Tema 4
Restaurant management system

Tehnici de programare

Ardeleanu Madalin-Florin

An academic : 2018 – 2019

Cuprins

1. Cerințe funcționale

Să se implementeze o aplicație care are scopul de a monitoriza activitatea dintr-un restaurant. Aplicația trebuie sa aibă trei tipuri de utilizatori: administrator, ospătar și bucătar. Ospătarul poate crea o nouă comandă pentru o masă, să adauge elemente din meniu și să calculeze nota de la masa respectivă. Administratorul poate să adauge, să editeze sau să șteargă un element din meniu. Bucătarul este notificat de fiecare dată când trebuie să gătească, prin intermediul ospătarului.

2. Obiective

2.1. Obiectiv principal

Obiectivul principal al proiectului este de a crea o aplicație prin intermediul căreia să se faciliteze activitatea dintr-un restaurant.

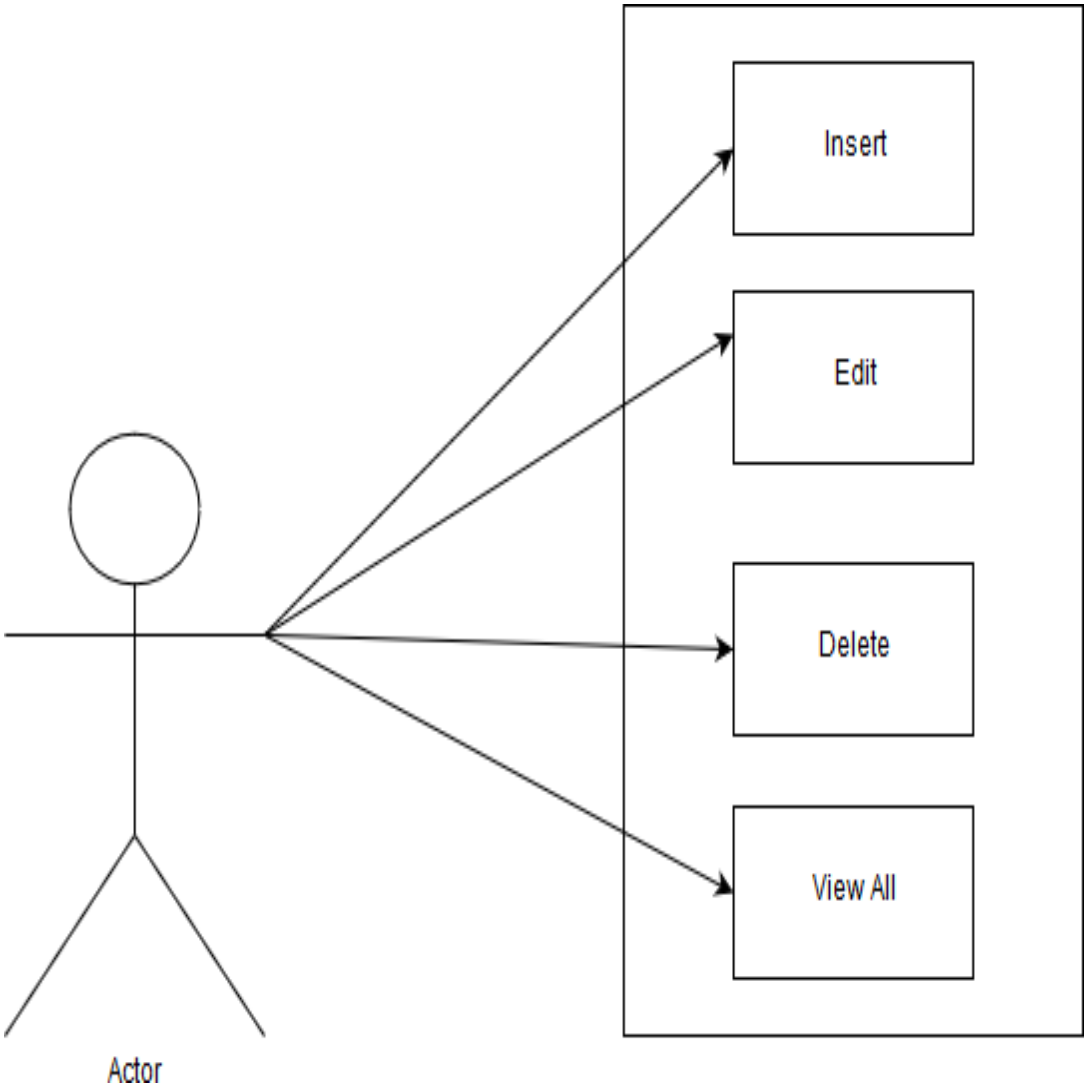
2.2. Obiective secundare

- Gruparea claselor înpachete
- Folosirea serializării
- Folosirea unor tehnici de programare : Observer Design Patter și Composite Design Pattern
- Alegerea structurilor de date

