**TEHNICI DE PROGRAMARE**

**ASSIGNMENT 2**

**QUEUES SIMULATOR**

**Chelemen Stefana-Bianca**

**GRUPA 30221**

**1.Obiectivul temei**

Aceasta tema are ca obiectiv implementarea unei simulari in timp real al unui sistem care lucreaza cu cozi pentru a determina si a minimiza timpul de asteptare al clientilor,,prin intermediul unui program Java. Datele de intrare pe care e bazata simularea, si anume numarul de clienti, numarul de cozi, intervalul de simulare, timpul minim/maxim de sosire, timpul minim/maxim de servire pot fi citite in doua moduri:

1.Dintr-un fisier text de intrare, caz in care rezultatele simularii se afiseaza intr-un fisier tezt de iesire.

sau

2.Din interfata grafica, introduse de catre utilizator, caz in care rezultatele simularii se afiseaza intr-o noua fereastra.

Ca obiective secundare putem mentiona :

* Folosirea multi-threadurilor ( un thread nou pentru fiecare coada )
* Simularea in timp real a timpului de sosire si de asteptare a clientilor
* Calculul timpului mediu de asteptare al clientilor
* Afisarea rezultatului simularii cozilor intr-un fisier
* Afisarea rezultatului simularii cozilor intr-o interfata grafica

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

***2.1.Analiza problemei***

Scopul problemei este acela de a implementa un program Java care realizeaza o simulare in timp real a unui sistem bazat pe cozi , utilizand paradigmele programarii orientate pe obiect .Problemele esentiale ale acestei aplicatii este de a gasi un mod eficient de reprezentare a obiectelor reale in clase, si afisarea cat mai simpla a rezultatelor simularii atat in fisier cat si in interfata grafica, pentru a putea fi inteleasa si de un utilizator neexperimentat. In implementarea acestei probleme ne vom confrunta cu problema concurentei (se vor rula mai multe parti din program in acelasi timp), deoarece dorim sa avem rezultate cat mai bune ale simularii fara a creste numarul de cozi, lucru care ar creste costurile.

***2.2.Modelare***

In cee ace priveste modelarea datelor, prima problema cu care ne care ne confruntam consta in reprezentarea datelor din viata reala (ex client, coada) . Pentru aceasta , vom implementa o clasa numita “ Client”, care va inmagazina informatiile despre un singur client necesare pentru simulare. O a doua clasa implementata este clasa “ Queue ” , care are printer altele, contine si o lista de clienti . Aici se reprezinta prin metode , sosirea clientului la o coada, ,asteptarea si eliminarea clientilor din cozi dupa ce acesta a fost servit . Vom folosi un thread per coada (multithreading) , pentru a rezolva problema concurentei.

Pentru a gestiona cozile de asteptare ,am implementat clasele “ MyQueueSimulation ” si “MyQueueSimulationFile” ( care sunt si acestea thread-uri ) . Clasa “MyQueueSimulation “ foloseste instante ale clasei “ DataIn “ ,unde sunt prelucrate date extrase din fisier/interfata grafica si introduse in program informatiile necesare implementarii simularii. Clasele “MyQueueSimulation” si “MyQueueSimulationFile” fac in esenta acelasi lucru, singura diferenta dintre acestea ar fi ca in clasa “MyQueueSimulationFile” vom avea o metoda run care porneste simularea si scrie rezultatele simularii intr-un fisier text de iesire, iar metoda run() din clasa “MyQueueSimulation” va afisa rezultatele simularii intr-o interfata grafica.

***2.3.Scenariul***

In cazul in care programul ruleaza fara erori , utilizatorul va vedea , intr – un fisier/in interfata grafica modul in care intra clientii in cozi , timpul petrecut in coada ,dar si clientii care asteapta . Iar , la finalulsimularii , in fisier/interfata grafica va aparea timpul mediu de asteptare al clientilor .

***2.4.Cazuri de utilizare***

In fereastra de simulare sau la deschiderea fisierului unde sunt sunt trecute rezultatele simularii aplicatiei , va aparea , la fiecare moment de timp ( de la 0 pana la timpul maxim de simulare , introdus in fisierul de intrare/in interfata grafica ) lista clientilor , al caror timp de sosire nu corespunde cu timpul curent , iar ,apoi , apar toate cozile , urmate de clientii care au intrat in coada . In cazul in care o coada este goala , langa aceasta va aparea mesajul “closed” . De asemena, la terminarea simularii, se va afisa in fisierul de iesire sau in interfata grafica timpul mediu de asteptare.

