DOCUMENTATIE

TEMA *2*

NUME STUDENT: BLENDEA ADRIAN

GRUPA: 30229

1.Obiectivul temei

Proiectarea si implementarea unei aplicatii cu scopul de analiza un sistem de „cozi” prin simularea venirii a N clienti, repartizarea acestora la Q cozi de asteptare si in final servirea acestor clienti si prelucrarea datelor privind timpul necesar realizarii acestor actiuni.

-recunoasterea claselor necesare implementarii si implementarea OOP a acestora;

-proiectarea unei interfete grafice user friendly si implementarea acesteia folosind java swing;

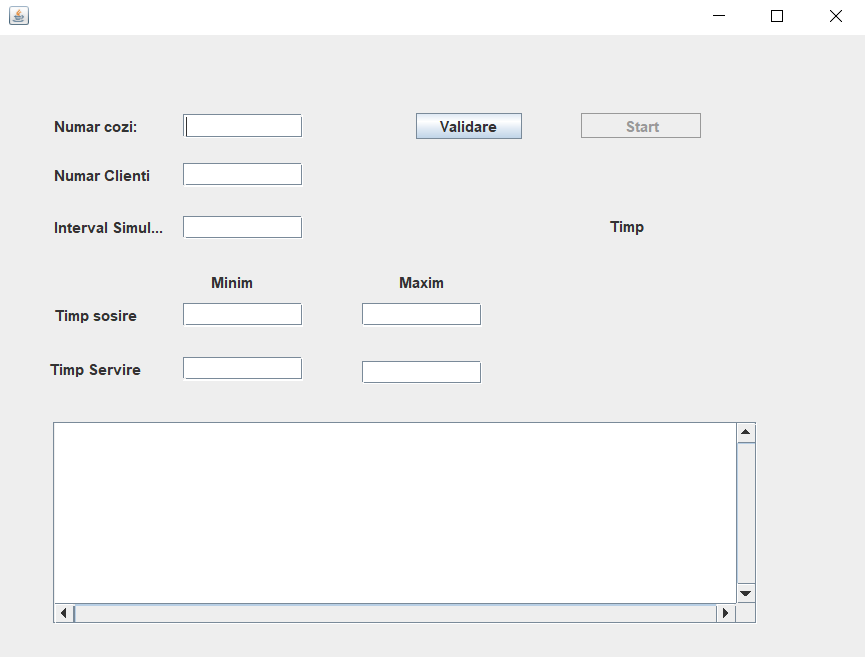
-preluarea datelor prin intermediul interfetei grafice create;

-crearea unei legaturi intre interfata si partea de calcul a programului folosind un patern MVC (model-view-controller)

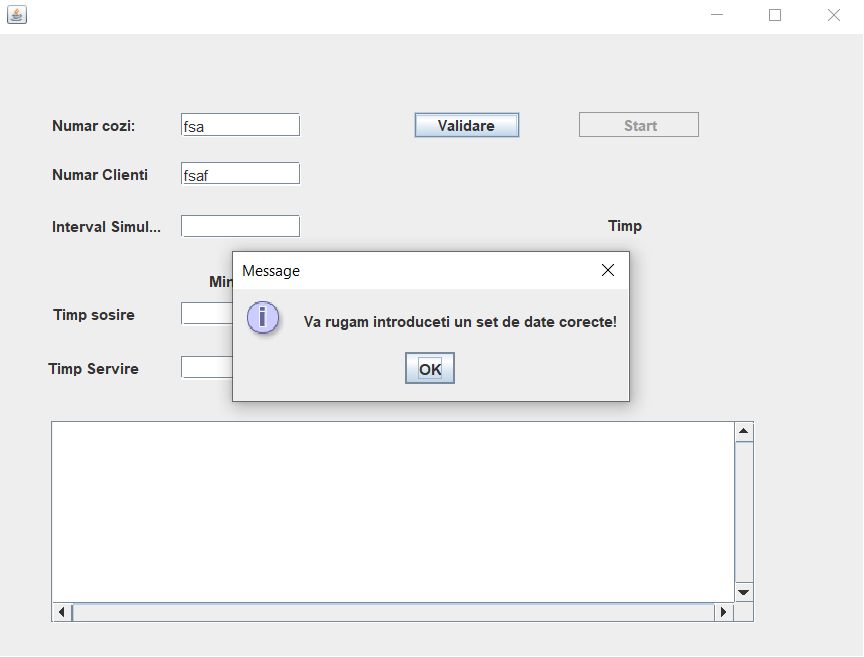
2. Analiza problemei, modelare, scenarii, cazuri de utilizare

