MINISTERUL EDUCAȚIEI NAȚIONALE

UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

FACULTATEA DE AUTOMATICA SI CALCULATOARE
SPECIALIZAREA CALCULATOARE SI TEHNOLOGIA INFORMATIEI

TEHNICI DE PROGRAMARE

- FOOD DELIVERY MANAGEMENT SYSTEM -

MURESAN GEORGIANA ROBERTA
GRUPA 30223 2020-2021

Cuprins

1. OBIECTIVUL TEMEI

Obiectivul principal al lucrarii e realizarea unei aplicatii ce simuleaza cat se poate de bine managementul unei aplicatii de food delivery. Chiar daca acest model e unul simplificat, el trebuie sa fie cat se poate de corect si apropiat de cazul real al prelucrarii datelor clientilor si ale restaurantului, pentru a putea oferi utilizatorului, adica clientului o intelegere in profunzime si o folosire usoara a aplicatiei.

Lucruruile mentionate mai sus sunt folosite si in interfata grafica, care de altfel este destul de complexa si bine definita, totodata usor de folosit, utilizatorul fiind indrumat la fiecare pas cu ce poate sau nu sa faca. Intreg proiectul a fost scris in limbajul Java. Au fost create mai multe clase, in special la interfata, pentru a putea sa fie usor de utilizat de catre client, organizate in mai multe clase, pachete si clase care vor fi descrise in cele ce urmeaza. Cu ajutorul acestora se simuleaza activitatea unui singur administrator, unui singur angajar si mai multor clienti.

Ca obiectiv secundar, această misiune a implicat:

1. Utilizarea modelului de design compozit
   - Această parte este descrisă în partea de implementare, și anume secțiunea 3
   - Acest model de proiectare constă din 2 clase și o interfață și a fost utilizat pentru clasele BaseProduct și CompositeProduct, bot implementând interfața MenuItem
   - Important de menționat că numărul de produse dintr-un produs compozit este 4 (aceasta a fost o decizie de implementare asumată de autor)
2. Implementarea interfeței IDeliveryService interface
   - Care conține toate operațiunile care pot fi efectuate pe datele restaurantului și care sunt implementate în clasa DeliveryService
3. Definirea clasei DeliveryService
   - Aceasta conține operațiunile care pot exista în sistemul de gestionare
   - În implementarea acestui punct, au fost identificate mai multe obiective
4. Folosind HashMap
   - HashMap a fost folosit pentru avantajul accesului rapid la memorie, și anume O (1)
   - Pentru a funcționa, este necesar ca unele metode să fie suprascrise
5. Crearea unei interfețe grafice pentru utilizator
   - A fost practicat înainte, într-o misiune anterioară
6. Utilizarea serializării
   - Este folosit pentru stocarea și încărcarea datelor despre restaurant, și anume meniul și comenzile plasate într-un fișier între rulările succesive ale aplicației

O parte foarte importanta din acest proiect o reprezeinta serializarea, asa ca o voi defini mai jos inainte de a trece mai departe:

Serializare= transformarea unui obiect intr-o secventa de octeti, din care se poate sa fie refacut ulterior obiectul serializat.

Procesul invers, de citire a unui obiect serializat pentru a-i reface starea originala, se numeste **deserializare**.

Referintele care construiesc starea unui obiect formeaza o intreaga retea de obiecte.

- DataOutputStream, DataInputStream
- ObjectOutputStream, ObjectInputStream

2. Analiza problemei, modelare, cazuri de utilizare

Aceasta tema simuleaza un sistem folosit in zilele noastre de majoritatea restaurantelor. In realizarea acestei teme trebuie sa luam in considerare mai multe scenarii.

Pentru administrator: daca se doreste introducerea unui produs care deja exista in meniu: atunci acesta va primii un mesaj de forma:"produsul deja exista",daca se doreste stergerea unui produs care nu exista atunci nu se va modifica cu nimic lista de produse.

Pentru creare de cont: daca exista contul nu se creeaza din nou, daca nu exista contul atunci utilizatorul nu poate da LOGIN.

Pentru operatiile cu produse: add, delete, modify: daca nu exista produsele respective, nu poate avea loc operatia.

