**DOCUMENTAȚIE**

Tema 2

NUME STUDENT: LAURA DAROLTI

GRUPA: 5

CUPRINS

1. Obiectivul temei

2. Proiectare

3. Implementare

4. Rezultate

5. Concluzii

6. Bibliografie

1. Obiectivul Temei

Obiectivul acestei teme este sa implementam o aplicație de gestionare a cozilor care atribuie clienților cozi astfel încât timpul de așteptare să fie minimizat. Obiectivul principal al unei cozi este de a oferi un loc pentru un client înainte de a primi un serviciu. Aplicația de gestionare a cozilor ar trebui să simuleze o serie de N clienți care sosesc pentru serviciu, intră în Q cozi, așteaptă, primesc serviciul și părăsesc în cele din urmă cozile. Toți clienții sunt generați la începutul simulării și sunt caracterizați prin trei parametri: ID, timpul de sosire și serviciul. Aplicația urmărește timpul total pe care fiecare client îl petrece în cozi și calculează timpul mediu de așteptare. Fiecare client este adăugat la coada cu cel mai scurt timp de așteptare atunci când timpul lor de sosire este mai mare sau egal cu timpul de simulare.

1. Proiectare

Pentru proiectarea OOP a aplicației, am creat următoarele clase:

Clasa “Client” reprezintă modelul pentru un client care sosește la coada . Acesta are trei atribute principale : id, timpul de sosire si timpul de servire; acestea sunt însoțite de câteva metode de acces si setare a atributelor.

Clasa “Queue” definește modelul pentru o coada de așteptare si are introduse valori pentru numărul maxim de clienți din coada, timpul de așteptare pentru clienții din coada si doua structuri de tipul unei cozi pentru coada de clienți in așteptare, si coada in care se introduc clienții aflați in așteptare apelând la metodele aferente construite.

Clasele “ConcreteStrategyTime”, ”ConcreteStrategyQueue” si interfața “Strategy”: Interfața este create pentru a defini cele doua moduri prin care se distribuie clientii,iar cele doua clase care implementează aceasta interfața implementează concret aceste strategii.

Clasa “Scheduler” introduce clienții in cozi după strategia aleasa, iar “Simulation Manager” gestionează simularea și interacțiunea cu utilizatorul. Acesta creează un număr de cozi și clienți și le supraveghează până când timpul de simulare este atins.

Diagrama UML de clase este următoarea:

Client\_and\_queue

Queue

addClient(c: Client): void

run(): void

getWaitingPeriod(): AtomicInteger

Client

- ID: int

- arrivalTime: int

- serviceTime: int

Imlplementation

Interface

ConcreteStrategyQueue

addClient(List<Queue> queues, Client c)

ConcreteStrategyTime

addClient(List<Queue> queues, Client c)

Strategy

void addClient(List<Queue> queues, Client c)

SimulationManager

generateClients():List<Client>

run():void

main()

Scheduler

- queues: List<Queue>

- strategy: Strategy

changeStrategy(...): void

getMaxClientsPerServer(): int

Structurile de date folosite :

Structura principală de date utilizată în acest proiect este BlockingQueue din pachetul java.util.concurrent, care este o colecție sigură pentru “Thread-uri” care permite inserarea și eliminarea elementelor într-un mod de blocare atunci când coada este goală sau plină.

Alte structuri utilizate: List,ArrayList etc.

1. Implementare

Clasa Client reprezintă un client care intră în sistem și are un ID unic, timpul de sosire și timpul serviciului.

Clasa ”Queue” reprezintă modelul pentru o coada ai are o capacitate maximă de clienți și o coadă de așteptare. Clasa coadă utilizează un BlockingQueue pentru a stoca clienții care așteaptă să fie serviți. De asemenea, ține evidența perioadei de așteptare, care reprezintă timpul total de așteptare al tuturor clienților din coada de așteptare. Metoda run() a clasei coadă este implementată pentru a simula trecerea timpului, în cazul în care timpul de serviciu al clientului în fruntea cozii este redus cu unul la fiecare secundă. Dacă ora de serviciu a clientului este zero, acesta este eliminat din coadă.

Clasa Scheduler este responsabilă pentru crearea și gestionarea cozilor și utilizează o interfață Strategy pentru a atribui clienți cozilor pe baza algoritmului ales. Sunt implementate două strategii: ConcreteStrategyQueue atribuie clienților cea mai scurtă coadă, iar ConcreteStrategyTime atribuie clienților coada cu cel mai scurt timp de așteptare.