Utilizarea programului presupune:

1.Introducerea datelor si in casutele text ale interfetei si apasarea butonului de validare pentru a se realiza validarea datelor de catre program:

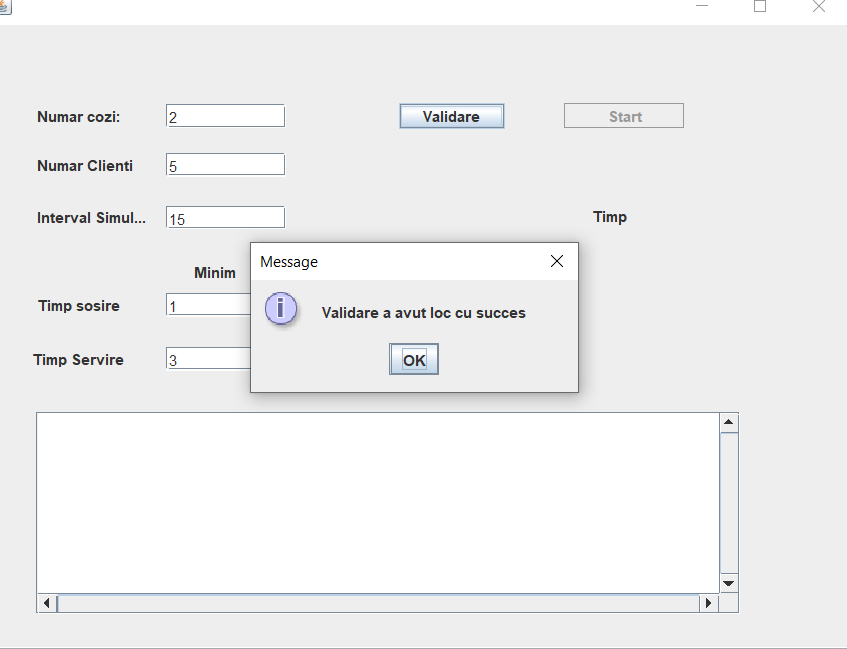


Butonul „Validare” poate produce unul dintre urmatoarele rezultate:

