Documentație tema 4

Cuprins

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

Obiectivul principal al temei este crearea unui sistem de management al unui restaurant care să graviteze în jurul a trei entități: administratorul, care poate prelucra meniul restaurantului, chelnerul, care preia comenzile, și bucătarul, care este notificat de fiecare dată când o comandă este creată.

Pentru realizarea obiectivului principal al temei a fost necesară îndeplinirea următoarelor obiective secundare:

- crearea a 3 interfețe grafice, câte una pentru fiecare entitate menționată mai sus, prin intermediul cărora se pot prelucra informațiile legate de restaurant

- alegerea unor structuri de date potrivite pentru a stoca produsele din meniu și comenzile

- salvarea listei de produse din meniu, prin intermediul serializării; astfel, la următoarea utilizare a aplicației se vor încărca informațiile din momentul salvării

- crearea unui fișier text în momentul în care chelnerul dorește să realizeze nota; acest fișier conține informații despre comandă

- folosirea Composite Design Pattern pentru a crea clasele ce modelează tipurile de produse ce pot fi incluse în meniu

- folosirea Design By Contract pentru a construi clasa în care se realizează modificări asupra produselor din meniu și comenzilor

- folosirea Observer Design Pattern, prin care bucătarul este anunțat de chelener că o nouă comandă a fost adăugată

1. Analiza problemei

Meniul restaurantului este format din 2 tipuri de produse: base și composite. Aceste produse sunt caracterizate de un nume și preț. În plus, un produs composite poate conține, la rândul său, alte produse composite sau base. Astfel, un produs composite este caracterizat și de o listă de produse.

În ceea ce privește comenzile, acestea sunt caracterizate de un id, data la care au fost efectuate, masa de la care a fost preluată si lista de produsce comandate.

Administratorul poate crea, șterge și edita produse din meniu, iar chelnerul poate crea o comandă, poate calcula și genera nota de plată pentru o comandă.

Rularea aplicației duce la apariția pe ecran a 3 interfețe. Fiecare interfață asigură funcționalitatea cerută pentru entitatea pe care o reprezintă.

Astfel, pentru adminstrator se remarcă următoarele condiții de utilizare:

- pentru crearea unui produs base se introduc informațiile cerute în chenare, nume și preț, după care se apasă butonul; în urma apăsării butonului se va observa modificarea tabelului din partea dreapta

- pentru crearea unui produs composite se introduce numele produsului în chenar, după care se apasă butonul; în tabel apare produsul, însă fără niciun produs component

- pentru modifcarea listei de produse dintr-un produs composite se folosesc următoarele 2 secțiuni din interfată: astfel, pentru adăugare se alege din primul drop list produsul composite, din al doilea drop list se alege din lista completă de produse produsul ce se dorește a fi adăugat, după care se apasă butonul; pentru ștergere funcționalitatea este similară, doar că în al doilea drop list vor apărea doar produsele care se află în componența produsului composite ales

- pentru editarea unui produs se alege din primul drop list produsul ce se dorește a fi eliminat, din al doilea drop list se alege câmpul ce va fi modificat, nume sau preț, în chenar se va introduce noua valoare a câmpului

- pentru ștergerea unui produs se selectează din drop list-ul ce conține toate produsele din meniu produsul ce se dorește a fi șters.

Pentru interfața chelnerului se remarcă următoarele condiții de utilizare:

- pentru crearea unei comenzi se introduc id-ul, data, masa, se selectează din drop list-ul ce conține toate produsele din meniu produsele din comandă, după care se apasă butonul additem, în urma căruia se va adăuga în chenar produsul selectat

- pentru calcularea prețului se introduc câmpurile comenzii pentru care se dorește calcularea notei; în urma apăsării butonului va apărea langă acesta valoarea calculta

- pentru generarea fișierului text ce conține informații despre o comandă se introduc valorile câmpurilor precizare mai sus

Interfața bucătarului conține un chenar care cuprinde informații despre comenzile primite de la chelner.

1. Proiectare

Proiectul este alcătuit din 3 pachete: data, business și presentation.

Pachetul data conține 2 clase: RestaurantSerializator, cu ajutorul căreia se vor salva informațiile din resaturant și se vor putea încărca ulterior informațiile salvate, și FileWriter prin care se va genera fișierul pentru o comandă.

