DOCUMENTATIE

TEMA 2

Nume student: Lucăcel Mălina

Grupa: 30221

CUPRINS

1. Obiectivul temei

Obiectivul principal al temei este simularea unei cozi de așteptare la casa de marcat pentru a distribui optim n clienți în q cozi, astfel încât timpul de așteptare să fie minim.

● Obiectivele secundare prezentate ca lista sunt:

● Colectarea datelor despre sosirea clienților la magazine

● Generarea de date aleatorii pentru a simula sosirea clienților la magazine

● Implementarea unui algoritm care distribuie clienții în cozi în funcție de timpul de așteptare minim

● Simularea procesului de așteptare pentru fiecare coadă și colectarea datelor relevante

● Analiza datelor colectate pentru a determina dacă algoritmul de distribuție a funcționat corespunzător

● Identificarea și implementarea de îmbunătățiri ale algoritmului de distribuție

● Testarea și validarea sistemului pentru a se asigura că este capabil să distribuie clienții în cozi în mod efficient

● Documentarea rezultatelor obținute și a îmbunătățirilor implementate pentru a fi utilizate în viitor

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

Scopul simularii în Java a cozilor de așteptare la casa de marcat utilizând thread-uri este de a reproduce comportamentul unei case de marcat reale într-un mediu controlat, unde pot fi monitorizate diferite aspecte, cum ar fi viteza de procesare a cozilor și timpul mediu de așteptare pentru clienți.

Thread-urile sunt unități de execuție independente care pot fi folosite pentru a simula simultaneitatea diferitelor procese care au loc într-o casă de marcat.

“Multithreading” înseamnă capacitatea unui program de a executa mai multe secvenţe de cod în acelaşi timp. O astfel de secvenţă de cod se numeşte fir de execuţie sau thread. Limbajul Java suportă multithreading prin clase disponibile în pachetul java.lang. În acest pachet există 2 clase Thread şi ThreadGroup, şi interfaţa Runnable. Clasa Thread şi interfaţa Runnable oferă suport pentru lucrul cu thread-uri ca entităţi separate, iar clasa ThreadGroup pentru crearea unor grupuri de thread-uri în vederea tratării acestora într-un mod unitar. Există 2 metode pentru crearea unui fir de execuţie: se creează o clasă derivată din clasa Thread, sau se creează o clasă care implementeză interfaţa Runnable. In acest proiect, nu voi lucre cu interfata Runnable.

Pentru a simula o casă de marcat, este necesar să se creeze mai multe thread-uri, fiecare reprezentând o coadă de așteptare diferită. Fiecare coadă de așteptare poate fi reprezentată prin intermediul unei structuri de date, cum ar fi o coadă(Blocking Queue pentru thread - safety), care va fi accesată de fiecare fir de execuție în mod concurent.

Simularea unei case de marcat utilizând thread-uri în Java este un proces complex și detaliat. Scopul acestei simulări este de a oferi o platformă controlată pentru a monitoriza și a optimiza procesul de așteptare și procesare a tranzacțiilor la casa de marcat.

Pentru a simula o coadă de așteptare, putem folosi o coadă în sine pentru fiecare casier și a introduce clienții în coada cu cel mai mic timp de așteptare. Bineinteles, acest lucru doar in cazul in care timpul de sosire a clientului la coada este mai mare sau egal cu timpul curent.

Cateva cazuri de utilizare:

1. Un magazin mare cu mai multe case de marcat dorește să optimizeze timpul de așteptare al clienților și să evite cozi lungi la o anumită casă de marcat.
2. Un magazin online care procesează mai multe comenzii simultan dorește să își optimizeze sistemul de cozi pentru a reduce timpul de așteptare al clienților și pentru a crește satisfacția acestora.
3. O instituție publică cu mai multe ghisee dorește să optimizeze timpul de așteptare al clienților și să evite aglomerația la o anumită ghiseu.