**3.Proiectare**

Respectand paradigmele programarii orientate pe obiect , am impartit programul in mai multe pachete, fiecare continand mai multe clase. Voi prezenta mai jos structura fiecarui pachet:

1.Pachetul “datamodels”

-contine clasele: Client, Queue, DataIn, SortingByArrival,SortingByWainting,SortingById

-acest pachet a fost gandit sa contina toate clasele a caror instante se vor folosi in simularea propriu-zisa

2.Pachetul “simulation”

-contine clasele:MyQueueSimulation si MyQueueSimulationFile

-acest pachet a fost gandit sa contina clasele cu simuarile-propriu zise

3.Pachetul “gui”

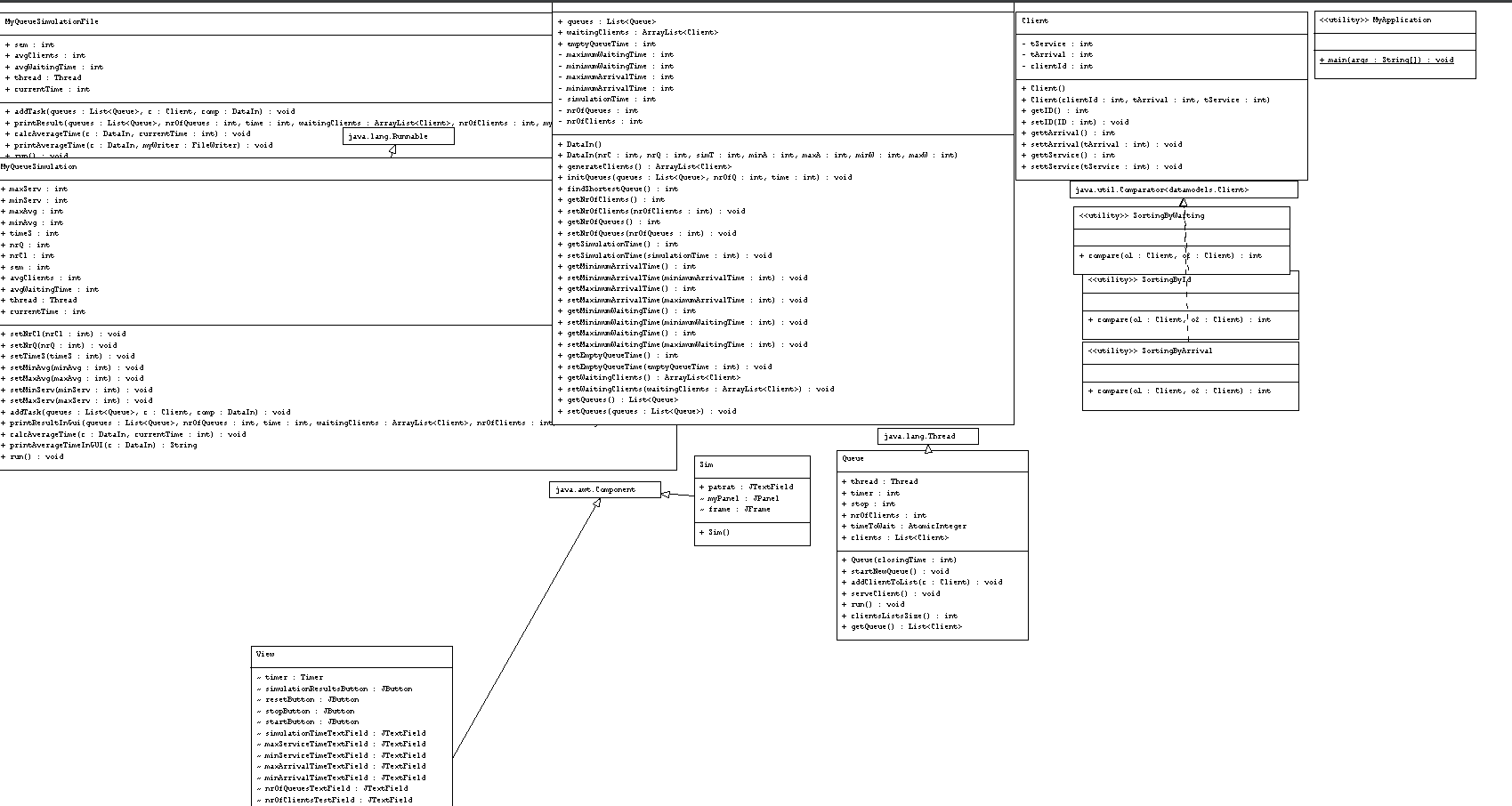
-contine clasele :View si Sim

-acest pachet contine doar clasele folosite pentru implemnetarea interfetei grafice: View are un constructor care creeaza fereastra principal “QueueSimulator” in care utilizatorul poate introduce datele de intrare; aceasta clasa mai are si un timer si un buton “START” care incepe simularea; constructorul clasei “Sim” creeaza o fereastra care afiseaza rezultatele simularii in timp real.

Programul contine si o clasa MyApplication care are o metoda main din care pornim aplicatia.

Pentru a da o reprezentare visual si mai clara a programului, voi adauga diagrama UML a aplicatiei:

… … … … … … … … … … … … … … … … … … … … … … … … … … … … … … … … … … … … … … …… … … … … … …



… … … … … … … … … … … … … … … … … … … … … … … … … … … … … … … … … … … … … … …… … … … … … …

Fig 1 – Diagrama UML a claselor

**4.Implementare**

1. Clasa “Client”

Aceasta clasa are ca atribute toate datele necesare pentru identificarea unui client in coada:

1. clientId: reprezinta ID-ul clientului
2. tArrival: timpul la care clientul a ajuns la servire
3. tService: durata in care a fost clientul servit

Ca metode, aceasta clasa are numai getteri si setter pentru a putea avea acces la atributele clasei, care sunt private:

1. getId() si setId()-returneaza/seteaza Id-ul clientului
