru.

* Serializer



Clasa primeste numele fisierului care va fi serializat sau deserializat.

Functia serializeInfo preia clasa Restaurant in starea ei curenta si o serializeaza in fisierul cu numele stabilit.

Functia deserializeInfo deserializeaza fisierul cu numele salavt in atributul clasei si returnreaza un obiect de tip Restaurant cu informatiile deserializate.

* Restaurant



Clasa Restaurant implementeaza metodele definite in interfata IRestaurantProcessing. Aceasta este definita de un meniu sub tip de lista, comenzile salvate intr-o structura de tip Map care contine detaliile despre comanda si produsele continute de aceasta.



Aceasta functie primeste ca argument o lista de BaseProduct-uri si un nume. Astfel se adauga la un composite product cu numele primit ca parametru acele BaseProduct-uri. Dupa formare, se adauga in meniul din clasa Restaurant.



Aceasta functiecauta prin meniul din clasa Restaurant si va strege toate produsele care au numele la fel cu cel trimis ca parametru.



Prima functie cauta prin meniu toate produsele care contin produsul sau produsele trimise ca parametru si cand vor fi gasite, li se updateaza informatiile dupa cele trimise ca parametru.



Aceasta functie construieste un String care contine informatii despre comanda trimisa ca parametru si apoi se foloseste de clasa FileWriting pentru a genera un fisier .txt cu informati construita.

* GUI



Clasa StartGUI este dedicata ferestrei de pornire care prezinta trei butoane: Administrator, Waiter, Chef. Fiecare buton va deschide o noua fereastra specifica tipului de angajat reprezentat de textul din el.



Clasa AdministratorGUI este dedicata ferestrei specifice rolului de administrator care contine sase butoane: Add Menu Item, Edit Menu Item, Edit List of Menu Items, Delete Menu Item, View Menu si Save.



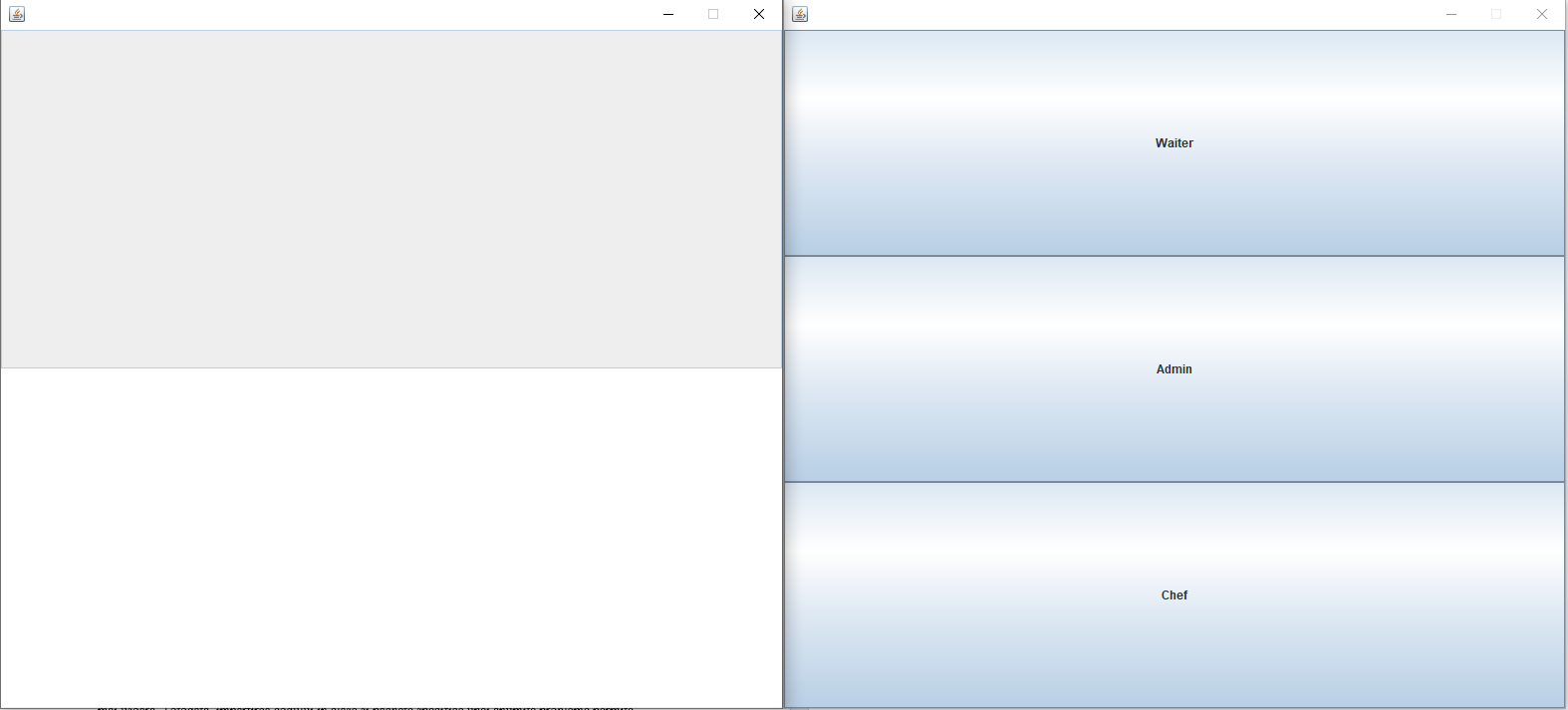
Clasa ChefGUI este dedicata ferestrei rolului de bucatar. Aceasta nu contine nici un buton, ci doua casete de text care se updateaza de fiecare data cand o noua comanda este introdusa. Aceasta updatare real-time a fost realizata cu ajutorul interfetei Observer si clasa Observable.