Pachetul business conține partea de procesare a informațiilor din restaurant. Interfața IRestaurantProcessing conține operațiile ce pot avea loc pentru restaurant. Clasa Restaurant implementează aceste operații. Clasa Order descrie caracteristicile unei comenzi. Clasa MenuItem descrie caracteristicile unui produs din meniu. Din această clasă se extind două clase: BaseProduct și CompositeProduct.

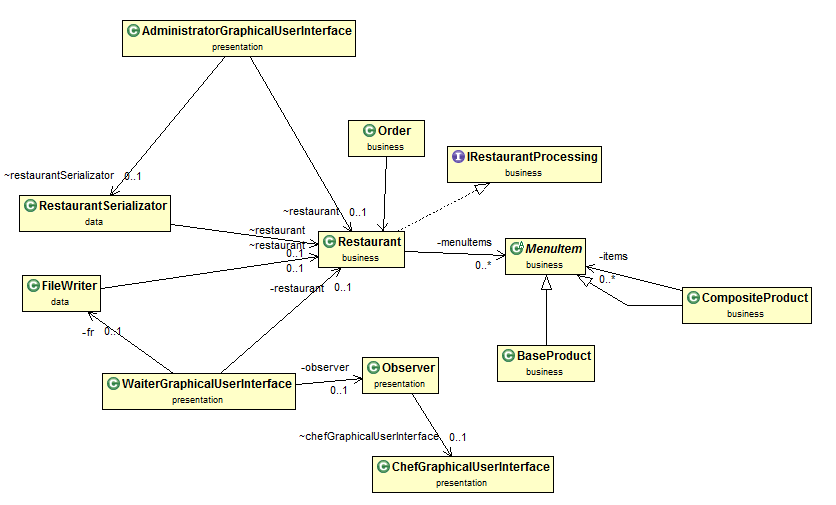
Pachetul presentation conține clasele ce definesc interfețele graficele.

Pentru proiectarea clasei Resaturant s-a folosit patternul Design By Contract, prin folosirea instrucțiunii assert. Astfel pentru fiecare funcție se definesc pre-condiții și post-condiții. Prin acest procedeu se asigură integritatea datelor obținute.

Pentru proiectarea claselor MenuItem, BaseProduct și CompositeProduct s-a folosit pattern-ul Composite Design. Astfel, clasele BaseProduct și CompositeProduct extind MenuItem, iar CompositeProduct conține o listă de MenuItem.

Interfața bucătarului se modifică prin folosirea pattern-ului Observer Design, cu ajutorul căruia ori de câte ori se creează o comandă prin intermediul interfeței clientului, interfața bucătarului afișează comanda primită.

Pentru stocarea produselor din meniu s-a folosit un ArrayList de obiecte MenuItem. Aceeași structură a folosit pentru stocarea produselor componente ale unui produs composite. Pentru stocarea comenzilor s-a folosit un HashMap, în care key-ul este calculat pe baza unui obiect Order, iar valoarea este dată de un ArrayList de obiecte MenuItem. În clasa Order s-au suprascris funcțiile hashCode() și equals().



1. Implementare

Pachetul business conține clasele BaseProduct, CompositeProduct, MenuItem, Order, Restaurant și interfața IRestaurantProcessing.

Clasa MenuItem este de tip abstract și implementează Serializable pentru a permite salvarea meniului în interfața administratorului. Are două câmpuri, nume și price, și conține metode de getter și setter pentru acestea. Funcția equals verifică egalitatea dintre două produse. Întrucât două produse nu pot avea același nume, ele pot fi egale doar dacă au același nume. Clasele BaseProduct și CompositeProduct extind clasa MenuItem. Clasa CompositeProduct are în plus implementată funcția computePrice(), în care i se calculează prețul, acesta obținându-se prin adunarea prețurilor produselor componente.

Clasa Order descrie caracteristicile unei comezi. Conține câmpurile id, date și table, pentru care există metode de get și set. În plus, acestă clasa conține metode suprascrise pentru hashCode() și equals(). Întrucât comenzile restaurantului vor fi introduse intr-un hashMap în care key-ul este dat de un obiect de tip Order, a fost necesară implementarea proprie a clasei Order a funcțiilor hashCode() și equals().

