Universitatea Tehnica din Cluj-Napoca

Facultatea de Automatica si Calculatoare

Sectia Calculatoare si Tehnologia Informatiei

Documentatie pentru aplicatie de restaurant

Student: Timotei Molcut

Cluj-Napoca

16 Mai 2019

Cuprins

1. Obiectivul temei
2. Analiza problemei, modelare, scenarii, cazuri de utilizare
3. Proiectare
4. Implementare
5. Rezultate
6. Concluzii
7. Bibliografie

Obiectivul temei

Obiectivul principal al temei de fata este de a realiza operatiile pe care administratorul unui restaurant si cu chelnerul le-ar face. Aceasta tema reprezinta practic o aplicatie de procesare a unui restaurant. Astfel se va usura modul de lucru al angajatiilor unui restaurant.

Pentru a se atinge acest obiectiv a fost necesara atingerea altor obiective secundare. Aceste obiective au si ele o importanta in implementarea aplicatiei. Printe aceste obiective se numara, folosirea de design pattern-uri precum Design by contract, Composite, Observer. De asemenea, serializarea a reprezentat un obiectiv secundar important.

De asemenea, nu trebuie sa uitam de faptul ca este nevoie de o interfata grafica. Aceasta interfata grafica usureaza intelegerea asupra a ceea ce se petrece in interiorul aplicatiei.

Pentru a se modela corect conceptul de produs s-a folosit design-ul composite. Astfel, in stock-ul restaurantului pot sa existe produse simple sau produse compuse.

De asemenea, serializarea poate sa salveze toate operatiile realizate asupra unui obiect. Astfel, si in aplicatia de fata obiectul care caracterizeaza restaurantul este deserializat la inceputul aplicatie, iar la final acesta este serializat. Astfel, se va tine o evidenta a tuturor operatiilor realizate in decursul a mai multe executii ale aplicatiei.

Un alt aspect foarte important reprezinta organizarea comenzilor facute in restaurant astfel incat fiecarei comenzi sa i se atribuie o colectie de produse comandate. Pentru acest aspect s-a folosit un HashMap, iar metoda hashCode() din clasa Order, care reprezinta cheia, s-a suprascris.

Despre toate aceste obiective se va discuta pe larg in capitolele urmatoare.

Analiza problemei, modelare, scenarii, cazuri de utilizare

Analizand problema de fata se poate observa ca este nevoie de o organizare pe nivele a aplicatiei. Astfel, toate pachetele importante in care s-a structurat aplicatia vor fi structurate de sus in jos, simbolizand adancimea. La nivelul de la suprafata, adica cel mai inalt nivel se afla prezentarea care tine de interfata grafica. Urmeaza apoi logica de business a aplicatiei. Din business se acceseaza obiectul restaurant in dataLayer, de unde se face serializarea si unde se salveaza fisierele pentru comenzile date in restaurant.

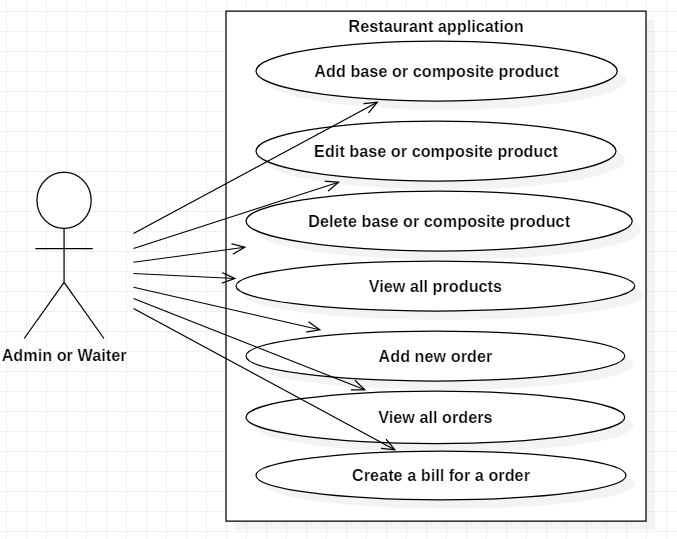
Pentru a extrage obiectul restaurant am implementat serializarea/deserializarea pentru acest obiect. Astfel, la inceputul aplicatiei se extrage(serializare) obiectul restaurant din fisierul „restaurant.ser”. Apoi se porneste interfata de administrator, care la randul ei porneste celelalte 2 interfete: de chelner si de bucatar. La final se face serializarea, adica se rescrie obiectul restaurant in acelasi fisier. E ca si cum la pornirea aplicatiei, user-ul foloseste „unealta” restaurant pentru a-si realiza operatiile dorite, iar la final pune „unealta” la loc.

