# DOCUMENTATIE

Tema IV

NUME STUDENT: Crisan Paul Teodor

GRUPA: 30229

# CUPRINS

[DOCUMENTATIE 1](#__RefHeading___Toc121_1929725558)

[CUPRINS 2](#__RefHeading___Toc123_1929725558)

[1. Introducere 3](#__RefHeading___Toc125_1929725558)

[2. Analiza problemei, modelare, scenarii, cazuri de utilizare 3](#__RefHeading___Toc127_1929725558)

[3. Proiectare 3](#__RefHeading___Toc95297887)

[4. Implementare 3](#__RefHeading___Toc95297888)

[5. Rezultate 3](#__RefHeading___Toc95297889)

[6. Concluzii 3](#__RefHeading___Toc129_1929725558)

[7. Bibliografie 3](#__RefHeading___Toc131_1929725558)

**Cateva cuvinte inainte de iceputul documentatiei**

Nu prea am habar cum sa ma exprim cel mai bine in documentatia asta asa ca m-am gandit sa folosesc o exprimare deschisa, mai non-formala, fiindca am observat ca ma ajuta sa prezint informatiile mai clar si mai bine, cu o nota friendly catre cititor.

# Obiectivul temei

Design and implement a food delivery management system for a catering company. The client can order products from the company’s menu. The system should have three types of users that log in using a username and a password: administrator, regular employee, and client

Realizarea obiectivului principal implica realizarea unor obiective secundare, respectiv crearea claselor pentru:

* Delivery Service
* Useri
* GUI-uri
* Produse
* Controller

# Proiectare

Am proiectat acest poiect in stil MCV tinand cont de paradigmele OOP

Pachetul “business” cuprinde: clasele necesare pentru functionarea proiectului nu stiu ce sa mai zic.

Pachetul “view” cuprinde: clasele ce se ocupa de interfata cu utilizatorul cuvinte in plus.

Pachetul “controller” cuprinde: clasa ce creeaza legatura intre interfata si model, implementeaza functionalitatea butoanelor, si clasele ce se ocupa de asignarea clientilor in cozi.

# Implementare

*Se va descrie fiecare clasa cu campuri si metodele importante. Se va descrie implemantarea interfetei utilizator.*

Clasele proietului:

Pachetul ‘Users’ :

* + - * cuprinde toate tipurile de utilizatori din proiect da da
      * clasa User e o clasa abstracta mostenita mostenita de de toate celelalte clase : Amin, Client si Employee
      * fiecare clasa are atributul de rol care se initializaraza automat cu instantierea claseor in functie de de tipul de user preatty smart right?

Clasa ‘Client’ :

* + - * are ca atribute id-ul clientului (identificatorul unic), timpul de sosire si timpul de servire al acestuia
      * dispune de un constructor basic
      * getters & setters
      * functia toString() folosita pentru afisarea in log/interfata

Pachetul ‘Product’:

* contine clasele pentru produse si interfata Menuitem folosite pentru nu stiu ce desing pattern pretty smart
* BaseProuct este clasa pt produsele basic
* Composit Item reporezinta clasa pentru produsele compuse si contine in plus o lista de produse basic
* interfata MenuItem e ca sa incapuslez sau nu mai stiu ce zicea acolo pe wikipeia habar nu am

Clasa ‘Order’:

* aceasta clasa reprezinta o coada propriuzisa folosita in proiecta
* deoarece proiectul foloseste cate un thread pentru fiecare coada, clasa noastra trebuie sa implementeze interfata *Serilizable*
* pentru stocarea propriu zisa in memorie a clientilor folosim structura sincronizata/concurenta BlockingQueue care functioneaza ca o coada normala
* clasa are si un atribut “waitingTime” care reprezinta timpul total de asteptare al cozii la un moment dat
* functia run() este apelata la pornirea programului. Aceasta decrementeaza cu 1 service timeul clientului din capul cozii la fiecare secunda pana ajunge la 0, apoi extrage clientul din coada
* functia addClient() este apelata in schedulet pentru a adauga clienti in cozi. Aceasta adauga clientul in BlockingQueue si incrementeaza timpul total de asteptare cu service time-ul clientului adaugat
* getters & setters
* functia toString() care afiseaza cate un “(I)” pentru fiecare client din coada, aceasta fiind folosita doar pentru estetica interfetei grafice

