Managementul unui resturant

Balint Simina Monica

Grupa 30226

Cuprins

1. Obiectivul temei

Obiectivul acestei teme de laborator este implementarea unu sistem de gestiune a unui restaurant. Acest sistem trebuie sa contina 3 tipuri de utilizatori: administrator, chelner si bucatar. Un admninistrator are posibilitatea de a adauga, a sterge si a modifica podusele existente in meniu. Chelnerul poate sa creeze o comanda noua pentru o anumita masa, sa adauge in comenzi elemente din meniu si sa creeze o facuta pentru masa respectiva. Bucatarul este notificat de catre chelner de fiecare data cand se primeste o comanda noua.

Programul in sine nu este unul complicat, mai ales avand la dispozitie toate resursele necesare, iar obiectivul acestui proiect este acela de a intelege mult mai bine conectarea dintre un sistem de gestiune a unui restaurant si un mediu de programare orientata pe obiecte. Abilitatea unei clase Java de a examina si manipula datele chiar din interiorul sau, nu pare sa sune a a fi ceva extrem de complicat sau elaborat, dar in alte limbaje de programare aceasta trasatura lipseste sau chiar nu exista. De exemplu, intr-un program care utilizeaza limbaje precum: Pascal, C, C++ este imposibila obtinerea unor informatii referitoare la functile sau procedurile definite in interiorul programului.

2. Analiza Problemei

Aplicația se va implementa urmarind urmatoarele cerinte:

1. Definirea interfetei RestaurantProcessing care va contine operatiile principale care vor putea fi executate de catre administrator si chelner, dupa cum urmeaza:
- Administratorul: adaugarea, stergerea si editarea unui item din meniu.
- Chelnerul: creeaza o comanda noua, calculeaza pretul total al unei comenzi si genereaza o factura pentru comanda în format .txt.
2. Se va folosi Composite Design Pattern pentru a defini clasele MenuItem, BaseProduct si CompositeProduct si Observer Design Pattern pentru a notifica bucatarul de fiecare data cand se face o noua comanda care contine un composite product.
3. Clasa Restaurant va fi implementata folosind structura de date hashtable. Cheia tabelei de dispersie va fi generată pe baza atributelor clasei Order, care poate avea asociate mai multe MenuItems.

Se va defini o structura de tip Map<Order, Collection<MenuItem>> pentru a stoca informatiile legate de comanda.

4. Produsele din meniu vor fi salvate într-un si dintr-un fisier folosind serializarea.

Pentru a remedia dezavantajele utilizarii structurilor cu acces direct sunt definite **tabelele de dispersie**, hash tables, ce reprezinta colectii de date in care pe baza unei functii hash cheia de cautare este pusa in corespondenta cu pozitia elementului in cadrul colectiei. Avantajul acestei tip de structura de stocare si cautare este dat de utilizarea mai eficienta a spatiului fara a se stoca memorie pentru elemente care nu sunt utilizate si implementarea acestora cu chei alfanumerice.

Utilizand fluxurile putem scrie aplicatii care salveaza si incarca datele in fisiere. Java permite si un mecanism mai avansat si anume serializarea obiectelor. In forma cea mai simpla serializarea obiectelor inseamna salvarea si restaurarea starii obiectelor. Obiectele oricarei clase care implementeaza interfata Serializable, pot fi salvate intr-un stream(fluxde date) si restaurate din acesta.

