Facultatea de Automatica si Calculatoare

Departamentul Calculatoare

**Queues Simulator**

(Documentatie)

Stef Paula-Elena

Grupa: 30221

An academic: 2020-2021

Disciplina: Tehnici de Programare fundamentala

Cuprins:

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

Simularea unei aplicatii care are in vedere analiza unui sistem bazat pe cozi in scopul determinarii si a minimizarii timpului mediu de asteptare al clientilor.

Aceasta aplicatie dezvolta urmatoarele obiective secundare, care urmeaza a fi detaliate in capitolul de **Implementare**:

- preularea datelor de intrare.(atat din intergata cat si din fisierul de input)

- generarea cozilor de asteptare si implementarea metodei de distribuire a clientilor.

- generarea unei liste de clienti cu date de caracterizare arbitrare si sortate in functie de timpul sosirii in coada.

- simularea acestei distribuiri si a implementarii cozilor de asteptare intr-un anumit interval de timp.

De asemenea, pentru a putea lucra cu aplicatia, era nevoie si de o interfata grafica prin care utilizatorul sa poata introduce anumite date necesare pentru derularea simularii, cum ar fi timpul simularii, numarul de clienti, numarul de cozi la care acestia vor fi distribuiti si doua intervale de timp pentru a se putea genera momentul sosirii si timpul necesar prelucrarii nevoilor clientului.

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

Scopul principal a acestei teme este de a rezolva problema asteptarii de lunga durata la o coada, care apare in zonele aglomerate sau la orele de varf. Scopul principal al unei cozi este de a oferi un loc unui client unde sa astepte momentul in care acesta va fi servit. Administrarea unui astfel de sistem se intereseaza in minimizarea perioadei de timp la care sunt supusi clientii in coada de asteptare inainte ca acestia sa fie serviti. Astfel se pune problema alegerii unei metode corespunzatoare pentru a reduce pe cat posibil timpul de asteptare, una dintre cele mai eficiente fiind alegerea cozii in care urmatorul client este plasat bazandu-ne pe timpul total de asteptare alclientiilor aflati deja la coada respectiva.Aceasta metoda,desi obtima in teorie este mai greu de adaptat in practica.O alta abordare ar mai fii sa alegem coada bazandu-ne pe numarul de clienti aflati la ea, insa aceasta abordare este mai putin obtima.

Aplicatia va trebui sa simuleze (fiind definit un anumit timp de simulare/rulare) N clienti care ajung la un anumit timp cu scopul de a fi serviti, intra in cozile de asteptare, asteapta, vor fi serviti si vor parasi definitiv cozile. Toti clientii vor fi generati intr-un mod aleator cand incepe aceasta simulare, iar fiecare client este caracterizat de 3 parametri: numar de identificare (un numar intre 1 si N), timpul de sosire (timpul simularii in care acesta se poate aseza la coda si timpul de servire (intervalul de timp sau durata necesara casierului de a servi clientul. Aplicatia tine cont de timpul total petrecut in coada a fiecarui client. Fiecare client este adaugat cozii reprezentate de cel mai mic timp de asteptare sau care are cei mai putini clienti,atunci cand timpul de sosire este mai mare sau egal cu cel ai timpului de simulare curent.

Datele de intrare pentru aplicatie trebuie considerate ca fiind preluate dintr-un fisier de tip text, avand urmatoarea structura:

- Numarul de clienti (N);

- Numarul de cozi(Q);

- Intervalul de simulare (timpul maxim de simulare);

- Timpul minim si maxim de sosire

(timp minim de sosire <= timp de sosire <= timp maxim de sosire);

- Timpul minim si maxim de servire

(timp minim de servire <= timp de servire <= timp maxim de servire);

Dupa procesarea acestor date se genereaza un scenariu aleator, care creeaza numarul de clienti specificat si le atribuie un ID si un timp de sosire si de procesare pentru a putea privi mai apoi simularea . Plasam fiecare client la coada care are timpul de asteptare cel mai mic, nu la coada cu cei mai putini clienti, deoarece am considerat ca aceasta abordare este mai practica.