Obiectele principale modelate de aceasta aplicatie sunt de tip MenuItem si Order. Order reprezinta comanda pe care chelner-ul o face. Pentru aceasta comanda trebuie selectata o lista de obiecte de tip MenuItem. Aceasta lista reprezinta practic comanda pe care o doreste un client la restaurant. Acest MenuItem poate sa aiba doua derivate: BaseProduct care reprezinta un tip de mancare de baza si un CompositeProduct care reprezinta un tip de mancare compus din alte MenuItem(adica BaseProduct).

Scenariile principale de utilizare sunt diferite de la administrator la chelner. Administratorul poate sa modifice meniul restaurantului, adica poate sa adauge un nou produs, poate sa editeze un produs sau poate sa stearga produsul cautat dupa nume. De asemenea, se poate vedea tot meniul adaugat pana la momentul de fata. Meniul poate sa contina si produse de baza si produse compuse.

Chelnerul, spre deosebire, nu poate „umbla la meniu”, adica nu este meseria lui sa faca acest lucru. Acesta poate doar sa creeze o noua comanda, pentru o anumita masa, la o anumita data. Pentru aceasta comanda trebuie asociata o lista de produse comandate. Astfel, chelnerul poate sa aleaga din meniu cateva produse. De asemenea, chelnerul poate sa creeze chitanta pentru comanda ceruta si oricand doreste poate sa vada un tabel cu toate comenzile introduse.

Toate aceste scenarii sunt descrise in diagrama UML use case de mai jos.



Proiectare

Dupa cum am mai spus deja, obiectele principale din aceasta aplicatie sunt restaurantul, menuItem si comanda. Deoarece nu am folosit baze de date, evidenta comenzilor si a produselor comandate trebuie salvata undeva. Asadar, considerand comanda, adica obiectul de tip Order, o cheie am mapat la aceasta cheie o lista de menuItem. Aceasta mapare am realizat-o intr-un atribut de tip HashMap. Pentru a asocia un Order la cheia hashmap-ului a trebuit sa suprascriu metoda hashCode(), care practic seteza pozitia la care cheia Order se va alinia.

De asemenea, pentru a tine evidenta meniului a fost nevoie de o lista de obiecte menuItem. Ori de cate ori administratorul doreste poate sa modifice acest meniu, dupa cum am descris in diagrama use case de mai sus.

Deoarece am dorit sa am si o evidenta a comenzilor, fara produsele lor asociate, am creat inca o lista cu obiecte de tip Order. Asadar, ori de cate ori adaug o comanda in hashmap-ul de comenzi voi adauga comanda respectiva si in aceasta lista.

Pentru a crea JTable-ul de afisare a produselor/comenzilor am folosit o metoda bazata pe reflection. Astfel, tabelul se poate construi dinamic cu coloanele si valorile necesare, iar acest lucru nu trebuie realizat manual.

Pentru serializare am creat 2 metode opuse care extrag/salveaza obiectul restaurant acolo unde trebuie.

Pentru a crea o comanda intr-un fisier .txt am folosit clasa FileWriter care implmenteaza o metoda folosita pentru a scrie in acel fisier comanda, produsele asociate si pretul total.

Detaliile de mai sunt sunt descrise sub forma unor diagrame UML de clase si pachete, mai jos.

