MINISTERUL EDUCAȚIEI ȘI CERCETĂRII ȘTIINȚIFICE

UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

Facultatea de Automatică și Calculatoare

Specializarea Calculatoare

# Tema 2: Aplicație simulare cozi

-documentație-

Strujan Florentina

Gr. 302210

An 2, semestrul 2

# Cuprins:

# 1. Obiectivul temei

Obiectivul temei de laborator a fost proiectarea și implementarea unei aplicații care simulează niște cozi cu clienți, având în vedere mai multe aspecte care ar putea sta la baza unei mai bune fluidizări a acestora. Se simuleaza o serie de clienți care ajung la locul obținerii unui serviciu.Se ține cont de plasarea acestora la niște cozi, urmărindu-se timpul în care aceștia ajung la cozi, timpul în care aceștia asteaptă la coada si timpul în care fiecare in parte este servit. Pentru a calcula timpul de asteptare este nevoie sa stim timpul sosirii,timpul necesar procesării clienților ajunși înainte la coadă și timpul de servire. Timpii de ajungere și de procesare sunt generați aleator pentru fiecare client in parte, în funcție de minimul și maximul timpului de sosire a clienților, dar si minimul si maximul timpului de servire a clientilor citite din fișier, astfel incat fiecare client este diferit. Pe lângă limite, din fișier se citesc și numărul de clienți, numărul de cozi și timpul maxim al simulării. După generarea fiecărui client cu timpurile lui de sosire si servire alocate random, acesta merge la casa cea mai liberă, stând la coadă și așteptând să fie servit.

Obiectivele secundare sunt:
-folosirea programării orientate pe obiect ( definirea de clase, metode, folosirea încapsulării etc.)
-folosirea limbajului de programare Java
-folosirea metodelor de maxim 30 linii și a claselor de maxim 300 linii
-folosirea generatorului Random pentru clienți
-folosirea thread-urilor, unul pentru fiecare coadă
-minimul de un test rulat si salvat în In-Test-1.txt
-un fișier .jar creat și configurat pentru a putea fi rulat conform cerințelor
-fiecare coadă cu structura de tip Collection cu proprietatea Synchronized
-închiderea și deschiderea dinamică a cozilor în funcție de existența și prelucrarea clinților din acestea.

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

Pentru implementarea aplicației cerute, vom analiza cerințele date, iar ulterior vom analiza tipul de programare „Programare Orientata pe Obiecte", deoarece conceptele acestui tip de programare stau la baza implementarii acestei aplicatii. Bazându-ne pe aceste concepte, putem incepe dezvoltarea aplicatiei. Mai mult decat atat, vom folosi si threaduri pentru a realiza concurenta.
Cozile sunt de obicei utilizate pentru a modela domenii din lumea reală. Obiectivul prin-cipal al unei cozi este pentru a oferi un loc pentru un „client" să aștepte înainte de a primi un „serviciu". Managementul coadă sisteme pe bază este interesat în minimizarea cantității tim-pul lor „clientii" sunt în așteptare în cozi înainte ca acestea sa fie servite.
Orice coadă poate fi văzută ca o pereche casă de servire - client care așteaptă la coadă, această corespondență fiind modelată după următorul aspect : fiecare coadă are clienți care trebuie procesați. Dar clientul poate să aleagă ( în funcție de numărul de case de servire puse la dispoziție) cărei case de servire să fie asociat. Modul de asociere depinde de cum este văzută problema.
Un scenariu în care ar putea fi folosit acest tip de aplicație: existand un hiper-market, avand mai multe cozi, deoarece este populat in majoritatea timpului de un numar ridicat de persoane, fiecare client care isi termina cumparaturile va cauta sa se aseze la coada care are cel mai mic numar de

persoane in ea. Presupunem ca managerul magazinului tine o evidenta legata de eficienta casierilor. Acest tip de aplicatie este folosit de utilizatorii care doresc realizarea simularea timpilor de asteptare pentru clienti la casa, eficienta casierilor.