3. Analiza problemei

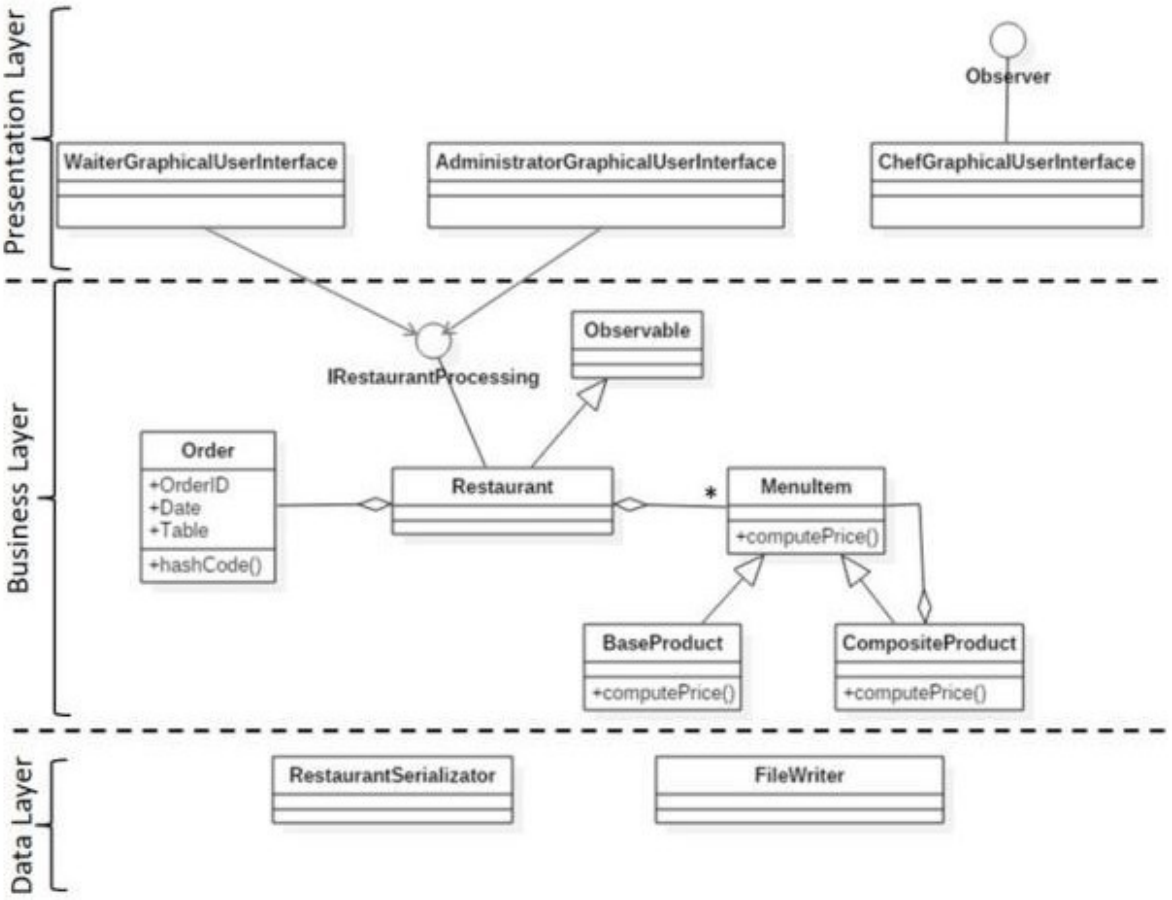
Un sistem de management al unui restaurant este o aplicație software pentru gestionarea eficientă a afacerilor de tip bar, restaurant, terasa, club, fast-food, cafenea etc. Softul oferă un control total asupra tuturor activităților unei astfel de afaceri, de la prelucrarea comenzilor, evidența meselor și a ospătarilor, până la emiterea notelor de plată.

