

Tema 2

Moldovan Radu Octavian
Grupa 30224

CUPRINS

1. Obiectivul temei	3
2. Analiza problemei si cazurile de utilizare	4
3. Proiectare	9
4. Implementare.....	12
5. Rezultate.....	13
6. Concluzii	14
7. Bibliografie	15

1. **Obiectivul temei**

Obiectivul principal:

Se dorește crearea unei aplicații utilizată în simularea gestiunii unui anumit număr de clienți ce trebuie atribuiți unor cozi, prin mai multe metode, astfel încât timpul de așteptare până la servire pentru fiecare client să fie minim.

Obiectivele secundare:

- Generarea listei de clienți în mod aleator, pe baza datelor de intrare furnizate de utilizator
- Ilustrarea componentei fiecărei cozi disponibile în timp real
- Verificarea și validarea datelor de intrare furnizate de către utilizator și semnalarea erorilor
- Calculul parametrilor specifici simulării, precum: numărul maxim de clienți în așteptare și ora evenimentului, timpul mediu de servire, timpul mediu de așteptare
- Interceptarea erorilor la rulare pentru a nu permite ca aplicația să se afle într-o stare nepermisă
- Furnizarea unei interfețe grafice pentru utilizator ce are rolul de a facilita introducerea parametrilor ce caracterizează simularea
- Oprirea funcționării programului în momentul în care a fost depășit timpul alocat simulării sau dacă fiecare client generat a fost servit

2. Analiza problemei si cazurile de utilizare

Analiza problemei:

Pentru realizarea simulării se creează un număr de cozi specificat de utilizator, acestea urmând să fie populate cu clienții generați aleator din lista datelor neasignate. Un client este plasat într-o coadă pentru a putea fi servit, însă doar în momentul în care timpul de simulare este egal cu timpul de sosire al clientului. Plasarea clientului se face astfel încât timpul de așteptare în coadă pentru acesta să fie minim, pe baza unei strategii care vizează plasarea într-o coadă cu timp de așteptare minim sau într-o coadă care conține un număr minim de clienți. La fiecare iterație a programului, timpul de servire pentru clientul care se afla față de coadă este decrementat, iar la finalizarea timpului de servire acesta este eliminat din coadă. Prelucrarea clienților se realizează în mod paralel, întrucât pentru fiecare coadă este atribuit câte un fir de execuție.

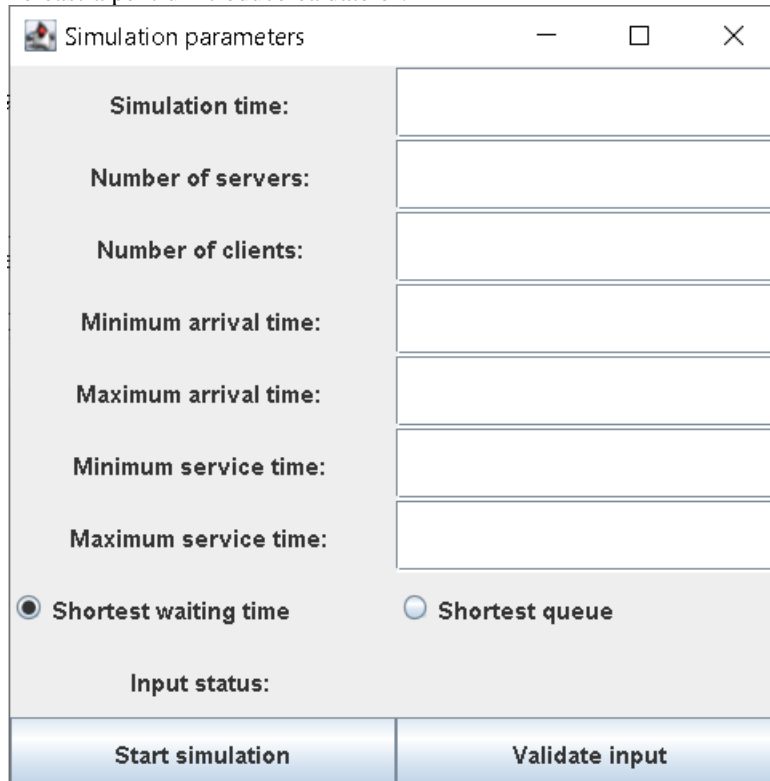
Erori de introducere a datelor:

1. Introducerea datelor de simulare în mod greșit, ce nu permit realizarea simulării, precum:
 - Introducerea de caractere altele decât cifre;
 - Introducerea de numere negative, care nu au sens în contextul simulării
 - Valorile maxime furnizate pentru timpii de sosire sau servire sunt mai mici decât valorile minime furnizate
 - Aceste erori sunt aduse la cunoștința utilizatorului în cadrul interfeței grafice
2. Neintroducerea parțială a unor date de simulare:
 - Necompletarea casetelor de text destinate introducerii datelor atrage furnizarea unui mesaj de eroare în cadrul interfeței grafice

Modelare:

În componenta aplicației se regăsesc 2 ferestre: o fereastră destinată introducerii parametrilor de simulare de către utilizator, precum și o fereastră al cărei scop este vizualizarea evoluției procesului de plasare și servire a clienților.

Fereastră pentru introducerea datelor:



Simulation parameters

Simulation time:

Number of servers:

Number of clients:

Minimum arrival time:

Maximum arrival time:

Minimum service time:

Maximum service time:

☒ Shortest waiting time ☐ Shortest queue

Input status:

Start simulation Validate input

Conține 7 casete pentru introducerea de text:

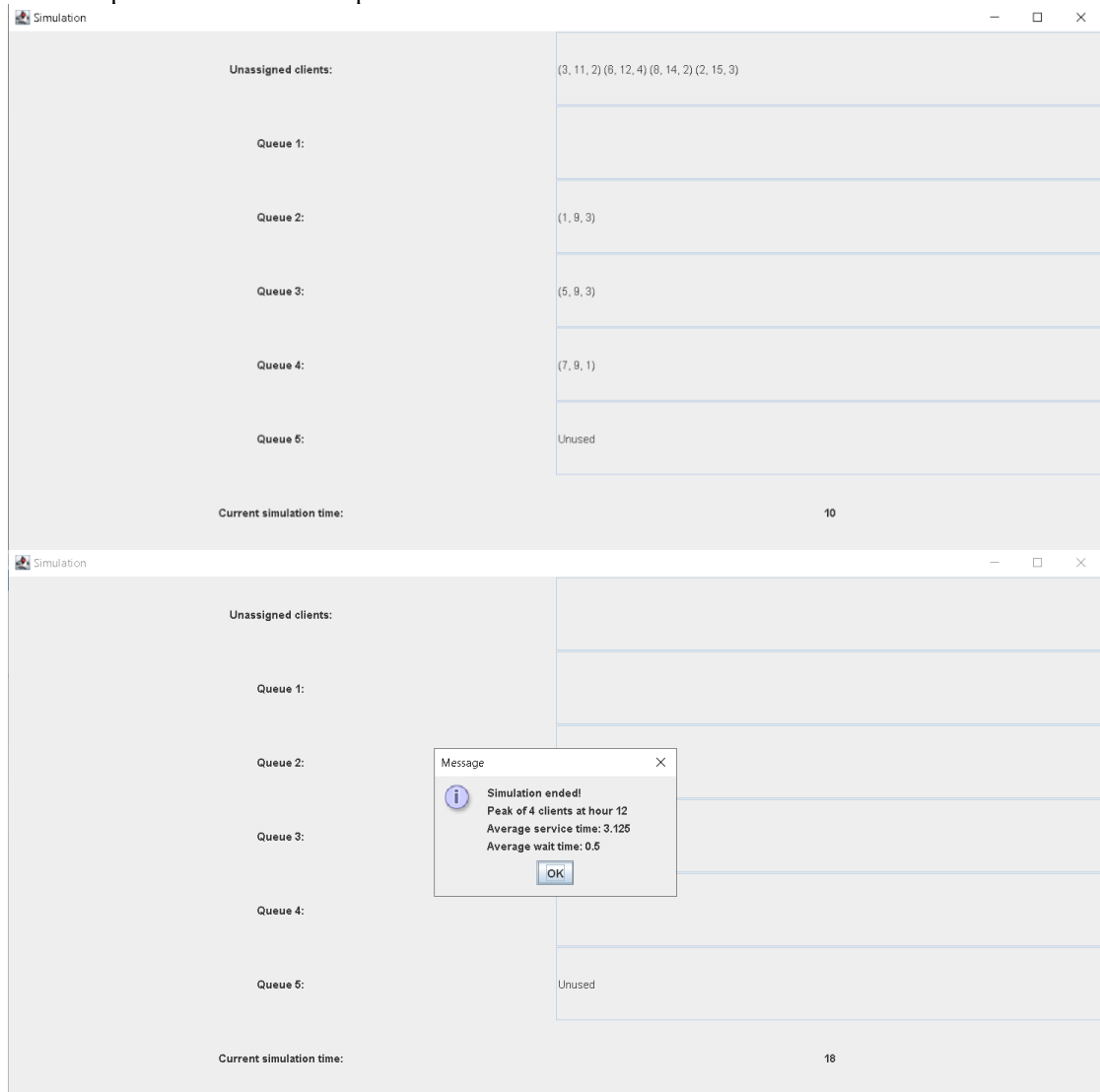
- O caseta pentru introducerea timpului de simulare
- O caseta pentru introducerea numărului de cozi
- O caseta pentru introducerea numărului de clienți
- O caseta pentru introducerea timpului minim de sosire
- O caseta pentru introducerea timpului maxim de sosire
- O caseta pentru introducerea timpului minim de servire
- O caseta pentru introducerea timpului maxim de servire