2. gettArrival() si settArrival()-returneaza/seteaza timpul de sosire al clientului
3. gettService() si settService()-returneaza/seteaza timpul de servire al clientului

2.Clasa “Queue”

Aceasta clasa implementeaza interfata “Runnable”, deoarece la creearea unei noi cozi, se creeaza un thread nou.

Atributele clasei “Queue”:

1. thread-reprezinta un Thread
2. timer- tine evidenta timpului in coada
3. stop- timpul fata de care ne raportam cand parcurgem coada
4. nrOfClients-numarul de client
5. timeToWait-timpul de asteptare;am utilizat tipul AtomicInteger deoarece lucram cu Threaduri
6. clients-este o lista sincrona de client; am ales acest tip de lista pentru a putea lucra cu threaduri

Metodele clasei sunt:

1. startNewQueue- metoda care incepe o noua coada si porneste threadul acesteia
2. addClientToList- metoda care adauga un client in lista de client
3. removeClientFromList()-metoda care scoate un client din lista de client
4. clientsListsSize()-metoda care returneaza dimensiunea listei de client
5. getQueue()-returneaza coada de client
6. run()-porneste Thread-ul unei cozi

3.Clasa DataIn

Atributele clasei sunt:

1. nrOfClients –numarul de client
2. nrOfQueues-numarul de cozi
3. simulationTime-intervalul de simulare
4. minimumArrivalTime/maximumArrivalTime-timpul minim/maxim de sosire la coada
5. minimumServicecTime/maximumServiceTime-timpul minim/maxim de servire
6. waitingClients-lista de client
7. queues-lista de cozi

Metodele clasei sunt:

1. metoda generateClients()-genereaza random client
2. metoda initQueues()-adauga o coada in lista de cozi si care porneste thread-ul acelei cozi
3. metoda findShortestQueue()-metoda care cauta in lista de cozi coada cea mai scurta

Pe langa aceste metode avem getteri si setter pentru atributele private ale clasei.

De asemenea, avem 2 constructori pentru citirea datelor din fisier sau din interfata grafica.