Pentru a putea observa progresul in “real time” am afisat cozile,clientii si timpul current in interfata aplicatiei, dar de asemenea pentru fiecare simulare ni se va genera un fisier output de tip txt care va contine jurnalul de executie al aplicatiei (starea timpului de simulare, lista clientilor in asteptarea timpului de sosire, cozile de asteptare cu clientii inserati pentru servire) si timpul mediu de asteptare in cozi pentru un viitor client.

De exemplu:

Input:

4

2

60

2, 30

2, 4

Output:

Time: 0

Waiting clients: (1, 30, 15);(2, 30, 19);(3, 30, 7);(4, 30, 5);

Queue 0: Closed

Queue 1: Closed

…………

Time: 29

Waiting clients: (1, 30, 15);(2, 30, 19);(3, 30, 7);(4, 30, 5);

Queue 0: Closed

Queue 1: Closed

Time: 30

Waiting clients: (1, 30, 15);(2, 30, 19);(3, 30, 7);(4, 30, 5);

Queue 0: Closed

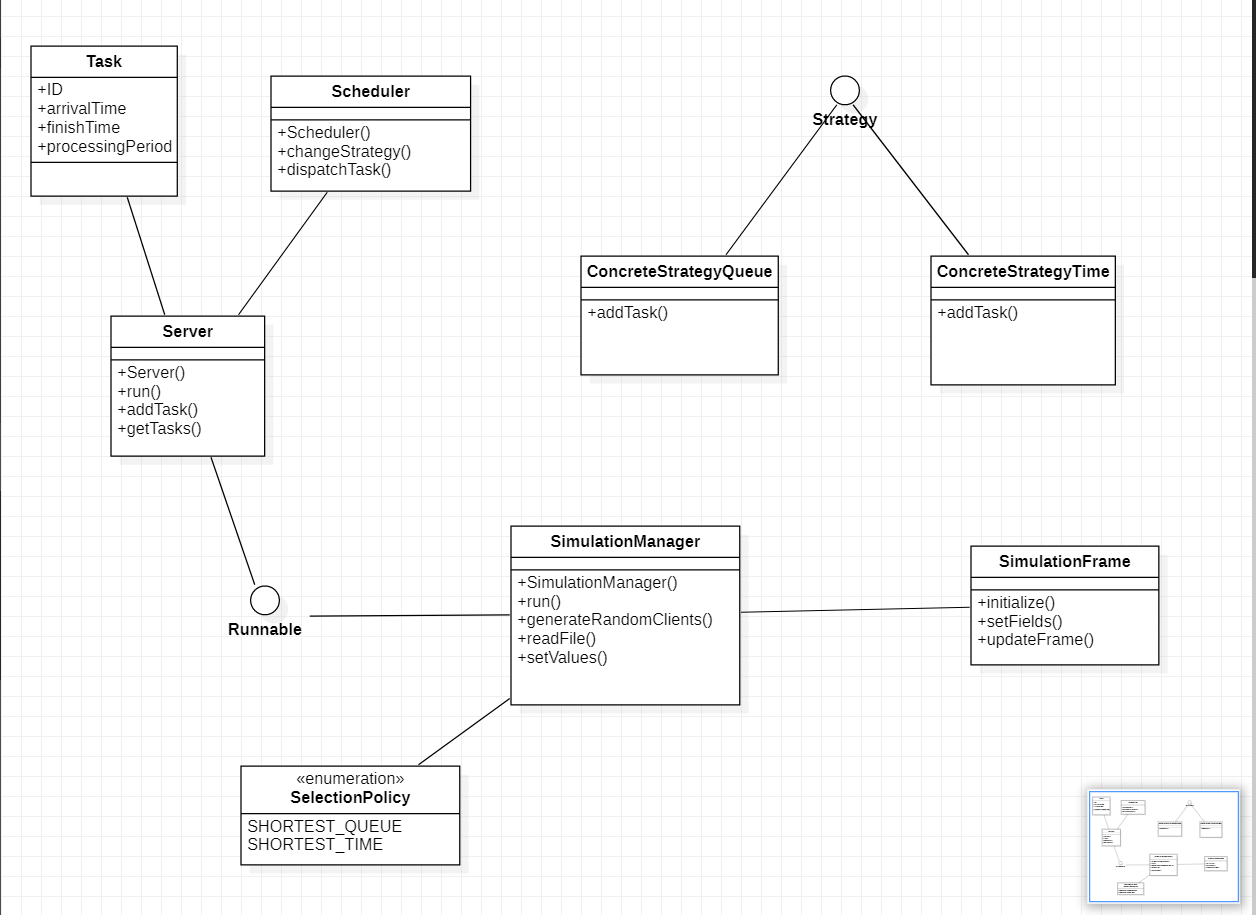
Queue 1: Closed

……………..