Conține 2 butoane, folosite pentru a selecta modul în care se realizează împartirea clienților în cozi:

- Shortest waiting time;
- Shortest queue.

Căte 2 butoane care sunt folosite pentru validarea datelor introduse și pentru a porni simularea: Validate input și Start simulation

Fereastra pentru simularea in timp real:



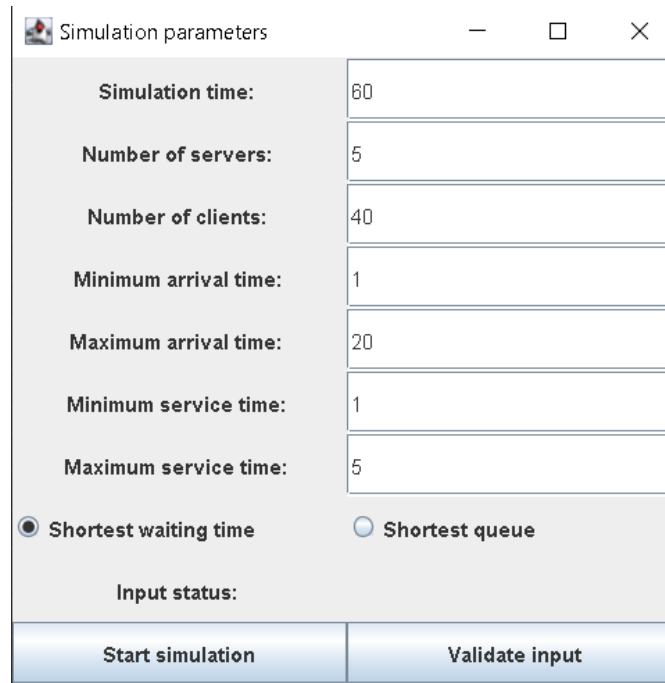
Este formata din 6 casete de text pentru afisarea continutului cozilor la un moment de timp. Cozile care nu sunt utilizate in simularea afiseaza in caseta de text „Unused” t:

- Queue 1 – arata continutul primei cozi la un moment de timp
- Queue 2 – arata continutul a cozii a 2-a la un moment de timp
- Queue 3 – arata continutul a cozii a 3-a la un moment de timp
- Queue 4 – arata continutul a cozii a 4-a la un moment de timp
- Queue 5 – arata continutul a cozii a 5-a la un moment de timp
- Unassigned clients – arata continutul cozii de asteptare la un moment de timp.
- Current simulation time – afiseaza timpul curent al simularii

La finalul simularii, apare un message box care informeaza utilizatorul cu privire la timpul mediu de asteptare pentru un client si timpul mediu de servire pentru un client, precum si timpul in care numarul de clienti din continutul cozilor a fost maxim.

Cazuri de utilizare:

Se introduc in fereastra de intrare parametrii pentru realizare simularii: 40 de clienti, 5 cozi, timp de simulare de 60, un timp minim de arrival de 1, un timp maxim de arrival de 20, un timp minim de service de 1 si un timp maxim de service de 5.



Simulation parameters	
Simulation time:	60
Number of servers:	5
Number of clients:	40
Minimum arrival time:	1
Maximum arrival time:	20
Minimum service time:	1
Maximum service time:	5
<input checked="" type="radio"/> Shortest waiting time <input type="radio"/> Shortest queue	
Input status:	
Start simulation	Validate input

Se selecteaza strategia „Shortest waiting time”.

Datele de intrare fiind corecte, se apasa pe Start simulation, iar apoi se incepe simularea.

The simulation window displays the following data:

Category	Value
Unassigned clients:	(10, 18, 4) (29, 18, 5) (13, 19, 5) (17, 19, 4) (32, 19, 2) (28, 20, 1)
Queue 1:	(2, 11, 2) (24, 14, 5)
Queue 2:	(5, 12, 2) (20, 18, 1) (31, 18, 2)
Queue 3:	(7, 12, 4) (15, 15, 4)
Queue 4:	(18, 11, 1) (14, 13, 4)
Queue 5:	(23, 12, 3) (4, 18, 5)
Current simulation time:	17

The second screenshot shows the simulation ending at 29 seconds. A message box displays the following statistics:

Simulation ended!
Peak of 13 clients at hour 12
Average service time: 3.325
Average wait time: 5.3

Simularea s-a finalizat la 29 de secunde, cu timp mediu de asteptare de 5.3 secunde, un timp mediu de servire de 3.325 secunde si un flux maxim de 13 clienti la secunda 12.

