**DOCUMENTAȚIE**

**Tema numărul 2**

**Nume student: Buretia Alex-Dian**

**Grupa: 30228**

**Cuprins**

1. Obiectivul temei..........................3
2. Analiza problemei, modelare, scenarii, cazuri de utilizare.......................................4
3. Proiectare...................................5
4. Implementare.............................7
5. Rezultate....................................11
6. Concluzii.....................................12
7. Bibliografie..................................12
8. **Obiectivul temei**

Obiectivul principal al temei numarul 2 este lucrul cu firele de executie (thread-uri) si intelegerea conceptelor legate de acestea.

Intrucat nu am mai avut ocazia sa lucrez cu astfel de fire de executie in trecut, mi-a dat ocazia sa invat si sa descopar lucrurui noi despre functionalitatile de baza ale unui procesor mai ales.

Thread-urile sunt task – uri realizate in paralel de catre procesor pe diferitele nuclee ale sale , astfel, procesorul reuseste sa rezolve in paralele task – uri. In momentul in care un fir de executie proneste alt fir de executie, primul poate astepta terminarea celui de- al doilea pentru a termina sau poate sa nu astepte. Aceasta asteptare se realizeaza cu metoda sleep in Java, atribuita unui obiect de tip Thread. Un fir de executie poate porni un infinit de alte fire, ajungand la un moment in care, daca acest numar depaseste numarul de nuclee al procesorului, firele de executie sunt puse o perioada bine determinata pe nucleu si sunt rulate, apoi peste un fir este pus altul tot pentru o perioada de timp bine determinata, pana cand toate firele se temrina. Acest fenomen cauzeaza posibilitatea rularii in paralel a mai multor programe( de pilda un browser, un compilator si un text-editor) de catre calculatorul nostru. Acest procedeu este numit Round Robin deoarece arata ca o inlantuire de task-uri care rezolva cerintele utilizatorului. Procesele contin mai multe fire de executie. Fiecare proces are memoria sa separata, registrii si stiva. Dar, firele de executie impart aceleasi registre si stive si au acces unul altuia in memorie. Astfel, singura metoda de a evita permisiunea accesului unui fir de executie in altul este prin sincronizarea sa ( synchronized ) . In Java, se pot sincroniza atat variabilele , cat si clasele sau metodele.

In plus, firele de executie se pot astepta unul pe altul pentru a prelucra un rezultat obitnut de alt fir de executie sau nu. Aceste functii sunt numite in Java ,,wait”. Metoda sleep pune pe pauza executia unui fir de executie, reusind astfel sa afiseze lucruruile in timp real.

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

Problema se pune la a genera niste clienti, si , pe baza numarului de cozi si a timpilor, sa fie distribuiti in acele cozi. Pentru a realiza acest lucru am utilizat IntelliJ. Cazurile de utilizare pe care am rulat proiectul meu au fost cele prezentate in documentatia pusa la dispozitie de catre doamna profesoara.

**3.Proiectare**

Procesul realizarii acestui proiect consta din: citirea datelor introduse de utilizator in interfata, generarea clientilor in mod corespunzator, determinara timpului minim de asteptare in toate cozile, introducerea clientului in coada cu timp minim de asteptare, stergerea clientului dupa ce acesta a fost prelucrat.

**4.Implementare**

Pentru a realiza acest proiect am lucrat cu 6 clase(View pentru a crea interfata grafica, Controller pentru a controla butoanele din interfata grafica, Main pentru rularea aplicatiei, Client pentru a implementa un client, Queue pentru a implementa un vector de clienti care reprezinta o coada si Scheduller pentru a implementa logica de executie). In cele ce urmeaza voi detalia modul de lucru al fiecarei clase.

Clasa View foloseste butoane, JtextField-uri pentru a citi variabilele introduse de utilizator, iar pentru a afisa rezultatul cerut de utilizator, un JtextArea. Pentru a citi variabilele din TextField-uri, avem nevoie de gettere, iar pentru a comanda butoanele din aplicatie avem nevoie de metode de Listener. Aceste metode vor apela metoda cu acelasi nume din clasa Controller si vor citi datele introduse de utilizator.