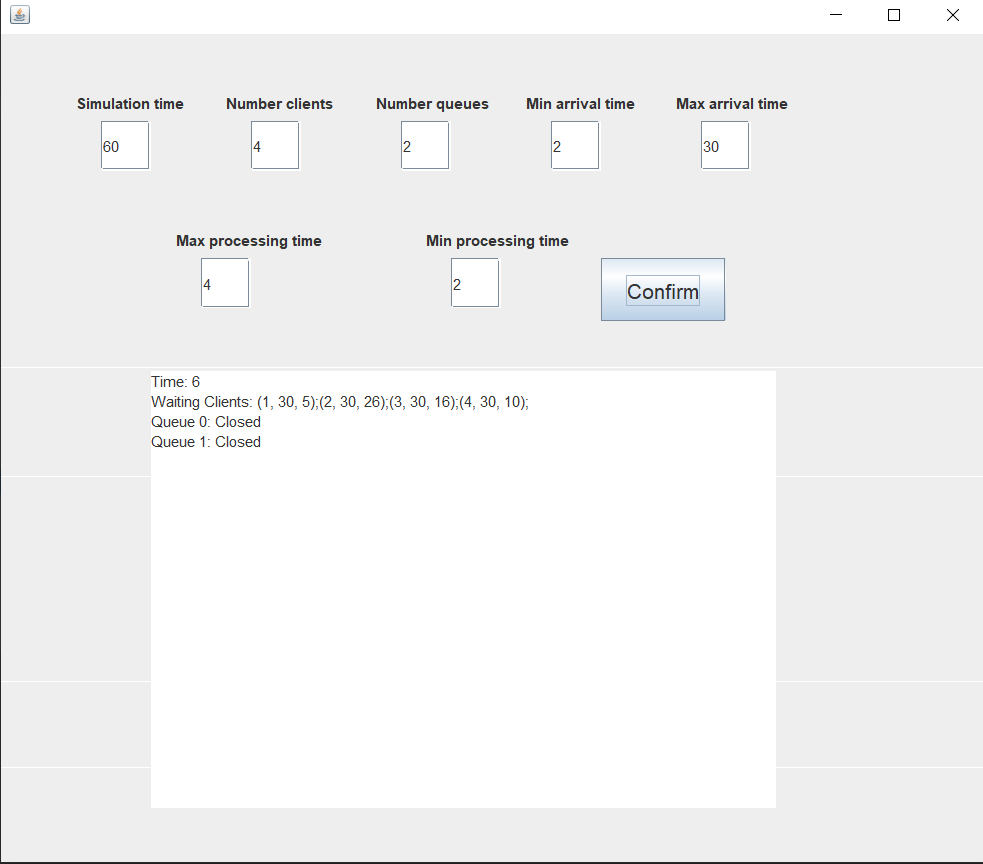
Average waiting time: 7.5

1. **Proiectare**

Diagrama UML:



Am rezolvat cerintele acestei probleme folosindu-ma de clasele: Task, SimulationManager, Server, SimulationFrame, Scheduler, ConcreteStrategyTime, ConcreteStrategyQueue, de enumeratia SelectionPolicy si de interfetele Strategy si Runnable. De asemenea am folosit pachetele java.util.\*, java.awt.\*, java.swing.\*, java.util.ArrayList, java.util.Random, oferind astfel o interfata user friendly si o aplicatie usor de folosit. Am avut nevoie de clasa SimulationFrame pentru a implementa interfata grafica care preia datele de la utilizator si afiseaza simularea in timp real, afisand pe rand fiecare client care apare intr-o coada si il sterge mai apoi cand iese. Clasa Task prezinta obiectul principal al aplicatiei noastre, adica clientul si clasa SimulationManager se ocupa de simularea intregului scenario,atribuind corespunzator clientii (Task-urile) catre coada corespunzatoare (Server).