Meniul va contine doua tipuri de produse: produse de baza si produse compuse. Waiter-ul va fi notificat doar daca in comanda exista produse compuse(deoarece am considerat ca produsele care necesita o anumita pregatire sunt produse compuse).

După cum este descris în secțiunea anterioară, aplicația poate trece prin aceste scenarii. Utilizatorul execută mai întâi aplicația, apoi apare un cadru care conține un mesaj de întâmpinare și trei butoane: ADMINISTRATOR, CLIENT, WAITER. Utilizând butoanele, utilizatorul selectează modul în care dorește să se „conecteze" si isi introduce USERNAME-UL si PAROLA. După selectarea tipului de utilizator, se deschide un nou cadru, corespunzător tipului de utilizator. În acel cadru, există componente de interfață care permit utilizatorului să își execute operațiunile.

Administratorul are optiunile urmatoare:

- Import products – o fereastra de SUCCES va aparea daca nu exista erori.
- DailyMenuOperations – o fereastra unde administratorul poate crea Daily Menus cu ajutorul ComboBox-urilor, unde I se afiseaza intr-un Jtable produsele compuse deja existente si poate sa stearga la alegere produse.
- Generate reports – I se cere administratorului sa introduce datele specifice acestor rapoarte.
- Products operations – se deschide o fereastra unde administratorul poate sa faca operatii de add, delete, modify si poate vizualiza produsele existente.

Clientul are optiunile urmatoare:

- Poate vizualiza produsele si daily menus in cadrul unor tabele, poate folosi optiunile de search pentru a cauta produsul in functie de preferintele sale, poate adauga produse in “basket”-ul lui, poate vizualiza cosul si vedea pretul total si, intr-un final, poate da send la comanda.

Waiter – aici nu s-au implementat caracteristicile dorite.

3. Proiectare (decizii de proiectare, diagrame UML, 7nteracti de date, proiectare clase, interfete, relatii, packages, algoritmi, interfata utilizator)

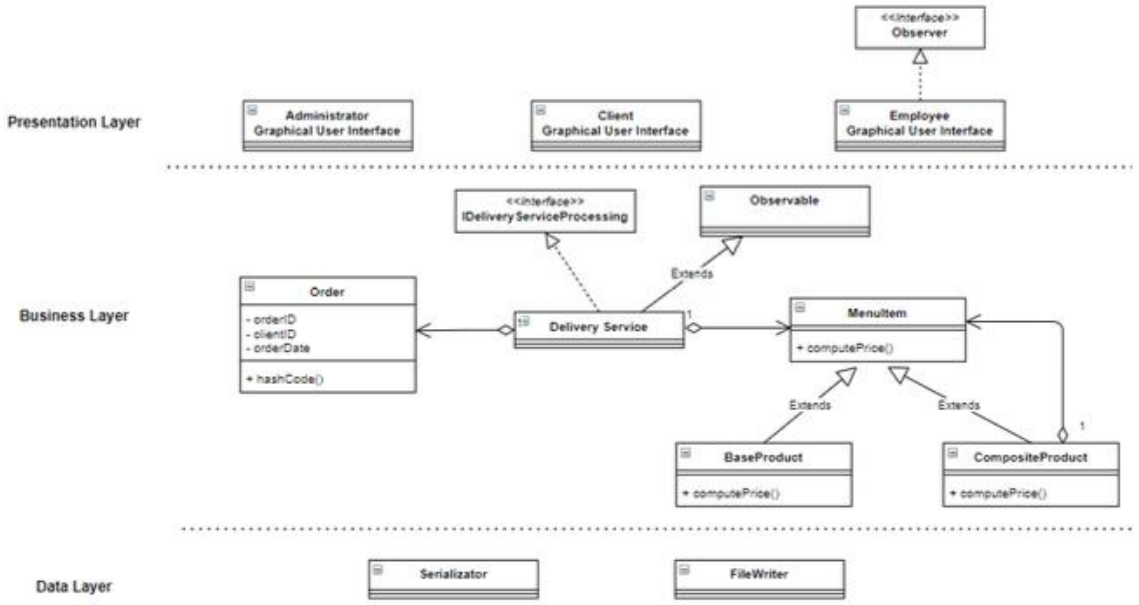


Figure 1: Class diagram to be considered as starting design of the system.