Exemple de cod:

public class View extends JFrame *{* private JFrame frame;  
 private JTextField t1;  
 private JTextField t2;  
 private JTextField i1;  
 //private File file;  
 private JTextField i2;  
 private JTextField nField;  
 private JScrollPane sc;  
 private JTextField qField;  
 private JTextField TmaxSiluationField;  
 private JTextArea rezultat;  
 private JLabel titlu, n, q, lblNewLabel, label;  
 private JButton generare;  
 public View*() {* this.setBounds*(*100, 100, 1079, 564*)*;  
 this.setDefaultCloseOperation*(*JFrame.*EXIT\_ON\_CLOSE)*;  
 this.getContentPane*()*.setLayout*(*null*)*;  
  
 titlu = new JLabel*(*"Tema 2"*)*;  
 titlu.setFont*(*new Font*(*"Tahoma", Font.*PLAIN*, 25*))*;  
 titlu.setBounds*(*415, 30, 102, 46*)*;  
 this.getContentPane*()*.add*(*titlu*)*;  
  
 n = new JLabel*(*"N="*)*;  
 n.setFont*(*new Font*(*"Tahoma", Font.*PLAIN*, 18*))*;  
 n.setBounds*(*45, 125, 31, 38*)*;  
 this.getContentPane*()*.add*(*n*)*;  
  
 q = new JLabel*(*"Q="*)*;  
 q.setFont*(*new Font*(*"Tahoma", Font.*PLAIN*, 18*))*;  
 q.setBounds*(*45, 209, 31, 22*)*;  
 this.getContentPane*()*.add*(*q*)*;  
  
 lblNewLabel = new JLabel*(*"TmaxSimulation"*)*;  
 lblNewLabel.setFont*(*new Font*(*"Tahoma", Font.*PLAIN*, 18*))*;  
 lblNewLabel.setBounds*(*45, 266, 128, 52*)*;  
 this.getContentPane*()*.add*(*lblNewLabel*)*;  
  
 t1 = new JTextField*()*;  
 t1.setFont*(*new Font*(*"Tahoma", Font.*PLAIN*, 18*))*;  
 t1.setBounds*(*45, 345, 60, 38*)*;  
 this.getContentPane*()*.add*(*t1*)*;  
 t1.setColumns*(*10*)*;  
  
 t2 = new JTextField*()*;  
 t2.setFont*(*new Font*(*"Tahoma", Font.*PLAIN*, 18*))*;  
 t2.setColumns*(*10*)*;  
 t2.setBounds*(*137, 345, 60, 38*)*;  
 this.getContentPane*()*.add*(*t2*)*;  
  
 i1 = new JTextField*()*;  
 i1.setFont*(*new Font*(*"Tahoma", Font.*PLAIN*, 18*))*;  
 i1.setColumns*(*10*)*;  
 i1.setBounds*(*45, 411, 60, 38*)*;  
 this.getContentPane*()*.add*(*i1*)*;  
  
 i2 = new JTextField*()*;  
 i2.setFont*(*new Font*(*"Tahoma", Font.*PLAIN*, 18*))*;  
 i2.setColumns*(*10*)*;  
 i2.setBounds*(*137, 411, 60, 38*)*;  
 this.getContentPane*()*.add*(*i2*)*;  
  
 nField = new JTextField*()*;  
 nField.setFont*(*new Font*(*"Tahoma", Font.*PLAIN*, 18*))*;  
 nField.setColumns*(*10*)*;  
 nField.setBounds*(*113, 125, 60, 38*)*;  
 this.getContentPane*()*.add*(*nField*)*;  
  
 qField = new JTextField*()*;  
 qField.setFont*(*new Font*(*"Tahoma", Font.*PLAIN*, 18*))*;  
 qField.setColumns*(*10*)*;  
 qField.setBounds*(*113, 201, 60, 38*)*;  
 this.getContentPane*()*.add*(*qField*)*;  
  