3. Proiectare

Aplicatia are urmatoarea structura:

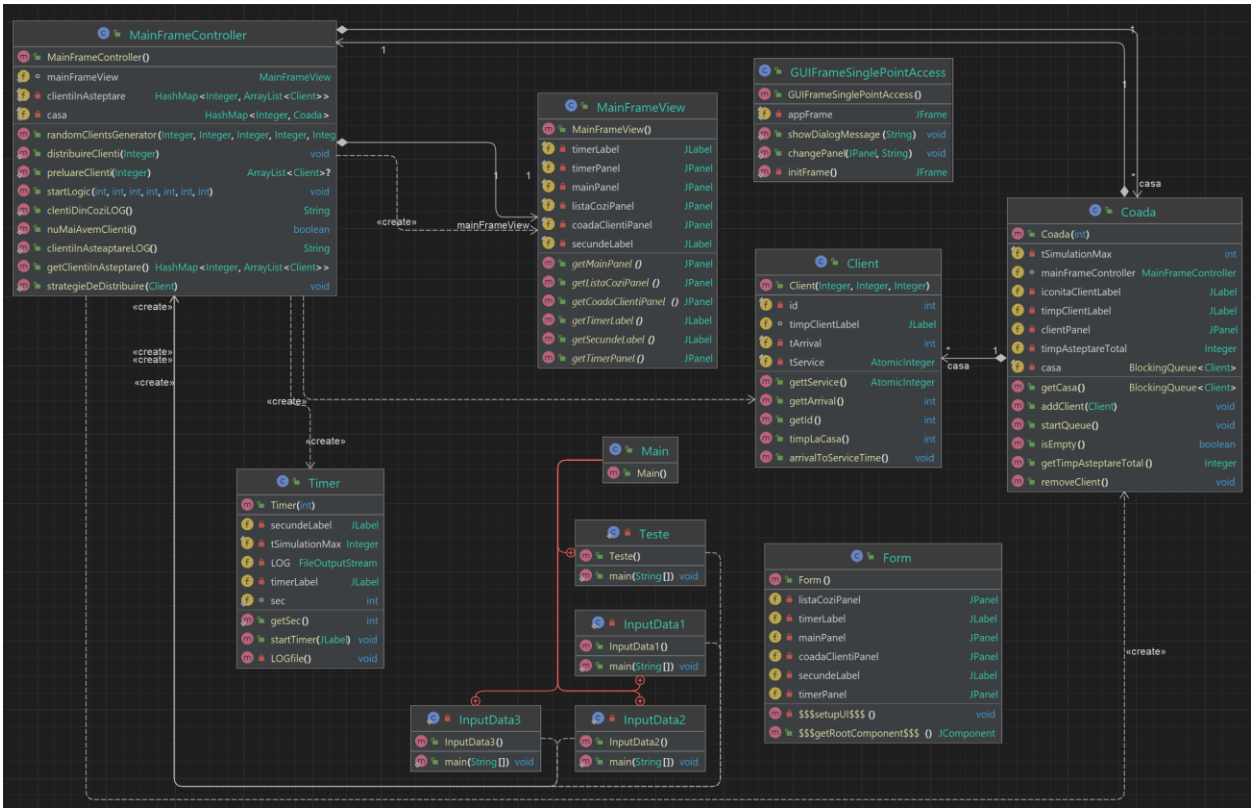


Figura1 – Diagrama UML a aplicatiei

Clasa Main este clasa principală pentru aplicația care simuleaza o coadă de așteptare la casa de marcat. Ea conține patru clase statice interne: InputData1, InputData2, InputData3 și Teste. Fiecare dintre aceste clase interne conține o metodă statică main care inițiază o instanță a clasei MainFrameController și apelează metoda startLogic a acestei instanțe cu anumiti parametrii in functie de ce dorim noi sa testam sau datele de intrare sugerate in cerinta temei. Scopul acestor metode este de a simula diferite scenarii de coadă de așteptare la casa de marcat, folosind valori diferite pentru numărul de clienți, numărul de cozi, timpul de procesare, timpul de sosire și altele.