1. Implementare

Clasa Task (care reprezinta clientii) are campurile

* ID : valoare intreaga, care reprezinta identificatorul task-ului (clientului)
* arrivalTime : valoare intreaga, care reprezinta timpul de sosire al task-ului (clientului)
* finishTime : valoare intreaga care reprezinta timpul la care clientul a parasit coada
* processingPeriod : valoare intreaga, care reprezinta timpul de servire (procesare) al task-ului

Toate aceste attribute sunt suficiente pentru a definii clientul.De asemenea avet getter-i si setter-I pentru anumite campuri, metoda toString care o folosim pentru afisarea atat in fisier cat si in interfata,iar clasa Task mai implementeaza si interfata Comparable<> pentru a putea ordona clientii in functie de timpul in care acestia ajung la coada.

Clasele ConcreteStrategyTime si ConcreteStrategyQueue implementeaza interfata Strategy, care contine metoda addClient. Diferenta intre aceste doua clase este politica dupa care se introduc clientii intr-o coada. In prima clasa, fiecare client este repartizat la coada cu timpul de asteptare minim, respectiv pentru cea de a doua clasa clientul este adaugat in coada care contine cei mai putini clienti. Am implementat ambele metode desi pentru simulare am decis sa folosesc doar metoda prima metoda, considerand ca este cea obtima.

Enumeratia SelectionPolicy contine cele doua politici de introducere a clientilor in cozi, si anume SHORTEST\_QUEUE, respectiv SHORTEST\_TIME.

Clasa Scheduler are ca atribute:

* ‘servers’ - de tip ArrayList<Server> care va memora lista de servere (cozi) dat de campul ‘numberOfServer’ din clasa SimulationManager.
* ‘maxNoServers’ – o valoare intreaga care imi va da numarul de cozi (ghisee) care vor fi create
* ‘maxTasksPerServer’ – numarul maxim de clienti care se pot afla la o coada
* ‘strategy’ – de tip SelectionPolicy si reprezinta metoda de a dispune clientii la cozi

In constructorul clasei se seteaza atributele, se craeaza si initializeaza cozile (server), si de asemenena se porneste thread-rul pentru fiecare coada, acestea realizandu-se in paralel.

Metoda changeStrategy schimba abordarea dorita pentru a plasa clientii la urmatoarea coada, in timp ce metoda dispatchTask imi asigura plasarea efectiva (adica adaugarea in lista de clienti ai cozilor) a clientiilor la coada dorita.

Clasa Server

* ‘tasks’ - lista de tip BlockigQueue<Task> care memoreaza task-uri (clienti) si are caracter de ThreadSafe pentru a fi utilizat in adaugarea si stergerea elementelor in interiorul unui Thread si

poate bloca Thread-ul curent in cazul in care nu mai are elemente.

* ‘waitingPeriod’ - de tip AtomicInteger deoarece asigura o stare stabila pentru un intreg cand acesta este accesat de mai multe Thread-uri si reprezinta timpul de asteptare pentru urmatorul task (client) adaugat la server (coada/queue) pentru a-i incepe servirea (procesarea).
* ‘totalWaitingTime’ - camp static cu rolul de a insuma totalitatea timpului de asteptare la fiecare perioada de timp trecuta (secunda) de la fiecare server (coada/queue) pentru a putea determina la final timpul mediu de asteptare al unui task (client).
* totalProcessedTasks – camp static care are rolul de a numara task-urile(clientii) care si-au incheiat timpul de procesare,sau mai bine zis,care au parasit coada,inainte de atingerea timpului limita
* runThreadTime – o variabila booleana care este creata pentru a verifica daca coada mai este sau nu deschisa, si utilizata in metoda run ca si conditie pentru contiunarea thread-ului.