Clasa SimulationManager implementează interfața Runnable, care îi permite să fie executată într-un “Thread” separat. Are mai multe câmpuri care pot fi setate pentru a controla parametrii de simulare, cum ar fi limita de timp, numărul de cozi și clienți, precum și timpii minimi și maximi de sosire și timpii de procesare pentru clienți. De asemenea, are un câmp selectionPolicy care specifică politica utilizată pentru a selecta coada pentru un client.

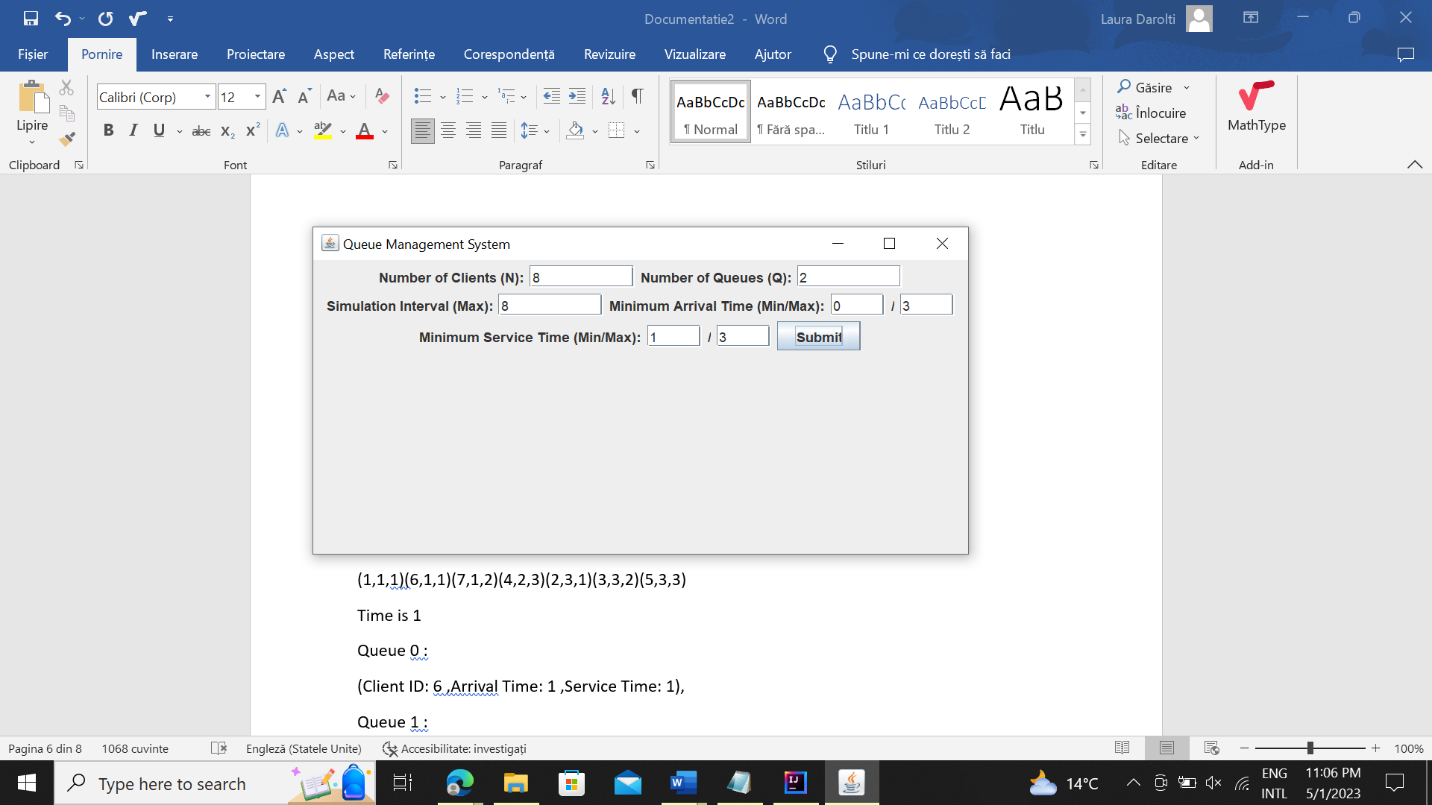
Metoda generateClients generează o listă de clienți cu timpi de sosire aleatorii și timpi de procesare, sortați în ordine crescătoare în funcție de ora sosirii.