4.Clasa SortingByArrival

Aceatsta clasa implementeaza interfata Comparator si am folosit-o pentru a suprascrie metoda “compare” astfel incat sa compare 2 obiecte de tip Client in functie de atributul tArrival(timpul de sosire). Aceasta clasa o instantiem atunci cand apelam metoda “sort” din clasa Collections, pentru a sorta clientii in functie de timpul de sosire in coada.

5.Clasa SortingByWaiting

Aceatsta clasa implementeaza interfata Comparator si am folosit-o pentru a suprascrie metoda “compare” astfel incat sa compare 2 obiecte de tip Client in functie de atributul tService (timpul de servire). Aceasta clasa o instantiem atunci cand apelam metoda “sort” din clasa Collections, pentru a sorta clientii in functie de timpul de servire.

6.Clasa “MyQueueSimulation”

In cadrul acestei clase se face simularea propriu-zisa. Simularea se face cu datele de intrare primite din interfata grafica.

Atributele clasei sunt:

1. Thread-thread
2. avgWaitingTime- pentru a calcula timpul mediu de asteptare al clientilor
3. nrCl;
4. nrQ;

atribute pe care le folosesc la siimularea in interfata grafica

1. timeS;
2. minAvg;
3. maxAvg;
4. minServ;
5. maxServ;

Metodele clasei sunt:

1. Metoda addTask()-metoda care cauta coada minima si adauga clientul in ea
2. Metoda printResultInGui()-metoda care scrie rezultatele in interfata grafica
3. Metoda calcAverageTime()-calculeaza timpul mediu de asteptare
4. Metoda printAverageTimeInGUI()-scrie in GUI timpul mediu de asteptare
5. Metoda run()

7.Clasa “MyQueueSimulationFile”

In cadrul acestei clase se face simularea propriu-zisa. Simularea se face cu datele de intrare primite dintr-un fisier text si scrie rezultatele simularii tot intr-un fisier text.

Atributele clasei sunt:

1. public int currentTime ;
2. public Thread thread;
3. public int avgWaitingTime;
4. public int avgClients;
5. public int sem;

Metodele clasei sunt:

1. Metoda addTask()-metoda care cauta coada minima si adauga clientul in ea
2. Metoda printResultInFile()-metoda care scrie rezultatele in fisier
3. Metoda calcAverageTime()-calculeaza timpul mediu de asteptare
4. Metoda printAverageTimeInFile ()-scrie in fisierul out timpul mediu de asteptare
5. Metoda run()

8.Clasa View

In aceasta clasa implementam interfata grafica propriu-zisa. Aceasta are un constructor care, la instantierea clasei creeaza o fereastra in care utilizatorul poate introduce datele de intrare pentru a testa simularea.

Clasa “View” are compnentele urmatoare:

1. Frame-fereastra interfetei grafice
2. Panel-panoul pe care sunt asezate componentele
3. TimerPanel-panoul pe care este asezat timerul
4. Timer-un timer
5. StartButton-un buton de start pentru a incepe simularea

Pe langa acestea avem label-uri si textField-uri pentru datele de intrare.

Metodele clasei “View”:

1. Start()-metoda care porneste timerul la apasarea butonului “START”
2. Stop()-metoda care opreste timerul
3. Restart()-metoda care reseteaza timerul

Pe langa aceste metode, am creat niste metode care iau datele de intrare introduse de catre utilizator sub forma de String si le transforma in numere intregi:

public int getNrOfClients()

public int getNrOfQueues()

public int getMinArrivalTime()

public int getMaxArrivalTime()

public int getMinServiceTime()

public int getMaxServiceTime()

public int getSimulationTime()

9.Clasa Sim

Aceasta clasa este istantiata atunci cand, dupa introducerea datelor de intrare, utilizatorul apasa

Butonul “START”. La istantierea acestei clase, va aparea o fereastra simpla in care vom vedea rezultatele simularii in timp real. In sectiunea “5.Rezultate” voi introduce niste imagini pentru a avea o idee cat mai clara cu privire la functionarea aplicatiei.

Atributele clasei “ Sim ” sunt:

1. JFrame frame; -fereastra propriu-zisa
2. JPanel myPanel;-panoul pe care vom aseza un textField cu rezultatele simularii
3. JTextField patrat;-o caseta de text in care scriem rezultatele simularii

10.Clasa MyApplication

Aceasta clasa are doar o metodamain in care se porneste in executie aplicatia .