In constructor ne intializam lista de task-uri si waitingPeriod, metoda addTask care adauga un nou client la lista de client ai cozii si aduna la waitingPeriod timpul de procesare al noului client. Metoda getTasks care doar imi returneaza lista de client sub forma de sir de task-uri pentru a fii mai accesibila in anumite metode, iar in metoda Run, in cadrul thread-ului (atata timp cat coada mai este deschisa) pentru fiecare client (luat cu ajutorul lui peek() ) ni se realizeaza o intarziere in secunde egala cu processingPeriod-ul respective task-ului current.Dupa aceasta clientul este eliminat din lista de client, el sfarsindu-si tranzactia.

Clasa SimulationManager

* ‘numberOfClients’ - numar intreg, care reprezinta numarul total de task-uri (clienti) care se vor genera arbitrar.
* ‘numberOfServers’ - numar intreg, care reprezinta numarul total de servere/queues (cozi) care se vor genera pentru a fi populate cu o distributie uniforma a task-urilor (clientilor).
* ‘timeLimit’ - numar intreg, care reprezinta numarul maxim de secunde pentru limita de simulare a acestei aplicatii.
* ‘minArrivalTime’ - numar intreg, care reprezinta timpul minim de sosire care poate fi generat pentru un task (client)
* ‘maxArrivalTime’ - numar intreg, care reprezinta timpul maxim de sosire care poate fi generat pentru un task (client).
* ‘minProcessingTime’ - numar intreg, care reprezinta timpul minim de servire (procesare) care poate figenerat pentru un task (client).
* ‘maxProcessingTime’ - numar intreg, care reprezinta timpul maxim de servire (procesare) care poate fi generat pentru un task (client).
* ‘scheduler’ - o instant/obiect al clasei Scheduler care face repartizarea task-urilor (clientilor) in servere(cozi).
* ‘generatedTasks’ - o instanta a tipului ArrayList<Task>, in care se vor stoca ‘numberOfClients’ de task-uri (clienti) sortati in ordine crescatoare.
* ‘writer’ - declarat ca si camp al clasei, deoarece se necesita vizibilitatea lui in metoda run() (fata deconstructorul clasei unde este initiat) pentru a executa comenzile de scriere a datelor infisierul de iesire.
* ‘selectionPolicy’ care imi indica tipul de strategie care va fi abordat in cadrul simularii
* ‘frame’ – de tip SimulationFrame cu ajutorul caruia imi realizez interfata

Din nou,in cadrul constructorului imi sunt initializate atributele si de asemenea imi este apelata metoda de generare a task urilor (clientiilor) in mod arbitrar, si anume metoda generateRandomTasks(),care imi alege in mod arbitrar (dar incadrate intre valorie minime si maxime date) un arrivalTime si processingTime si imi creeaza si adauga un nou client care are aceste attribute. Se creeaza “numberOfClients” task-uri care dupa sunt sortate dupa timpul lor de sosire.

Avem metoda ReadFile care este folosita pentru a introduce datele necesare dintr-un fisier daca se doreste acest lucru.

Metoda run() adauga pe rand fiecare client la coada potrivita, in functie de strategia aleasa si la momentul in care timpul este acelasi cu timpul de sosire al clientului.Dupa se updateaza fereastra care ii prezinta utilizatorului parcursul simularii folosind un obiect din clasa SimulationFrame (frame). Tot in cadrul acestei clase avem si metoda setValues prin intermediul careia se preiau valorile introduce in cadrul interfetei si se seteaza atributele corespunzatoare din cadrul clasei (echivalentul citirii din fisier).