# 3. Proiectare

## 3.1Decizii de proiectare

Pe plan intern, pentru a implementa aplicatia, am setat un client să aibă un id, un timp de sosire și un timp de asteptare, iar fiecare coada a serviciului să conțină unul sau mai multi clienți, la un anumit timp. Asignand fiecarei cozi de clienti câte un thread, reușesc astfel să procesez clientii din fiecare coada simultan. Clientii sositi la procesare, vor fi aseazati automat la casa care are timpul total de asteptare cel mai scurt in momentul respectiv.

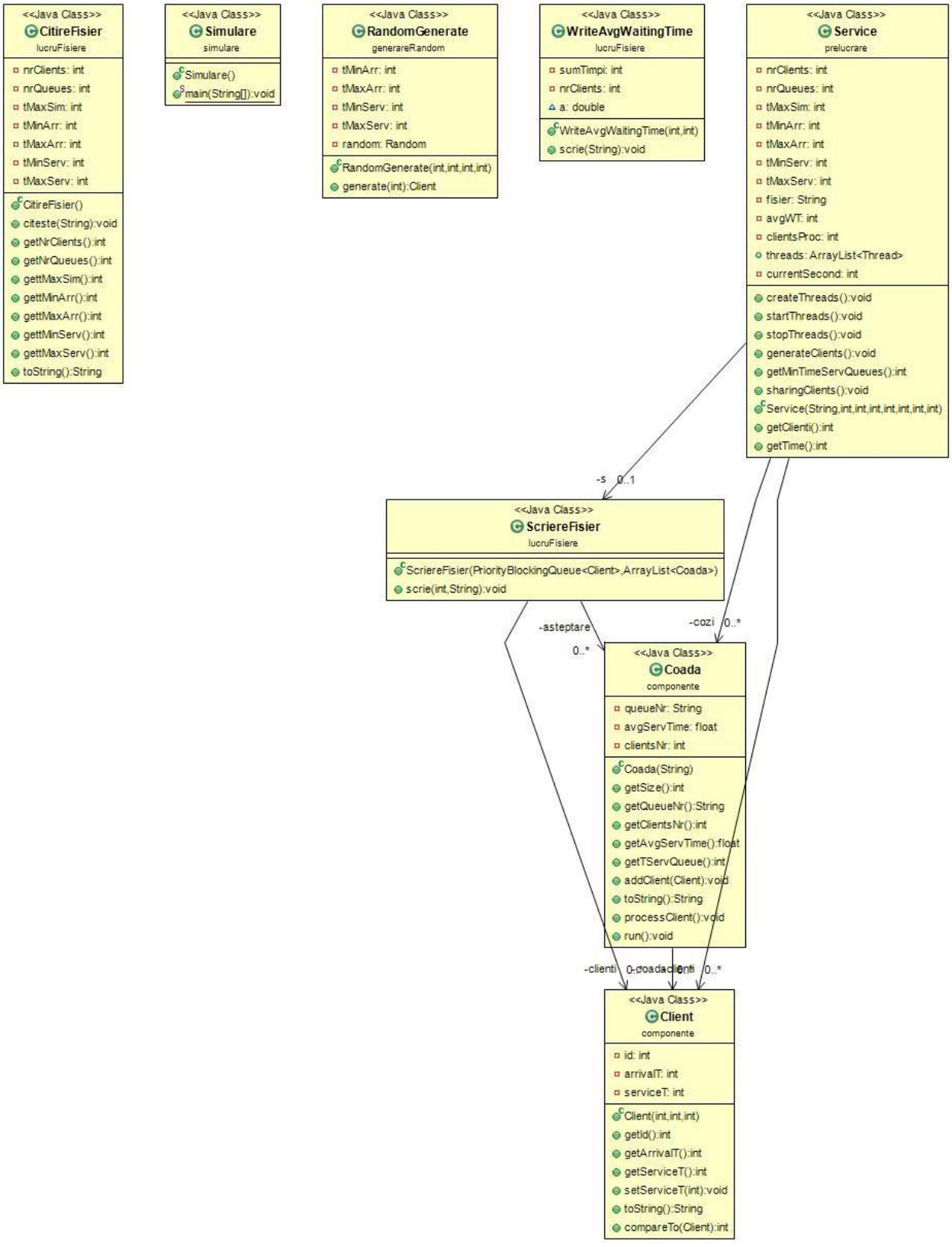
Pe plan extern, vom afișa într-un fișier situația fiecărei cozi în momentul de timp în care se procesează cel puți un client, pentru reducerea numărului de afisări inutile, doar de decrementare a timpului clienților aflați la cozi. La finalul fișierului se va afișa și timpul mediu de așteptare a unui client în funcție de datele introduce dintr-un alt fișier, de intrare.

