**FACULTATEA DE AUTOMATICĂ ȘI CALCULATOARE**

**DEPARTAMENTUL CALCULATOARE**

**DOCUMENTATIE**

**TEMA 4**

**An academic:**

**2021-2022**

**NUME STUDENT: VASILACHE ANDREI-FLORIN**

**GRUPA: 30223**

**CUPRINS**

1. Obiectivul temei ..................................................................................................................................................... 3

2. Analiza problemei, modelare, scenarii, cazuri de utilizare .................................................................................... 3

3. Proiectare ............................................................................................................................................................... 4

4. Implementare ......................................................................................................................................................... 6

5. Rezultate ................................................................................................................................................................ 9

6. Concluzii ................................................................................................................................................................ 10

7. Bibliografie ............................................................................................................................................................ 10

1. **Obiectivul temei**

Obiectivul acestei teme a fost de a crea o aplicatie de delivery pentru o companie de catering. In aceasta aplicatie a fost nevoie sa folosim doua design patterns: Composite Design Pattern si Observer Design Pattern.

Obiectivul secundar al acestui proiect este:

* de a crea clasa DeliveryService care este esentiala pentru acest proiect;
* de a crea interfetele grafice de utilizator pentru Administrator, Client si Employee;
* de a deserializa produsele la deschiderea aplicatiei si serializarea produselor la inchiderea acesteia;
* de a crea comenzile comandate de client;
* de a crea bonurile la cererea clientului cu toate produsele comandate si pretul total de platit;
* de a implementa functionalitati de cautare a produselor dupa anumite criterii (dupa nume, calorii, rating, pret, sodiu etc).



1. **Analiza problemei, modelare, scenarii, cazuri de utilizare**

In aceasta aplicatie se poate simula in interfata grafica aceasta aplicatie de menegeriere a comenzilor, dar cu cateva reguli (care vor fi prezentate in capitolele urmatoare).

Descrierea use-case-urilor pentru simulare in GUI:

* Se deschide aplicatia;
* Se ruleaza metoda main din clasa LoginGUI din pachetul presentation;
* Se completeaza campurile cu administrator/administrator pentru a se face logarea de administrator sau cu client/client pentru a se face logarea cu client.
* Administrator:
* pe butonul de "ADD BASE PRODUCT" se apasa doar cand toate campurile au fost completate pentru a putea adauga un produs de baza;
* pe butonul de "ADD COMPOSITE PRODUCT" se apasa doar daca s-au selectat unul sau mai multe produse din lista prezenta in interfata grafica;
* pe butonul de "DELETE PRODUCT" se apasa doar daca se selecteaza un singur produs;
* pe butonul de "MODIFY" se apasa doar dupa ce s-a selectat un produs de baza si toate campurile au fost completate, in campul pentru nume introducandu-se acelasi nume al produsului.
* Client:
* pe butonul de "ORDER" se apasa doar cand au fost selectate unul sau mai multe produse din lista prezenta;
* pe butonul de "BILL" se apasa daca clientul este sigur daca a efectuat cel putin o comanda;
* pe butonul de "SEARCH" se apasa daca se doreste cautarea unor produse ce au ca si campuri ce este completat in unul din TextField-uri;
* pe butonul de "View all" se apasa daca se doreste vizualizarea tuturor produselor (de obicei este folosit dupa ce se efectueaza o cautare).

1. **Proiectare**

**Diagrama de clase:**

Unified Modeling Language (UML) este un limbaj standard pentru descrierea de modele si specificatii software. Este folosita pentru reprezentarea vizuala a claselor si a inter-dependentelor si a relatiilor de multiplicitate dintre ele. Diagramele de clasa sunt folosite si pentru reprezentarea concreta a unor instante de clasa, asadar obiecte, si a legaturilor dintre acestea.

* Prima decizie pe care am luat-o a fost sa ma ajut de prezentarea oferita de domul profesor;
* Dupa am implementat clasele prezentate si am urmat instructiunile;
* Dupa am implementat restul claselor prezente;
* Mi-am creat toate metodele pe care le folosesc in clasa DeliveryService;
* Mi-am creat cele 3 clase pentru interfata grafice pentru utilizator;
* Intr-un final mi-am creat JavaDoc-ul, mi-am pus pre si post conditiile si dupa invariatiile.