Main este clasa care este declanșată în momentul rulării programului. Aceasta conține o metodă main, care nu face altceva decât să inițieze instanțele claselor interne InputData1, InputData2, InputData3 și Teste. InputData1 va fi folosita pentru primul set de teste, InputData2 pentru al doilea set de teste, iar InputData3 pentru al treilea set de

teste. Clasa interna Teste este folosita pentru testele pe care utilizatorul doreste sa le faca(acestea nu sunt predefinite in cerinta temei. Toate rezultatele vor fi stocate in fisiere text denumite strategic.

Clasa "MainFrameController" conține metode care implementează logica simulării de cozi, inclusiv generarea aleatoare de clienți, distribuirea acestora în cozi și preluarea clienților din cozi. Aceste metode sunt apelate de către interfața grafică a aplicației.

Variabilele de clasă "mainFrameView", "clientiInAsteptare" și "casa" sunt private și statice. Variabila "mainFrameView" este de tip "MainFrameView" și este utilizată pentru a face referință la obiectul principal al interfeței grafice. Variabila "clientiInAsteptare" este un hash map care reține lista de clienți care așteaptă să fie distribuiți în cozi. Variabila "casa" este un hash map care reține lista de cozi efective.

Metoda "startLogic" primește ca parametri numărul de clienți, numărul de cozi, timpul maxim de simulare, timpul minim și maxim de sosire al clienților și timpul minim și maxim de servire a clienților. În această metodă se creează obiectul "mainFrameView", se generează clienții aleatori, se creează cozi și se pornește timerul pentru simulare.

Metoda "randomClientsGenerator" primește ca parametri numărul de clienți, timpul minim și maxim de sosire al clienților, timpul minim și maxim de servire a clienților și obiectul "clientPanel". Această metodă generează clienți aleatori și îi adaugă în obiectul "clientPanel". De asemenea, metoda adaugă clienții în hash map-ul "clientiInAsteptare", în funcție de timpul de sosire.

Metoda preluareClienti(Integer tCurrent) preia clienții din coada de așteptare a casei de marcat la momentul tCurrent (timpul curent) și îi elimină din această coadă. Dacă nu există clienți în coada de așteptare la momentul tCurrent, metoda returnează null. În plus, metoda elimină clienții preluați din coada de așteptare din interfața grafică a programului.

Metoda distribuireClienti(Integer tCurrent) preia clienții din coada de așteptare și îi distribuie în casele de marcat disponibile, folosind metoda strategieDeDistribuire(Client client). Dacă nu există clienți în coada de așteptare la momentul tCurrent, metoda se termină.

Metoda strategieDeDistribuire(Client client) primește un client și îl distribuie în casa de marcat care are cel mai mic timp total de așteptare. Dacă există o casă de marcat fără clienți, acolo va fi plasat clientul.

Metoda nuMaiAvemClienti() verifică dacă nu mai există clienți în nicio coadă și returneaza true dacă această condiție este îndeplinită, altfel returneaza false.

Metoda clientiInAsteptareLOG() construieste un string folosind un StringBuilder, care contine informatii despre clientii care asteapta sa fie serviti. In primul rand, se adauga string-ul "Waiting clients: " la StringBuilder, iar apoi, prin intermediul unui loop, se itereaza prin valorile din clientiInAsteptare, care este un Map care contine liste de clienti care asteapta in coada. Pentru fiecare client din fiecare coada, se adauga la StringBuilder un string care contine informatii despre id-ul clientului, momentul de sosire si timpul de servire. La final, metoda returneaza string-ul construit..

Metoda clentiDinCoziLOG() construieste un string folosind un StringBuilder, care contine informatii despre clientii care sunt deja in procesul de servire in cadrul fiecarei case de marcat. Parcurgem map-ul “casa” si cu ajutorul functiilor getKey() si getValue() extragem numarul cozii la care suntem (key), respective clientii care sunt in coada respective. Pentru fiecare client din fiecare coada, se adauga la StringBuilder un string care contine informatii despre id-ul clientului, momentul de sosire si timpul de servire. La final, metoda returneaza string-ul construit..

Clasa Client este o clasa care modeleaza un client dintr-un magazin. Ea extinde clasa JLayeredPane pentru a putea fi afisata pe interfata grafica a aplicatiei.