**5.Rezultate**

Rezultatele simularilor le putem observa in fisierele de test de iesire:

Pentru fisierul urmator “ test1In.txt ”:

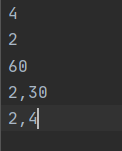
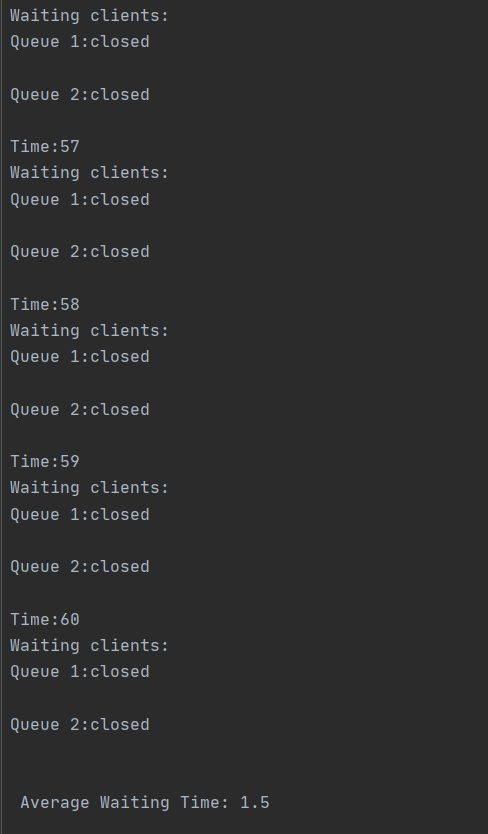


Fig 2 - Exemplu de fisier de intrare .

Avem rezulatele simularii iin fisierul “ test1Out.txt ” :



Am ales sa pun capturi numai cu prima si ultima parte din fiser deoaece timpul de simulare este destul de mare (60) si sunt multe rezulate

Fig 3 - Exemplu fisier de iesire .

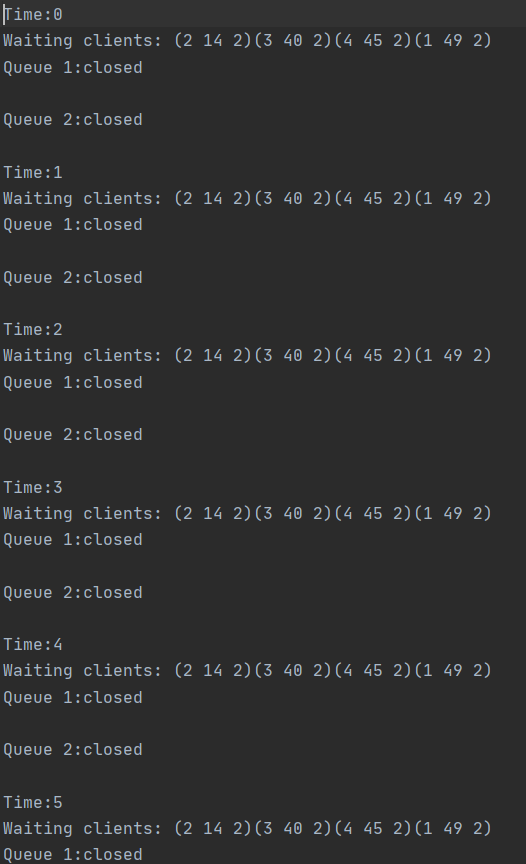


Fig 4 - Exemplu fisier de iesire .

In acelasi fel, se fac simularile pentru fisierle “test2In.txt” si “test3In.txt” si se scriu rezultatele in fisierle

“test2Out.txt” si “test3Out.txt” .