## 3.2 Diagrama UML

UML este notaţia internaţională standard pentru analiza şi proiectarea orientată pe obiecte. Diagramele UML de clase sunt folosite in modelarea orientata obiect pentru a descrie structura statica a sistemului, modului in care este el structurat. Ofera o notatie grafica pentru reprezentarea: claselor - entitati ce au caracteristici comune relatiilor - relatiile dintre doua sau mai multe clase ,

Unified Modeling Language sau UML pe scurt este un limbaj standard pentru descrierea de modele si specificatii pentru software. 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.

Reprezentarea UML a claselor.


<<Java Class>>
CitireFisier
lucruFisiere
nrClients: int
nrQueues: int
tMaxSim: int
tMinArr: int
tMaxArr: int
tMinServ: int
tMaxServ: int
CitireFisier()
citeste(String):void
getNrClients():int
getNrQueues():int
gettMaxSim():int
gettMinArr():int
gettMaxArr():int
gettMinServ():int
gettMaxServ():int
toString():String
<<Java Class>>
Simulare
simulare
Simulare()
main(String[]):void
<<Java Class>>
RandomGenerate
generareRandom
tMinArr: int
tMaxArr: int
tMinServ: int
tMaxServ: int
random: Random
RandomGenerate(int,int,int,int)
generate(int):Client
<<Java Class>>
WriteAvgWaitingTime
lucruFisiere
sumTimpi: int
nrClients: int
a: double
WriteAvgWaitingTime(int,int)
scrie(String):void
<<Java Class>>
Service
prelucrare
nrClients: int
nrQueues: int
tMaxSim: int
tMinArr: int
tMaxArr: int
tMinServ: int
tMaxServ: int
fisier: String
avgWT: int
clientsProc: int
threads: ArrayList<Thread>
currentSecond: int
createThreads():void
startThreads():void
stopThreads():void
generateClients():void
getMinTimeServQueues():int
sharingClients():void
Service(String,int,int,int,int,int,int,int)
getClienti():int
getTime():int
-s
0..1
<<Java Class>>
ScriereFisier
lucruFisiere
ScriereFisier(PriorityBlockingQueue<Client>,ArrayList<Coada>)
scrie(int,String):void
-asteptare
0..*
-cozi
0..*
<<Java Class>>
Coada
componente
queueNr: String
avgServTime: float
clientsNr: int
Coada(String)
getSize():int
getQueueNr():String
getClientsNr():int
getAvgServTime():float
getTServQueue():int
addClient(Client):void
toString():String
processClient():void
run():void
-clienti
0..*
-coadaClienti
0..*
<<Java Class>>
Client
componente
id: int
arrivalT: int
serviceT: int
Client(int,int,int)
getId():int
getArrivalT():int
getServiceT():int
setServiceT(int):void
toString():String
compareTo(Client):int

## 3.3 Proiectare clase

In cadrul acestei aplicatii, exceptand clasele pentru prelucrarea fișierelor și generarea clienților ,intalnim patru clase : Client, Coada, Service si Smulare. Clasele Coada si Service sunt clasele de baza ale aplicatiei. Clasa Coada contine metoda de run() a threadurilor, iar aici se produce procesarea clientilor. Se extrage cate un client din BlockingQueue-ul de clienti, si se proceseaza in functie de timpul clientului de servire. In timpul procesarii, thread-ul cozii respective va fi pe sleep, in functie de timpul de servire al clientului. Dupa, clientul respectiv este scos din Coada si se trece la procesarea urmatorului client.
In clasa Service, cream cozile, threadurile si apeland metoda generate() din clasa de generare a clientilor, generam clienti care sunt introdusi in coada de asteptare. Dupa , sunt distribuiti in coada cu timpul de asteptare cel mai scurt. În clasa Simulare se instantieaza un obiect de tip Service și în loc de path-urile catre fisierele de intrare si iesire de folosesc args[0], args[1] pentru compilarea din terminal.