Clasa Client contine trei variabile membru: id (un numar intreg care identifica unic clientul), tArrival (timpul de sosire al clientului in magazin) si tService (timpul necesar pentru ca clientul sa fie deservit la casa de marcat). Toate cele trei variabile sunt initializate in constructorul clasei.

Metoda arrivalToServiceTime() este apelata cand clientul este preluat de la coada si incepe sa fie deservit la o casa de marcat. In panel-ul clientului, in momentul in care clientul ajunge la casa, timpul de sosire(tArrival) este inlocuit cu timpul pe care clientul il va petrece in coada(tService), acesta fiind decrementat la fiecare secunda.

Metoda timpLaCasa() va returna timpul pe care clientul il mai are de petrecut la casa de marcat. Aceasta metoda este folosita pentru a actualiza label-ul din panoul clientului.

Clasa “Coada” implementează o coadă de clienți într-un sistem de simulare a unei case de marcat. Clasa extinde clasa JPanel și are ca membri de date o referință către controllerul clasei principale a aplicației (MainFrameController), un panou pentru client, o etichetă pentru imaginea clientului, o etichetă pentru timpul de așteptare și un timp de simulare maxim.

Constructorul clasei primește timpul maxim de simulare și inițializează câmpurile necesare, setează dimensiunea și aspectul vizual al cozii. Metoda addClient() primește un obiect de tip Client și adaugă acest client la coadă și îl afișează în interfața grafică.

Metoda startQueue() pornește un fir de execuție care se execută în mod continuu și își propune să gestioneze clientii aflați în coadă. Pentru fiecare client din coadă, timpul de așteptare se decrementează cu o unitate la fiecare secundă, iar dacă timpul de petrecut la casa de marcat al clientului a expirat, atunci clientul este eliminat din coadă.

Metoda removeClient() elimină primul client din coadă și actualizează interfața grafică.

Metoda getTimpAsteptareTotal() returnează timpul total de așteptare a tuturor clienților din coadă.

Metoda isEmpty() returnează adevărat dacă coada este goală și fals altfel. Metoda getCasa() returnează coada de clienți.

Clasa Timer este o subclasa a clasei JPanel și este utilizată pentru a afișa in interfata grafica cate secunde au trecut de la inceperea rularii programului și pentru a gestiona distribuirea clienților în cadrul simulării de cozi. Aceasta conține un câmp static sec pentru a stoca numărul de secunde scurse de la începutul simulării și un câmp tSimulationMax pentru a specifica timpul maxim de simulare.

