**- TEMA IV -**

**SISTEM DE MANAGEMENT A UNUI RESTAURANT**

Marc Andrei – Aurel

Grupa 30224

Laborator – Tehnici de Programare

Cuprins

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

Obiectivul temei

Composite Design Pattern descrie un grup de obiecte care sunt tratate ca o singura instanta a aceluiasi tip de obiect. Intentia C. D. P – ului este de a uni obiecte intr-o structura de arbore si lasa utilizatorul sa utilizeze structura, cerand fiecarui nod din arbore sa indeplineasca o sarcina. Pattern-ul are 4 „participanti” : componenta, care declara interfata pentru obiectele din compozitie si acceseaza componentele – copil; frunza, care descrie comportamentul pentru obiectele primitive din compozitie; compozitia, stocheaza componentele – copil si implementeaza relatiile dintre copii in interfata componentei; clientul, care manipuleaza obiectele din compozitie prin interfata componentei.

Observer Design Pattern este folosit unde avem multe relatii one – to – many intre obiecte, astfel incat, cand un obiect este modificat, obiectele ce depind de el sa fie notificate automat. Acest design pattern se incadreaza in categoria Behavioral pattern. Are trei componente : subiectul, observer-ul si clientul. In cazul nostru, clasa Restaurant va trebui sa implementeze Observable, iar chef-ul, care trebuie sa fie notificat, va trebui sa aiba o clasa ce implementeaza Observer.

HashMap-ul este o parte din colectiile Java si implementeaza o interfata de Map. Stocheaza perechi de chei si valori si pentru accesul la o valoare este nevoie de cheie. Metoda hashCode este folosita pentru a determina in ce bucket este stocat un entry, in tema curenta va trebui suprascrisa, la fel ca si metoda equals.

Design by contract este o abordare a design-ului software. Aceasta presupune in definirea formala si precisa prin preconditii, postconditii si invarianti. Aceste specificatii sunt referite ca si contracte. In general, pentru conditii sunt folosite assert-uri.

Serialization este procesul translatiei unei structuri de date sau obiect intr-un format care poate fi stocat, de obicei intr-un fisier, sau transmis si reconstruit mai tarziu. In aceasta aplicatie, serializarea va fi folosita la scrierea si citirea din fisier a produselor din meniu si a comenzilor introduse prin intermediul interfetei, in vederea salvarii datelor pentru urmatoarele utilizari ale aplicatiei, de catre administrator, chelner sau chef.

**Obiectivul temei** este dezvoltarea unei aplicatii pentru managementul unui restaurant. Sistemul trebuie sa aiba trei tipuri de useri: administrator, chelner si chef. Administratorul beneficiaza de operatiile de a adauga, sterge si modifica produse din meniu. Chelnerul va putea adauga o comanda unei mese, adauga elemente din meniu si a genera o factura pentru o comanda. Se va considera, pentru simplificarea problemei, ca restaurantul are doar un administrator si un chelner, pentru a nu necesita o fereastra de log in. Diagrama UML a claselor ce trebuie implementata este data in cerinta, si este prezentata mai jos.

Analiza problemei

Tema consta in implementarea unui sistem de management al unui restaurant. Vom defini interfata RestaurantProcessing ce va contine operatiile ce pot fi efectuate de chelner si administrator: in cazul administratorului, se va putea crea, sterge si edita un produs din meniu, iar pentru chelner se va putea crea o comanda, calcula pretul pentru o comanda si genera o factura in format .txt, care ofera detalii in legatura cu pretul, data la care s-a efectuat comanda si produsele comandate.