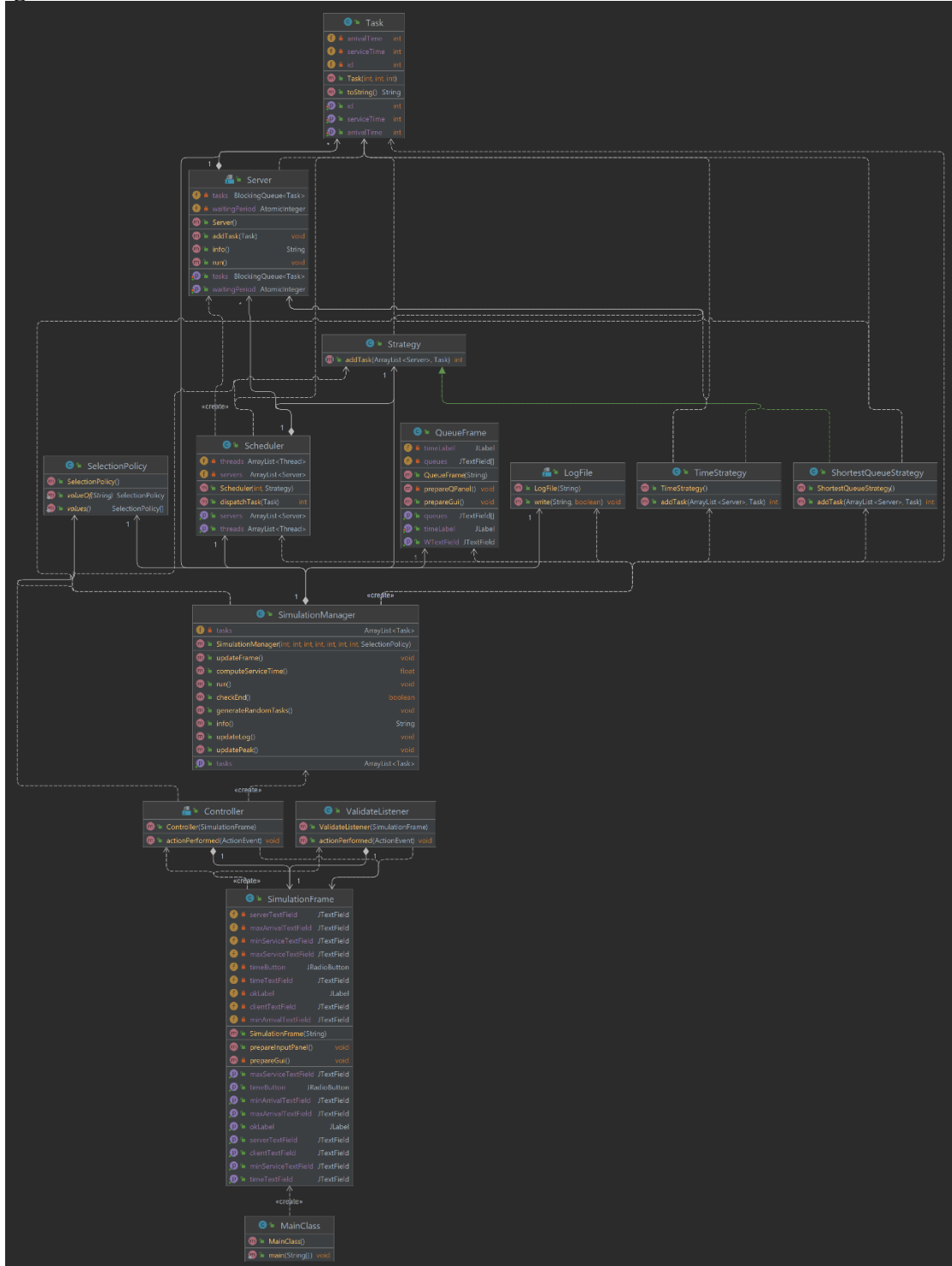
3. Proiectare

Pentru proiectarea aplicatiei s-a ales structurarea fisierelor sursa in mai multe pachete precum: Business, GUI si Model.

