A picture containing drawing

Description automatically generated

# Tema 2

# QUEUES SIMULATOR

Indrumator de laborator: Moldovan Dorin

Student: Manici Valentin-Dan

Grupa: 30226

# Contents

Tema 2 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 1

QUEUES SIMULATOR . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .1

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

Obiective secundare. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .1

[2.Analiza problemei, modelare, scenarii, cazuri de utilizare. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 2](#_Toc36769939)

[3. Proiectare. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 3](#_Toc36769940)

[4.Implementare. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 5](#_Toc36769945)

[5.Concluzii. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .8](#_Toc36769946)

[6.Bibliografie. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .8](#_Toc36769947)

# 1.Obiectivul temei

Proiectarea și implementarea unui simulator de cozi.

## Obiective secundare:

1. Generator de clienti aleatori
2. Implementarea corecta a distribuirii clientilor pe cozi

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

Problema cere ca datale sa fie citite dintr-un fisier text sis a fie afisate in alt fisier text. Astfel programul trebuie doar rulat si urmarit in cee ace a afisat in fisierul de output. Un fisier de input trebuie sa contina numarul de client, numarult de cozi, timpul maxim de simulare, tmpul minim la care clientul ajunge in coada cat si timpul maxim, de asemenea trebuie sa contina timpul minim si timpul maxim de servire al unui client. Un astfel de exemplu cu datele detinute de fisierul test este:

4

2

60

2,30

2,4

Programul permite rularea de mai multe ori cu acelasi date de intrare deoarece generatorul de client aleatory va genera de fiecare data date diferite crean diferite scenario, chiar si cazut in care mai multi client au acelasi timp de ajungere. Programul genereaza in fisierul de output modul in care clientii sunt tratati de catre cozi.

Daca se doreste rularea programului cu date diferite ori se creeaza un nou fisier, ori se modifica datele din fisierul de input current.

# 3. Proiectare

## Decizii de implementare

Am implementat cozile cu ajutorul colectilor din java, de asemenea prima data am folosit un array list pentru stocarea clientilor generate aleaotoriu.

## Pachete

Programul este organizat in trei pachete:

* “client” : Acest pachet contine clasele Client si ReadFile care sunt necesare pentru realizarea generarii clientilor si citirii datelor din fisier de input;
* “tema2”: Acest pachetul contine clasa Queue folosita pentru crearea cozii, adaugarea de client si scrierea in fisier a pasilor realizati de program
* “Main”(default package): Contine clasa Main care apeleaza simularea .

## Proiectare clase

1. Clasa Client

* Contine campurile Id, arrivalTime – timpul de ajungere a clintului in coada, serviceTime – timpul in care clientul trebuie servit
* Constructor: public Client(int Id, int arrivalTime, serviceTime) { }
* Metode:
* public int getId()
* public int getArrivalTime()
* public int getServiceTime()
* public String toString()
* public static Comparator()

1. Clasa ReadFile

* Contine campurile client care este reprezentat de un arraylist de client, nrQueues care reprezinta numarul de cozi, maxSimulationTime care reprezinta timpul maxim de simulare, nrClienti – numarul total de client, ind – o variabila folosita ca index
* Constructor: cel default ReadFile() { }
* Metode:
* public static void readFile()
* public static Client getClient()
* public static int getSim()

1. Clasa Queue implementeaza interfata Runnable

* Contine campurile queue care reprezinta o coada si campul queueNr care reprezinta numarul cozii
* Constructor: public Queue(int queueNr) { }
* Metode:
* public void fileWrite(String s) { }
* public void addClient(Client c) { }
* public void run() { }

1. Clasa Simulate implementeaza interfata Runnable

* Contine campurile nrOfQueues – numarul de cozi, maxSimulationTime – timpul maxim permis simularii, campul queue care este un array de tipul clase Queue si campul thread reprezinta un array de threaduri
* Constructor: Simulate() { }
* Metode:
* public int leastPopulateQueue() { }
* public void run() { }