Interfața IRestaurantProcessing conține definiția operațiilor suportate de administrator și chelner. Functțiile createNewItem, deleteMenuItem și editMenuItem sunt suportate de administrator, iar funcțiile createOrder, computePrice și generateBill sunt suportate de către chelner.

Clasa Restaurant implementează interfața IRestaurantProcessing. Conține două câmpuri: orders, de tip HashMap în care se stochează informațiile despre comenzi, și menuItems, de tip ArrayList, în care se stochează informațiile despre produsele din meniu. Implementarea acestei clase s-a realizat cu pattern-ul Design By Contact. Astfel, s-a creat o metodă de tip ”well formed”, numită invariant(), în care se verifică dacă cele două câmpuri de mai sus sunt nule sau nu. Această funcție se va apela la începutul fiecărei funcții din această clasă prin intermediul instrucțiunii assert, întrucât fiecare funcție trebuie să indeplinească, inainte de rulare, condiția ca cele două structuri sa nu fie nule.

Pattern-ul Design by Contact presupune verificarea unor condiții pre si post rulare pentru o funcție. Acest lucru se poate realiza prin intermediul instrucțiunilor de tip assert. Pentru a permite executarea instrucțiunilor assert este necesară adăugarea parametrului -ea atunci când se va rula din linia de comanda fișierul .jar.

Funcția createNewItem primește ca parametru un obiect de tip MenuItem, pe care îl adaugă in meniu. Funcția apelează la început metoda invariant, după care verifică dacă obiectul primit ca parametru este diferit de null. După adăugarea in listă, funcția se asigură ca marimea listei a crescut cu unu în urma execuției.

Funcția addItemtoComposite primește ca paramentri un obiect de tip CompositeProduct și un obiect de tip MenuItem. Funcția adaugă în lista de produse a obiectului de tip CompositeProduct obiectul de tip MenuItem. Funcția verifică la început condiția invariant(), după care verfică dacă cele doua obiecte primite ca parametru sunt diferite de null. Dupa ce se adaugă obiectul în listă, se verifică dacă marimea listei de produse a obiectului de tip CompositeProduct a crescut unu.

Funcția removeItemFromComposite primește ca parametri un obiect de tip CompositeProduct și un obiect de tip MenuItem. Funcția șterge din lista de produse a obiectului composite obiectul de tip MenuItem primit ca parametru. Funcția verifică la început condiția invariant(), după care verifică dacă cele două obiecte primite ca parametru sunt diferite de null. După executarea ștergerii, funcția se asigură ca lista de produse a produsului composite a pierdut un element.

Pentru ștergerea unui element din meniu am decis că pe lângă ștergerea produsului propriu-zis se vor șterge și obiectele composite care îl conțin. Pentru o ștergere de acest fel, a fost nevoie sa implementez o funcție ce caută recursiv un produs într-un composite, findItemInComposite, care primește ca parametru un obiect composite și un obiect MenuItem. Se verifică la început condiția invariant și faptul că cele două obiecte primite ca parametru sunt diferite de null. Această funcție verifică prin lista de produse a obiectului composite dacă unul dintre produse este egal cu obiectul MenuItem primit ca parametru. Dacă se găsește, se returnează true, iar dacă obiectul la care s-a ajuns este composite se apelează recursiv funcția, primind ca parametru obiectul composite la care s-a ajuns. Dacă nu se găsește obiectul dorit se returnează false. Acestă funcție va fi apelată în funcția de ștergere.

Funcția deleteMenuItem șterge produsul primit ca parametru. Se verifică invariantul și dacă lista de produse are elemente de șters, deci dacă are marimea mai mare sau egală cu 1. Funcția caută prin listă obiectele composite pentru a se apela funcția findItemInComposite asupra lor. Pentru fiecare composite la care funcția de find returnează true, acesta se va șterge. Abia după ce posibilele ștergeri s-au produs, se va căuta produsul de șters în listă și se va șterge. Funcția se asigură la final ca lista de produse și-a micșorat numărul de elemente.

