**FACULTATEA DE AUTOMATICĂ ŞI CALCULATOARE**

**CATEDRA CALCULATOARE**

FUNDAMENTAL PROGRAMMING TECHNIQUES

ASSIGNMENT 4

**FOOD DELIVERY MANAGEMENT SYSTEM**

**Viman Andrei-Liviu**

**Grupa: 30226**

**Cuprins**

1.Obiectivul

2.Analiza problemei si scenarii de utilizare

3.Proiectare

4.Implementare

5.Rezultat

6.Concluzii

7.Bibliografie

1.Obiectivul

Obiectivul acestei teme este de a proiecta si de a implementa, un program care proceseaza operatiile pe care un restaurant le poate face. Aplicatia permite logarea mai multor clienti si a unui singur administrator si a mai multor angajati care sunt notificati ca trebuie sa se ocupe de o comanda. Fiecare utlizator poate efectua diferite operații,în funcție de rolul lui.Astfel administratorul poate importa produsele,să gestioneze produsele(să steargă,să adauge,să editeze),să genereze rapoarte. Clientul poate să vizualizeze meniul,să caute un produs în meniu pe baza unui criteriu / criterii,să plaseze o comandă,să solicite bonul in format pdf. Angajatul primeste o notificare de fiecare dată când este plasată o comandă de către client deoarece acesta este nevoit sa se ocupe de nevoile clientului. Aceasta teme respecta toate paradicmele POO si respecta si patternul mvc dare si design pattern.

2.Analiza problemei si scenarii de utilizare

Aceasta tema a ridicat mai multe probleme, cea a folosirii lambda expressilor dar si a serializarii si a deserializarii si folosirea HasMapurilor pentru stocarea datelor in memorie. Modelarea ei a fost facuta exact asa cum s-a cerut pe mai multe nivele de executie cu ajutorul mai multor pachete, fiecare pachet reprezentand un anumit nivel:

1.buisnessLayer aici se implementeaza logica de buisness acea logica unde se poate adauga o comanda un client un meniu .

2.dataLayer este pachetul care contine metoda cu care creem bonul dar si metoda cu care am citit toate produsele din products.csv

3.model este pachetul care contine obiectele cele mai importantee din restaurant.

4.presentatioLayer este pachetul care implementeaza interfata grafica prentru o utilizare mul mai usoara a restaurantului dar tot aici ne ocupam de implementarea ascultatorilor pentru butoane.

Pentru a putea simula mai usor acest asa zis magazin am ales sa implementez si o interfata grafica care sa il ajute pe administrator sa poata urmarii mai usor evolutia comenzilor pe care clientii le plaseaza. Interfata are mai frameuri unul pentru a alege daca esti un client sau un administrator unul pentru logarea clientilor unul pentru administrator carea area mai multe txt fildeuri pentru a afisa produsele comenzile dar si produsele pe care el vrea sa le compuna. Tot aici ami gasim mai multe butoane pentru fiecare operatie pe care administratorul o poate face. Pentru client tot asa 2 text fildeuri unde poata sa vada tote produsele si produsela pe care le-a comandat si butoane pentru fiecare operatie pe care un client o poate face. Frameu angajatului afisieaza un simplu mesaj care ne notifica ca comandan noastra este iin curs de presgatire.