## 3.4Metode

### Metode din clasa Client

Pe langa medodele de get si set , avem metoda folosita pentru corectitudinea adaugarii in structura de tip coada, ce determina implementarea Comparable<Client> de catre clasa.

```
public int compareTo(Client c) {
		return arrivalT - c.getArrivalT();
			}
```

### Metode din clasa Coada

Pe langa metodele de get si set, avem si metodele run() si processClient de timp synchronized.
Metoda extrage cate un client din BlockingQueue-l cozii respective,
il proceseaza( pune pe sleep threadul in functie de service time-ul clientului) si ulterior il scoate din coada.

```
public synchronized void processClient() {
		Client client = coada.peek();
		avgServTime += client.getServiceT();
		clientsNr++;
		System.out.println("--PROCESSING-- Clientul cu id ul " + client.getId()
+ " are timpul de procesare "
				+ client.getServiceT() + " secunde si momentul ajungerii "
+ client.getArrivalT());
		try {
			Thread.sleep(client.getServiceT());
		} catch (InterruptedException e) {
```

```
                e.printStackTrace();
            }
            coada.remove();
            for (Client c : coada) {
                c.setServiceT(c.getServiceT() - client.getServiceT());
            }
            System.out.println("--EXIT-- Clientul" + client.getId() + " s-a procesat
si a fost eliminat din" + queueNr);
            }
```

Metoda reprezinta “munca” pe care threadurile o executa. Cat timp cozile nu sunt goale, se va apela metoda processClient() de mai sus. Astfel fiecare thread asignat unei cozi va procesa concurent fata de celelalte threaduri.

```
@Override
    public void run() {
            System.out.println("--STARTING-- " + Thread.currentThread().getName() +
" reprezinta " + queueNr);
            while (true) {
                if (!coada.isEmpty()) {
                    System.out.println("--INFO-- Nr de clienti in " + queueNr
+ " este " + getSize());
                    processClient();
                }

            }
            }
```

## Metode din clasa RandomGenerate

Metoda genereaza in functie de limitele date, un timp de sosire randm si un timp de procesare random si returneaza un Client cu prametrii generate random si id ul dat.

```
public Client generate(int id) {
            int arrivalT = tMinArr + random.nextInt(tMaxArr - tMinArr + 1);
            int serviceT = tMinServ + random.nextInt(tMaxServ - tMinServ + 1);
            return new Client(id, arrivalT, serviceT);
            }
```

## Metode din clasa CitireFisiere

Metoda primeste ca parametru calea spre fisierul din care vor fi citite informatiile necesare bunei functionari a aplicatiei, si il sectioneaza pe linii pentru, apoi in functie de delimitatorul “,”. Metoda arunca si exceptiile corespunzatoare posibilei erori la citire.

```
public void citeste(String fisier) {
            String st;
```

```
		try {
			File file = new File(fisier);
			BufferedReader br = new BufferedReader(new FileReader(file));
			if ((st = br.readLine()) != null)
				nrClients = Integer.parseInt(st);
			if ((st = br.readLine()) != null)
				nrQueues = Integer.parseInt(st);
			if ((st = br.readLine()) != null)
				tMaxSim = Integer.parseInt(st);
			if ((st = br.readLine()) != null) {
				String aux[] = st.split(",");
				tMinArr = Integer.parseInt(aux[0]);
				tMaxArr = Integer.parseInt(aux[1]);
			}
			if ((st = br.readLine()) != null) {
				String aux[] = st.split(",");
				tMinServ = Integer.parseInt(aux[0]);
				tMaxServ = Integer.parseInt(aux[1]);
			}
			br.close();
		} catch (FileNotFoundException e) {
			System.out.println("Fisierul nu a fost gasit");
		} catch (IOException e) {
			System.out.println("Eroare la citire");
		}

		}
```