 TmaxSiluationField = new JTextField*()*;  
 TmaxSiluationField.setFont*(*new Font*(*"Tahoma", Font.*PLAIN*, 18*))*;  
 TmaxSiluationField.setColumns*(*10*)*;  
 TmaxSiluationField.setBounds*(*221, 280, 60, 38*)*;  
 this.getContentPane*()*.add*(*TmaxSiluationField*)*;  
  
 sc=new JScrollPane*(*rezultat*)*;sc.setBounds*(*500, 86, 515, 407*)*;  
 rezultat = new JTextArea*()*;  
 rezultat.setFont*(*new Font*(*"Tahoma", Font.*PLAIN*, 18*))*;  
 rezultat.setBounds*(*478, 86, 515, 407*)*;  
 this.getContentPane*()*.add*(*rezultat*)*;  
 rezultat.setColumns*(*10*)*;  
 sc.setVerticalScrollBarPolicy*(*ScrollPaneConstants.*VERTICAL\_SCROLLBAR\_ALWAYS)*;  
  
 this.getContentPane*()*.add*(*sc*)*;  
  
 label = new JLabel*(*"New label"*)*;  
 label.setBounds*(*704, 239, 45, 13*)*;  
 this.getContentPane*()*.add*(*label*)*;  
  
 generare = new JButton*(*"Generare"*)*;  
 generare.setFont*(*new Font*(*"Tahoma", Font.*PLAIN*, 16*))*;  
 generare.setBounds*(*77, 496, 128, 21*)*;  
 this.getContentPane*()*.add*(*generare*)*;  
 this.setVisible*(*true*)*;  
  
 *}* public void addGenerareListener*(*ActionListener action*) {* generare.addActionListener*(*action*)*;  
 *}* public void setText11*(){* nField.setText*(*""*)*;  
 *}* public void setText12*(){* qField.setText*(*""*)*;  
 *}* public void setText13*(){* TmaxSiluationField.setText*(*""*)*;  
 *}* public void setText14*(){* t1.setText*(*""*)*;  
 *}* public void setText15*(){* t2.setText*(*""*)*;  
 *}* public void setText19*(*String c*){* rezultat.setText*(*c*)*;  
 *}* public void setText16*(){* i1.setText*(*""*)*;  
 *}* public void setText17*(){* i2.setText*(*""*)*;  
 *}* public String getText11*(){* return nField.getText*()*;  
 *}* public String getText12*(){* return qField.getText*()*;  
 *}* public String getText13*(){* return TmaxSiluationField.getText*()*;  
 *}* public String getText14*(){* return t1.getText*()*;  
 *}* public String getText15*(){* return t2.getText*()*;  
 *}* public String getText16*(){* return i1.getText*()*;  
 *}* public String getText17*(){* return i2.getText*()*;  
 *}* // public void setSC(String c){sc.setView;}

Clasa Controller este o clasa menita sa controleze interfata vizuala a aplicatiei, atat butoanele cat si logica din spate. Singurul buton pe care l-am implementat in aceasta clasa este cel de Generare, care preia datele introduse de utilizator si le transmite mai departe clasei de Scheduler pentru a le procesa. In controller am apelat metoda din Scheduler care genereaza datele pentru clienti, utilizand metoda Math.random( genereaza un numar subunitar pozitiv ), precum si calculul secundei maxime cu ajutorul unui vector de frecventa, initializat initial pe 0 si se modifica valoarea din vector daca contorul de parcurgere se afla pe intervalul de venire – (prelucrare + venire ) al clientului .In plus, tot in aceasta clasa este determinat si timpul mediu de asteptare, pe baza timpilor de sosire ai clientilor, determinandu-se media aritmetica. Aceasta valoare este scrisa in fisierul rezultat.txt cu ajutorul unui FileWriter.

Generarea de clienti, atat in clasa Scheduler, cat si constructorul clasei Client.