Graphical user interface

Description automatically generatedGraphical user interface, Word

Description automatically generatedGraphical user interface

Description automatically generatedGraphical user interface, application, table

Description automatically generated

3.Proiectare

Unified Modeling Language (prescurtat UML) este un limbaj standard pentru descrierea de [modele](https://ro.wikipedia.org/w/index.php?title=Modelare_orientat%C4%83_pe_obiect&action=edit&redlink=1) și [specificații](https://ro.wikipedia.org/w/index.php?title=Specifica%C8%9Bie&action=edit&redlink=1) pentru [software](https://ro.wikipedia.org/wiki/Software). Limbajul a fost creat de către consorțiul [Object Management Group](https://ro.wikipedia.org/w/index.php?title=Object_Management_Group&action=edit&redlink=1) (OMG) care a mai produs printre altele și standardul de schimb de mesaje intre sisteme [CORBA](https://ro.wikipedia.org/w/index.php?title=CORBA&action=edit&redlink=1). UML a fost la bază dezvoltat pentru reprezentarea complexității programelor orientate pe obiect, al căror fundament este structurarea programelor pe clase, și instanțele acestora (numite și obiecte). Cu toate acestea, datorită eficienței și clarității în reprezentarea unor elemente abstracte, UML este utilizat dincolo de domeniul IT. Așa se face că există aplicații ale UML-ului pentru management de proiecte, pentru business Process Design etc.

Diagram

Description automatically generated

Pentru a gasi o solutie cat mai buna, am folosit mai multe pachet cu nume sugestiv si am ales sa folosesc o structura pe mai multe nivele. In model am incapsulat date si am implementat functionalitati, in controller am implementat cate o subclasa care care implementiaza action listener, mai exact ascultatorii pentru butoare care asteapta ca noi sa apasam pe unul dintre ele. In view am implementat asa cum s-ar spune, asa zisa legatura dintre utilizator si calculator. Acest pattern sta la baza majoritara a aplicatiilor java cu interfata grafica. De remarcat ca am folosit pachetul java.SWING pentru implementarea interfetei.

La ce revine analiza problemei? Este foarte simplu, aceasta tema are un singur scop acela fiind de a ne invata sa lucram cu serializarea si deserializarea si cu expresii landa dar si cu structuri de date mai complexe , deoarece sa recunoastem ca in ziua de azi orice program ca sa ruleze are nevoie de o baza de date, ele fiind indispensabile vietii curente.

A screenshot of a computer

Description automatically generated with medium confidence

4.Implementare

Am decis sa implementez clasele cat mai intuitiv posibil, cu un cod cat mai lizibil si cat mai usor de inteles deoarece in viitor va fi nevoie ca o ehipa intreaga sa inteleaga codul pentru a putea lucra toti la acel proiect. O sa descriu fiecare clasa pe rand:

Graphical user interface, text

Description automatically generated **Clasa Serializator** este clasa care contine metode pentru as scrie un obiect in fisier, mai exact este folosita pentru fiecare structura de stocare pentru a o scrie in fisier si atunci cand se lanseaza aplicatia toate datele vor fi incarcate din nou pentru a relua aplicatia de unde a ramas deoarece avem nevoie de o aplicatie cat mai exacta ca sa putem sa ne continuam treaba de unde am ramas.

**Clasa FileWriter** este clasa cu care am citit datele din fisierul products.csv unde initial erau toate produsele care ereau importate in magazin, un fel de stoc al restaurantului. Metoda care citeste din fisier am implementat-o cu expresii landa deoarece asa era mult mai usor si codul era mai scurt. In aceasta clasa mai este metoda cu care creez bonul comenzilor cu numele clientului si cu produsele cumparate ora achizitiei dar si totalul comenzii.

Graphical user interface, text

Description automatically generated

Text

Description automatically generated **Clasa Main** am instantiat un view si un DeliveryService care le am transmis ca parametrii noului controller care va deschide primul frame pentru selectia persoanei care acceseaza aplicatia care poate sa fie un client sau un administrator.

**Clasa MenuItem** ete clasa generala aunui meniu si ea implementeaza serializarea pentru a putea scrie in fisier datele. Aceasta clasa are ca si atribut titul meniului, am implementat asa deoarece trebuie sa alcatuim si meniuri compuse si astfel acest lucru putand sa il implementez mult mai usor. In aceasta clasa am suprascris hasCode dar si equals pentru a putea sa mapam obiectela deoarece in java daca vrei sa adaugi elemente intr-o colectie, aceste doua metode trebuie suprascrise.

Text

Description automatically generated

Text

Description automatically generated **Clasa LoggingAccount** este clasa care are doua atribute, un nume de utilizator si o parola. Am creeat aceasta clasa deoarece in orice aplicatie trebuie sa existe mai mult client cara sa poata comanda mai mule lucruri dintr-o lista de produse deja existente. Si aceasta clasa implementeaza serializarea deoarece trebuie sa stocam lista cu clientii intr-un fisier ca la pornirea aplicatiei ei sa existe si sa nu trebuiasca sa creem de fiecare data client noi.