Pentru clasele MenuItem, BaseProduct si CompositeProduct, se va folosi Composite Design Pattern. Pentru a notifica chef-ul la fiecare comanda de CompositeProduct, se va folosi Observer Design Pattern, la fiecare adaos in map-ul de comenzi fiind trimisa o notificare. Clasa Restaurant va avea un HashMap care va contine comenzile. Cheia hashmap-ului va fi obiectul de tip Order, pentru care metoda hashCode() va fi suprascrisa pentru a obtine valoarea hash de la atributele obiectului Order. De asemenea, va contine un ArrayList ce va stoca produsele din meniu. Clasa va fi implementata folosind metoda Design by contract, avand pre conditii, post conditii, invarianti si assert-uri.

Clasa **Restaurant**, care implementeaza interfata RestaurantProcessing, o sa implementeze metodele mostenite din interfata, si anume: crearea unui produs in meniu, stergerea si editarea unui produs din meniu; crearea unei noi comenzi, calculul pretului unei comenzi si generarea unei facturi in format text. De asemenea, pe langa getter-ele si setter-ele comune pentru colectiile de comenzi si cea pentru meniu, vom implementa o metoda care adauga un produs in lista de produse deja existente. La implementarea acestei metode, vom adauga si notificarea catre chef, care va rezulta in apelul metodei update din partea chef-ului, care va adauga o comanda in tabel.

Clasele **BaseProduct** si **CompositeProduct** extind clasa **MenuItem**, iar acestea reprezinta clasele ce vor incapsula produsele din meniul restaurantului. MenuItem este o clasa abstracta ce are ca atribute pretul si numele produsului. BaseProduct extinde aceasta clasa, si nu are alte atribute. In schimb, vom implementa o metoda toString() pentru a obtine mai usor campurile obiectului. CompositeProduct extinde, de asemenea, MenuItem, si are un atribut suplimentar, si anume o lista de BaseProduct, deoarece un obiect de tip CompositeProduct este, de fapt, o combinatie de BaseProduct, privita ca o singura entitate.

Clasa **Order** este clasa ce retine informatii despre comanda, si anume id-ul comenzii, data comenzii si masa la care s-a facut comanda. In clasa Order, vom suprascrie metodele equals si hashCode. In equals vom decide ce campuri ar trebui sa corespunda astfel incat doua obiecte de tip Order sa fie egale. In hashCode, vom returna id-ul comenzii.

Clasele **AdministratorGraphicalUserInterface**, **ChefGraphicalUserInterface** si **WaiterGraphicalUserInterface** sunt clase ce extind JFrame si reprezinta interfetele grafice pentru administrator, chef si chelner. Ferestrele administratorului si al chelnerului contin butoane ce efectueaza functiile lor.

Structura proiectului dezvoltat este Maven, un sistem de build și administrare a proiectelor, scris în Java. Proiectul este descris într-un fișier XML, denumit POM (Project Object Model) , care conține informații despre module, precum și despre dependențele proiectului. Acesta îi definește proiectului o denumire unică, formată din group ID și artifact ID.