Figura de mai sus reprezintă punctul de plecare pentru proiectarea aplicației. Există trei straturi, care sunt în mod concret trei pachete. A fost adăugat un pachet suplimentar pentru a conține clasa cu metoda principală. Această metodă conține definiția pentru primul cadru care este deschis și ascultătorii de acțiune pentru cele trei butoane care deschid celelalte cadre și, de asemenea, invocarea metodelor de inițializare pentru aceste cadre.

Începând de jos, stratul de date conține clasele necesare pentru scrierea datelor în fișierul .csv (și crearea fișierului dacă este cazul). Serializator oferă, de asemenea, punctul de plecare al aplicației, deoarece creează obiectul clasei Restaurant pe care funcționează întreaga aplicație.

Stratul Business conține clase și interfețe care reprezintă date pentru lucrul efectiv în interiorul restaurantului, deoarece există toate componentele a 8nterac înseamnă de fapt un restaurant în acest context.

Aici, modelul de proiectare compozit a fost utilizat 8nteracti clasele BaseProduct, CompositeProduct și interfața MenuItem. Acestea se facilitează având în vedere că există doar un tip de element de meniu și este stocat în colecțiile de restaurante, dar poate fi de fapt un produs de bază sau un produs compozit, produsul de bază este o formă simplă care conține doar numele și prețul, iar produsul compozit se bazează pe o colecție (în acest caz ArrayList) în care pot fi stocate multe elemente de meniu. Aceste elemente de meniu pot fi de fapt fie Produse de bază, fie Produse 8nteracti. De fapt, interfața MenuItem are metodele necesare, care sunt, de asemenea, implementate de clase, care sunt 8nteractiv pentru a ține evidența produselor din meniu sau într-o comandă, fără a fi cu adevărat importante dacă este un produs de bază sau un produs compozit.

Modelul de proiectare a observatorului este de asemenea 8nterac și implică clasele Restaurant și Order și interfețele Observer și IrestaurantProcessing. În DeliveryService, metoda de notificare este apelată în metoda în care este 8ntera o nouă comandă. Această notificare constă în adăugarea listei de articole din ordine la lista de articole a bucătarului, articolele pe care trebuie să le gătească. Acest lucru se face prin metoda de actualizare, care se numește în notificareChef.

În cele din urmă, există stratul de prezentare care conține cadrele corespunzătoare celor trei tipuri de utilizatori. Deoarece sunt clase GUI, au 8nteract de 300 de linii de cod și conțin metode cu 8nteract de 30 de linii. Toate componentele care pot fi văzute pe cadre au fost create și adăugate într-un anumit loc din cadru. Aceasta înseamnă că metoda setBounds a fost utilizată pentru toate aceste componente.

4. IMPLEMENTARE

Această secțiune este dedicată codificării reale și prezentării claselor care compun aplicația și, de asemenea, anumitor decizii de implementare. Unul dintre acestea a fost menționat, dar merită discutat din nou este că un produs compozit poate avea maximum patru componente. Acest lucru s-a făcut gândindu-ne la o situație realistă și, de asemenea, la ușurința implementării, deoarece a fost mai greu să creezi un tabel cu toate produsele din meniu, fără să știi numărul coloanelor sale, deci s-a redus la nume, preț, element 1, element 2, item3 și item4.

Clasele BaseProduct, CompositeProduct, Order și interfețele IdeliveryService și ImenuItem nu prezintă nicio implementare specială. În principal, acestea conțin operațiunile, metodele, getters și getters care sunt, de asemenea, prezente în cerințele misiunii.

DeliveryService Class va fi discutat în paragrafele următoare.