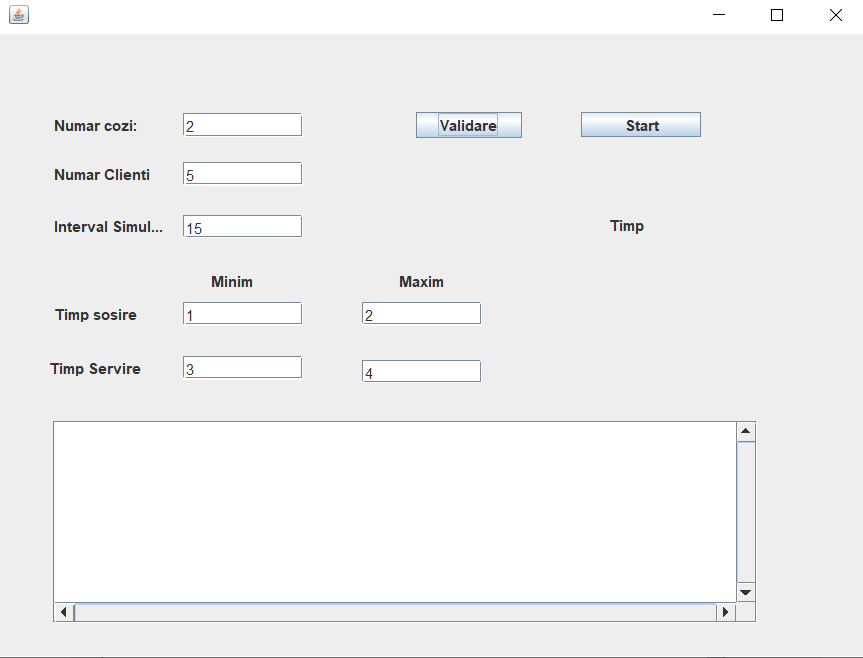


Cazul de eroare care marcheaza faptul ca datele de intrare introduse sunt eronate si trebuie reintroduse in mod corect

Sau

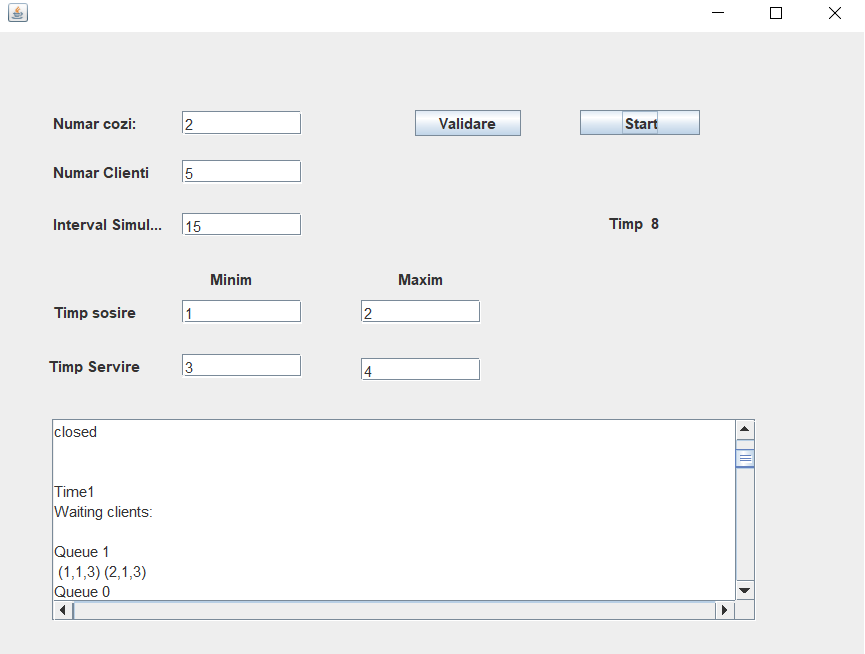


Cazul functional, in care validarea datelor este facuta cu succes , in acest caz programul va debloca programul de start:



2.Dupa validarea datelor se poate face pornirea programului apasand butonul de start.Datele completate raman in text fielduri la apasarea butonului de start!

3.Programul ruleaza tinand o evidenta a timpului si a situatiei cozilor la fiecare moment din timp,situatia este afisata in zona text din partea de jos a ferestrei .



Functionarea programului respecta documentatia temei :

Use Case: setup simulation

Primary Actor: user

Main Success Scenario:

1. The user inserts the values for the: number of clients,

number of queues, simulation interval, minimum and

maximum arrival time, and minimum and maximum

service time

2. The user clicks on the validate input data button

3. The application validates the data and displays a message

informing the user to start the simulation

Alternative Sequence: Invalid values for the setup parameters

- The user inserts invalid values for the application’s setup

parameters

- The application displays an error message and requests the

user to insert valid values

- The scenario returns to step 1

# 3.Proiectare

Proiectul este structurat cu un pattern MVC asadar avem 3 pachete: model,view,controller.