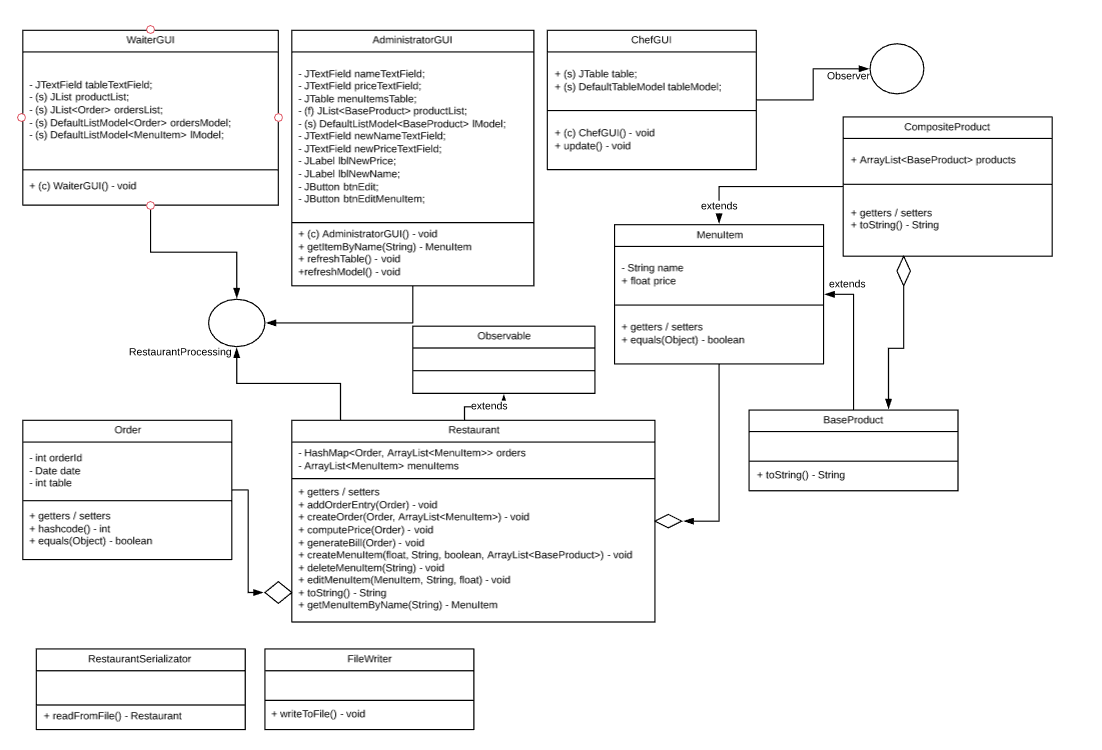
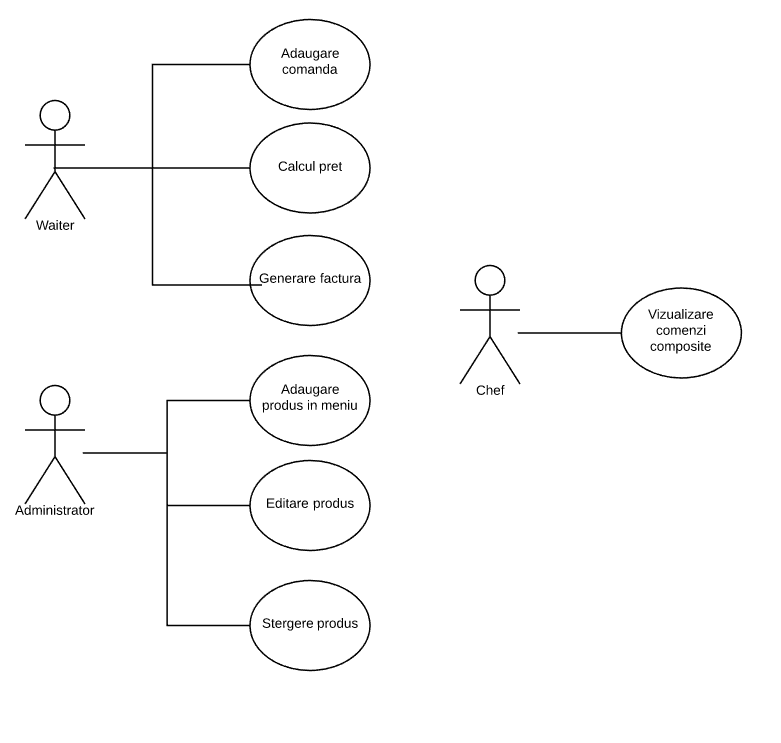
Diagrama UML implementata este cea oferita in cerinta problemei.

Diagrama Use Case



Implementare

Clasa **RestaurantProcessing** – interfata care descrie metodele ce pot fi implementate de catre administrator si chelner. Are ca metode neimplementate: **createMenuItem()**, **deleteMenuItem()**, **editMenuItem()**, pentru administrator si **createOrder()**, **computePrice()**, **generateBill()** pentru chelner. Metodele sunt implementate in clasa prezentata ulterior.