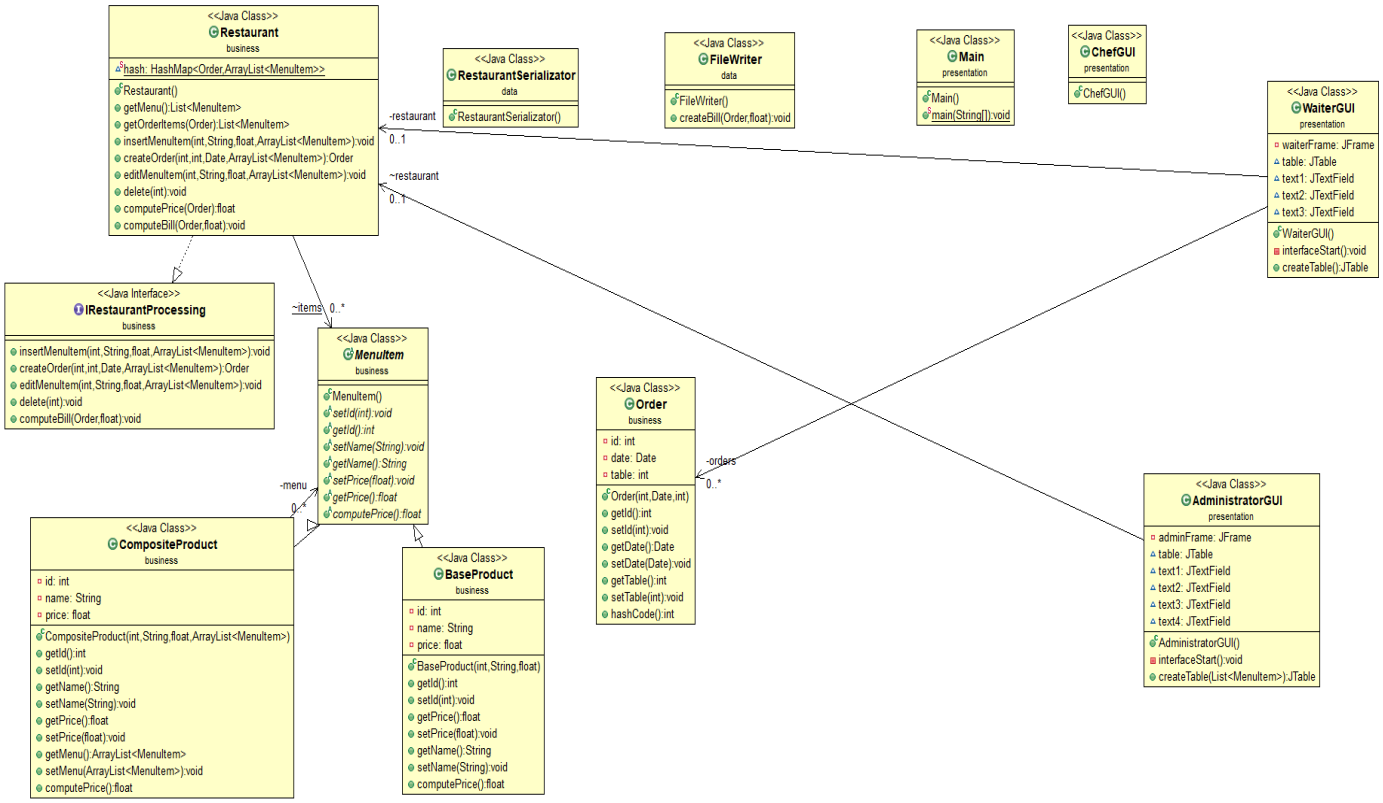
3. Proiectare

Am ales utilizarea colectiilor in rezolvarea unor cerinte. De asemenea in realizarea clasei Restaurant am utilizat o structura de date HashMap, iar pentru stocarea produselor din meniu am folosit tip de date ArrayList.

Fiecare utilizator are la dispozitie mai multe optiuni pe care le poate alege oricand. In tabelul de la administrator, acesta poate sa introduca date sau sa le stearga, de asemenea daca acesta doreste poate si sa modifice valorile din tabel.

In cazul in care utilizatorul doreste sa creeze o comanda, programul este gandit in asa fel incat daca, comanda este plasata cu succes, iar stocurile sunt disponibile, se va returna o factura pe numele clientului in format txt.