Această clasă implementează interfața IdeliveryService, deci implementează metodele definite acolo. Ca câmpuri, aici sunt prezente două statice care stochează meniul și comenzile. Sunt statice, deoarece în aplicație există un singur meniu și un singur pachet de comenzi, deci în întreaga aplicație nu va exista decât o singură instanță din clasa restaurant.

```
Public boolean WellFormed(){
    if(menu == null)
        return false;
    if(dailyMenu == null)
        return false;
    if(ordersInformations == null)
        return false;
    return true;
}
```

In metoda ImportProducts():

```
@Override
public List<MenuItem> ImportProducts()
```

In aceasta metoda dau import la produsele din fisierul .csv si cu ajutorul stream-ului fac split.

```
List<MenuItem> menuItems =
br.lines().skip(1).map(mapToBaseProduct).collect(collectingAndThen(toCollec
tion(() -> new TreeSet<>(comparing(MenuItem::getTitle))),
        ArrayList::new));
```

Tot aici se poate observa partea de “Design by contract: preconditions and postconditions” alaturi de assert :

```
/**
 *   @pre product != null
 *   @pre product.price > 0
 */
@Override
public void AddProduct(MenuItem product) {

}

/**
 *   @pre product != null
 *   @pre product.price > 0
 *   @post menu.size() = @pre menu.size() - 1
 */

@Override
public void DeleteProduct(MenuItem product) {

}
/**
 *   @pre product != null
 *   @pre product.price > 0
 */

@Override
public void ModifyProduct(MenuItem product) {

}

/**
 *   @pre productName1 != null
 *   @pre productName2 != null
 *   @pre productName3 != null
 *   @pre productName4 != null
 */
@Override
public void CreateDailyMenu(String productName1, String productName2,
```

```
String productName3, String productName4) {

}

@Override
public void GenerateReports() {

}

/**
 *   @pre order != null
 *   @pre order.id > 0
 */
@Override
public void CreateNewOrder(Order order) {

}
```

Serializator – aceasta clasa este una foarte importanta, cu ajutorul ei se fac toate procesele de serializare si deserializare, de la serializarea produselor de baza si composite product, pana la datele administratorului, clientilor, meniuri, orders si deliveryService.

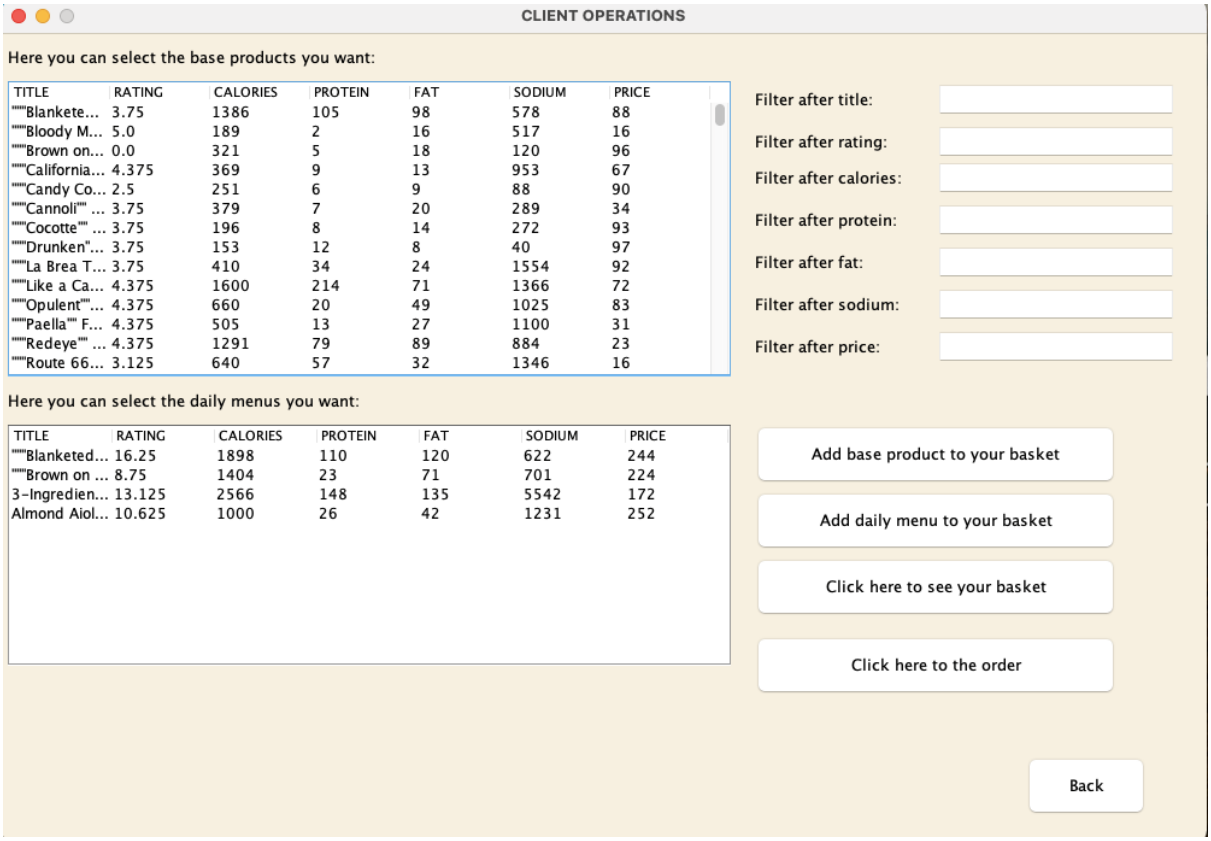
Un exemplu ar fi urmatorul:

```
public static void DeliveryServiceSerialization(DeliveryService
deliveryService, String filename){

    try {
        FileOutputStream fileOutputStream
                = new FileOutputStream(filename);
        ObjectOutputStream objectOutputStream
                = new ObjectOutputStream(fileOutputStream);
        objectOutputStream.reset();
        objectOutputStream.writeObject(deliveryService);
        objectOutputStream.close();
        fileOutputStream.close();

        System.out.println("The delivery service has been serialized");

    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

Revenind la partea de import Products, am definit clasele BaseProduct si Composite product cu urmatorele field-uri: title, rating, calories, proteins, fats, sodium, price.