Diagrama UML de clase

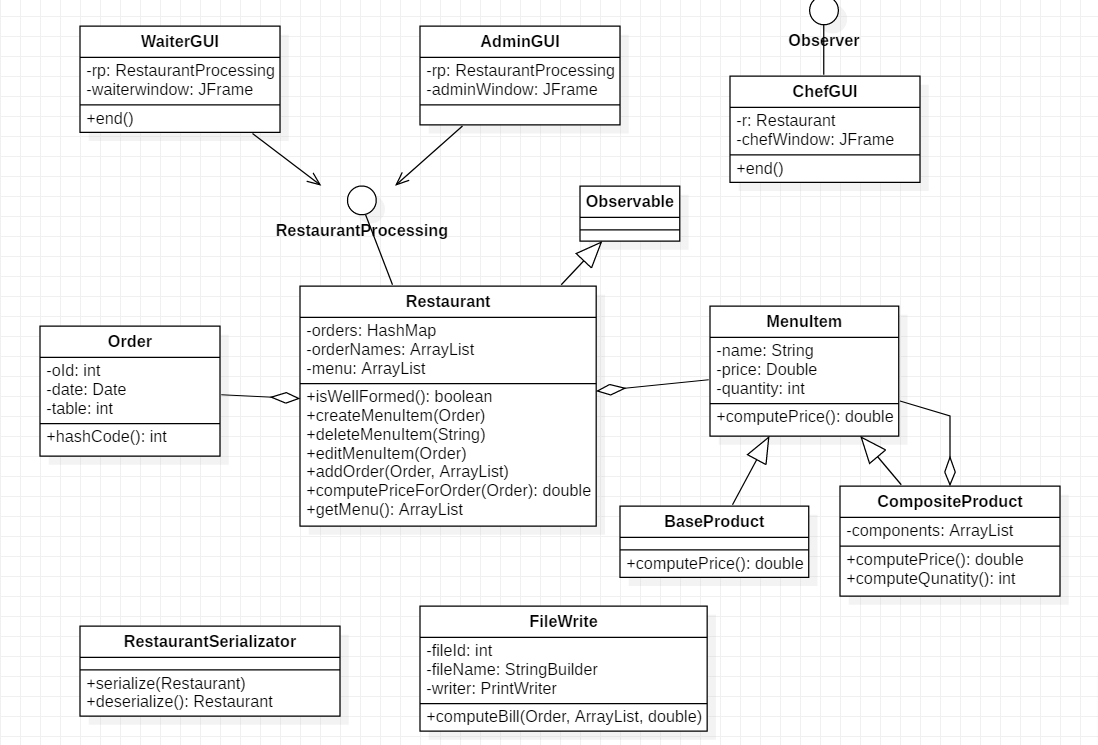
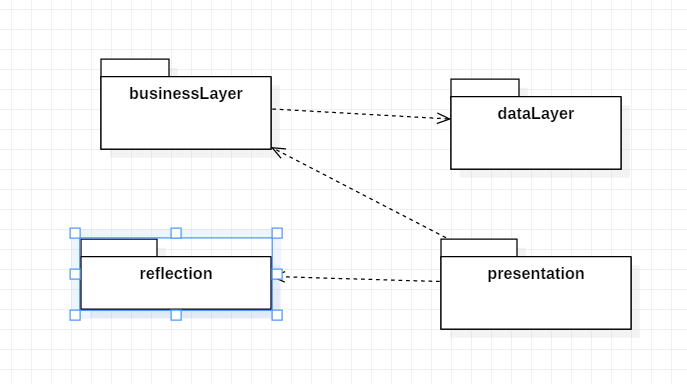


Diagrama UML de pachete de mai jos, surprinde organizarea aplicatiei despre care am tot vorbit mai sus. PresentationLayer este la suprafata, dependent de catre businessLayer, iar businessLayer continua dependenta pana la dataLayer.



Implementare

Acest capitol este dedicat implementarii aplicatiei. Deoarece aceasta aplicatie este dependenta de un obiect care contine un hashmap implementarea a fost mai costisitoare decat in general avand o structura ierarhica despre care am vorbit mai sus.

Obiectele principale ale acestei aplicatii sunt Restaurant, MenuItem si Order. In jurul acestora se „joaca” celelalte obiecte care au rolul de modelare a acestor obiecte. De asemenea, aceste obiecte sunt singurele reale, adica palpabile in lumea reala. Voi incepe cu descrierea implementarii acestor obiecte.

Obiectul Order contine atributele: oId, date, tabel. Acest obiect contine 2 constructori: unul cu toate parametrele necesare fiecarui atribut, iar cel din urma contine valori pentru toate atributele mai putin pentru id. Acest constructor din urma este folosit cel mai des, deoarece in implementarea aplicatie, cheia primara adica id-ul este auto-incrementat, asadar nu trebuie specificat in adaugarea comenzii in hashMap. Acest aspect este valabil si pentru celelalte clase. Order mai contine si get-ere si set-ere pentru fiecare atribut, care este privat, si se suprascrie metoda toString pentru afisarea prietenoasa a acestui obiect. Aceste aspecte sunt valabile si pentru MenuItem. Spre deosebire de aceasta, MenuItem contine atributele: name, price si quantity. De asemenea, dupa cum am mai spus MenuItem are 2 derivate: BaseProduct si CompositeProduct. Design pattern-ul composite a determinat ca acest CompositeProduct sa contina o lista de obiecte pe care le mosteneste adica MenuItem. De asemenea, MenuItem mai contine si o metoda de computePrice(), care sa va suprascrie in CompositeProduct. Prin urmare, Order va trebui sa aiba in lista sa de produse cerute, produse existente in meniul restaurantului.