Clasa **Restaurant** – implementeaza metodele descrise in interfata RestaurantProcessing, si contine o colectie de comenzi, precum si una de produse din meniu. Sunt implementate metodele **createMenuItem()**, **deleteMenuItem()**, **editMenuItem()**, care lucreaza pe produsele din meniu: creare, stergere, respectiv editare, si **createOrder()**, **computePrice()**, **generateBill()**, care lucreaza pe comenzi: creare, calcul pret si generarea facturii. Pe langa acestea, sunt implementate getter-e si setter-e pentru colectii, precum si o metoda care adauga o comanda in HashMap, care va notifica chef-ul.

Clasa **BaseProduct** – extinde clasa MenuItem si nu are niciun atribut suplimentar. Ca metode, este implementata o metoda toString(), care returneaza, sub forma unui string, numele si pretul produsului. De asemenea, este implementat si constructorul, cu parametrii pret si nume.

Clasa **CompositeProduct** – extinde clasa MenuItem si are ca atribute suplimentare un Array List de Base Product. Aceasta lista retine produsele simple ce alcatuiesc acest produs compus. Ca metode, este implementat constructorul clasei, cu parametrii pret, nume si lista ce trebuie copiata, getter-e si setter-e pentru lista de produse, o metoda ce adauga un produs in lista, si o metoda listToString(), care returneaza lista de produse ce alcatuiesc produsul nostru compus, sub forma unui String, pentru a usura insertia in tabelul administratorului.

Clasa **MenuItem** – este clasa abstracta care este extinsa de clasele BaseProduct si CompositeProduct. Aceasta implementeaza interfata Serializable, pentru ca obiectele de tipul MenuItem, sau orice subclasa, sa poata fi scrise in fisier, ca apoi sa fie luate si utilizate la popularea restaurantului. Are ca atribute pretul si numele unui produs din meniul restaurantului. Ca metode, avem getter-e si setter-e pentru atributele definite, precum si o metoda equals() care defineste cum trebuie sa fie un obiect, astfel incat acesta sa fie egal cu un obiect de tipul MenuItem, si returneaza true sau false in functie de egalitatea obiectelor.

Clasa **Order** – este clasa ce retine informatii despre o comanda. Are ca atribute id-ul, data si masa la care s-a produs comanda. Ca metode, are getter-e si setter-e pentru atributele definite, o metoda toString (), care returneaza id-ul si masa la care s-a produs comanda, sub forma unui String, o metoda equals(), care defineste cum trebuie sa fie un obiect, astfel incat acesta sa fie egal cu un obiect de tipul Order, o metoda hashCode(), care suprascrie hash code-ul care va fi inscris in Hash map. Aceasta implementeaza interfata Serializable, pentru ca obiectele de tipul Order sa poata fi scrise in fisier, ca apoi sa fie luate si utilizate la popularea restaurantului.

Clasa **AdministratorGraphicalUserInterface** – este clasa care ofera interfata grafica a administratorului. Ofera posibilitatea de a utiliza functionalitatile unui administrator, si anume adaugarea unui produs in meniu, editarea unui produs din meniu si stergerea unui produs din meniu. De asemenea, contine un JTable care ofera informatii despre produsele deja existente in meniu, precum si un JList care afiseaza toate produsele simple, de tip Base Product, deja existente in meniu, pentru a le putea adauga mai usor la adaugarea unui produs de tip Composite Product in meniu. Am folosit JButton-uri pentru executarea functionalitatilor, fiecare avand un nume sugestiv pentru functionalitatea indeplinita.

Clasa **ChefGraphicalUserInterface** – este clasa care ofera interfata grafica a chef-ului si extinde clasa Observer. Are o metoda update(), care update-aza tabelul afisat in interfata, adaugand o comanda de Composite Product. Tot ce este afisat in interfata este acest tabel.