1. Clasa Main incepe simularea

## Diagrame UML

A screenshot of a cell phone

Description automatically generated

# 4.Implementare

1. Clasa Client

Fiecare client este descries de trei parametri:id, arrivalTime, serviceTime. Constructorul Client primeste trei argumente, Id-ul de tip int, arrivalTime de tip int si serviceTime de tip int si este folosit la crearea unei instante a clasei Client si initializeaza cei trei parametrii respectiv this.Id cu Id, this.arrivalTime cu arrivalTime si this.serviceTime cu serviceTime

* + public int getId() returneaza Id-ul clientului
  + public int getArrivalTime() returneaza timpul de ajungere in coada al clientului
  + public int getServiceTime() returneaza timpul de servire al clientului
  + public Stirng toString() returneaza un mod de afisare al clientului, folosit pentru a fi sigur de generarea random
  + public static Comparator<Client> ComparatorClient = new Comparator<Client>() compara clientii in functie de timpul de ajungere al acestora

1. Clasa ReadFile

Sunt declarate urmatoarele variabile: un arraylist de client numit clients, numarul de cozi, nrQueuse, timpul maxim de simulare, maxSimulationTime, numarul de client nrClienti si un index ind initializat cu valoarea minus unu.

public static void readFile() {

ArrayList<String> lista = new ArrayList<String>();

try {

File myObj = new File("f1");

Scanner myScanner = new Scanner(myObj);

while (myScanner.hasNextLine()) {

String data = myScanner.nextLine();

lista.add(data);

}

myScanner.close();

Random rand = new Random();

*nrClienti* = Integer.*parseInt*(lista.get(0));

*nrQueues* = Integer.*parseInt*(lista.get(1));

*maxSimulationTime* = Integer.*parseInt*(lista.get(2));

String[] s1 = lista.get(3).split(",", 2);

int minArrivalTime = Integer.*parseInt*(s1[0]);

int maxArrivalTime = Integer.*parseInt*(s1[1]);

String[] s2 = lista.get(4).split(",", 2);

int minServiceTime = Integer.*parseInt*(s2[0]);

int maxServiceTime = Integer.*parseInt*(s2[1]);

for (int i = 0; i < *nrClienti*; i++) {

int index = i + 1;

int arrivalTime = minArrivalTime + rand.nextInt(maxArrivalTime - minArrivalTime + 1);

int serviceTime = minServiceTime + rand.nextInt(maxServiceTime - minServiceTime + 1);

Client client = new Client(index, arrivalTime, serviceTime);

*clients*.add(client);

}

Collections.*sort*(*clients*, Client.*ComparatorClient*);

} catch (FileNotFoundException e) {

System.*out*.println("Error!");

}

* Se declara un arraylist de strings in care se vor stoca datele citite din fisierul dat ca input, datele de pe fiecare linie vor reprezenta un string. ArrivalTime si ServiceTime sunt date ca avand un minim si un maxim, aceste minime si maxime se obtin in ambele cazuri cu ajutorul functiei predefinite split (). Exemplu split(“,”,2) va imparti un string in doua stringuri eliminand si caracterul virgula care le desparte. De asemena utilizez functia rand.nextInt() pentru a genera a valoare aleatoare pentru arrivalTime si serviceTime. La final se ordoneaza clientii din arraylist crescator in functie de arrivalTime
* public static Client getClient() { } incrementeaza indexul ind si returneaza clientul care se afla la indexul respective in arraylist
* public static int getSim() { } returneaza timpul maxim de simulare care a fost citit din fisier .

1. Clasa Queue

Contine numarul cozii si o coada implementata cu ajutorul colectilor

* public void fileWrite(String s){

try(FileWriter file = new FileWriter("log.txt", true);

BufferedWriter buffer = new BufferedWriter(file);

PrintWriter print = new PrintWriter(buffer))

{

print.println(s);

} catch (IOException e) {

System.*out*.println("Error!");

}

}

Cu ajutorul acestei functii se face scrierea in fisier

* public void addClient(Client c){

queue.add(c);

fileWrite("("+c.getId()+","+c.getArrivalTime()+","+c.getServiceTime()+")"+"added to queue "+queueNr);

}

Se adauga un client c in coada si se scrie acels client in fisier sub un anumit format.