```
Public BaseProduct(String title, float rating, int calories, int protein,
int fat, int sodium, int price)
```

Legat de partea de interfata, am ales sa am una interactiva, cu multe Jframe-uri si prezenta Jtable-urilor si a JcomboBox-urilor pentru o interactiune mai buna cu utilizatorul:



5. REZULTATE

Precum am mentionat la inceput, scopula acestui proiect este de a realiza o implementare a unui sistem de management al livrării alimentelor pentru o companie de catering, obiectiv care este dus aproape pana la capat.

Toate operatiile administratorului sunt prezente si functioneaza la calitate maxima, iar cat despre client, acesta se bucura nu doar de o comanda care poate fi dusa la capat si o complexitate ridicata atunci cand vine vorba de produsele oferite, ci si de o interfață primitoare care ii usureaza procesul prin care trece.

Proiectul ar putea fi imbunatatit daca s-ar implementa pana la capat caracteristicile specific chelnerului.

6. CONCLUZIE

Ca si incheiere, as spune ca pana acum a fost cea mai interesanta tema, mi-a placut pentru ca era putin mai subiectiva decat celelalte. Intr-adevar am muncit mai mult ca si pana acuma, dar a meritat, deoarece am invatat mai multe despre OOP, ceea ce imi va fi de folos in continuare.

Ca si implementari ulterioare, in ceea ce priveste simularea aplicatiei, ar putea sa existe un sistem de logare pana si pentru administratori si angajati, putand exista mai multi angajati care sa prepare comenzile primite de la diversi clienti. As fi putut sa implementez stergerea comenzilor, sau stergerea componentelor dintr-un produs compus.

7. Bibliografie

[1] http://javahungry.blogspot.com/2013/08/hashing-how-hash-map-works-in-java-or.html
[2] http://javahungry.blogspot.com/2014/03/hashmap-vs-hashtable-difference-with-example-java- interview-questions.html
[3] http://javahungry.blogspot.com/2013/08/how-sets-are-implemented-internally-in.html
[4] https://en.wikipedia.org/wiki/Red%E2%80%93black_tree
[5] https://en.wikipedia.org/wiki/Hash_table
[6] https://www.baeldung.com/java-serialization
[7] https://www.geeksforgeeks.org/serialization-in-java/
[8] https://docs.oracle.com/javase/8/docs/api/java/io/Serializable.html