In ceea ce prieste interfata grafica rezultatele sunt scrise in felul urmator :

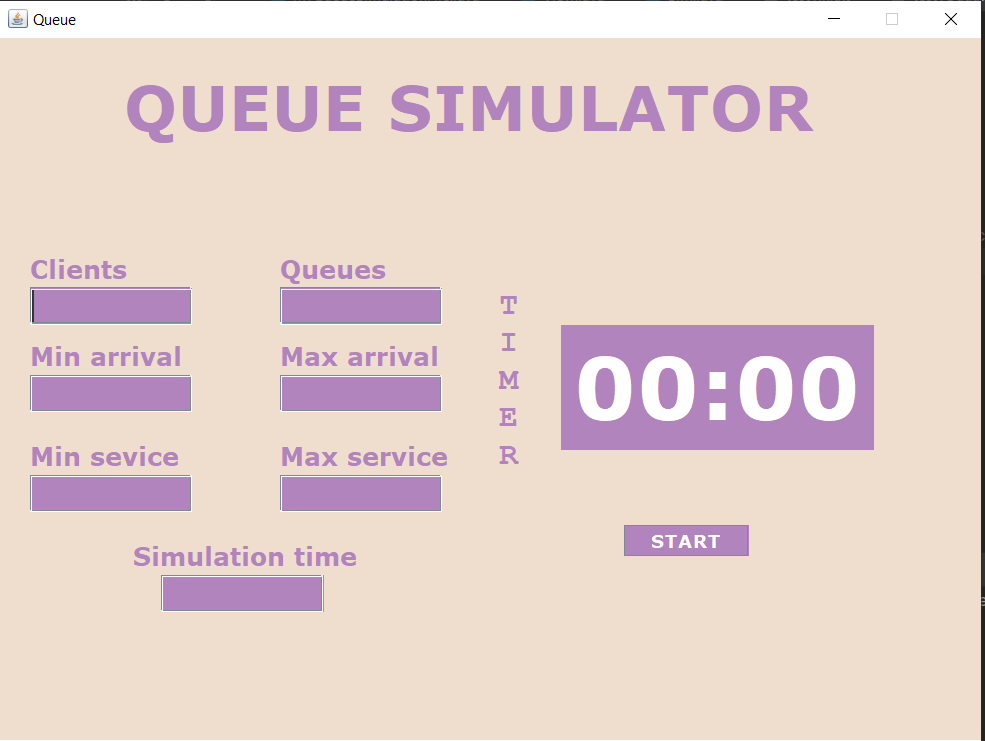


Fig 5 – Pagina de pornire vazuta de catre utilizator la rularea aplicatiei .

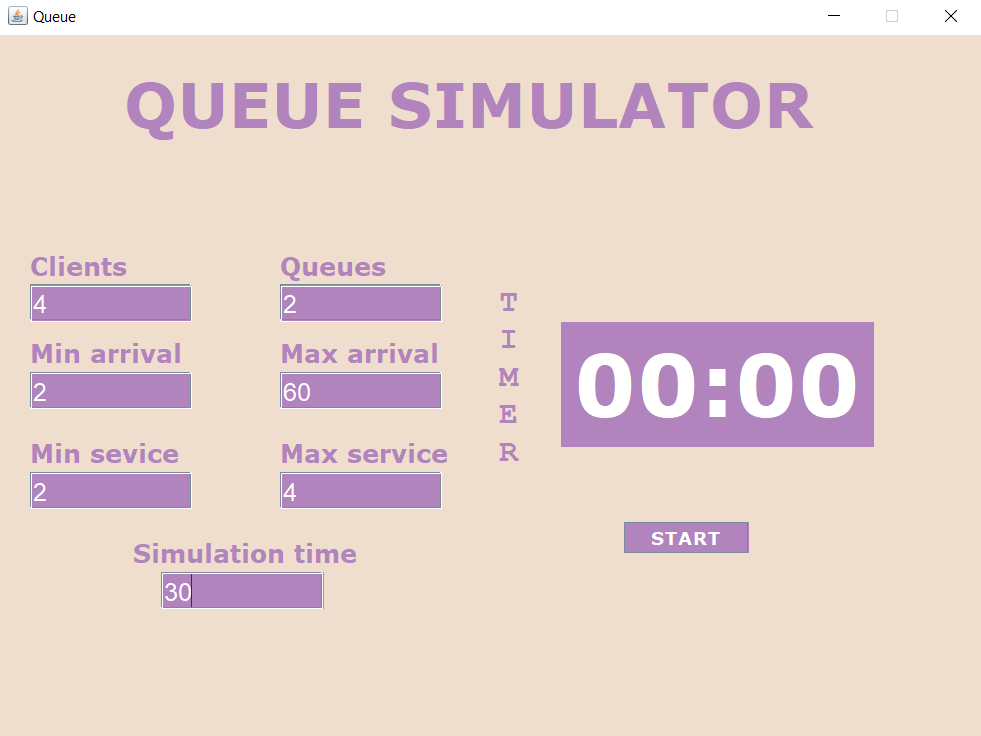


Fig 6 – Pagina de pornire dupa introducerea unor date de intare .