Clasa Simulation Frame

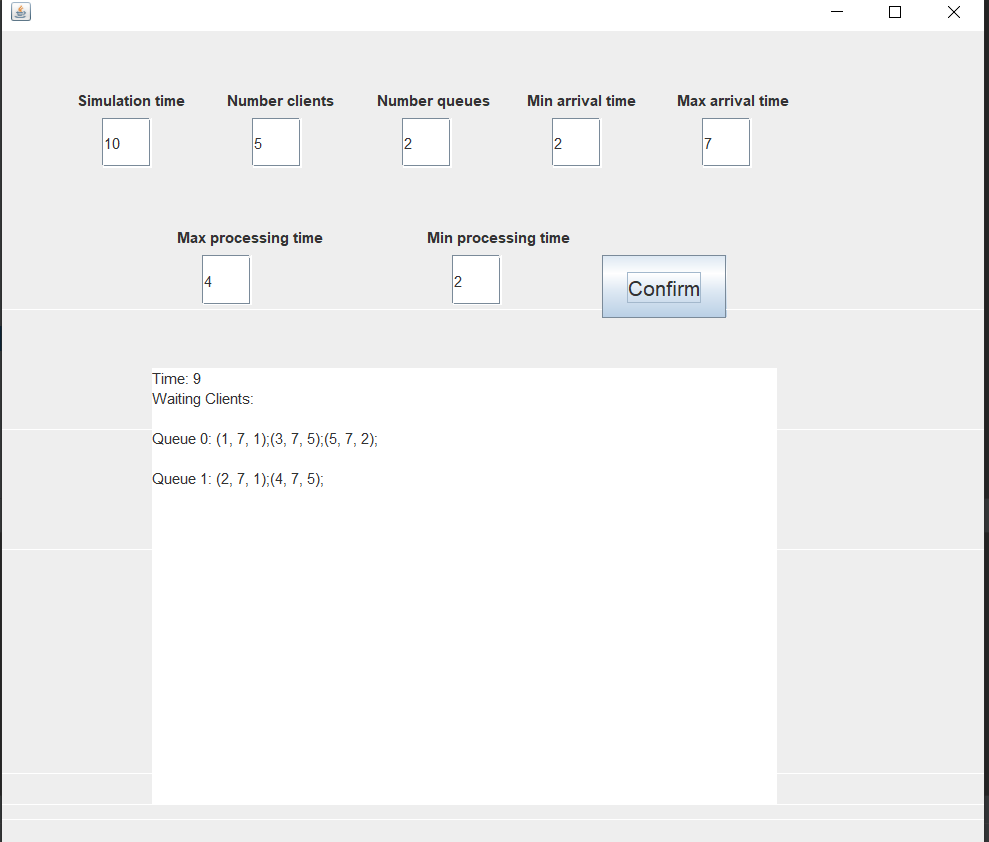
In cadrul acestei clase avem mai multe campuri de tip JTextField care imi ofera posibilitatea de a introduce datele necesare de input prin intermediul interfetei, un camp de tip JButton care in momentul cand este apasat imi incepe simularea si un JTextArea in care mi se va afisa in timp real simularea : Timpul curent, clientii care inca nu s-au asezat la coada si toate cozile si clientii care se afla in moment la ele. Astfel in metodele initialize(),setFields() doar imi initializez si setez toate aceste campuri,impreuna cu label-uri corespunzatoare pentru a face utilizarea aplicatiei practica. In metodata updateFrame imi este actualizat campul result (JTextArea) astfel incat acesta sa imi afiseze corespunzator si lizibil timpul current,clientii care asteapta sa intre la cozi si de asemenea, cozile si clientii care se afla la fiecare dintre acestea.Aceasta metodaeste apelata in SimulationManager cu scopul de a modifica in “timp real” cozile si clientii.

1. **Rezultate**

Din implementarea acestei aplicatii a rezultat o simulare a cozilor de asteptare cu clienti, care optimizeaza timpul de asteptare al clientilor, adaugandu-l pe fiecare client la coada in care va avea cel mai scurt timp de asteptare pana la preluarea.

Aceste rezultate pot fi observate atat ca rezultat total in fisierul txt ,cat si in timp real prin intermediul interfetei.

Resultatul interfetei:



Resultatul fisierului output.txt pentru testul 1:

4

2

60

2,30

2,4

Time: 0  
Waiting clients: (3, 7, 29);(1, 17, 15);(2, 25, 26);(4, 26, 14);  
Queue 0: Closed   
Queue 1: Closed   
  
Time: 1  
Waiting clients: (3, 7, 29);(1, 17, 15);(2, 25, 26);(4, 26, 14);  
Queue 0: Closed   
Queue 1: Closed

……………………

Time: 22  
Waiting clients: (2, 25, 26);(4, 26, 14);  
Queue 0: (3, 7, 14);  
Queue 1: (1, 17, 10);  
  
Time: 23  
Waiting clients: (2, 25, 26);(4, 26, 14);  
Queue 0: (3, 7, 13);  
Queue 1: (1, 17, 9);

…………..