Astfel, un asemenea sistem are nevoie de trei tipuri de useri : administrator, ospătar și bucătar. Administratorul va putea să gestioneze informațiile din meniu, în timp ce ospătarul va putea crea comenzi, adăugând elemente din meniu. De asemenea, ospătarul va putea calcula nota de plată pentru o anumită masă și va notifica bucătarul când va trebui să gătească.



4. Proiectare

4.1. Diagrama de claseUML



4.2. Clase și algoritmifolosiți

Pachetul **BusinessLayer** conține clasele care conțin partea de logică a aplicației. În pachet se găsesc clasele *Order, Restaurant, MenuItem, BaseProduct, CompositeProduct* și cele 2 interfete *RestauratProcessingAdministrator* și *RestaurantProcessingWaiter.*

Clasa *Order* conține informațiile necesare despre o comandă. Aceasta va reprezenta cheia pentru structura Map din clasa Restaurant. Am suprascris metoda hashcode() în cadrul acestei clase.
Fiecare comanda este caracterizata de un id unic, de o data(aici ne referim la data la care s-a efectuat comanda) și de numărul mesei pentru care s-a realizat comanda.
În afara de metoda de hashcode(), în cadrul clasei se găsesc gettere și settere pentru fiecare atribut al acesteia.
Acestea sunt urmatoarele:
getOrderId() - returneaza id-ul comenzii(valoare unica)
getDate() - returneaza data la care s-a efectuat comanda respectiva
getTable() - returneaza numărul mesei pentru care s-a luat comanda
setOrderId(long orderId) – seteaza id-ul comenzii
setDate(Date date) – seteaza data comenzii
setTable(int table) – seteaza numărul mesei

Clasa *MenuItem* este o clasă abstractă și modelează un obiect din meniu. Clasele *BaseProduct și CompositeProduct vor* moșteni clasa MenuItem.

Cele două clase sunt similare, cu excepția faptului că un obiect de tipul CompositeProduct va conține o listă de obiecte de tipul MenuItem.
Astfel se respectă tehnica **Composite Design Pattern**.
Tot aici sunt declarate metode computePrice() și computeProducts() ce urmează să fie implementate în subclase.
Prima metoda va fi folosită pentru a calcula totalul de plata pentru o anumita comanda iar cea de-a doua va fi folosită pentru a forma un string cu produsele ce au fost cumpărate, aceste 2 data urmând să fie folosite la generarea facturii unei anumite comenzi.

Clasa BaseProduct extinde clasa abstracta MenuItem, are un constructor în care se apeleaza super și 3 metoda, una de computePrice(), computeProducts() și generateBill();

Clasa CompositeProduct la fel ca și clasa BaseProduct, extinde clasa abstracta MenuItem. Ca și atribut aici găsim un ArrayList de MenuItem, lucru ce implica folosirea a Composite Design Pattern-ului.
Aici se găsesc metode ca computePrice() ce implementeaza metoda abstracta din MenuItem, aici se va calcula suma totala a produselor ce sunt comanda la o masa, o alta metoda este computeProduct() care la fel implementeaza metoda abstracta din MenuItem, cu ajutorul acestei metoda se formeaza un string ce conține detalii despre denumirea tuturor produselor ce au fost cumpărate, prețul fiecarui produs în parte și prețul total al comenzii calculat anterior cu ajutorul metodei computePrice().
Se mai poate observa un setter pentru ArrayList-ul de MenuItem, care este folosit pentru o indica pentru ce produse vrem sa calculam totalul de plata și detalii despre acestea(ce produse sunt și prețul fiecaruia).
Și nu în ultima rând avem metoda generateBill() care ar trebui sa genereze o factura pentru o anumite comanda.

Clasa *Restaurant modelează* restaurantul propriu-zis. Această clasă conține o listă de obiecte de tipul MenuItem care va reprezenta meniul restaurantului și un obiect de tipul Map care va avea ca și cheie un obiect de tipul Order, iar ca valoare va avea o listă de obiecte de tipul MenuItem (care vor reprezenta produsele comandate la o masă).
În aceasta clasa sunt prezente gettere și settere pentru cele 2 atribute:
getMenuItems() - ce returneaza lista cu produsele disponibile din restaurant