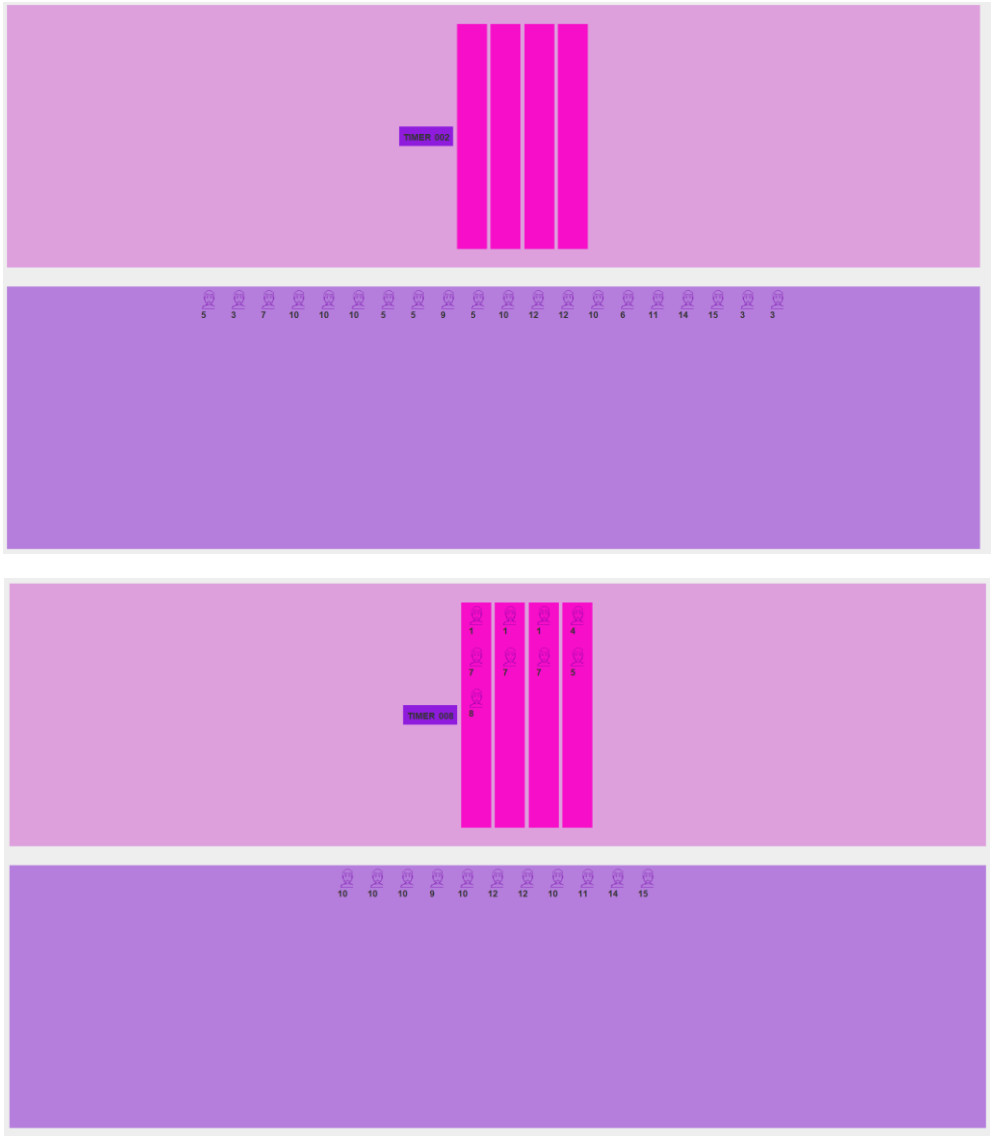
Constructorul primește ca argument tSimulationMax și initializează câmpul corespunzător. De asemenea, clasei i se atașează un fișier LOG prin intermediul câmpului LOG, în care se vor scrie informații despre starea simulării, atat pentru testele propuse in cerinta temei (LOG_test1, LOG_test2, LOG_test3), cat si pentru teste pe care utilizatorul doreste sa le faca(LOG_test).