* public int getQueueSize(){} returneaza dimensiunea cozii

Aceasta clasa implementeaza interfata Runnable astfel ca este nevoie de implementarea metodei run() care nu trenuie sa contina parametri. In functie de procesarea clientilor, cu ajutorul threadului, se scriu pasii in fisierul de ouyput.

Conceptul de thread (fir de executie) defineste cea mai mica unitate de procesare ce poate fi programata spre executie de catre sistemul de operare. Este folosit in programare pentru a eficientiza executia programelor, executand portiuni distincte de cod in paralel in interiorul aceluiasi proces. Cateodata insa, aceste portiuni de cod care constituie corpul threadurilor , nu sunt complet independente si in anumite momente ale executiei, se poate intampla ca un thread sa trebuiasca sa astepte executia unor instructiuni din alt thread, pentru a putea continua executia propriilor instructiuni . Aceasta tehnica prin care un thread asteapta executia altor threaduri inainte de a continua propria executie, se numeste sincronizarea threadurilor.

Threadurile sunt diferite fata de clasicele procese gestionte de sistemele de operare ce suporta multitasking, in principal prin faptul ca, spre deosebire de procese, toate threadurile asociate unui proces folosesc acelasi spatiu de adresare . Procesele sunt in general independente , in timp ce mai multe threaduri pot fi asociate unui unic proces. Procesele stochează un numar semnificativ de informatii de stare, in timp ce threadurile dintr-un proces impart aceeasi stare, memorie sau alte resurse. Procesele pot interactiona numai prin mecanisme de comunicare interproces speciale oferite de sistemul de operare (semnale, semafoare, cozi de mesaje și altele asemenea) . Cum impart acelasi spatiu de adresare, threadurile pot comunica prin modificarea unor variabile asociate procesului si se pot sincroniza prin mecanismele proprii. În general este mult mai simplu și rapid schimbul de informatii intre threaduri decât între procese.

Atat firele de executie, cat si procesele au stari ce pot fi sincronizate pentru a evita problemele ce pot aparea din cauza faptului ca impart diverse resurse . In general , fiecare fir de executie are o sarcina specifica și este programat astfel incat sa optimizeze utilizarea procesorului.

1. Clasa Main incepe simularea cozilor
2. Clasa Simulate

In aceasta clasa sunt declarate numarul de cozi nrOfQueues, timpul de simulare maxSimulationTime, un array queue de tipul clasei Queue si un array de thread – uri. Constructorul acestei clase instatiaza variabilele mentionate.

Metode:

public int leastPopulatedQueue(){

float min=999;

int index=0;

int[] a=new int[nrOfQueues];

for(int i=0;i<nrOfQueues;i++){

a[i]=queue[i].getQueueSize();

}

for(int i=0;i<nrOfQueues;i++){

if(min>a[i]){

min=a[i];

index=i;

}

}

return index;

}

In metoda public int leastPopulatedQueue() se decide care coada e mai putin populate, adica a carei dimensiuni dintre cozi este mai mica fata de celelalte.

Deoarece si aceasta clasa implementeaza interfata Runnable am implementat metoda run() fara argumente care in care clientul se adauga in cea mai putin populate coada, in functie de timpul curent si timpul la care trebuie sa ajunga in coada clientul.

public void run() {

int currentTime = 0;

while(currentTime<maxSimulationTime){

Client c = ReadFile.*getClient*();

queue[leastPopulatedQueue()].addClient(c);

try {

Thread.*sleep*(100);

} catch (InterruptedException e) {

System.*out*.println("Error!");

}

currentTime++;

}

System.*exit*(0);

}

# 5.Concluzii

Am invatat cate ceva despre threaduri, despre modul in care se poate lucra cu acestea.

# 6.Bibliografie

<https://ro.wikipedia.org/wiki/Fir_de_execu%C8%9Bie>

<https://dzone.com/articles/java-thread-tutorial-creating-threads-and-multithr>

<https://stackoverflow.com/questions/10459258/multithreading-why-one-thread-is-doing-all-of-the-work>

<https://www.w3schools.com/java/java_files_read.asp>

<https://www.geeksforgeeks.org/generating-random-numbers-in-java/>

<https://docs.oracle.com/javase/7/docs/api/java/util/Queue.html>