Time: 57  
Waiting clients:   
Queue 0: Closed   
Queue 1: (2, 25, 1);  
  
Time: 58  
Waiting clients:   
Queue 0: Closed   
Queue 1: Closed   
  
Time: 59  
Waiting clients:   
Queue 0: Closed   
Queue 1: Closed   
  
Average waiting time: 3.25

Output pentru testul 2:

50

5

60

2,40

1,7

Time: 0  
Waiting clients: (21, 2, 15);(43, 3, 20);(5, 4, 6);(33, 7, 3);(44, 7, 36);(2, 9, 12);(12, 9, 40);(17, 9, 35);(29, 9, 16);(16, 10, 11);(38, 10, 24);(30, 12, 11);(23, 13, 25);(26, 13, 13);(47, 14, 31);(4, 15, 25);(42, 16, 18);(37, 17, 2);(24, 18, 1);(35, 18, 28);(48, 19, 32);(15, 20, 37);(20, 20, 40);(22, 21, 30);(27, 22, 3);(32, 22, 39);(46, 22, 12);(49, 22, 35);(1, 23, 23);(11, 23, 37);(13, 23, 4);(40, 25, 33);(45, 26, 29);(41, 27, 23);(25, 29, 31);(7, 30, 36);(3, 31, 19);(36, 31, 26);(6, 32, 24);(10, 32, 38);(18, 32, 15);(31, 32, 10);(8, 34, 5);(39, 36, 29);(9, 37, 10);(34, 37, 40);(50, 37, 35);(14, 39, 27);(19, 40, 32);(28, 40, 5);  
Queue 0: Closed   
Queue 1: Closed   
Queue 2: Closed   
Queue 3: Closed   
Queue 4: Closed   
  
Time: 1  
Waiting clients: (21, 2, 15);(43, 3, 20);(5, 4, 6);(33, 7, 3);(44, 7, 36);(2, 9, 12);(12, 9, 40);(17, 9, 35);(29, 9, 16);(16, 10, 11);(38, 10, 24);(30, 12, 11);(23, 13, 25);(26, 13, 13);(47, 14, 31);(4, 15, 25);(42, 16, 18);(37, 17, 2);(24, 18, 1);(35, 18, 28);(48, 19, 32);(15, 20, 37);(20, 20, 40);(22, 21, 30);(27, 22, 3);(32, 22, 39);(46, 22, 12);(49, 22, 35);(1, 23, 23);(11, 23, 37);(13, 23, 4);(40, 25, 33);(45, 26, 29);(41, 27, 23);(25, 29, 31);(7, 30, 36);(3, 31, 19);(36, 31, 26);(6, 32, 24);(10, 32, 38);(18, 32, 15);(31, 32, 10);(8, 34, 5);(39, 36, 29);(9, 37, 10);(34, 37, 40);(50, 37, 35);(14, 39, 27);(19, 40, 32);(28, 40, 5);  
Queue 0: Closed   
Queue 1: Closed   
Queue 2: Closed   
Queue 3: Closed   
Queue 4: Closed

…………………………………

Time: 58  
Waiting clients:   
Queue 0: (4, 15, 19);(20, 20, 40);(13, 23, 4);(40, 25, 33);(3, 31, 19);(10, 32, 38);(19, 40, 32);  
Queue 1: (42, 16, 18);(15, 20, 37);(11, 23, 37);(7, 30, 36);(8, 34, 5);(9, 37, 10);(14, 39, 27);  
Queue 2: (26, 13, 4);(37, 17, 2);(24, 18, 1);(35, 18, 28);(27, 22, 3);(32, 22, 39);(45, 26, 29);(6, 32, 24);(39, 36, 29);(28, 40, 5);  
Queue 3: (47, 14, 23);(22, 21, 30);(49, 22, 35);(25, 29, 31);(18, 32, 15);(34, 37, 40);  
Queue 4: (23, 13, 10);(48, 19, 32);(46, 22, 12);(1, 23, 23);(41, 27, 23);(36, 31, 26);(31, 32, 10);(50, 37, 35);  
  