Clasa “EmployeeView” :

* este clasa care folosita pentru asignarea clientilor la in functie de selection policy
* contine lista tuturor deliveryServicelor folosite in program
* dispune de obiectul strategy care este selection polyciul adica obiectul ceva random nu stiu ce cacat sa mai scriu in mizeria asta de documentatie asa ca bag cateva cuvinte de umplutura sper sa nu observe nimeni si daca observa va rog nu ma exmatriculati fiindca o sa fiu dezmostenit de parinti si am belit-o grav de tot multumesc pentru intelegere
* contrtuctorul primeste numarul total de deliveryService si de la 0 la nr total de deliveryService -1 initializeaza fiecare coada, si porneste cate un thread pentru fiecare coada, precum si strategia de selectie
* functia changeSelectionPolicy() nici numai stiu ce sa zic de ea ca am zis efectiv tot despre proiectu asta stau si ma uit ca bou la poarta noua la fisieru asta word si imi vine sa arunc laptopu pe geam
* functia calculateAWT() este folosita in SimulationManager pentru a calcula timpul mediu de asteptare al deliveryServicelor
* functia addClient() asigneaza clienti deliveryServicelor in functie de strategia de selectie
* getters & setters
* functia toString() formeaza un string frumos elegant pentru afisarea sa in Event Log

Clasa ‘Client’ :

* + - * are ca atribute id-ul clientului (identificatorul unic), timpul de sosire si timpul de servire al acestuia
      * dispune de un constructor basic
      * getters & setters
      * functia toString() folosita pentru afisarea in log/interfata

Pachetul ‘View’:

* are interfata strategy cu metoda addClient, care adauga clientiin cozi
* clasa “ClientsStrategy” reprezinta strategia de adaugare a clientilor in cozi in functie de cel mai mic numar de clienti. Mereu se adauga clientul in prima coada, apoi sortam cozile in functie de numarul de clienti in asa fel incat coada cu cel mai mic numar de clienti sa fie pe peima pozitie in lista cozilor
* clasa “TimeStrategy” reprezinta strategia de adaugare a clientilor in cozi in functie de cel mai scurt timp de asteptare al cozilor. Mereu se adauga clientul in prima coada, apoi sortam cozile in functie de timpul de asteptare in asa fel incat coada cu cel mai mic rimp de asteptare sa fie pe peima pozitie in lista cozilor

Clasa ‘DeliveryService’:

* in clasa asta se intampla toata magia
* are mai multe atribute care reprezinta itemele din meniu, orderul curent, userii aplicatiei, si multe alte alea
* in primu rand in constructor avem initializarile si imi si adaug clientii in userList(ii harcodez no), basically numai claientii pot sa se inregistreze
* am metoda de importat products.csv in aplicatia mea
* foloseste ceva streamuri, mapToItem nu am prea inteles exact ideea e ca tranforma un .csv file in arrayList de MenuItem si o face rapid
* metoda de creat un order se creeaza un order nou cu sizeul listei ca id ca sa fie unic si frumos, si la acest order se trimite idul clientului care a facut orderul, de asemenea se face si un text bill adica un fisier text cu nota de plata si se returneaza o nota de plata care este trimisa mai departe in employee view pt notify
* compute oreder price itereaza prin toate itemele orderului curent si calculeaza pretul total al orderului si in returneaza
* add selected item adauga itemul selectat din view in order
* add composite iteam adauga un item composit si reinitializaaza slected items list
* get menu items string returneaza un string facut frumos cu lista itemelor din menu si degn ala il folosesc ca sa ii dau display in liste

Clasa ‘View’ :