**Structuri de date:**

Tema fiind facuta in Java, aceasta dispune de anumite clase, interfete si colectii, deoarece este un limbaj de tip OOP (Object Oriented Programming). Exista 15 clase in total: BaseProduct, Client, CompositeProduct, DeliveryService, MenuItem, Order, AdministratorGUI, LoginGUI, etc. Acestea ajuta la inglobarea design-ului intregii aplicatii. Fiecare clasa extinde JFrame, acestea fiind customizate după placul programatorului.

Obiectul de tip Frame reprezinta fereastra in care se afla proiectul, iar cel de tip Panel este ceea ce se afla in interiorul ferestrei, adica zona de butoane, casete de text si etichete. Butoanele sunt obiecte de tip JButton, fiecare are un rol bine stabilit prin intermediul functionalitatii oferite de clasele ce implementeaza interfata ”ActionListener”. Butoanele au denumiri sugestive pentru a interactiona cu interfata mult mai usor si placut pentru orice tip de utilizator.

De asemenea, structura de date cea mai utila in acest proiect a fost hash table-ul, pentru a stoca datele produselor si a le accesa mult mai eficient. Nici listele nu au fost uitate, fiind folosite pentru a memora lista de produse, cat si clientii.

**Relatii:**

* Relatie de agregare intre: DeliveryService si Order;
* Relatie de dependenta intre: DeliveryService si IdeliveryServiceProcessing, Observer si EmployeeGUI;
* Relatie de mostenire intre: DeliveryService si Observable, MenuItem si BaseProduct & CompositeProduct.

Mai jos am atasat diagrama UML reprezentativa proiectului avand cele 3 pachete, fiecare avand clasele lor specifice:

A picture containing text

Description automatically generated

**Algoritmi:**

Algoritmii utilizati in realizarea acestui simulator in timp real sunt cei ce opereaza pe date cu ajutorul metodei de serializare si deserializare implementata prin biblioteca java.io.Serializable. De asemenea, s-au utilizat regex-urile pentru a imparti un lung sir de caractere in mai putine string-uri, cu scopul de a manipula mai usor. Composite Design Pattern si Observer Design Pattern au fost utilizate pentru a structura codul si pentru a notifica angajatul daca clientul a efectuat o comanda.

1. **Implementare**

In acest proiect am implementat 15 de clase, adunate in 3 pachete diferite, folosite pentru a da o tenta estetica proiectului si pentru a fi mult mai usor de vizualizat continutul proiectului.

1. Business package:

* Clasa BaseProduct:

Aceasta clasa extinde clasa MenuItem si implementeaza Serializable (pentru a fi posibila serializarea).

In aceasta clasa am 7 variabile instanta ce reprezinta pentru un produs de baza: numele (title: String) rating-ul (rating: float), caloriile (calories: int), proteinele (protein: int), grasimi (fat: int), sodiu (sodium: int), pret (price: int).

In constructor imi initializez toate aceste variabile instanta.

Am gettere si settere pentru fiecare dintre acestea si mai am o metoda getTitle pentru a returna un produs de acest tip (eficient pentru cand am un BaseProduct intr-o lista de MenuItem).

* Clasa Client:

Aceasta clasa implementeaza interfata Serializable.

Am doua variabile instanta ce reprezinta id-ul clientului (id: int; eficient pentru comenzi) si numele acestuia (nume: String).

In constructor initializez ambele variabile instanta.

Am gettere si settere pentru ambele.

* Clasa CompositeProduct:

Aceasta clasa extinde clasa MenuItem si implementeaza clasa Serializable.

In aceasta clasa am 3 variabile instanta: numele un produs compus din mai multe produse de baza (name: String), lista de produse de baza (baseProducts: ArrayList<BaseProduct>()), si pretul total calculat din suma preturilor tuturor produselor de baza continute in acest produs compus.

In constructor initializez toate aceste variabile instanta.

Am gettere si settere pentru fiecare din acestea.

Am o metoda de adaugare a unui produs de baza care adauga produsul in lista si incrementeaza totalPrice.

* Clasa Order:

Aceasta clasa implementeaza interfata Serializable.

Am 3 variabile de instanta: id-ul comenzii (orderID: int), id-ul clientului (clientID) si data comenzii (orderData: String)

In constructor initializez toate aceste variabile instanta.