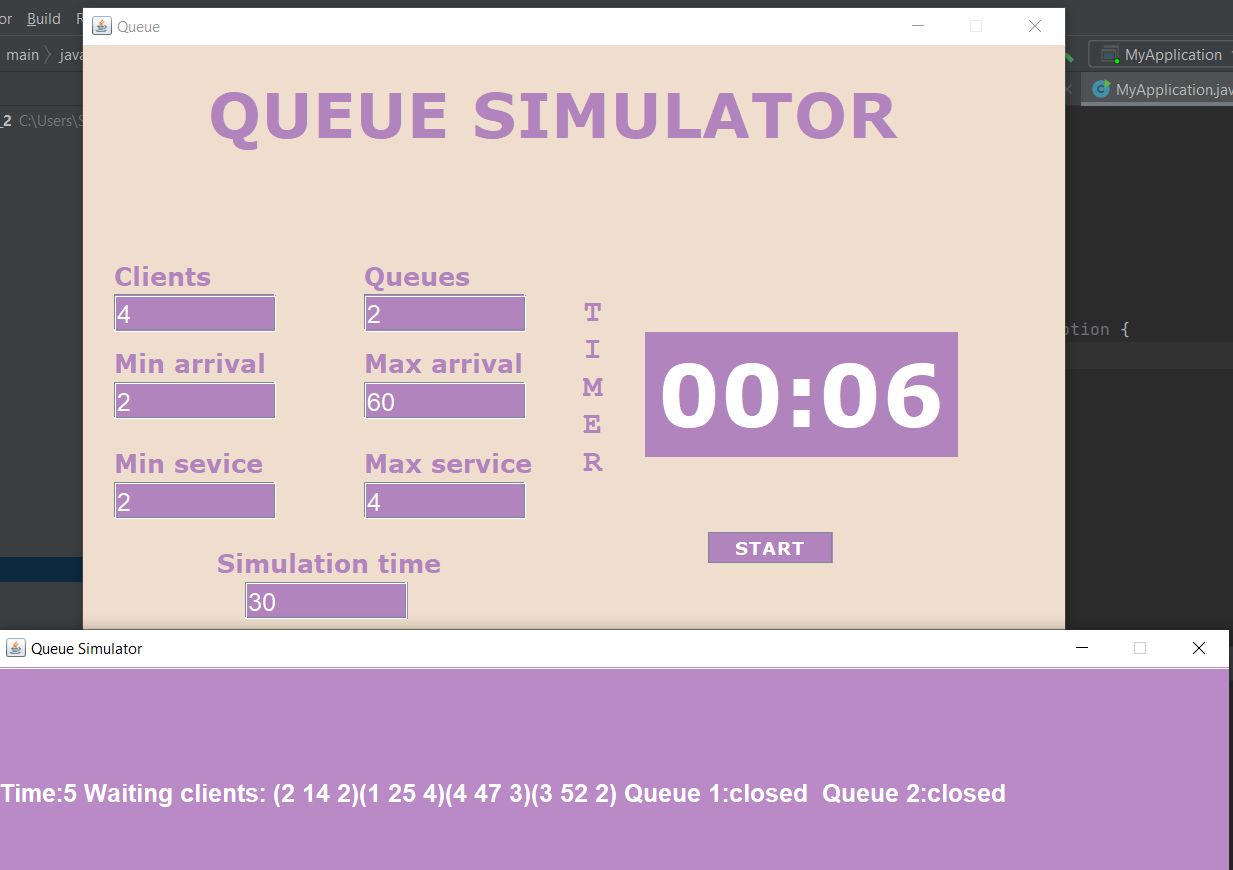


Fig 6 – Pagina de pornire si pagina de rezultate ale simularii, dupa apasarea butonului “START” .