public Vector*<*Client*>* generareClient*(*int n, int q, int t1, int t2, int i1, int i2*) {* for *(*int i = 1; i <= n; i++*) {* Client c = new Client*(*i, t1, t2, i1, i2*)*;  
 clientVector.add*(*c*)*;  
 *}* for *(*int i = 0; i < n; i++*) {* for *(*int j = i + 1; j < n; j++*) {* if *(*clientVector.elementAt*(*i*)*.getArrivalTime*()* > clientVector.elementAt*(*j*)*.getArrivalTime*()) {* Collections.*swap(*clientVector, i, j*)*;//ordonam vectorul crescator in functie de timpul de sosire  
 *}  
 }  
 }* return clientVector;  
*}*

public Client*(*int id,int t1,int t2, int i1, int i2*) {* int arrivalTime=*(*int*)((*Math.*random()* \* *(*t2-t1*))*+t1*)*;  
 int serviceTime=*(*int*)((*Math.*random()*\**(*i2-i1*)*+i1*))*;  
 this.id = id;  
 this.arrivalTime = arrivalTime;  
 this.serviceTime = serviceTime;  
*}*

Clasa Client am utilizat – o pentru a defini un client, pe baza argumentelor primite din interfata grafica, si anume timpul minim si maxim de asteptare si de sosire. Astfel, un client este definit de id-ul sau, tipmul de sosire si cel de asteptare. Getterele si setterele sunt metode care dau posibilitatea prograatorului sa acceseze valoarea unui camp din afara clasei, pentru a o modifica.

Clasa Scheduler este utilizata pentru prelucrarea datelor. Aceasta clasa implementeaza metoda run() corespunzatoare firului de executie pornit in clasa Controller. In afara de aceasta metoda, clasa mai contine si generarea de clienti, care transmite parametrii constructorului din Clienti( care face logica cu math.random) , dar si ordoneaza vectorul de clienti dupa timpul de sosire in mod crescator. Metoda run este cea care implementeaza logica, in mare parte, a aplicatiei mele. La inceput, metoda run porneste cate un fir de executie pentru fiecare coada din lista de cozi queue, thread care modifica timpul de prelucrare al clientului si, dupa ce a expirat acel interval, sterge clientul a carui comanda s-a livrat. Pentru acest lucru, metoda utilizeaza sleep-ul, timp in acre este prelucrat acel client. Acest run este implementat in clasa Queue. Apoi, in clasa Scheduler, metoda run() parcurge timpul de simulare, determina timpul minim de asteptare din toate cozile, si, parcurgand lista de clienti in asteptare, daca gaism un timp de sosire egal cu cel curent din simulare, afisam faptul ca l-am introdus in coada cu timp de asteptare minim, scazand vectorul de clienti in asteptare. In cele ce urmeaza sunt afisati si restul clientilor in asteptare, dar si clientii aflati in cozi in acel moment, valori pe care aplicatia le scrie in JtextArea.

Clasa Queue implementeaza cu succes un vector de clienti, lucru care reprezinta coada, care are un anumit timp de asteptare minim, care creste o data cu timpul de prelucrare a clientului nou introdus in coada. Acest lucru este realizat in metoda adaugaInCoada(Client c).

Clasele Queue si Scheduler, deoarece folosesc fire de executie, implemeteaza Runnable.