* am o gramada de viewuri unele mai tari ca altele
* adminview e clasa pt adaugat produse si tot felu, si mai genereaza si reportusi si pot importa products punct scv
* employee view nu face prea multe
* client view cuprinde listele alea de cautare si lista de iteme pe care le selectez si chestii
* viewul de login are doua text fielduri simple care primesc acolo useru si parola si dupa se merge in controller pe actionlistener si se verifica daca alea exista in baza de date si tot felu ii da
* mai bag aici niste cuvinte in plus ok multumesc
* cuprinde toate elementele ce formeaza JFrame-ul afisat pe ecran (butoane, label-uri, text field-uri).
* creez un JPanel in care aranjez corespunzator toate elementele apoi il adaug pe JFrame
* nu am folosit nici un layout pt JPanel, am preferat sa asez toate elementele la liber (eu am scris codul 100%, nu e generat sau ceva)
* butoanele si label-urile folosesc alt font size pentru estetica
* am stat vreo doua ore sa aliniez pixel cu pixel elementelele alea
* are toate metodele care adauga ActionListener-e pt fiecare buton
* functia updateView() preia toate informatiile curente ale simularii si le afiseaza in interfata (timpul curent, clientii ramasi in afara cozilor, clientii din cozi)
* are cate un getter pentru fiecare informatie necesara pornirii simularii care este apelat in controller la momentul apasarii butonului de simulare
* folosesc doar 4 cozi in interfata ca sa arat ca merge bine treaba si daca foloseam mai multe se vedea naspa
* am facut o smecherie si la pornirea programului cozile nefolosite sunt disabeled si se vede blana
* clientii in cozi sunt repezentati de cate un “(I)” care ar trebui sa fie un om
* mai avem si lista tuturor clientilor generati care se goleste pe masura ce timpul sosirii acestora este egal cu timpul curent
* mi-am dat niste filde destul de grave legate de view, am incercat sa afisez cozile cu niste Jlabels si nu prea o iesit mare chestie, arata oribil but at least it works. Trebuia sa fac in loc de Jlabels un JtextArea ca si fisierul text EventLog (inca doua cuvinte ca sa ajung la limita ok multumesc)

Clasa ‘Order’ :

* + - * are ca atribute id-ul clientului (identificatorul unic), timpul de sosire si timpul de servire al acestuia
      * dispune de un constructor basic
      * getters & setters

Clasa “Controller” :

* face legatura intre butoane si functionalitatea acestora
* cuprinde toate clasele corespunzatoare pentru ActionListener-elor butoanelor
* preia din interfata toate informatiile necesare pornirii simulari, adica timpul de simulare , intervalul timpului de sosire al clientilor, intervalul timpului de serviciu al clientilor, numarul total de cozi, numarul total de clienti si strategia de selectie
* are o gramada de butoane pentru fiecare view si basically tot de e la mine in program acolo comunica e ft smart

Clasa “Main” :

* in main am testat in mare parte functionalitatea interfetelor sa vad cum lucrreaza in spate
* pe langa asta mi-am creat obiectele principale ‘view’ si ‘controller’
* am considerat ca nu mai e nevoie si de un obiect separat ‘model’ si ca e mai bine sa initializez componentele model-ului in crearea controller-ului
* acum ca ma gamdesc mai bine cred ca era mai elegant sa fac si un ‘model’ dar nu mai am timp asa ca asta este

# Reports & JavaDoc

Am folosit si ceva java doc pre conditii si post contidtii pe care le scriu in interfata Idelivery care are ceva metode in Delivery Service, basically tre sa verific daca paramnetrii de intrare si de iesire a metodelor is diferite de null ca sa se poata executa treaba aia asa ca in bodyul metodei verifica asta cu assert carearunca o exceptie in cazulin care conditia nu e true tre sa verific daca paramnetrii de intrare si de iesire a metodelor is diferite de null ca sa se poata executa treaba aia asa ca in bodyul metodei.

Reporturile nu prea merg merg numa doua da e ok, am hardcotat basically multe chestii acolo ca nu am apucat sa mai fac view cu ele n shit sper ca e ok

# Concluzii

Mi s-a parut o tema ft cool si destul de challenging. Imi place destul de tare OOP- ul si as vrea sa merg mai departe pe treaba asta, iar tema I m-a invatat sa nu mai fiu mocan cand vine vorba de scris cod. Ar trebui sa caut mereu cai mai usoare si elegante precum si Debuger in loc de split() si verificarea cu System.out.print(). Am invatat si sa scriu cod mai eficient si frumos si sa fac JUnitTesting.

# Bibliografie

Regex:

https://stackoverflow.com/questions/36490757/regex-for-polynomial-expression

GuiGenie:

http://guigenie.com/

Impertire polinoame:

https://www.youtube.com/watch?v=FsotIB0Usvw

• Lambda expressions and stream processing o 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 o 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 o

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.html o <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#tag