Metoda run() este metoda principală a clasei și conține logica de simulare. Inițializează obiectul cu numărul specificat de servere și clienți, generează lista de clienți în așteptare și apoi simulează procesarea clienților prin cozi până când se atinge limita de timp sau nu mai există clienți în cozi. În timpul fiecărei iterații a buclei de simulare, metoda afișează lista clienților în așteptare, trimite clienții la cozi dacă au sosit, simulează procesarea clienților prin cozi pentru o singură unitate de timp și afișează starea cozilor.

Metoda main creează o instanță a clasei SimulationManager și o transmite clasei Interface pentru a fi executată.

Am utilizat clasa TeeOutputStream din biblioteca Apache Commons IO pentru a redirecționa rezultatele consolei atât la consolă, cât și la un fișier numit "Output.txt".

1. Rezultatele unei simulări ale sistemului de management al cozilor:



Waiting clients:

(0,0,1)(1,1,1)(6,1,1)(7,1,2)(4,2,3)(2,3,1)(3,3,2)(5,3,3)

Time is 0

Queue 0 :

(Client ID: 0 ,Arrival Time: 0 ,Service Time: 1),

Queue 1 :

Waiting clients:

(1,1,1)(6,1,1)(7,1,2)(4,2,3)(2,3,1)(3,3,2)(5,3,3)

Time is 1

Queue 0 :

(Client ID: 6 ,Arrival Time: 1 ,Service Time: 1),

Queue 1 :

(Client ID: 1 ,Arrival Time: 1 ,Service Time: 1), (Client ID: 7 ,Arrival Time: 1 ,Service Time: 2),

Waiting clients:

(4,2,3)(2,3,1)(3,3,2)(5,3,3)

Time is 2

Queue 0 :

(Client ID: 4 ,Arrival Time: 2 ,Service Time: 3),

Queue 1 :

(Client ID: 7 ,Arrival Time: 1 ,Service Time: 2),

Waiting clients:

(2,3,1)(3,3,2)(5,3,3)

Time is 3

Queue 0 :

(Client ID: 4 ,Arrival Time: 2 ,Service Time: 2), (Client ID: 2 ,Arrival Time: 3 ,Service Time: 1), (Client ID: 5 ,Arrival Time: 3 ,Service Time: 3),

Queue 1 :

(Client ID: 7 ,Arrival Time: 1 ,Service Time: 1), (Client ID: 3 ,Arrival Time: 3 ,Service Time: 2),

Waiting clients:

Time is 4

Queue 0 :

(Client ID: 4 ,Arrival Time: 2 ,Service Time: 1), (Client ID: 2 ,Arrival Time: 3 ,Service Time: 1), (Client ID: 5 ,Arrival Time: 3 ,Service Time: 3),

Queue 1 :

(Client ID: 3 ,Arrival Time: 3 ,Service Time: 2),

Waiting clients:

Time is 5

Queue 0 :

(Client ID: 2 ,Arrival Time: 3 ,Service Time: 1), (Client ID: 5 ,Arrival Time: 3 ,Service Time: 3),

Queue 1 :

(Client ID: 3 ,Arrival Time: 3 ,Service Time: 1),

Waiting clients:

Time is 6

Queue 0 :

(Client ID: 5 ,Arrival Time: 3 ,Service Time: 3),

Queue 1 :

Waiting clients:

Time is 7

Queue 0 :

(Client ID: 5 ,Arrival Time: 3 ,Service Time: 2),

Queue 1 :

5.Concluzii

In concluzie, tema prezentata implementează un sistem de managment cu mai multe cozi și algoritmi de atribuire si procesare ai clienților .

6.Bibliografie

[Redirecting System.out.println() Output to a File in Java - GeeksforGeeks](https://www.geeksforgeeks.org/redirecting-system-out-println-output-to-a-file-in-java/)

[Java Threads (w3schools.com)](https://www.w3schools.com/java/java_threads.asp)

[Guide to java.util.concurrent.BlockingQueue | Baeldung](https://www.baeldung.com/java-blocking-queue)