## Metode din clasa ScriereFisier

Metoda primeste ca parametri numarul ce va fi scris la fiecare iteratie a unei cozi si fisierul in care se va scrie.

```
public void scrie(int time, String fisier) {
		FileWriter f = null;
		try {
			f = new FileWriter(fisier, true);
			f.write("Time " + time + "\n");
			f.write("Waiting clients" + clienti.toString() + "\n");
			for (Coada c : asteptare)
				f.write(c.toString() + "\n");
			f.flush();
			f.close();
		} catch (IOException e) {
			System.out.println("Eroare la scriere");
		}
		}
```

## Metode din WriteAvgWaitingTime

Metoda va scrie in fisier timpul mediu de asteptare a unui client in functie de timpul asteptat de toti clientii si numarul de client procesati.

```
public void scrie(String fisier) {
		FileWriter f = null;
		try {
			a = (double) sumTimpi / (double) nrClients;
			f = new FileWriter(fisier, true);
			f.append("Average waiting time:  " + a);
			f.flush();
			f.close();
		} catch (IOException e) {
			System.out.println("Eroare la scriere");
		}
		}
```

## Metode din clasa Service

Metodele de creare, pornire si oprire a thread-urilor

```
public void createThreads() {
		for (int i = 0; i < nrQueues; i++) {
			int aux = i + 1;
			Coada c = new Coada("Queue " + aux + ": ");
			cozi.add(c);
			Thread t = new Thread(cozi.get(i));
			threads.add(t);
			threads.get(i).setName("Thread " + aux + ": ");
		}
	}

	public void startThreads() {
		for (Thread thread : threads) {
			thread.start();
		}
	}

	public void stopThreads() {
		for (Thread thread : threads)
			thread.interrupt();
		}
```

Metoda de generare random a clientilor in functie de numarul acestora

```
public void generateClients() {
```

```
		RandomGenerate r = new RandomGenerate(tMinArr, tMaxArr, tMinServ,
tMaxServ);
		for (int i = 0; i < nrClients; i++) {
			Client c = r.generate(i + 1);
			clienti.add(c);
		}
		}
```

Metoda de determinare a cozii cu timpul minim de asteptare

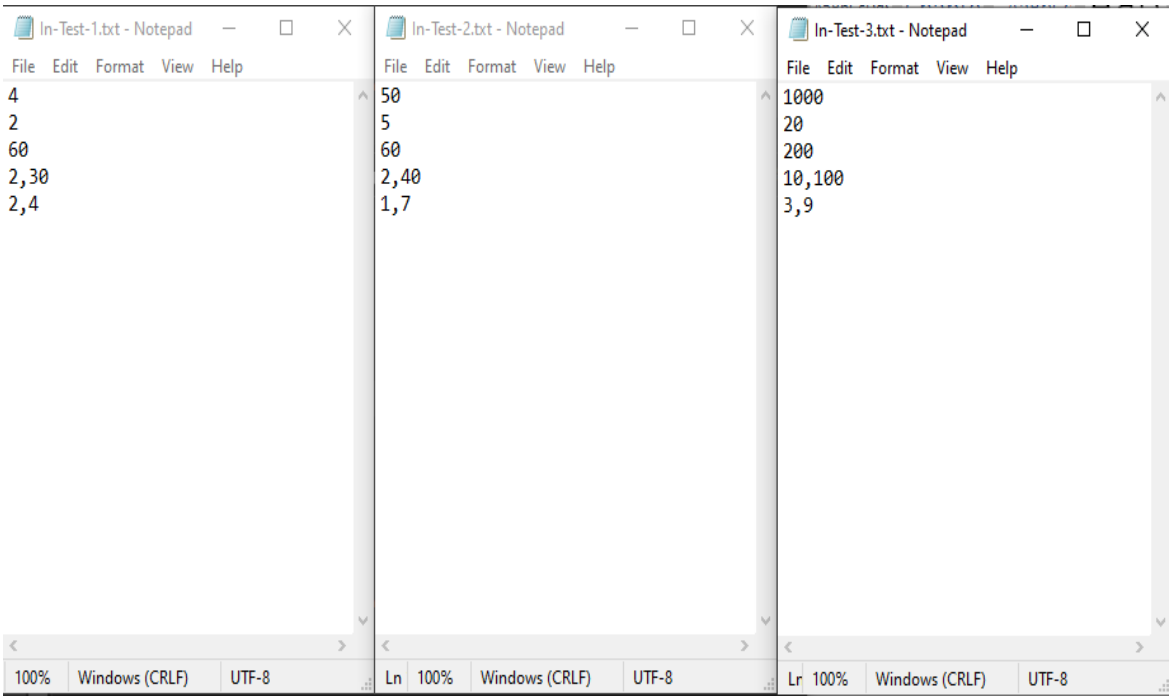
```
public int getMinTimeServQueues() {
		int tMinServQueues = 99999;
		int minQueue = 0;
		for (int i = 0; i < nrQueues; i++) {
			if (cozi.get(i).getTServQueue() < tMinServQueues) {
				tMinServQueues = cozi.get(i).getTServQueue();
				minQueue = i;
			}
		}
		return minQueue;
		}
```