Clasa WaiterGUI este dedicata rolului de ospatar. Aceasta contine trei butoane: Add Order, Compute Bill for Order si View Orders.

Rezultate

Aplicatia porneste initial cu aceste doua ferestre:



Fereastra din stanga este cea specifica pentru Chef. Utilitatea ei este ca la fiecare Order nou introdus, aceasta se updateaza cu detaliile lui.

Mod de folosire:

Add Order: Se introduce in caseta de text: <nr mesei>;<Produs>;<Produs>…

Compute Bill for Order: Se introduce in caseta de text: <nr\_comanda>

Add Menu Item: <produs> <pret> <produs> <pret>…;<Nume\_Produs>

Edit Menu Item: <produs> <pret>

Edit List of Menu Items: <produs> <pret> <produs> <pret>…;<Nume\_Produs>

Delete Menu Item: <Nume\_Produs>

View Menu: Deschide o noua fereastra care afiseaza informatiile despre meniu.

Save: Salveaza informatiile despre restaurant din momentul apasarii butonului.

Concluzii

Ideea cea mai importanta care a rezultat din realizarea acestei teme este, cu sigurant, importanta folosirii unui pattern sau mai multe care sa corespunda cu necesitatile de implementare ale proiectului. Astfel, se folosesc pattern-uril Composite Design pentru a defini elementele din meniul restaurantului, Model-View Controller pentru a implementa interfata grafica, implementarea unei interfete dupa standardul Design by Contract.

Este prezenta si nevoia de realizare a unui cod “curat”. Necesitatea de a realiza un cod atat de inteligibil incat nu are nevoie de a fi comentat este imperativa, deoarece reintoarcerea la el si/ sau modificarea acestuia devine mult mai usoara. Totodata, impartirea codului in clase si pachete specifice unei anumite probleme permite reutilizarea acestuia si evita scrierea unor bucati largi de cod care vor fi folosite doar o singura data. Toate acestea sunt garantate de catre folosirea pattern-urilor speicifice si Design by Contract.

Elementul care necesita dezvoltare ulterioara este interfata grafica. Momentan este foarte simpla, doar pentru a demonstra functionalitate. Bineinteles, poate fi folosita cu instructiunile de folosire de rigoare, dar nu este facila folosirea ei.

Bibliografie

Map: <https://www.w3schools.com/java/java_hashmap.asp>

Patterns: <https://en.wikipedia.org/wiki/Composite_pattern>

Design by Contract: <https://en.wikipedia.org/wiki/Design_by_contract>

File I/O: <https://stackabuse.com/reading-and-writing-files-in-java/>