Clasa **WaiterGraphicalUserInterface** – este clasa care ofera interfata grafica a chelnerului. Aceasta ofera posibilitatea utilizarii functionalitatilor chelnerului, si anume crearea unei noi comenzi, calculul pretului unei comenzi si generarea unei facturi pentru o comanda selectata. Sunt afisate toate comenzile prin id si masa, precum si produsele ce pot fi comandate. In momentul crearii unei noi comenzi, se adauga id-ul si masa de la care s-a efectuat comanda, in lista comenzilor.

Clasa **RetrieveItemsFromFile** – este clasa in care se realizeaza scrierea si citirea din fisierul in care sunt scrise detaliile restaurantului, si anume produsele din meniu si Hash Map-ul de comenzi, impreuna cu lista produselor comandate, aceste functionalitati fiind facute dupa principiul serializarii. Metoda readFromFile() realizeaza citirea obiectelor din fisier, folosind clasele FileInputStream si ObjectInputStream, instantele fiind apoi inchise. Pentru a citi obiectele, se foloseste metoda readObject(). Metoda writeToFile() realizeaza scrierea obiectelor in fisier, de fiecare data detaliile restaurantului fiind suprascrise peste detaliile anterioare. Se folosesc clasele FileOutputStream si ObjectOutputStream, mai apoi instantele lor fiind inchise, dupa ce obiectul de tip Restaurant este scris in fisier.

Rezultate

Pentru testarea aplicatiei, am initializat un numar mic de produse, de tipul BaseProduct, cat si de tipul CompositeProduct. Le-am adaugat in lista si am testat functionalitatile aplicatiei pe aceste produse. De asemenea, pentru testarea functionalitatilor chelnerului, am instantiat obiecte de tip Order, si le-am pus in Hash Map-ul restaurantului. Am testat fiecare functionalitate pe parcursul implementarii lor, pentru asigurarea unei implementari corecte. Pentru a rula aplicatia propriu – zisa, va trebui sa instantiem obiecte de tipul Chef Graphical User Interface, Waiter Graphical User Interface si Administrator Graphical User Interface, si le vom seta drept vizibile. Inainte de acest lucru, se vor incarca in obiectul de tip Restaurant, datele din fisierul in care se scrie la inchiderea aplicatiei, pentru a pastra toate modificarile suferite pe parcursul utilizarii aplicatiei. Pentru salvarea datelor in fisier, exista un buton, „Exit application”, care salveaza datele, iar mai apoi inchide aplicatia prin System.exit (), pentru ca apoi la urmatoarea rulare, datele scrise in fisier sa fie incarcate in obiectul Restaurant pentru utilizarea de catre administrator, respectiv chelner.

Concluzii

In aplicatie s-a realizat implementarea diferitelor functionalitati dintr-un restaurant, functionalitati ce pot fi utilizate de administrator si chelner, bucatarul putand doar sa primeasca comenzi. Aplicatia a asigurat o buna cunoastere ulterioara a unor Design Pattern-uri, si anume Composite Design Pattern, Design by Contract si Observer Design Pattern, dar si a utilizarii principului de serializare, aspecte deosebit de importante ale programarii orientate pe obiect. Dupa realizarea aplicatiei, pot afirma ca am clarificat multe aspecte neclare in legatura cu design pattern-urile prezentate mai sus, precum si in legatura cu serializarea.

Ca dezvoltare ulterioara, poate fi luata in considerare posibilitatea de a permite existenta mai multor administratori si chelneri, printr-o interfata de login.

Bibliografie

<https://www.geeksforgeeks.org/composite-design-pattern/>

<https://www.geeksforgeeks.org/java-util-hashmap-in-java/>

<https://www.tutorialspoint.com/design_pattern/observer_pattern.htm>

<https://en.wikipedia.org/wiki/Design_by_contract>

https://lucidchart.com/