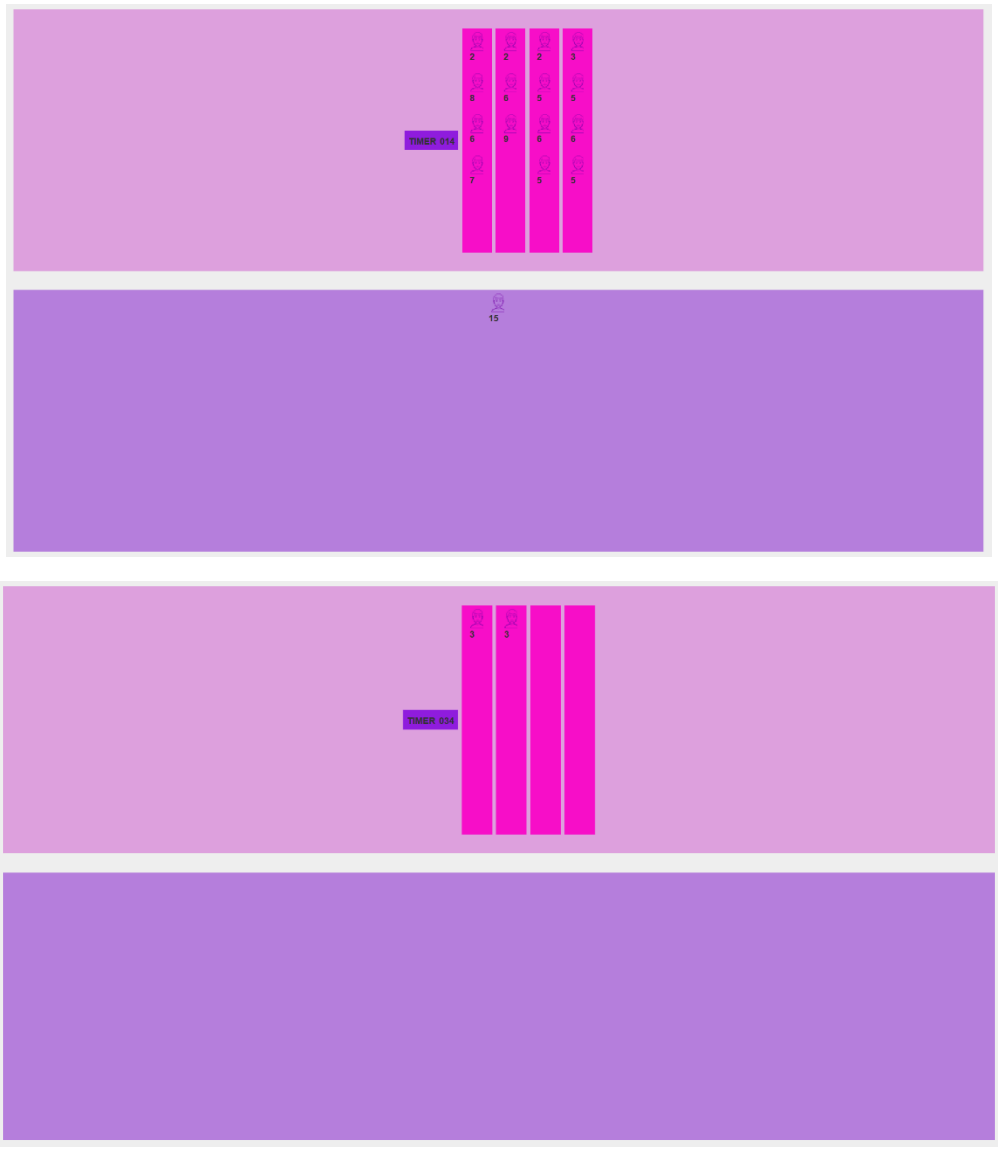
Metoda startTimer(JLabel secundeLabel) pornește un fir de execuție care afișează numărul de secunde pe eticheta secundeLabel, distribuie clienții către cozi prin intermediul metodei distribuireClienti() din clasa MainFrameController și înregistrează informații în fișierul LOG prin apelul metodei LOGfile(). De asemenea, metoda verifică dacă a fost atins timpul maxim de simulare (tSimulationMax) și închide fortat aplicația în consecință. De asemenea, aplicatia va fi inchisa si daca s-au terminat toti clientii de servit(atat clientii care asteptau sa intre la casa, cat si cei de la case).

Metoda LOGfile() scrie informații despre starea simulării în fișierul LOG. Cand se ruleaza programul, in fisierul LOG_testX (unde X este numarul testului, sau X nu exista daca suntem la un test aleatory) se va scrie constant clientii in asteptare, iar pentru fiecare coada se va scrie ce client se afla la timpul current in ea. Daca nu avem niciun client in coada, se va afisa “Closed” pentru coada respectiva.

4. Implementare

In urmatoarele randuri voi prezenta cum functioneaza interfata grafica.





5. Rezultate

La aceasta tema am avut propuse 3 teste pentru a arata corectitudinea functionarii aplicatiei. Este ineficient si imposibil sa ilustrez rezultatele testelor prin intermediul interfetei grafice, asa ca voi atasa fisierele text (LOG-urile de mai sus).