In continuare voi prezenta diagrama UML a proiectului:



Pe baza aceste diagrame se poate observa agregarea dintre MenuItem si Restaurant si Order si Restaurant. Se observa, de asemenea, si relatiile dintre BaseProduct, CompositeProduct și MenuItem, care scot in evidenta utilitatea folosirii Composite Design Pattern-ului.

4. Implemetare

Am ales sa structurez acest proiect pe 3 pachete diferite: business, data si presentation.

Pachetul business contine clasele de baza si functionale ale proiectului: BaseProduct, CompositeProduct, Restaurant, Order, clasa abstracta MenuItem, dar si o interfata IrestaurantProcessing.

Pachetul data contine clasele FileWriter si ResturantSerializator.

Pachetul presentation contine clasele care se ocupa de interfetele grafice ale aplictiei si anume: AdministratorGUI, WaiterGUI si ChefGUI, dar si o clasa Main de unde se lanseaza aplicatia propiu zisa.

In clasele BaseProduct si CompositProduct sunt implementati constructorii claselor si gettere si settere.

Clasa Restaurant implementeaza interfata IRestautantProcessing care are definite toate metodele pe care apoi le-am suprascris in clasa Restaurant. Aceasta clasa are urmatoarele metode:

- insertMenuItem,
- createOrder,
- editMenuItem,
- delete,
- computePrice,
- computeBill.

In clasa FileWriter se utilizeaza tipul de date print writer pentru implementarea metodei createBill care va fi apelata in clasa Resturant.

Clasa AdministratorGUI implementeaza interfata grafica pentru administrator. Aceasta are in componenta 3 butoane, fiecare pentru cate o operatie pe MenuItem implementata în clasa Restaurant. De asemenea, am 4 text field-uri în care introduc datele dorite, care apoi sunt procesate si afisate într-un tabel de tip Jtable.

Clasa WaiterGUI reprezinta interfata pentru chelner. Acesta poate sa adauge o comanda, sa genereze o factura si sa calculeze pretul total al comenzii. Fiecare dintre aceste functionalitati este pusa in actionListener-ul cate unui buton.

5. Rezultate

La administrator, în funcție de operația aleasă, se vor putea observa elementele de meniu într-un tabel care va afișa id-ul, numele produsului și prețul.

6. Concluzii

Mediul de programare Java ofera posibilitatea reala ca majoritatea programelor sa fie scrise intr-un limbaj sigur de tip. Java se extinde cu un mecanism pentru polimorfism parametric, care permite definirea si implementarea adstractizarilor generice. Acest proiectul ofera un design complet pentru acest limbaj.

Eu personal, am invatat si am inteles mai bine cum sa folosesc clasele si modificatorii-cei care stabilesc daca o clasa este publica sau privata. Tema a 4-a de laborator mi-a aprofundat

cunostintele referitoare la limbajul de programare Java întrucat am invatat sa utilizez Observer Design Patter.

In ceea ce priveste evolutiile ulterioare am cateva idei pentru a modifica interfata pentru a o face mai atractiva, deoarece in zilele noastre asta este ceea ce cauta un utilizator. El sau ea nu are nicio ingrijorare cu privire la modul in care este realizata o aplicatie la nivel de cod au de toate resursele necesare. Aplicatia ar putea fi dezvoltata ulterior in special pe planul interfetei grafice. As putea adauga, de asemenea, functionalitati in plus, cum ar fi transmiterea catre bucatar a retetei produsului din meniu comandat.

7. Bibliografie

https://docs.oracle.com/javase/tutorial/uiswing/components/combobox.html

https://www.1keydata.com/sql/sqlwhere.html

http://tutorials.jenkov.com/java-reflection/index.html

http://www.tutorialspoint.com/java/java_serialization.htm

https://beginnersbook.com/2013/12/how-to-serialize-hashmap-in-java/

https://javarevisited.blogspot.ro/2011/02/how-hashmap-works-in-java.html

https://www.geeksforgeeks.org/

https://www.wikipedia.org/

https://stackoverflow.com/