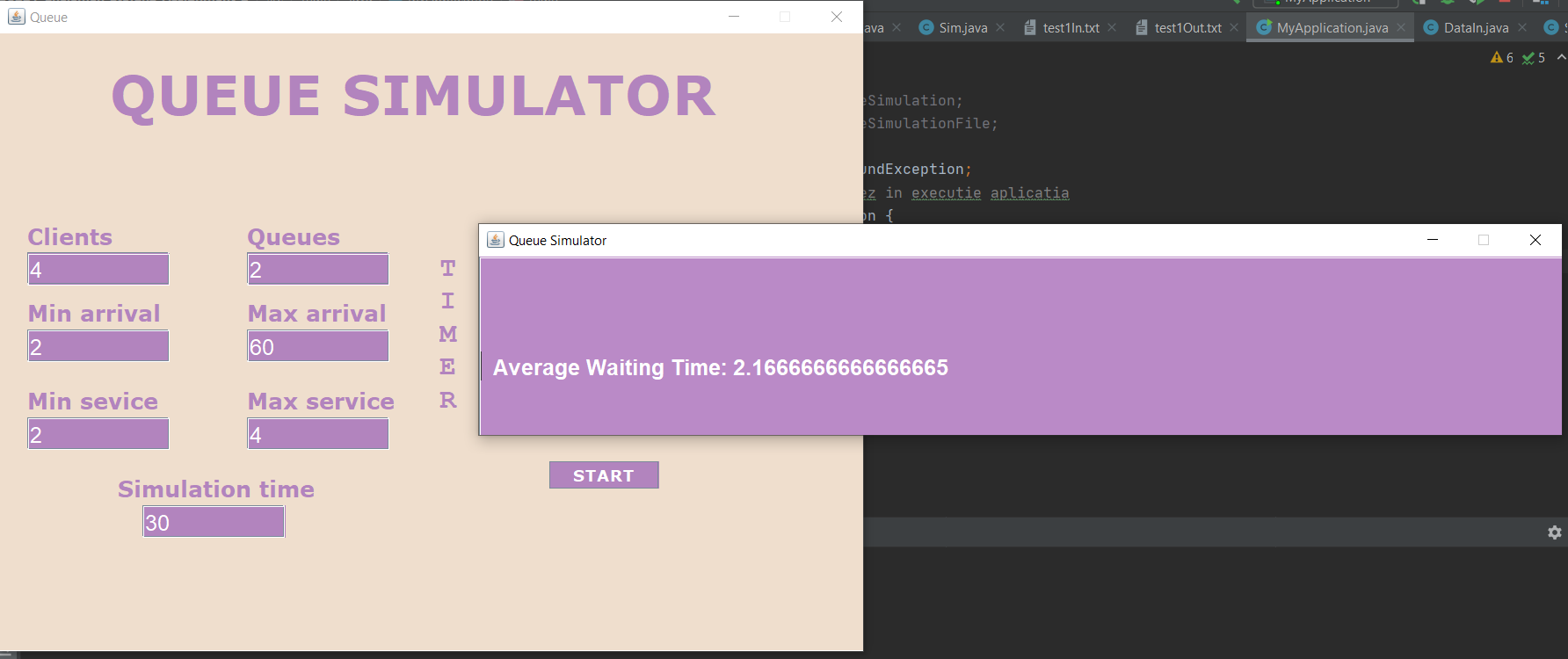


Fig 7 - Pagina de pornire si pagina de rezultate ale simularii, la sfarsitul simularii .

**6.Concluzii**

In concluzie, in opinia mea aceasta tema a fost utila pentru reimprospatarea conostiintelor dobandite pana acum in ceea ce pirveste limbajul de programare Java, paradigmele programarii orientate pe obiect a interfetelor grafice, dar mai ales a Thread-urilor si a conceptului de “Multithreading” . Proiectul ar putea fi imbunatatit daca am putea gasi o soluctie pentru o sincronizare mai buna a cozilor .

**7.Bibliografie**

1 . https://docs.oracle.com/javase/tutorial/essential/concurrency/index.html

2 . https://docs.oracle.com/javase/8/docs/api/java/util/Timer.html

3. http://www.tutorialspoint.com/java/util/timer\_schedule\_period.htm

4. http://www.javacodegeeks.com/2013/01/java-thread-pool-example-using-executors-and-threadpoolexecutor.html