Time: 59  
Waiting clients:   
Queue 0: (4, 15, 18);(20, 20, 40);(13, 23, 4);(40, 25, 33);(3, 31, 19);(10, 32, 38);(19, 40, 32);  
Queue 1: (42, 16, 17);(15, 20, 37);(11, 23, 37);(7, 30, 36);(8, 34, 5);(9, 37, 10);(14, 39, 27);  
Queue 2: (26, 13, 3);(37, 17, 2);(24, 18, 1);(35, 18, 28);(27, 22, 3);(32, 22, 39);(45, 26, 29);(6, 32, 24);(39, 36, 29);(28, 40, 5);  
Queue 3: (47, 14, 22);(22, 21, 30);(49, 22, 35);(25, 29, 31);(18, 32, 15);(34, 37, 40);  
Queue 4: (23, 13, 9);(48, 19, 32);(46, 22, 12);(1, 23, 23);(41, 27, 23);(36, 31, 26);(31, 32, 10);(50, 37, 35);  
  
Average waiting time: 76.24

Output pentru fisierul 3:

1000

20

200

10,100

3,9

Time: 199  
Waiting clients:   
Queue 0: (217, 18, 50);(116, 22, 100);(741, 25, 9);(156, 26, 20);(958, 26, 78);(18, 29, 15);(512, 29, 96);(234, 32, 99);(355, 36, 68);(409, 38, 40);(151, 40, 16);(226, 41, 67);(140, 43, 74);(632, 45, 43);(899, 46, 61);(888, 48, 55);(53, 51, 58);(810, 52, 30);(276, 53, 67);(348, 55, 47);(606, 57, 48);(379, 58, 64);(948, 60, 39);(659, 62, 41);(739, 63, 54);(696, 65, 52);(428, 67, 46);(963, 68, 9);(408, 69, 30);(521, 70, 75);(841, 72, 32);(594, 74, 96);(279, 77, 44);(87, 79, 89);(785, 81, 41);(117, 84, 20);(653, 84, 37);(515, 85, 29);(114, 87, 72);(37, 90, 82);(661, 92, 86);(230, 95, 95);(246, 98, 89);  
Queue 1: (290, 19, 14);(75, 21, 80);(557, 23, 71);(309, 26, 17);(994, 26, 69);(829, 28, 65);(877, 30, 6);(911, 30, 36);(47, 32, 6);(223, 32, 90);(592, 35, 54);(695, 37, 50);(72, 39, 92);(817, 42, 45);(656, 44, 71);(271, 46, 30);(798, 47, 29);(708, 48, 94);(808, 51, 69);(453, 53, 42);(607, 54, 89);(15, 58, 83);(372, 60, 57);(25, 63, 24);(285, 63, 96);(729, 66, 23);(786, 67, 23);(535, 68, 78);(283, 71, 31);(173, 72, 37);(467, 73, 38);(800, 75, 69);(245, 77, 9);(824, 77, 92);(724, 80, 50);(676, 83, 23);(853, 83, 48);(222, 85, 55);(294, 87, 15);(104, 88, 63);(450, 90, 53);(391, 92, 4);(545, 92, 94);(416, 95, 60);(123, 97, 72);(48, 100, 7);(108, 100, 14);(380, 100, 93);

………………

Average waiting time: 1238.511

**6.Concluzii**

Din aceasta tema s-au putut invata idei importante in ceea ce privesc programele care necesita implementari cu Thread-uri, sincronizarea lor, intelegerea si invatarea metodelor de abordare si executie a acestora la fel ca si utilizarea unor tipuri de obiecte/variabile specifice lucrului cu Thread-uri (sincronizate).

**7.Bibliografie**

* <http://docs.oracle.com/javase/tutorial/essential/concurrency/index.html>
* <https://www.w3schools.com/java/java_threads.asp>
* <https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/atomic/AtomicInteger.html>
* <https://docs.oracle.com/javase/7/docs/api/java/util/concurrent/BlockingQueue.html>