Pachetul model prezinta modul de reprezentare al polinoamelor si al operatiilor pe acestea.Este alcatuit din 3 clase: Client,Queue si Scheduler

Cel de al 2 lea pachet, view, contine doar o clasa numita View care reprezinta interfata grafica a programului

Pachetul controller contine de asemenea doar o clasa numita Controller , acesta are rolul de a face legatura dintre view si model.

4.Implementare

In continuare vor fi prezentate clasele ce alcatuiesc proiectul si functionarea acestora:

Clasa Client – este folosita pentru a reprezenta clientii sub forma de 3 parametrii relevanti pentru studiul nostru: un id (numeric , numele si prenumele clientului nu este relevant pentru situatia de fata), timpul de sosire de casa si timpul necesar servirii clientului respectiv.Aceasta clasa implementeaza Comparable , criteriul de sortare pentru clienti este arrivalTime (crescator);

Atribute:

ID (int ) : reprezinta id-ul fiecarui client

arrivalTime (int) : reprezinta timpul de sosire la casa al fiecarui client

serviceTime (int): reprezinta timpul necesar servirii clientului

Metode si constructori:

Public Client (constructor) : primeste 3 parametri intregi (id,arrivalTime,serviceTime) si ii asigneaza atributelor obiectului

getID (int) : getter pentru id

getArrivalTime (int) : getter pentru arrivalTime

getServiceTime (int) : getter pentru serviceTime

print (void) : functie de afisare a unui client

toString (string) : generarea unui string pentru a reprezenta un client sub forma de cuvinte

toString2 (string) : reprezentarea unui client ca o secventa de 3 intregi de forma (x,y,z),x reprezentat id ul, y arrivalTime iar z serviceTime

compareTo (int) : functia de compararea a clientilor (sortare crescatoare in functie de timpul de sosire )

Clasa Queue - este folosita pentru a reprezenta functionarea unei linii de astepare in care clientii stau pentru a fi serviti.Aceasta extinde clasa Thread(pentru a implementa functionalitatea mai multor linii care functioneaza deodata), si implementeaza Comparable (criteriul de sortare este timpul de asteptare, cozile sunt sortate in crescator in functie de timpul de asteptare)

Atribute:

Clients (BlockingQueue<Client) -reprezinta coada de clienti in care sunt stocate efectiv datele despre clientii aflati la coada

waitingTime (int) – contine timpul de asteptare curent pentru fiecare coada

queueNumber (int) -identificator pentru fiecare coada

Metode si constructori:

Public Queue (un singur parametru de tip int) – constructorul primeste ca parametru doar numarul cozii il atribuie, de asemenea initializeaza timpul cozii create cu 0 si lista de clienti

addClient (void) : adauga clienti in coada folosind metoda add a colectii si incrementeaza timpul de asteptare al cozii cu timpul de servire al clientului adaugat

toString (string ) : parcurge lista de clienti folosind un iterator si adauga intr un string fiecare client folosind metoda toString2 din clasa Client

getTotalTime (int) : parcurge lista de clienti cu un for each si calculeaza timpul de asteptare total curent al cozii si il returneaza

compareTo (int) - metoda de comparare, este folosita pentru a compara coziile si a le sorta crescator in functie de timpul de asteptare

getQueueNumber ( int ) – getter pentru queueNumber

getWaitingTime ( int ) – getter pentru waitingTime

run ( void ) – descrie functionarea linii de servire, clientii sunt extrasi din coada in ordine ( din capul cozii , folosind metoda poll) apoi treadul este pus in „sleep” pentru un timp egal cu timpul de servire al clientului procesat, dupa care clientul este eliminat din coada iar timpul de astepare al cozii este scazut cu timpul necesar servirii clientului. Pentru a evita erori tot acest proces este inconjurat de un if care verifica daca coada are elemente in ea.