getOrders() - ce returneaza o lista ce conține fiecare comanda care a fost făcută și ceea ce s-a cumpărat în cazul fiecarei comenzi.
setMenuItems(ArrayList<MenuItem> menuItems) – ce este folosită pentru o seta un anumit meniu unei comenzi
setOrders(HashMap<Order,Collection<MenuItem>> orders) – ce seteaza o lista ce va conține toate comenziile ce au fost făcute și ce produse s-au achizitionat în cadrul fiecarei comenzi.
O alta metoda prezenta este cea de isWellFormed() care verifica practic dacă exista elemente în meniu, aceasta metoda este utila în cazul stergerii unui element din meniu, nu vrem sa stergem ceva din meniu dacă nu exista nimic acolo.
Totodata cu ajutorul acestei metode, se realizeaza o parte din design by contract folosind un invariant.
Se poate observa ca în metodele de adaugare,editare,stergere fie ca e vorba de un meniu sau de o comanda, se folosesc asertiuni pentru fiecare anumitor condiții, în unele cazuri sunt folosite precondiții, in alte postconditii iar în alte chiar amândouă.

Clasa Restaurant va implementa și metodele din interfețele RestaurantProcessingAdministrator și RestaurantProcessingWaiter

- addNewMenuItem – adaugă un nou produs în meniu, aceasta operație este executata de către administrator
- editMenuItem() – editează un produs din meniu, aceasta operație este executata de către administrator
- deleteMenuItem() – șterge un produs din meniu, aceasta operație este executata de către administrator
- addNewOrderItem() – creează o comandă nouă, aceasta operație este executata de către ospatar

Pachetul **DataLayer** cuprinde doua clase: FileWriter ce este folosită pe post de logger în cazul generarii facturii pentru o anumita comanda și RestaurantSerializator ce va serializa datele despre continutul unui restaurant la începutul rularii programului, va deserializa aceste date ce vor fi folosite mai departe în clasele AdministratorGraphicalUserInterface și WaiterGraphicalUserInterface pentru a adauga,edita sau sterge anumite produse în cazul administratorului și de a adauga, afisa și a crea o nota de plata pentru o anumita comanda iar la închiderea aplicației, aceste date vor fi

serializate din nou, pentru a putea fi utilizate ulterior cum este și logic, deoarece și în viața reala dacă salvam niște date, nu vrem sa le pierdem după închiderea aplicației.

Clasa RestaurantSerializator implementează două metode : serialize() și deserialize() care sunt specifice serializării, respectiv deserializării.

Sunt folosite cu scopul de a salva și încarca datele salvate cu privire la restaurant.

Pachetul **PresentationLayer** cuprinde clasele folosite pentru interfața grafică: AdministratorGraphicalUserInterface, ChefGraphicalUserInterface, WaiterGraphicalUserInterface și clasa LoginUI. Clasele AdministratorGraphicalUserInterface, ChefGraphicalUserInterface și WaiterGraphicalUserInterface reprezintă ferestrele pentru cei trei utilizatori: administrator, bucătar și ospătar.

În clasa AdministratorGraphicalUserInterface este realizata interfata grafica pentru administrator, aceasta este formata dintr-un Jframe, un Jpanel și 3 butoane ce corespund urmatoarelor informații: adaugarea unui nou produs în meniu, editarea unui produs din meniu și stergerea unui produs din meniu.
În aceasta interfata se poate vedea și tabelul cu produsele disponibile, aceasta fiind creat cu ajutorul unui Jtable.
La apasarea butonului de adaugare se va deschide o noua fereastra ce va conține 2 JtextField-uri și 1 buton, cele 2 JtextField-uri corespund denumirii produsului și prețul ce vrea administratorul sa le introduca.
La apasarea butonului de „Add new item”, un nou produs cu caracteristicile precizate va fi adaugat în lista de meniu, iar tabelul din primul Jframe va fi actualizat automat.

Pentru editarea unui produs, trebuie selectata linia din tabela ce corespunde produsului respectiv, urmând să fie apăsat butonul de „Edit item”, dacă nu a fost selectat niciun produs, se va deschide o fereastra mica de avertizare în care se va preciza ca nu a fost selectat niciun produs, în cazul în care este selectat un produs, după apasarea butonului de „Edit item” se va deschide o