Am gettere si settere pentru toate acestea.

* Clasa MenuItem:

Aceasta clasa implementeaza interfata Serializable.

Am un constructor gol.

Am o metoda de getTotalPrice care este suprascrisa in clasa CompositeProduct.

* Interfata IDeliveryServiceProcessing:

In aceasta clasa se afla majoritatea metodelor necesare pentru implementarea clasei DeliveryService.

* Clasa DeliveryService:

Aceasta este cea mai importanta clasa din tot proiectul. Ea implementeaza 2 interfete: IDeliveryServiceProcessing care este creata de mine si contine metodele necesare pentru functionalitatea aplicatiei; Serializable pentru a putea mentine informatiile din aceasta clasa.

Ca variabile instanta am 3: un Map pentru a tine minte comenziile (ca si cheie) si lista de produse comandate sub forma unui ArrayList<MenuItem> (ca si valoare).

In constructor se initializeaza cele 3 variabile instanta si se apeleaza metoda de citire din fisierul de serializare, metoda denumiat serializationRead().

* Metoda de addClient adauga in lista de clienti un client nou cu id-ul si numele transmise ca si parametrii.
* Metoda importProduct are rolul de a adauga un produs in lista de produse, produs care poate sa fie ori de baza ori compus din mai multe produse de baza. Mai pe scurt, poate sa fie de tip MenuItem.
* Metoda addOrder are ca scop adaugarea unei noi comenzi, iar parametrii reprezinta comanda de tip Order si lista de produse de tip ArrayList<MenuItem>.
* Metoda checkTheSameCompositeProducts are ca rol de a returna adevarat daca 2 parametri sunt egali, iar prin egali ma refer ca au aceeasi lista de produse de baza. Mai intai se verifica daca au acelasi numar de produse, iar in caz contrar se returneaza false. In caz afirmativ se trece pe ramura de else si se verifica daca fiecare produs in parte de pe aceeasi pozitie au acelasi nume.
* Metoda deleteProduct are rolul de a sterge un produs atat din lista de produse (si din cele de tip Compostie) cat si din comenzi. Mai intai se face stergerea din lista de produse (daca faca parte dintr-un tip Composite, atunci trebuie sters si din el) dupa care se parcurge lista de comenzi si se aplica acelasi algoritm (numai ca acum trebuie sa ne "jucam" cu cheile si valorile din Map).
* Metoda convertToMinutes are rolul de a returna cate minute au trecut de la 00:00 pana la ora transmisa ca si parametru.
* Metoda reportTimeInterval are scopul de a scrie in fisierul de raport toate comenzile care au fost efectuate intre 2 ore: o ora de inceput si una de sfarsit. Fisierul destinatie este transmis ca si parametru, iar cele 2 ore sunt si ele transmise ca si parametrii.
* Metoda reportProducts are scopul de a scrie in acelasi fisier de raport produsele care au fost comandate de mai mult de un anumit numar de ori. Acest numar este transmis ca si parametru. Pentru fiecare produs din lista de produse se cauta de cate ori a fost comandat (cu ajutorul unui contor). Dupa parcurgerea tuturor comenzilor se verifica daca contorul este mai mare decat acel numar de ori. In caz afirmativ, se scrie in fisier.
* Metoda reportClient se scrie in acelasi fisier de raport clientii ce au comandat de un numar minim de ori (nrOfTimes: int - transmis ca si parametru) si pretul comenzilor lor care trebuie sa fie mai mare decat un anumit pret (orderValue: int - transmis ca si parametru).
* Metoda reportProductsDate are rolul de a scrie in acelasi fisier de raport toate produsele ce au fost comandate intr-o anumita data (data transmisa ca si parametru). Daca da, atunci atunci ele se vor scrie in fisier.
* Metoda generateReport are ca rol de a apela ultimele 4 metode pentru a putea efectua un raport. Numele raportului il reprezinta numele transmis ca si parametru in aceasta metoda (reportName).
* Metoda getAllProducts are rolul de a returna toate produsele scrise in fisierul de serializare sub forma unui String[], metoda folositoare pentru afisarea lor in interfatele grafice pentru utilizatori. Se parcurge lista de produse, iar daca produsul este de baza atunci se afieaza toate caracteristicile lui, iar daca este de tip Composite, atunci se afieaza numele lui si toate numele produselor de baza ce sunt in acesta.
* Metoda getSearchedProducts are rolul de a returna toate produsele dupa anumite criterii. Mai exact daca au aceleasi valori ca parametrii metodei. Aceasta metoda este folosita pentru cautarea clientilor in interfata grafica pentru utilizator pentru client. Poate clientul vrea sa caute un produs doar dupa nume sau doar dupa numerul de proteine, dar in acelasi timp poate sa caute dupa toate. Daca nu se doreste cautarea dupa un anumit camp (diferit de nume), atunci o sa se introduca valoarea -1. Daca nu se doreste dupa nume, atunci pur si simplu nu se scrie nimic la nume.
* Metoda modifyProduct are rolul de a modifica un produs de baza. Se parcurg toate produsele si unde se intalneste acelasi nume atunci produsul vechi este sters si cel nou este adaugat. Daca in produs este de tip Composite atunci se parcuge lista de BaseProduct din el si se sterge cel care are acelasi nume si dupa se adauga cel nou.
* Metoda serializationWrite face serializarea obiectelor intr-un fisier din pachetul serialization. Se parcurge lista de produse si se scrie in acel fisier.
* Metoda serializationRead face deserializarea obiectelor din fisierul “products.ser” si le stocheaza in allProducts.
* Metoda generateBill are rolul de a genera bonul pentru un client. Aceastra trebuie generata DOAR daca clientul este sigur ca a efectuat cel putin o comanda. Se creaaza in pachetul bills un fisier cu numele clientului unde o sa fie afisat numele lui urmat de toate produsele comandate, iar la final pretul total pe care il are de platit.