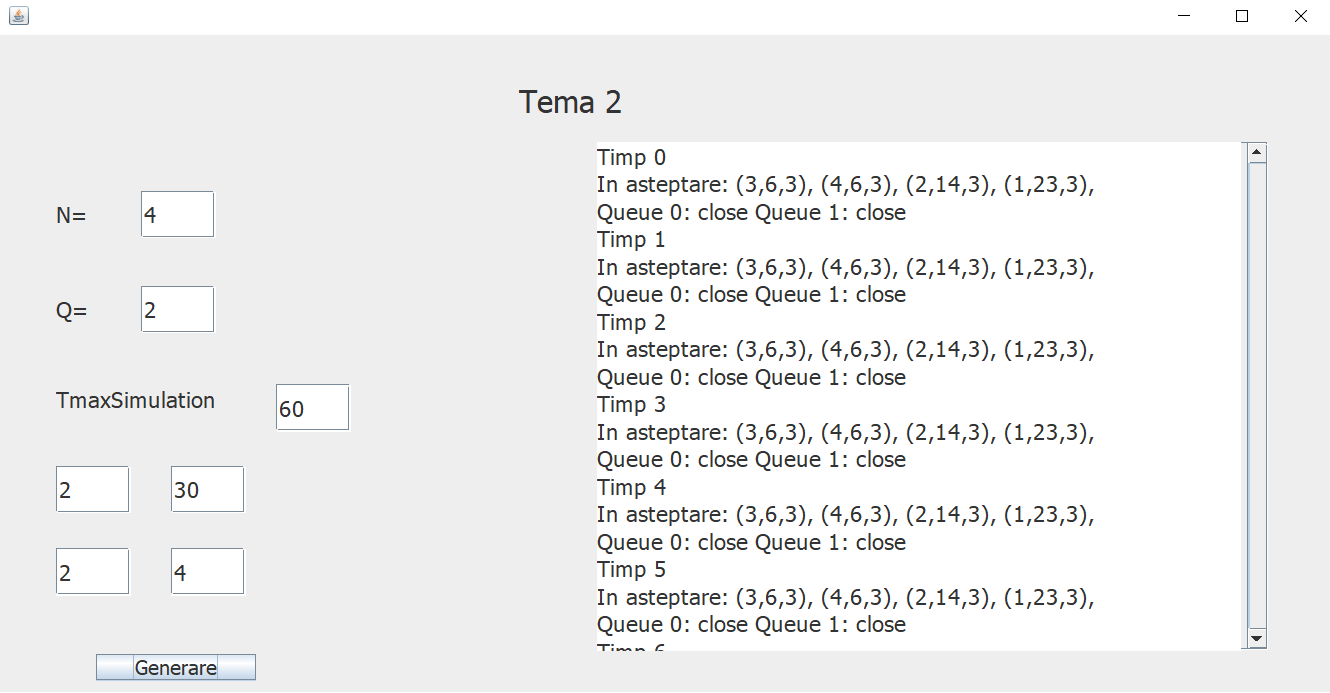
Metda run() din Queue:

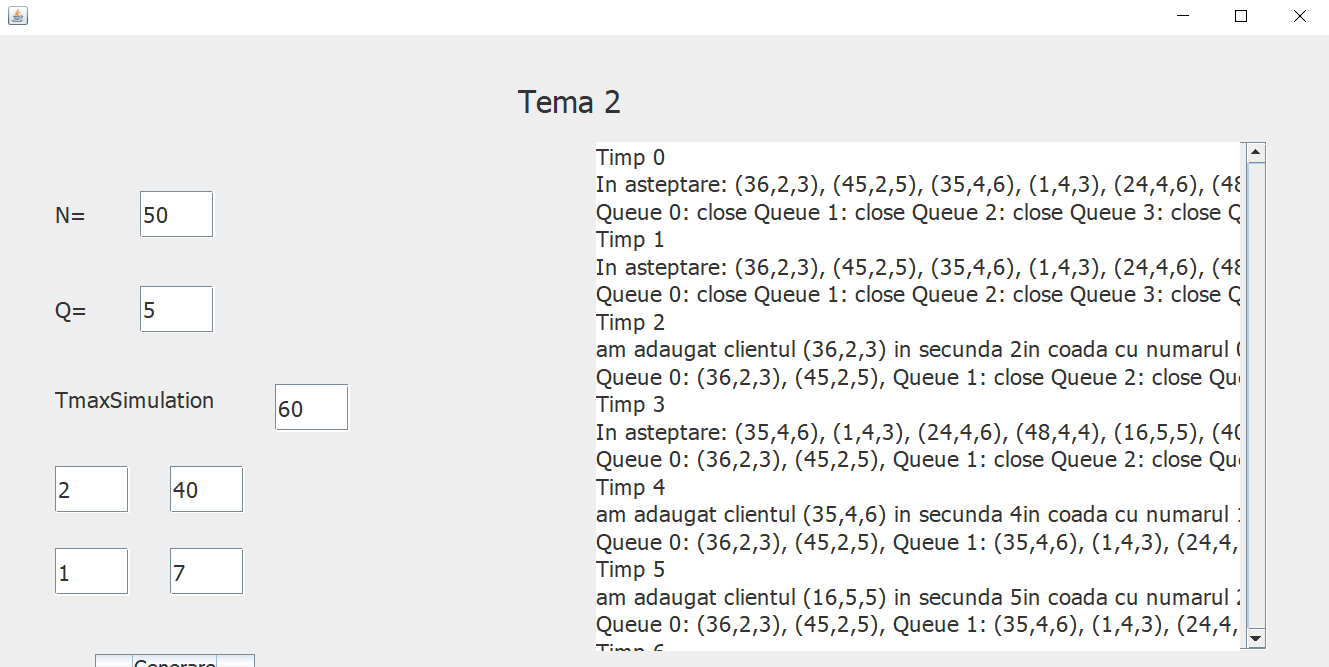
public void run*() {* int i = 1;  
 //scadem timpul de procesare al primului client din fiecare coada  
 //se face pe o coada  
 while *(*true*) {* if *(*!clientVector.isEmpty*()) {* Client client = clientVector.elementAt*(*0*)*;  
 try *{* Thread.*sleep(*client.getServiceTime*()* \* 100L*)*;  
 *}* catch *(*InterruptedException e*) {* e.printStackTrace*()*;  
 *}* waitingPeriod -= client.getServiceTime*()*;  
 clientVector.remove*(*0*)*;  
 *}* try *{* t.*sleep(*100*)*;  
 *}* catch *(*InterruptedException e*) {* e.printStackTrace*()*;  
 *}  
 }  
}*