Test 1 :

Date de intrare:

Test 1
N = 4 Q = 2 $t_{simulation}^{MAX}$ = 60 seconds $[t_{arrival}^{MIN}, t_{arrival}^{MAX}] = [2, 30]$ $[t_{service}^{MIN}, t_{service}^{MAX}] = [2, 4]$

Rezultat:

```
Time 1
Waiting clients: (4,22,3);  (3,24,3);  (1,11,4);  (2,28,3);
Queue 1:Closed
Queue 2:Closed

Time 2
Waiting clients: (4,22,3);  (3,24,3);  (1,11,4);  (2,28,3);
Queue 1:Closed
Queue 2:Closed

Time 3
Waiting clients: (4,22,3);  (3,24,3);  (1,11,4);  (2,28,3);
Queue 1:Closed
Queue 2:Closed

Time 4
Waiting clients: (4,22,3);  (3,24,3);  (1,11,4);  (2,28,3);
Queue 1:Closed
Queue 2:Closed

Time 5
Waiting clients: (4,22,3);  (3,24,3);  (1,11,4);  (2,28,3);
Queue 1:Closed
Queue 2:Closed

Time 6
Waiting clients: (4,22,3);  (3,24,3);  (1,11,4);  (2,28,3);
Queue 1:Closed
Queue 2:Closed

Time 7
Waiting clients: (4,22,3);  (3,24,3);  (1,11,4);  (2,28,3);
Queue 1:Closed
Queue 2:Closed

Time 8
Waiting clients: (4,22,3);  (3,24,3);  (1,11,4);  (2,28,3);
Queue 1:Closed
Queue 2:Closed

Time 9
Waiting clients: (4,22,3);  (3,24,3);  (1,11,4);  (2,28,3);
Queue 1:Closed
Queue 2:Closed

Time 10
Waiting clients: (4,22,3);  (3,24,3);  (1,11,4);  (2,28,3);
Queue 1:Closed
Queue 2:Closed

Time 11
Waiting clients: (4,22,3);  (3,24,3);  (1,11,4);  (2,28,3);
Queue 1:Closed
Queue 2:Closed

Time 12
Waiting clients: (4,22,3);  (3,24,3);  (2,28,3);
Queue 1:(1,11,3);
Queue 2:Closed
```

```
Time 13
Waiting clients: (4,22,3);  (3,24,3);  (2,28,3);
Queue 1:(1,11,2);
Queue 2:Closed

Time 14
Waiting clients: (4,22,3);  (3,24,3);  (2,28,3);
Queue 1:(1,11,1);
Queue 2:Closed

Time 15
Waiting clients: (4,22,3);  (3,24,3);  (2,28,3);
Queue 1:Closed
Queue 2:Closed

Time 16
Waiting clients: (4,22,3);  (3,24,3);  (2,28,3);
Queue 1:Closed
Queue 2:Closed

Time 17
Waiting clients: (4,22,3);  (3,24,3);  (2,28,3);
Queue 1:Closed
Queue 2:Closed

Time 18
Waiting clients: (4,22,3);  (3,24,3);  (2,28,3);
Queue 1:Closed
Queue 2:Closed

Time 19
Waiting clients: (4,22,3);  (3,24,3);  (2,28,3);
Queue 1:Closed
Queue 2:Closed

Time 20
Waiting clients: (4,22,3);  (3,24,3);  (2,28,3);
Queue 1:Closed
Queue 2:Closed

Time 21
Waiting clients: (4,22,3);  (3,24,3);  (2,28,3);
Queue 1:Closed
Queue 2:Closed

Time 22
Waiting clients: (4,22,3);  (3,24,3);  (2,28,3);
Queue 1:Closed
Queue 2:Closed

Time 23
Waiting clients: (3,24,3);  (2,28,3);
Queue 1:(4,22,2);
Queue 2:Closed

Time 24
Waiting clients: (3,24,3);  (2,28,3);
Queue 1:(4,22,1);
Queue 2:Closed
```