1. Pachetul **Model** cuprinde doua clase, **Server** si **Task**. Aceste clase reprezinta tipurile de date care vor fi prelucrate in cadrul simularii.
 - Clasa **Task** reprezinta clientul care va fi plasat in coada. Aceasta clasa contine:
 - Campuri private:
 - „**id**” – **int** ce reprezinta id-ul unic prin care se identifica fiecare client;
 - „**arrivalTime**” – **int** ce reprezinta timpul de sosire al clientului;
 - „**serviceTime**” – **int** ce reprezinta timpul de service al clientului.
 - Clasa **Server** reprezinta coada care are rolul de a asigura servirea clientilor. Clasa contine:
 - Campuri private:
 - „**waitingPeriod**” – **AtomicInteger** ce reprezinta timpul de asteptare actualizat la fiecare secunda din executia programului;
 - „**tasks**” – care reprezinta un **BlockingQueue** ce contine obiecte de tip **Task**.
 - Metodele, dintre care unele vor fii descrise ulterior:
 - **public String info();**
 - **public void addTask(Task newTask);**
 - **public synchronized void run();**
2. Pachetul **GUI** contine clasele responsabile pentru functionarea interfetei grafice a aplicatiei si este structurat in clase precum: **Controller**, **QueueFrame**, **SimulationFrame**, **ValidateListener**.
3. Pachetul **Business** contine clasele responsabile pentru prelucrarea obiectelor de tip **Server** si **Task**, acestea reprezentand intreaga logica a aplicatiei:
 - Enumeratia **SelectionPolicy**, utilizata pentru a selecta strategia de plasare a clientilor intr-o coada;
 - Clasa **Log**, utilizata pentru a scrie informatii referitoare la functionarea aplicatiei in fisierul text „queue_log.txt”;
 - Clasa **Scheduler**, care este un planificator al task-urilor, in cadrul caruia se determina cum anume vor fii pozitionati clientii in cozi la anumite momente de timp. Aceasta clasa porneste thread-urile aferente fiecarei cozi. Clasa este formata din:
 - Campuri private:
 - „**servers**” – **ArrayList** care contine referintele la cozile disponibile
 - „**threads**” – **ArrayList** care contine referinte la thread-urile corespunzatoare cozilor
 - „**maxNoServers**” – numarul de cozi
 - „**strategy**” – strategia de selectie a clientilor
 - Metodele, dintre care unele vor fii descrise ulterior:
 - **public Scheduler(int maxNoServers, Strategy strategy);**
 - **public void dispatchTask(Task task);**
 - Interfata **Strategy**, folosita pentru implementarea independenta a metodei:
 - **public void addTask(List<Server> servers, Task task);**
 - Clasa **ShortestQueueStrategy**, implementeaza **Strategy**, si functioneaza prin introducerea unui client in coada care contine cei mai putini clienti.
 - Clasa **TimeStrategy**, implementeaza **Strategy**, si functioneaza prin introducerea unui client in coada pentru care timpul de asteptare este minim.
 - Clasa **SimulationManager**, care are rolul de a centraliza celelalte clase responsabile pentru functionarea aplicatiei si de a controla modul in care se desfasoara simularea. Aceasta se ocupa de gestiunea ferestrelor interfetei grafice, retinerea parametrilor de simulare, generarea aleatoare a listei de clienti ce urmeaza sa fie prelucrati, actualizarea statisticilor precum timpul mediu de asteptare, servire si fluxul maxim de clienti, actualizarea GUI si scrierea informatiilor in fisierul de Log:
 - Campuri private:
 - „**scheduler**” – obiectul de tip **Scheduler** responsabil pentru plasarea clientilor in cozi;

- „**frame**” – fereastra de simulare responsabila pentru afisarea evolutiei cozilor;
- „**tasks**” – lista de clienti care este generata la inceputul simularii;
- „**selectionPolicy**” – politica de plasare a clientilor in coada;
- „**minArrivalTime**” – reprezinta timpul minim de arrival;
- „**maxArrivalTime**” – reprezinta timpul maxim de arrival;
- „**minServiceTime**” – reprezinta timpul minim de service;
- „**maxServiceTime**” – reprezinta timpul maxim de service;
- „**timeLimit**” – reprezinta durata maxima de timp alocata pentru simulare
- „**numberOfClients**” – numarul de clienti ce urmeaza sa fie serviti;
- „**numberOfServers**” – numarul de cozi in care vor fi plasati clientii;
- „**currentTime**” – timpul curent al simularii;
- „**peakClients**” – numarul maxim de clienti la un anumit moment de timp;
- „**peakTime**” – momentul de timp la care se inregistreaza numarul maxim de clienti;
- „**avgTime**” – folosit pentru calculul timpului mediu de asteptare pentru un client;
- „**log**” – instanta a clasei LogFile responsabila pentru scrierea in fisier a rezultatelor simularii;
- Metodele, dintre care unele vor fii descrise ulterior:
 - public **SimulationManager**(...);
 - public void **generateRandomTasks**();
 - public String **info**();
 - public void **updateLog**();
 - public void **updateFrame**();
 - public void **checkEnd**();
 - public void **updatePeak**();
 - public float **computeServiceTime**();
 - public synchronized void **run**()

Diagrama UML a claselor:



4. Implementare

Dintre metodele mai importante, amintim:

- În clasa **Server**:
 - Metoda **run()** este folosita pentru a gestiona thread-urile pornite din constructorului clasei Scheduler astfel: in fiecare secunda se decrementeaza timpul de servire al clientului aflat in fata cozii, iar daca perioada de servire se finalizeaza, atunci clientul se elimina din lista
- În clasa **Scheduler**:
 - Constructorul acestei clase initializeaza un numar de maxNoServers cozi, la care le ataseaza si porneste cate un thread pentru fiecare coada, pentru a asigura functionarea paralela cozilor. De asemenea, este responsabila pentru setarea strategiei de plasare a clientilor in cozi.
 - Metoda **dispatchTask** realizeaza apelul functiei specifice fiecarei dintre cele doua strategii pentru a plasa clientii.
- Clasa **ShortestQueueStrategy** implementeaza o singura metoda, **addTask**, care adauga un task in coada cu cei mai putini clienti.
- Clasa **TimeStrategy** implementeaza o singura metoda, **addTask**, care adauga un task in coada pentru care timpul de asteptare este minim..
- În clasa **SimulationManager**:
 - Metoda **generateRandomTasks** este utilizata pentru a genera in mod aleator, conform parametrilor de intrare o lista de clienti care urmeaza sa fie serviti.
 - Constructorul **SimulationManager** are rolul de a seta attributele clasei, de a pregati sistemul de scriere in fisierul log, apeleaza metoda generateRandomTasks pentru a crea lista de clienti, initializeaza elementele interfetei grafice si porneste firele de executie asociate fiecarei cozi.
 - Metoda **run** este folosita pentru a prelucra la un interval de o secunda, lista clientilor care inca nu au fost atribuiti niciunei cozi. De asemenea, metoda este responsabila pentru calculul datelor statistice, actualizarea interfetei grafice, precum si pentru scrierea in fisierul de log a datelor specifice simularii.

5. Rezultate

Pentru a testa aplicatia am folosit testele stabilite pentru acest assignment. Am realizat 3 teste, care se regasesc in folderul **appTesting**, test1.txt, test2.txt si test3.txt.

Primul test:

- Number of clients = 4;
- Number of queues = 2;
- Simulation interval = 60 s;
- [Minimum arrival time, Maximum arrival time] = [2, 30];
- [Minimum service time, Maximum service time] = [2, 4].
- Cu rezultatele:
 - Avg waiting time: 0.75 s
 - Avg service time: 3.25 s
 - Peak of 3 clients at hour 9.

Al doilea:

- Number of clients = 50;
- Number of queues = 5;
- Simulation interval = 60 s;
- [Minimum arrival time, Maximum arrival time] = [2, 40];
- [Minimum service time, Maximum service time] = [1, 7].
- Cu rezultatele:
 - Avg waiting time: 6.04 s
 - Avg service time: 3.72 s
 - Peak of 11 clients at hour 34

Al treilea test:

- Number of clients = 1000;
- Number of queues = 20;
- Simulation interval = 200 s;
- [Minimum arrival time, Maximum arrival time] = [10, 100];
- [Minimum service time, Maximum service time] = [3, 9].
- Cu rezultatele:
 - Avg waiting time: 104.62 s
 - Avg service time: 5.992 s
 - Peak of 684 clients at hour 100

6. Concluzii

In concluzie, consider ca aceasta tema m-a ajutat sa descopar conceptul de Thread-uri si sa imi consolidez cunostiintele de programare orientata pe obiecte in limbajul Java. De asemenea, consider important faptul suntem incurajati sa utilizam platforma Gitlab, intrucat reprezinta o modalitate foarte facila de a administra si actualiza codul sursa al unei aplicatii.

7. Bibliografie

1. https://dsrl.eu/courses/pt/materials/A1_Support_Presentation.pdf
2. <https://www.thoughtco.com/using-java-naming-conventions-2034199>
3. <http://www.javacodegeeks.com/2013/01/java-thread-pool-example-using-executors-and-threadpoolexecutor.html>
4. <http://docs.oracle.com/javase/tutorial/essential/concurrency/index.html>