Clasa Scheduler – aceasta clasa se ocupa cu evidenta cozilor si repartizarea clientilor la acestea.

Atribute:

queues( ArrayList <Queue> ) : contine toate cozile functionale

numberofQueues ( int ) : contine numarul de cozi

Metode si constructori:

Public Scheduler ( primeste un singur parametru de tip int ce reprezinta numarul de cozi ce vor fii create) - asigneaza valoarea primita la numberofQueues si initializea array listul de cozi

addQueue ( void ) – adauga o coada primita ca parametru in lista de cozi si sorteaza lista de cozi folosint metoda Collections.sort. Cozile vor fi sortate conform functiei compareTo din clasa Queue , metoda descrisa anterior.

dispatchClient ( int ) : aceasta metoda adauga primeste ca parametru un Client si il adauga mereu la coada de pe pozitia 0 ( coada cu cel mai mic timp de asteptare ) , de asemenea aceasta obtine timpul de asteptare al cozii dupa adaugarea clientului apeland metoda getWaitingTime din clasa Queue pe care il returneaza , de asemenea metoda resorteaza lista de cozi pentru a aduce din nou coada cu cel mai mic timp de asteptare pe pozitia 0 a listei de cozi folosind de asemenea metoda Collections.sort .

getQueues ( ArrayList < Queues > ) : getter pentru lista de cozi

Clasa View – interfata utilizator a programului

Atribute:

private JPanel contentPane;  
private JTextField sosireMinTextField;  
private JTextField sosireMaxTextField;  
private JTextField ServireMintextField;  
private JTextField servireMaxTextField;  
private JLabel queuesLabel;  
private JTextField queuesTextField;  
private JLabel clientsLabel;  
private JTextField intervalTextField;  
private JLabel lblNewLabel;  
private JLabel timpLabel;  
private JLabel minimLabel;  
private JLabel maximLabel;  
private JLabel serviceTimeLabel;  
private JButton validareButton;  
private JButton startButton;  
private JLabel timeLabel;  
JTextField clientsTextField;  
JTextArea textArea;  
JScrollPane scroll;

Metode:

validareListener ( void ) : listener pentru butonul de validare

startListener (void) : listener pentru butonul de start

getQueues ( String ) : returneaza continutul text fieldului queuesTextField

getClients ( String ) : returneaza continutul text fieldului clientsTextField

getInterval ( String ) : returneaza continutul text fieldului intervalTextField

getMinTimpSosire( String ) : returneaza continutul text fieldului sosireMinTextField

getMaxTimpSosire ( String ) : returneaza continutul text fieldului sosireMaxTextField

getMinServire ( String ) : returneaza continutul text fieldului ServireMintextField

getMaxServire ( String ) : returneaza continutul text fieldului servireMaxTextField

getStartButton ( JButton ) : getter pentru startButton

getTextArea ( JTextArea ) : getter pentru textArea

getTimpLabel ( JLabel) : getter pentru timpLabel

getTimeLabel ( JLabel ) : getter pentru timeLabel

getScroll ( JScrollPane) : getter pentru Scroll

Clasa Controller – este cea mai importanta clasa a programului, face legatura dintre model si view, de asemenea controleaza ce afiseaza programul, preia datele din interfata cu utilizator , pune in functiune threadurile necesare pentru cozi ,genereaza random clienti etc

Atribute:

mainView (view) – interfata cu utilizatorul;

scheduler (Scheduler) - se ocupa cu evidenta cozilor si repartizarea clientilor la acestea

clients ( ArrayList < Client > ) – arraylist de clienti in care se for adauga clienti generati random de o metoda ce va fi prezentata ulterior

timeLimit ( int ) – reprezinta intervalul pe care va rula simularea

start ( int ) – este folosit pentur a verifica daca s-a apasat butonul start de pe interfata mainView a controlerului

prt ( String ) – in acest string se va adauga textul ce va fi afisat de catre controller in zona text din partea de jos a interfetei

serviceAverage – aici se va calcula media de servire a tutoror clientilor procesati de cozi

waitingAverage – aici se va calcula metia de asteptare a tutorur clientilor procesati de cozi

Metode si constructori :

public Controller (View mainView) – controlerul primeste ca parametru un View , constructorul are un rol foarte important in executia programului asa ca vom parcurge implementarea acestuia pas cu pas.