**Clasa Order** este o clasa unde se salveaza detaliile cele mai importante ale unei comenzi, acestea sunt: numarul de ordine a unei comenzi, id-ul clientului care a pus comanda dar zi data si ora la care a fost plasata ea si nu in ultimul rand pretul tota ai ei. Si aceasta clasa implementeaza serializarea deoarece trebuie sa stocam lista cu comenz intr-un fisier ca la pornirea aplicatiei ele sa existe pentru ca administratorul sa poata creea mai multe rapoarte.

Text

Description automatically generated

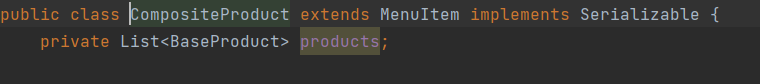
**Interfata IDeliveryServiceProcessing** sunt toate metodele care ulterior vor fi implementate in clasa **DeliveryService.** Aici am scris informatiile necesare JavaDocului sdeoarece aceasta era o cerinta a temei si asa am crezut de cuvvinta ca trebuie implementata chiar daca am copiat ulterior dupa ce am terminat de scris clasa DeliveryService am completat si interfata cua antetele metodelor.

**Clasa DeliveryService** este clasa care implementeaza **IdeliveryServiceProcessing** dar in acelasi timp ea si extinde clasa Observable pentru a putea notifica in timp real angajatul ca un client a pus o comanda. Aici se gasesc toate datele aplicatiei, lista de clienti, lista de comenzi, listaa de produse, dar si lista de comenzi mapate impreuna cu produsele aleasa de client. Metodele dein acesta clasa sunt destul de simple deoarece aici se adauga elemente in liste, de fiecare data cand vom adauga ceva nou se va realiza si serializarea in acelasi timp. Desrializarea are loc o singura data, in momentul in care pornim aplicatia. Încă de acum se poate observa că am respectat structura impusă de laborator,cu mici modificări și adăugări de noi clase,care să îmi usureze implementarea aplicației.

Text

Description automatically generated

**Clasa CompositeProduct** este clasa care extinde MenuItem dar in acelasi timp implementeaza si serializarea. Aceasta clasa este creeata pentru administrator care poate sa adauge produse multiple pentru a creea un meniu nou cu ele. Ceea ce inseamna ca aici o sa avem nevoie de metode care sa faca din toate produsele compuse un produs nou, adica sa adune toate caracteristicile unui produs de baza intre ele.

****

**Text

Description automatically generated Clasa BaseProduct** este clasa care extinde MenuItem dar in acelasi timp implementeaza si serializarea. Aceasta clasa contine toate atributele necesare descrierii unui produs, aceste fiind: numele lui, numarul de calorii pe care il are, cat de cumparat este, cat sodiu contine, cate proteine contine, cate grasimi are dar si care este pretul lui. Pentru fiecare atribut am generat cate un getter si cate un setter. Am ami generat si suprascrierea metodelor equals si hashCode pentru maparea intr-o colectie Java. Toate aceste produse se cites din fisierul dat cu toate produsele magazinului.