1. Presentation package:

* Interfata Observer:

Aceasta interfata este pentru a informat un angajat ca s-a efectuat o comanda.

* Clasa LoginGUI:

Aceasta clasa e pentru interfata grafica de utilizator pentru a se putea loga fie ca administrator fie ca si client

In primul textField se introduce username-ul si in al doilea parola dupa care se apasa pe buton de LOGIN. Daca un camp nu a fost introdus corect atunci nu se intampla nimic.

* Clasa AdministratorGUI:

Aceasta clasa este pentru interfata grafica de utlizator pentru administrator. In ea sunt prezente 4 butoane ce au urmatoarele functionalitati:

* "ADD BASE PRODUCT": Dupa ce se introduc toate campurile in TextField-uri se poate apasa pe acest buton care adauga in lista de produse produsul nou de tipul BaseProduct ce contine campurile introduse. Dupa ce se creeaza noul obiect, se adauga in lista si se apeleaza metoda de serializare a obiectelor.
* "DELETE PRODUCT": Dupa ce se selecteaza un obiect din lista de produse se apasa pe acest buton pentru a-l sterge din lista aceasta. La final se apeleaza metoda de serializare a produselor.
* "MODIFY": Dupa ce se completeaza toate TextField-urile si dupa ce se selecteaza un obiect (campul de nume trebuie sa fie exact acelasi cu numele produsului selectat) se poate apasa pe buton pentru a modifica caracteristicile produsului respectiv. La final se apeleaza metoda de serializare a obiectelor.
* "ADD COMPOSITE PRODUCT": Dupa ce se selecteaza mai multe produse din lista prezenta si se completeaza textField-ul de nume (cu numele dorit pentru produsul de tip CompositeProduct) se apasa pe buton pentru a adauga noul produs in lista de produse. La final se apeleaza metoda de serializare a obiectelor.
* Clasa ClientGUI:

Aceasta clasa este pentru interfata grafica de utlizator pentru client. In ea sunt prezente 4 butoane ce au urmatoarele functionalitati:

* "ORDER": Dupa ce se selecteaza mai multe produse se apasa pe acest buton pentru a adauga o comanda.
* "SEARCH": Dupa ce se completeaza unele campuri din TextField-uri (unde nu se doreste cautarea dupa un criteriu se pune -1) se apasa pe acest buton pentru a reduce numarul de produse la cele de interes.
* "View all": Daca se apasa pe acest buton se afiseaza din nou toate produsele (de obicei este folosit dupa un search).
* "BILL": Acest buton trebuie sa fie apasat daca clientul stie daca a facut cel putin o comanda pentru a i se putea generea bonul.