Metoda in care se realizeaza distribuirea clientilor( generati random si pusi pe lista de asteptare ) in functie de coada cu timpul minim de asteptare. In aceasta metoda se tine evidenta thread-urilor si se realizeaza afisarea in fisier.

```
public void sharingClients() throws InterruptedException {
		while (currentSecond <= tMaxSim) {
			s.scrie(currentSecond, fisier);

			if (currentSecond > tMaxSim || clientsProc == nrClients) {
				for (Thread thread : threads) {
					System.out.println("--STOPPING-- " +
thread.getName());
					thread.interrupt();
				}
				break;
			}
			Client c = clienti.peek();
			int getRightQueue = getMinTimeServQueues();
			cozi.get(getRightQueue).addClient(c);
			avgWT += cozi.get(getRightQueue).getTServQueue();
			clientsProc++;
			Thread.sleep(c.getServiceT());
			clienti.remove();
			currentSecond += c.getServiceT();
		}
		WriteAvgWaitingTime w = new WriteAvgWaitingTime(avgWT, clientsProc);
		w.scrie(fisier);}
```

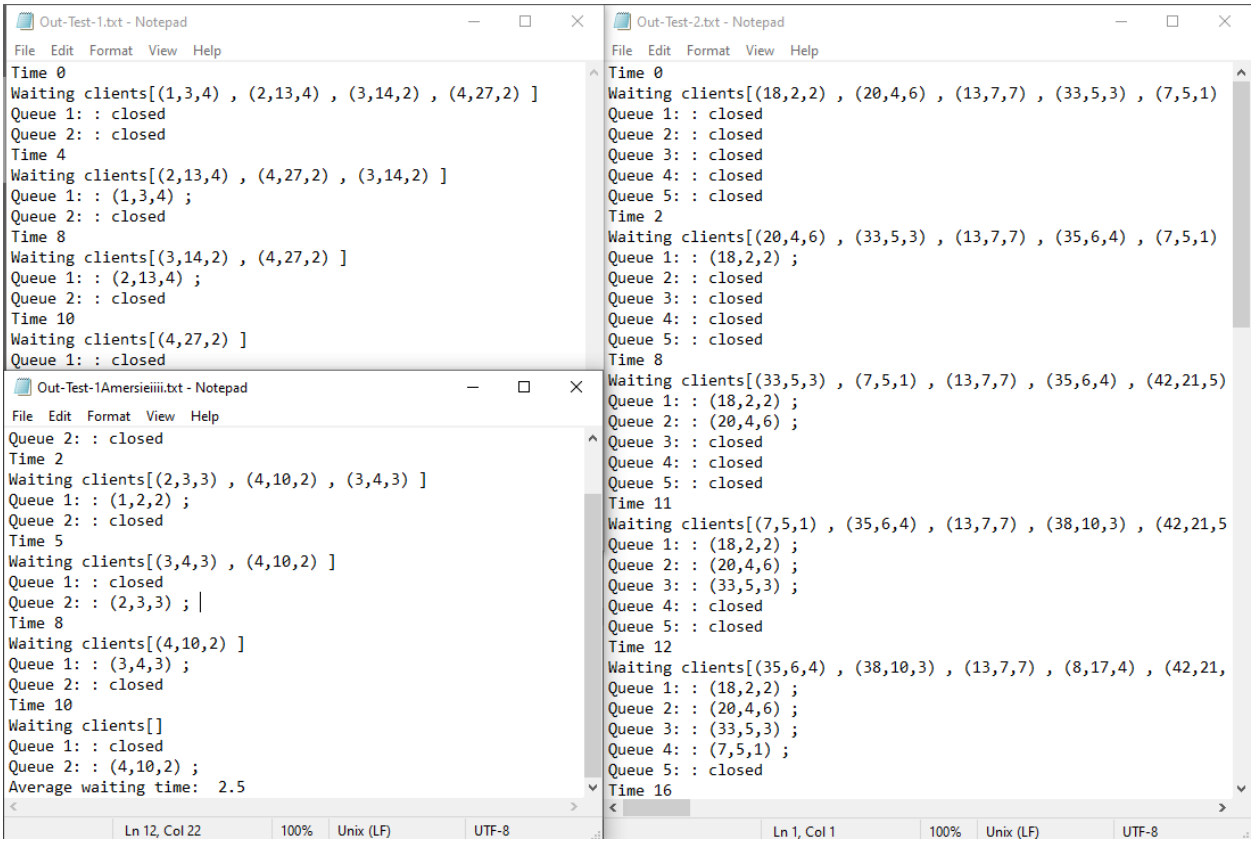
# 4.Implementare




Initial, trebuie creat un fisier text cu datele de intrare in ordinea prestabilita.

La rularea aplicatiei, in functie de datele de intrare, utilizatorul va putea vizualiza in timp real continutul cozilor si timpul mediu de asteptare a unui client la coada in functie de datele de intrare si procesarea cozilor . Informatii despre starea threadurilor si desfasurarea interna a aplicatiei vor fi disponibile in consola.