noua fereastra formata dintr-un Jframe, 2 Jtextfield-uri și un buton, la fel ca la adaugarea unui produs.
Primele 2 Jtextfield-uri vor corespunde noilor valori pe care vrem sa le atribuim produsului și anume, un nou nume sau un nou preț, după introducerea acestor valori, se apasa pe butonul denumit „Edit item", se va vedea ca produsul selectat initial va fi redenumit în funcție de nume, de preț sau de ambele după valorile introduse.
Aceasta operație de editare, în modul în care am implementat-o eu consta în 2 operatii succesive, una de stergere a produsului selectat, urmata de operație de introducere a produsului cu noile valori pe care le-am introdus, așa ca dacă va așteptați ca produsul redenumit sa apara pe aceeași linie cu produsul initial, acest lucru nu se va întâmpla decât dacă este vorba de ultimul produs adaugat.
Și nu în ultimul rând în fereasta principala mai avem butonul pentru stergerea unui produs, modul de utilizare este identic cu cel de la editare, adică înainte de apasarea butonului de stergere, trebuie selectat produsul care se dorește a fi șters urmând după aceea apasarea butonului de stergere.
În cazul în care nu s-a selectat niciun produs și se apasa butonul de stergere, se va deschide o noua fereastra în care va apărea un mesaj de avertizare cum ca nu a fost selectat niciun produs, în cazul în care este selectat în produs, după apasarea butonului de stergere.

În clasa WaiterGraphicalUserInterface este realizate interfata pentru ospatar, aceasta fiind formata din Jframe, Jpanel, 1 Jtable și 3 butoane ce corespund urmatoarelor operatii: adaugarea unui comenzi nou, afisarea tuturor comenziilor și crearea unei facturi pentru o anumita comanda.
În cazul în care se apasa pe primul buton, se va deschide o noua fereastra ce va conține un Jframe, un Jpanel, 2 butoane și 2 Jtextfield-uri.
Cele 2 Jtextfield-uri trebuie completate cu informațiile referitoare la ce produs se dorește a fi comandat și de la ce masa se face comanda, pentru adaugarea produsul, se va apasa pe butonul „Add product", cât timp nu se iese din aceasta fereastra se pot adauga oricate produse pe aceeași comanda, dacă se iese în fereastra și se intra la loc, în cazul în care ai adaugat niște produse înainte și nu ai finalizat comanda, produsele se pierd și trebuie introduse din nou.
Așadar după adaugarea tuturor produselor pe care le dorim(produsele care se doresc a fi introduse trebuie sa existe în meniul creat de administrator, dacă produsul descris de noi nu exista în meniu și apăsat pe butonul de adaugare pentru produs, se va deschide o mica fereastra care să ne avertizeze ca produsul respectiv nu exista pe stock), se apasa pe butonul de order și astfel tabelul de comenzi este actualizat, elementele din tabel sunt după cum urmează: un id unic pentru comanda, ce este incrementat automat după fiecare adaugare a unei comenzi, data la care s-a efectuat

comanda(aceasta va fi fix data curenta), masa la care s-a făcut comanda și numărul de produse comandata.
Al doilea buton este folosit pentru afisarea tuturor comenzilor care s-au adaugat.
Iar ultimul buton este folosit pentru crearea unei facturi pentru o anumita comanda, înainte de apasarea acestui buton, trebuie selectata comanda pentru care se dorește să fie tiparita factura.
Dacă nu este selectat nicio comanda, iar butonul de tiparire este apăsat se va deschide o noua fereastra cu un mesaj de avertizare ca nu a fost selectata nicio comanda.
După selectarea comenzii și apasarea butonului se va scrie într-un fisier txt ce produse s-au cumpărat pentru comanda respectiv, cât a costat fiecare și prețul total de plătit.
Totodata aceste informații se vor afisa și în consola cu ajutorul unui logger.

Și nu în ultimul rând, cu ajutorul claselor Subject din bussinesLayer, Observer și ChefGraphicalUserInterface din presentationLayer se realizeaza Observer Design Pattern-ul.

5. Concluzii și dezvoltăriulterioare

Aplicația implementată monitorizează în mod eficient operațiile dintr-un restaurant, putând fi folosită în același timp de trei tipuri de useri: administrator, ospătar și bucătar. Astfel, eficiența atinge un grad ridicat. Un alt aspect important al aplicației este serializarea meniului. Acesta este încărcat odată cu deschiderea aplicației și actualizat la fiecare operație. Pentru a edita, respectiv șterge informațiile,utilizatorul va trebui doar să selecteze informația respectivă,iar apoi să introducă informațiile dorite în tabel. Astfel am evitat deschiderea unor ferestre suplimentare care probabil ar fi dus la confuzie utilizatorul.

6. Bibliografie

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

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

docs.oracle.com/javase/8/docs/technotes/guides/language/assert.html

https://www.javaworld.com/article/2074956/icontract--design-by-contract-in-java.html

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

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

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