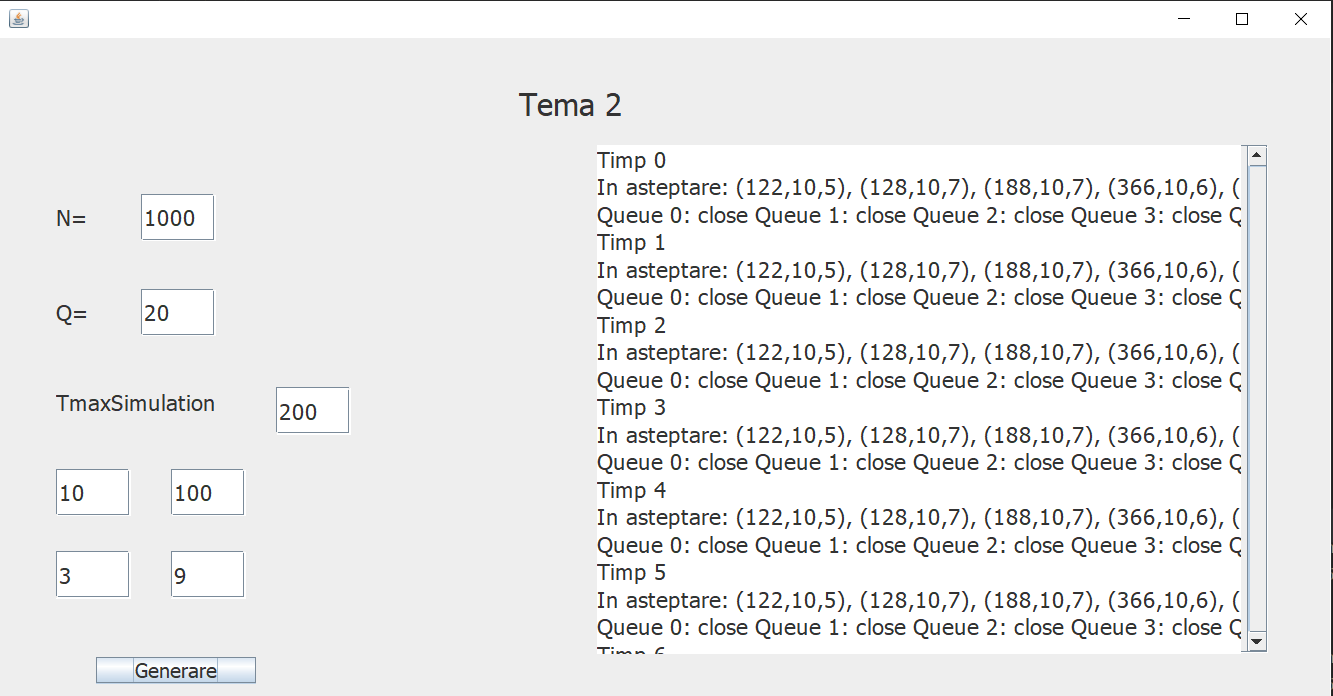
Metoda run() din Scheduler:

public synchronized void run*() {* String ggg=new String*(*""*)*;  
 for *(*Queue w : queue*) {* w.setTminAsteptare*(*0*)*;//la inceput timpul de asteptare e 0 pt fiecare coada  
 Thread t = new Thread*(*w*)*;  
 t.start*()*;//pornim un nou fir de executie pt fiecare casierie in sine  
 *}* int timpCurent = 0;  
 while *(*timpCurent <= TmaxSim*) {* int minPeriod = queue.elementAt*(*0*)*.getWaitingPeriod*()*;  
 int minPeriodIndex = 0;  
 for *(*Queue q : queue*) {* if *(*minPeriod > q.getWaitingPeriod*()) {* minPeriod = q.getWaitingPeriod*()*;  
 minPeriodIndex = queue.indexOf*(*q*)*;  
 *}  
 }* String rezultat = new String*(*"Timp " + timpCurent + "\n"*)*;  
 for *(*int wQueueSize = 0; wQueueSize < clientVector.size*()* && wQueueSize >= 0; wQueueSize++*) {* if *(*clientVector.elementAt*(*wQueueSize*)*.getArrivalTime*()* == timpCurent*) {* int g = 1;  
 int minWaitingTime = 100000;  
 int p = 0, kkk = 0;  
 rezultat+=*(*"am adaugat clientul (" + clientVector.elementAt*(*wQueueSize*)*.getId*()* + "," + clientVector.elementAt*(*wQueueSize*)*.getArrivalTime*()* + "," + clientVector.elementAt*(*wQueueSize*)*.getServiceTime*()* + ")" + " in secunda " + timpCurent + "in coada cu numarul " + minPeriodIndex*)*;  
 queue.elementAt*(*minPeriodIndex*)*.adaugaInCoada*(*clientVector.elementAt*(*wQueueSize*))*;  
 queue.elementAt*(*minPeriodIndex*)*.setTminAsteptare*(*clientVector.elementAt*(*wQueueSize*)*.getServiceTime*())*;  
 clientVector.remove*(*wQueueSize*)*;  
 wQueueSize--;  
  
 *}  
 }* rezultat += "In asteptare: ";  
 if *(*!clientVector.isEmpty*()) {*//daca avem clienti  
 for *(*Client c : clientVector*) {* rezultat += c.toString*()* + ", ";  
 *}* rezultat += "\n";  
 *}* else *{* rezultat += "closed\n";  
 *}* for *(*Queue mmm : queue*) {* rezultat += "Queue " + queue.indexOf*(*mmm*)* + ": ";  
 if *(*!mmm.getClientVector*()*.isEmpty*()) {* for *(*Client c : mmm.getClientVector*()) {* oradeVarf++;  
 rezultat += c.toString*()* + ", ";  
 *}  
 }* else rezultat += "close ";  
 *}* System.*out*.println*(*rezultat + "\n"*)*;  
 ggg+=*(*rezultat+"\n"*)*;  
 timpCurent++; view1.setText19*(*ggg+"\n"*)*;  
 try *{* Thread.*sleep(*100*)*;  
 *}* catch *(*InterruptedException e*) {* e.printStackTrace*()*;  
 *}  
 }  
  
}*

**5.Rezultate**







Rezultatele au fost obtinute ruland aplicatia mea, pe datele gasite in barem.

**6.Concluzii**

In opinia mea, acest proiect a fost foarte de folos pentru ca m-a facut sa inteleg conceptul de fir de executie, intelegand astfel cum lucreaza un thread, dar si cum il pot implementa in cod. Proiectul a necesitat o inten sa documentare, d eoarece nu lucrasem cu thread-uri inainte de realizarea p roiectului. Consider ca aceste teme sunt folositoare atat pentru noi, cat si pe nt ru cariera noastra de mai tarziu.

Nemailucrand pana acum cu fire de executie, a fost mai greu la inceput sa inte leg cum functi neaza acestea, dar dupa o documentare amanuntita am reusit sa inteleg mod ul de l ucru si cum le pot implementa in cod.

**7.Bibliografie**

**https://www.w3schools.com/java/java\_files\_create.asp**