In primul rand constructorul initializeaza atributul start cu valaorea 0 , stringul prt cu valoarea „ „ , atribuie view ul dat ca parametru la view l obiectului si initializeaza array listul de clienti.

In continuare controlerul va implementa metodele action performed pentru action listenerele butoanelor de validare si de start.Pentru butonul de validare acesta va incerca sa preia ca intregi(Integer.parseInt) valorile din text fielduri ( prin metodele de get prezentate in clasa de view) , daca apare vreo problema (campurile nu contin intregi) programul va arunca o exceptie de tipul NumberFormatException care va fi prinsa de un bloc try catch. Daca nu sunt probleme blocul try continua prin a arata un mesaj de succes si prin a seta intregul timeLimit la valoarea citita din interfata iar in cazul ca sunt probleme programul va intra pe blocul catch si va afisa un mesaj de eroare.

Pentru implementarea action performed a butonului start nu mai e nevoie de bloc try and catch deoarece datele au fost validate deja, se vor prelua datele din nou, se va creea un scheduler cu un numar de cozi preluat din interfata , de asemenea se vor porni threadurile cozilor si se vor genera clienti random folosind metoda generateClients pe care o vom descrie in continuare.

generateRandom ( 5 int ) – primeste 5 intregi ca parametri ( numar clienti , timp minim si maxim sosire respectiv servire ) genereaza clienti random folosint un obiect de tip Random

si metoda rand.nextInt se adauga clienti in array listul de clienti , si se sorteaza arraylistu folosind metoda Collections.sort. De asemenea in timpul generarii clientilor se calculeaza timpul mediu de servire al clientilor.

afisare (String) – este o metoda folosita pentru afisarea evenimentelor ca in exemplu din documentatie .

run (void) – descrie ce va face controlerul odata pus in functiune in functia main:

Folosind un while acesta va astepta cate o secunda (sleep ) pentru fiecare iteratie a while ului pentru a avea o simulare real time. In interiorul while -ului metoda:

-va seta labelul de timp la timpul curent

-va procesa lista de clienti si va selecta clientii care au timpul de sosire egal cu timpul curent ,si ii va trimite la cozi prin functia dispatchClient a din Scheduler ,in retur va primi de scheduler timpul de asteptare al fiecarui client (timpul cozii) si il va adauga la waitingAverage

La finalul while va imparti averageTime la nr de clienti si il va afisa impreuna cu media de serviceTime obtinuta anterior

setStart ( void ) – setter pentru start

getStart ( void ) -getter pentru start

Main – se va crea un obiect de tip view , unul de tip controller si se va folosi structura

while(c.start==0)  
{  
System.*out*.println(c.start);  
}

System.*out*.println("A pornit");  
c.start();

Pentru a intarzia pornirea procesului pana la apasarea butonului start al interfetei

5.Concluzii

Tema 1 a fost foarte utila in recapitularea cunostintelor de java dobandite in semestrul anterior dar si asimilarea unor noi cunostinte cum ar fi folosirea threadurile, care desi au fost mentionate semestrul trecut , nu au fost indeajuns aprofundate

Dezvoltarea ulterioara a programului ar trebui sa fie modul de afisare al datelor. De exemplu s-ar putea crea inca o fereastra de text in care sa se afiseze exact cand un client a fost servit si de catre ce casa ( in cuvinte ) pentru a putea fi urmarita mai cu atentie functionalitatea programului