1. Data package:

* Clasa FileWriter:

Aceasta clasa este folosita pentru a scrie intr-un fisier.

* Clasa Serialization:

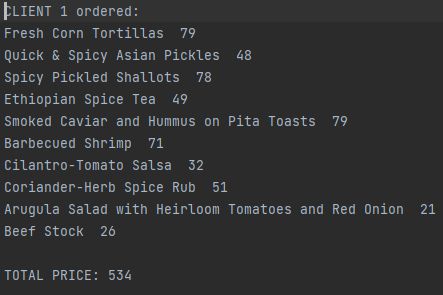
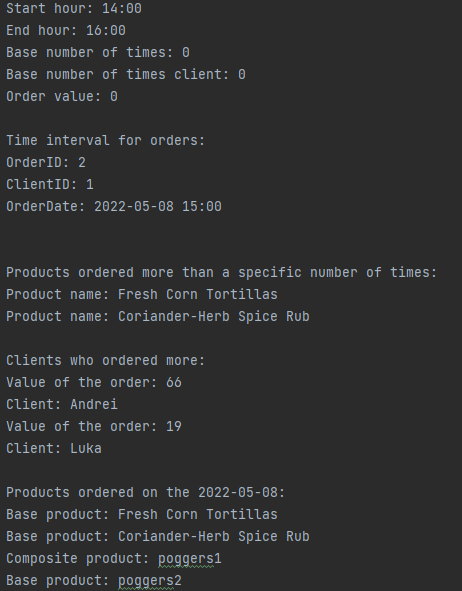
Aceasta clasa este folosita pentru a serializa lista de produse si invers.

Graphical user interface, application

Description automatically generated

1. **Rezultate**

Rezultatele operatiilor, adica comenzile fiecarui client o sa fie scrise intr-un fisier numit “bills.txt” in care putem observa numele clientului care a facut plata, totalul facturii, produsul comandat si suma fiecaruia. De asemenea, in celalalt fisier denumit si “report.txt” vom regasi intervalul de timp la care s-a efectuat plata, cat si produsele care s-au cumparat de mai multe ori de catre un anumit client.

1. **Concluzii**

In concluzie, am implementat tot ce se cerea in lista de cerinte mentionate. Este de recomandat sa se respecte regulile precizate anterior pentru a asigura o functionalitate cat mai buna a aplicatiei.

Implementarea acestui proiect m-a ajutat sa-mi aprofundez cunostintele despre conceptele OOP, cat si asigurarea unei mai bune intelegere a modului de funcționare a unei interfete grafice si mai ales de gestionarea a datelor cu ajutorul serializarii si deserializarii folosind biblioteca java.io.Serializable.

Este necesara o documentare profunda despre modul in care functioneaza o baza de date conectata simultan cu aplicatia in JAVA si mai ales este necesar sa fi inteles toata documentatia consultata.

**Dezvoltari ulterioare:**

Alte capabilitati ulterioare ale proiectului ar putea fi implementarea unei interfete mult mai friendly, in care utilizatorul ar avea mult mai multe optiuni de selectare a produselor dorite. De asemenea, se poate utiliza un feature in care sa se urmareasca comanda in timp real si, ulterior, urmarirea curierilor, ca in cele mai utilizate aplicatii de food shopping din tara. Ba chiar mai mult, optiunea meniului principal in mai multe limbi ar fi recomandata, daca aplicatia va fi folosita in viitorul apropiat de un magazin recunoscut pe plan international.

De asemenea, este important sa avem o interfata cat mai usor de folosit de catre utilizatori, deoarece nimeni nu o sa foloseasca un program care este greu de inteles sau de utilizat. Asadar, este esential sa avem un program functional, eficient si usor de utilizat de catre toate tipurile de users.

1. **Bibliografie**

* https://docs.oracle.com/javase/tutorial/java/javaOO/lambdaexpressions.html
* https://winterbe.com/posts/2014/07/31/java8-stream-tutorial-examples/
* http://www.tutorialspoint.com/java/java\_serialization.htm
* http://javarevisited.blogspot.ro/2011/02/how-hashmap-works-in-java.html
* http://docs.oracle.com/javase/8/docs/technotes/guides/language/assert.html
* https://docs.oracle.com/javase/7/docs/technotes/tools/windows/javadoc.html#tag