# 5.Rezultate




Am obtinut o implementare a cerintei proiectului , punand in practica simularea unui sistem de cozi, afisand in timp real continutul acestora si medi de asteptare a unui client. Daca datele sunt introduce corespunzator in fisierul de intrare, rezultatele generate sunt corecte si utile pentru utilizator.

# 6.Concluzii

Sunt de părere că în urma realizării acestei teme am reușit să-mi reamintesc și să aprofundez materia de semestrul trecut, să-mi îmbunătățesc tehnicile de programare în acest limbaj, am deprins o mai buna aptitudine in a rezolva algoritmi pe obiecte si am aprofundat concepte legate de structuri de date de tip BlockingQueue, cat si threaduri. De asemenea, am invatat cum functioneaza concurenta in Java.

Ca o posibila dezvoltare ulterioara, aplicatiei ii mai pot fi aduse unele imbunatatiri, ca de exemplu:

- afisarea in timp real a timpului de asteptare la cozi
-luarea in considerare a timpului de sosire in concordanta cu timpul simularii pentru adaugarea in cozi

# 7.Bibliografie

http://users.utcluj.ro/~igiosan/Resources/POO/Curs/POO11.pdf
https://www.tutorialspoint.com/java/java_multithreading.htm
http://coned.utcluj.ro/~salomie/PT_Lic/4_Lab/Assignment_2/Java_Concurrency.pdf
https://www.youtube.com/watch?v=i9FGIlqUEtw&feature=youtu.be
https://stackoverflow.com/questions/3906081/how-do-i-generate-a-random-value-between-two-numbers