Lasand modelul la o parte, putem incepe in descrierea ierarhiei de sus in jos. Asa cum am mai spus deja prezentarea contine trei clase. Cea principala este denumita AdminGUI. Aceasta contine doua component de tip JComboBox folosite pentru pentru a selecta operatia si meiul. De sus in jos se pot intorduce mai intai un BaseProduct, iar apoi avand o selectie a meniului se poate adauga inca un CompositeProduct cu baseProduct existente. Este necesar ca un compositeProduct sa fie format din cel putin 2 baseproduct. Din cele descrise mai sus rezulta ca este nevoie de doua butoane. Primul este folosit pentru a aduga un baseProduct in posibilul CompositeProduct, iar al doilea folosit pentru crearea unui CompositeProduct propriu-zis. Dupa cum le spune si numele acesti ascultatori apeleaza o metoda ori de cate ori butoanele corespunzatoare sunt apasate. Pentru metoda asculatatorului se vor verifica selectiile din fereastra principala de la cele doua componente JComboBox. In functie de selectia aleasa se vor realiza anumite operatii, dar nu inainte de a se extrage datele din JTextField-uri necesare. Doar in cazul operatiei de vizualizare se va crea o noua fereastra care va contine datele din tabelul corespunzator din baza de date. In cazul in care se semnaleaza erori legate de operatiile existente, exista un component de tip JLabel folosite pentru a arata utilizatorului aceste erori(acelasi pentru fiecare buton).

Coborand mai jos in ierarhie se ajunge la blocul de business. Aici se vor implementa metodele din restaurant descrise in interfara RestaurantProcessing. Aproape fiecare metoda foloseste un validator care verifica daca obiectul dat ca parametru respecta conditiile necesare pentru baza de date. Aici se foloseste Design by contract adica se vor folosi asserts pentru metode astfel incat sa se verifice daca comportamentul metodelor si argumentele sunt valide. Aceasta metoda poate arunca o exceptie numita, in cazul in care atributele obiectului nu sunt bune. Astfel in acest caz order este adaugat in hashmap impreuna cu toate obiectele pe care le-a folosit, adica o lista de MenuItem, iar utilizatorul va fi semnalat de acest lucru.

In cazul in care se doreste crearea unei comenzi, in WaiterGUI, se va putea selecta o masa pentru comanda respectiva. De asemenea, se va selecta o lista de produse impreuna cu cantitatea dorita pentru aceste produse. In caz de orice fel de erori utilizatorul va fi semnalat printr-un text afisat intr-un JLabel. Din WaiterGUI se poate crea chitanta pentru ultima comanda inserata in restaurant. Astfel, se va apela metoda statica din FileWriter care practic creeaza un fisier de tip text cu numele comenzii, produsele existente si pretul total. Desigur ca aceste comenzi se realizeaza incremental, cu un anume fileId. Toate aceste fisiere sunt salvate in aplicatie.

De asemenea, la orice creare de comanda ChefGUI, adica bucatarul este anuntat ca trebuie sa faca mancare, adica sa prepare pentru comanda data de chelner. Astfel, in ChefGUI va apare textul cu comanda noua creata.

Un aspect particular despre care nu am vorbit reprezinta metoda createTable din clasa Reflection care foloseste tehnici de reflection pentru a extrage campurile(Field) pentru o lista de obiecte, creand coloanele unui tabele cu numele acestori campuri, iar valorile existente in atributele fiecarui camp vor fi inserate in tabel pe coloana corespunzatoare. Aceasta implementare variaza de la Order la MenuItem. In cazul in care se apeleaza pentru Order se va omite campul static inc folosit doar pentru auto-incrementare, la fiecare creare de new Order(). Spre deosebire, pentru MenuItem se va dori accesarea campurilor din clasa mama, astfel se va mai adauga metoda getSupperClass() dupa metoda getClass(). Campurile se acceseaza astfel: obiectul „o” apeleaza metoda getClass(), iar apoi se apeleaza metoda getDeclaredFields() care va returna campurile corespunzatoare clasei obiectlui. Valoarea unui camp al unui obiect se acceseza astfel: campul „field” apeleaza metoda get() asupra obiectului de la care se doreste valoarea respectiva. Astfel, cu un tablou unidimensional cu numele coloanelor si inca un tablou bidimensional cu valorile pentru fiecare coloana si fiecare obiect din lista data ca parametru pentru metoda se va crea un JTable care va fi si returnat. Aceasta metoda va fi apelata in interfata grafica pentru operatia de vizualizare.

Rezultate

Rezultate bune(pe masura asteptarilor).

Concluzii

In concluzie, pot sa spun ca am dobandit cunostinte noi legate de posibilitatea de a salva precum intr-o baza de date intr-un hashmap cu cheie Order si lista de menuItem. De asemenea, toate tipurile de design pettern au fost ceva nou si interesant, iar serializarea a reprezentat cel mai interesant lucru nou invatat.

Imbunatatirile pe care as vrea sa le aduc sunt legate de buna organizare a acestei aplicati.

Bibliografie

* World Wide Web
* <https://stackoverflow.com>
* <https://www.tutorialspoint.com>
* <https://stackoverflow.com/questions/13901339/java-observer-update-function>
* <https://www.tutorialspoint.com/java/java_serialization.htm>
* <https://www.geeksforgeeks.org/iterate-map-java>