Funcția editMenuItem primește ca parametri un MenuItem, un String ce denotă câmpul pentru care se dorește modificarea și un String ce denotă noua valoare ce trebuie introdusă. Întrucât un produs are două proprietăți, nume și preț, se verifică dacă Stringul ce denotă câmpul are una din cele două valori.

Funcția createOrder creează o comandă cu ajutorul parametrilor și o adaăuga in map. Funcția computePrice() calculează prețul total al unei comenzi primite ca parametru. Funcția generateBill() generează un fișier text ce conține informații despre o comandă primită ca parametru.

Pachetul presentation conține clasele interfețelor.

Clasa AdministratorGraphicalUserInterface creează interfața ce trebuie folosită de administrator.

Pentru fiecare operație ce poate fi întreprinsă de administrator s-a creeat o secțiune pe ecran, iar rezultatul propriu-zis al operației apare în urma apăsării unui buton. Fiecare action listener a fost implementat în câte o clasă interioară clasei acestei interfețe. Pe lângă aceste clase, au fost create niște funcții ajutătoare pentru a actualiza anumite elemente de pe interfață ce suferă modificări în urma unor operații. Funcția actualizareTabel() redesenează tabelul, adăugându-se in el produsele din meniu la momentul apelării acetei funcții. De aceea, această funcție este apelată la finalul funcțiilor din clasele unde este descrisă acțiunea produsă de butoane prin care se ajunge la modificarea listei de produse. Funcția selectCompositeItems() actualizează lista de produse composite ce apare în secțiunile unde se poate adăuga sau șterge un element dintr-un composite. Această funcție este folosită la finalul funcției unde este implementată adăugarea unui composite în urma apăsării butonului corespunzător și la finalul funcției de ștergere a unui produs, astfel încât, dacă este șters un composite, acesta să dispară din cele doua liste menționate mai sus. Funcția itemsList() este asemănătoare, însă se adresează produselor de ambele tipuri. Aceasta rescrie lista ce apare atunci când administratorul dorește să adauge un produs într-un composite, lista ce apare atunci când dorește să șteargă un element, lista ce apare atunci dorește să editeze un produs. Totodată, în urma operațiilor se modifică și lista produselor ce apare atunci când chelnerul dorește să adauge produse într-o comandă.

Clasa WaiterGraphicalUserInterface creează interfața ce trebuie folosită de chelner. Are același schelet precum interfața pentru administrator, doar că va cuprinde alte operații, cele ce pot fi suportate de către chelner. În plus, apare un obiect de tip Observer, care va actualiza interfața bucătarului la fiecare adăugare a unei comenzi.

Interfața bucătarului este alcătuită dintr-un chenar unde vor apărea comenzile pe care trebuie să le prepare.

Clasa RestaurantSerializator primește ca parametru în constructor un obiect de tip Restaurant. Funcția save() salvează într-un fișier denumit restaurant.ser produsele din meniu. Această funcție este folosită la implementarea acțiunii butonului save din interfața administratorului. Funcția load(), ce primește ca parametru fișierul în care s-au salvat informațiile, va seta lista de produse a restaurantului cu lista conținută în acel fișier. Această funcție este apelată in clasa Main, de unde se va rula progamul.

1. Concluzii

Această aplicație simulează activitatea unui restaurant. Aceasta poate fi folosită de trei utilizatori: administratorul, chelnerul și bucătarul, pentru fiecare creându-se câte o interfață grafică. Aministratorul poate crea, edita și șterge produse din meniu, chelnerul poate crea comenzi și să calculeze nota pentru o anumită comandă, iar bucătarul este atențional de fiecare creare a unei comenzi. Pentru atingerea funcționalității dorite s-au folosit diferite desing pattern-uri, cum ar fi Composite Design Pattern, Observer Design Pattern, Design by Contract. Importantă a fost alegerea structurilor de date pentru stocarea produselor din meniu și pentru stocarea comenzilor. Pentru produsele din meniu am ales ArrayList, iar pentru comenzi HashMap.

1. Bibliografie

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

<http://javarevisited.blogspot.ro/2011/02/how-hashmap-works-in-java.html>

<https://intellij-support.jetbrains.com/hc/en-us/community/posts/207014815-How-to-enable-assert>

https://intellij-support.jetbrains.com/hc/en-us/community/posts/207014815-How-to-enable-assert