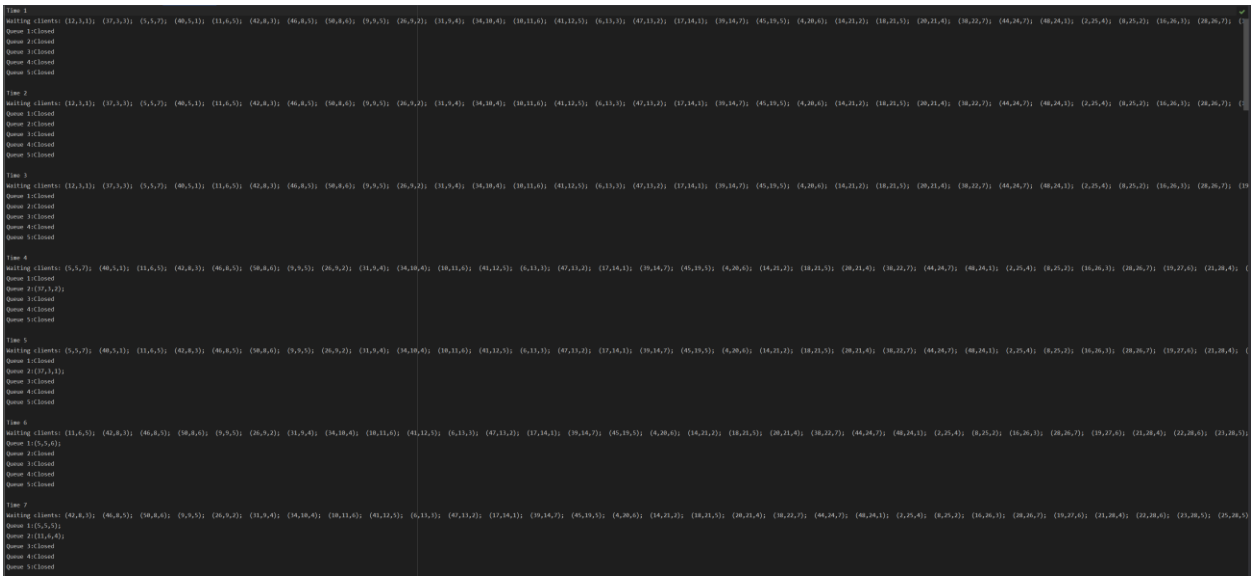
```
Time 25
Waiting clients: (2,28,3);
Queue 1:Closed
Queue 2:(3,24,2);

Time 26
Waiting clients: (2,28,3);
Queue 1:Closed
Queue 2:(3,24,1);

Time 27
Waiting clients: (2,28,3);
Queue 1:Closed
Queue 2:Closed

Time 28
Waiting clients: (2,28,3);
Queue 1:Closed
Queue 2:Closed

Time 29
Waiting clients:
Queue 1:(2,28,2);
Queue 2:Closed

Time 30
Waiting clients:
Queue 1:(2,28,1);
Queue 2:Closed

Time 31
Waiting clients:
Queue 1:Closed
Queue 2:Closed
```

Test 2 :

Date de intrare

Test 2
N = 50 Q = 5 $t^{MAX}_{simulation}$ = 60 seconds $[t^{MIN}_{arrival}, t^{MAX}_{arrival}]$ = [2, 40] $[t^{MIN}_{service}, t^{MAX}_{service}]$= [1, 7]

Rezultat : (voi atasa inceputul si finalul testului, datorita timpului necesar programului sa ajunga la finalul servirii clientilor)



6. Concluzii

Concluzionand, acest proiect m-a ajutat sa-mi aprofundez cunostintele in limbajul Java, precum structurile de date, utilizarea corecta si eficienta a obiectelor etc.

Realizand proiectul am invatat lucruri noi despre thread-uri, despre cum functioneaza acestea si cum le utilizam. De asemenea am invatat sa scriu date intr-un fisier text folosind “FileOutputStream” si sa lucrez cu “AtomicInteger”.

7. Bibliografie

https://www.digitalocean.com/community/tutorials/java-blockingqueue-example

https://howtodoinjava.com/java/multi-threading/atomicinteger-example/

https://www.educative.io/answers/how-to-generate-random-numbers-in-java

https://www.scaler.com/topics/java/exit-in-java/

https://www.javatpoint.com/java-fileoutputstream-class

Cursuri TP