**Clasele de View** sunt niste clasa destul de complexe deoarece ele trebuie sa se ridice la asteptarile clientilor si ale administratorului care sa poata sa faca o multime de operatii pe aceste date. Mai departe o sa descriu fiecare clasa din componenta View-ului. Frameul de star contine doua butoane unde putem alege din ce perspectiva vrem sa folosim aplicatia, adica ca si cliend sau ca si administrator. Daca alegem sa folosim aplicatia ca si client si dam click pe butonul cu client, se va deschide un nou frame cu meniul de login a clientului care se poate loga daca deja are cont sau poate sa se inregistreze daca nu are cont. Daca clientul a reusit sa se logheze i se va deschide un nou frame unde poate sa vizualizeze produsele care se afla in restaurant poate sa plaseze o comanda cu produsele pe care le vrea dar poate sa aplice si mai multe criterii de cutare in structura produselor. Daca alege sa plaseze o comanda cu ajutorul observatorilor am reusit sa notific frameul de angajat care sa ii spuna clientului ca comanda lui a fost flasata si ca se ocupa de ea. Revenind al primul frame daca am fi ales sa apasam pe butonul de administrator s-ar fi deschis frameul administratorului care poate sa faca iport la produse care pote sa compuna un meniu din mai multe produse dar in acelasi timp poata sa stearga sa insereze si sa editeze produse. El mai poate genera patru rapoarte care in mare parte tin de statistica. El poate sa vizualizeze istoricul cu comenzile plasate de clienti.

**Clasele de Controller** are mai multe subclase, cate o subclasa pentru fiecare frame, dar si cate o clasa o clasa pentru pecare ascultator al fiecarui buton. Am ca si atribute un model si un view deoarece controllerul trebuie sa faca legatura intre model si view.

Pentru ascultatorii butoanelor am transformat acolo unde a fost nevoie datele in intregi sau in numere cu virgula flotanta de exemplu atunci cand un adminitrator vrea sa insereze un produs nou in restaurant. Pot spune ca controllerul este foarte strans legat de view, el avand ca prim atribut un obiect de tipul view.

5.Rezultat

Aceasta tema are ca rezultat invatarea studentilor sa foloseasca expresii landa dar si lucrul cu colectii in java. Acest lucru reusind sa il fac cu ajutorul interfetei grafice facand asfel o baza de date mult mai interactiva si mult mai usor de folosit. Daca era cu interogari mai complexe si sa aiba o stabilitae mai buna ea ar putea fi folosita chiar si de magazinele mici deoarece functioneaza asemanator cu apleicatiile pe care proprietarii magazinelor le cumpara cu abonament lunar.

6.Concluzii

Pot spune ca a fost prima data in viata mea cand am lucrat cu expresii landa in java si chiar a fost destul de interesant. Am intelezi cum trebuie sa se modeleze viata reala datele din mai multe domenii . Principalul obiectiv al acestei teme este de a ne invata sa lucram cu expresii landa si cu colectii deoarece acest lucru ne poate usura foarte mult viata deoarece nu mai suntem nevoiti sa scriem asa de mult cod; sa recunoastem ca un programator bun va sti intodeauna sa isi refoloseasca codul pentru a creea o mai buna stabilitate peste timp a programului. Tot aici pot spune ca este prima data in viata mea cand am folosit JavaDoc, am generat-o si se poate vedea mai jos:

Graphical user interface, table

Description automatically generated with medium confidence

7.Bibliografie

• Lambda expressions and stream processing https://docs.oracle.com/javase/tutorial/java/javaOO/lambdaexpressions.html o https://docs.oracle.com/javase/tutorial/java/javaOO/methodreferences.html o https://www.oracle.com/technical-resources/articles/java/ma14-java-se-8- streams.html o https://winterbe.com/posts/2014/07/31/java8-stream-tutorial-examples/ o <https://howtodoinjava.com/java8/java-stream-distinct-examples/>

• Java serialization

http://www.tutorialspoint.com/java/java\_serialization.htm o https://www.baeldung.com/java-serialization o https://www.geeksforgeeks.org/serialization-in-java/ o <https://docs.oracle.com/javase/8/docs/api/java/io/Serializable.html>

• Java HashMap

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

• Java assert o http://docs.oracle.com/javase/8/docs/technotes/guides/language/assert.html o http://javarevisited.blogspot.ro/2012/01/what-is-assertion-in-java-java.htmlo http://stackoverflow.com/questions/11415160/how-to-enable-the-java-keywordassert-in-eclipse-program-wise o https://intellij-support.jetbrains.com/hc/en-us/community/posts/207014815-How-toenable-assert

• Adding custom tags to javadoc o https://docs.oracle.com/javase/7/docs/technotes/tools/windows/javadoc.html#ta