**Documentatie Simulator**

**Cozi**

Pop Mihali Marco Silviu

1. Obiectivul temei

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

3. Proiectare

4. Implementare

5. Rezultate

6. Concluzii

7. Bibliografie

**1. Obiectivul temei**

Obiectivul principal al temei este de a crea o aplicatie java care preia dintr-o interfata grafica mai multi parametri de simulare, si cu valoarea lor se va simula mai multe cozi ce proceseaza clienti generat aleator pe thread-uri diferite. Simularea va fi afisata in timp real in GUI, impreuna cu timpul mediu asteptat, timpul mediu de procesare si ora de varf.

Obiective secundare:

* Crearea interfetei grafice (cap 4 )
* Preluarea inputului de la utilizator(cap 4)
* Procesarea pe thread-uri diferite (cap 4)
* Vizualizarea in timp real (cap 4)

Crearea interfetei grafice:

Crearea unei interfete grafice ce va lasa utilizatorul ce va lasa utilizator sa introduca parametrii ce controleaza simularea.

Preluarea inputului de la utilizator:

Textul introdus este preluat din campurile de tip text si este stocat.

Procesarea pe thread-uri diferite

Pentru fiecare coada simulata se asociaza un thread diferit care sa efectueze operatiile necesare.

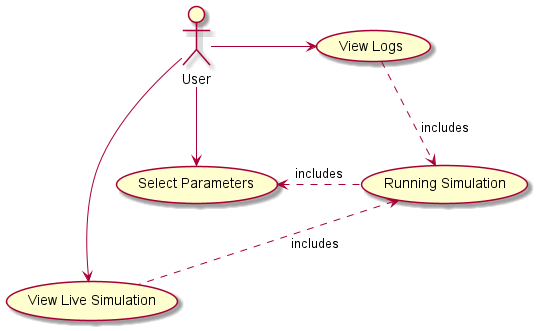
Vizualizarea in timp real

In interfata grafica se afiseaza icon-uri pentru pentru case si pentru clienti si este simulat numarul de clienti la fiecare casa. Update-urile se vor face o data la fiecare secunda.

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

Cerinte functionale:

* Posibilitatea de a alege parametrii simularii: timeLimit, maxProcessingTime, minProcessingTime, maxArrivalTime, minArrivalTime, numberOfServers, numberOfClients si maxNbOfClientsPerQueue in interfata grafica
* Posibilitatea de a porni simularea dupa algerea parametrilor
* Simularea propriu-zisa unde fiecare coada este simulata de un thread diferit
* Calcularea timpului mediu asteptat, mediu de procesare si a orei de varf dinamic pentru a fi afisat.
* Afisarea simularii folosind un alt frame al interfetei grafice



Use-case diagram

Scenariul principal:

* Utilizatorul introduce parametrii de simulare in interfata
* Simularea isi incepe functionarea folosind parametrii introdusi
* In timpul simularii se creare un fisier log, unde este scris pentru fiecare moment al simularii situatia codizilor si a clientilor care nu au ajuns inca in cozi
* In timpul simularii este afisata o reprezentare grafica a situatiei cozilor plus timpii medii si ora de varf
* Simularea se incheie

**3. Proiectare**

Structura, organizarea in pachete: policy, simulation, utils, view

* policy : contine clase si interfete ce ajuta la alegerea metodei de decizie a clientilor care sunt la capatal listei de asteptare si vor sa se puna la o coada
* simulation: contine clasele care se ocupa de simulare propriu-zisa, impartirea clientilor la coada respectiva la momentul potrivit
* utils : mai multe enumeratii si interfete ce ofera valori necesare pentru usurarea scrierii codului, respectiv care calculeaza si scriu in fisier
* view: clasele in care sunt declarate atribute din Java Swing, adica toate clasele ce au elemente ce vor face parte din interfata grafica si care contribuie la afisarea simularii in timp real

Structuri de date folosite:

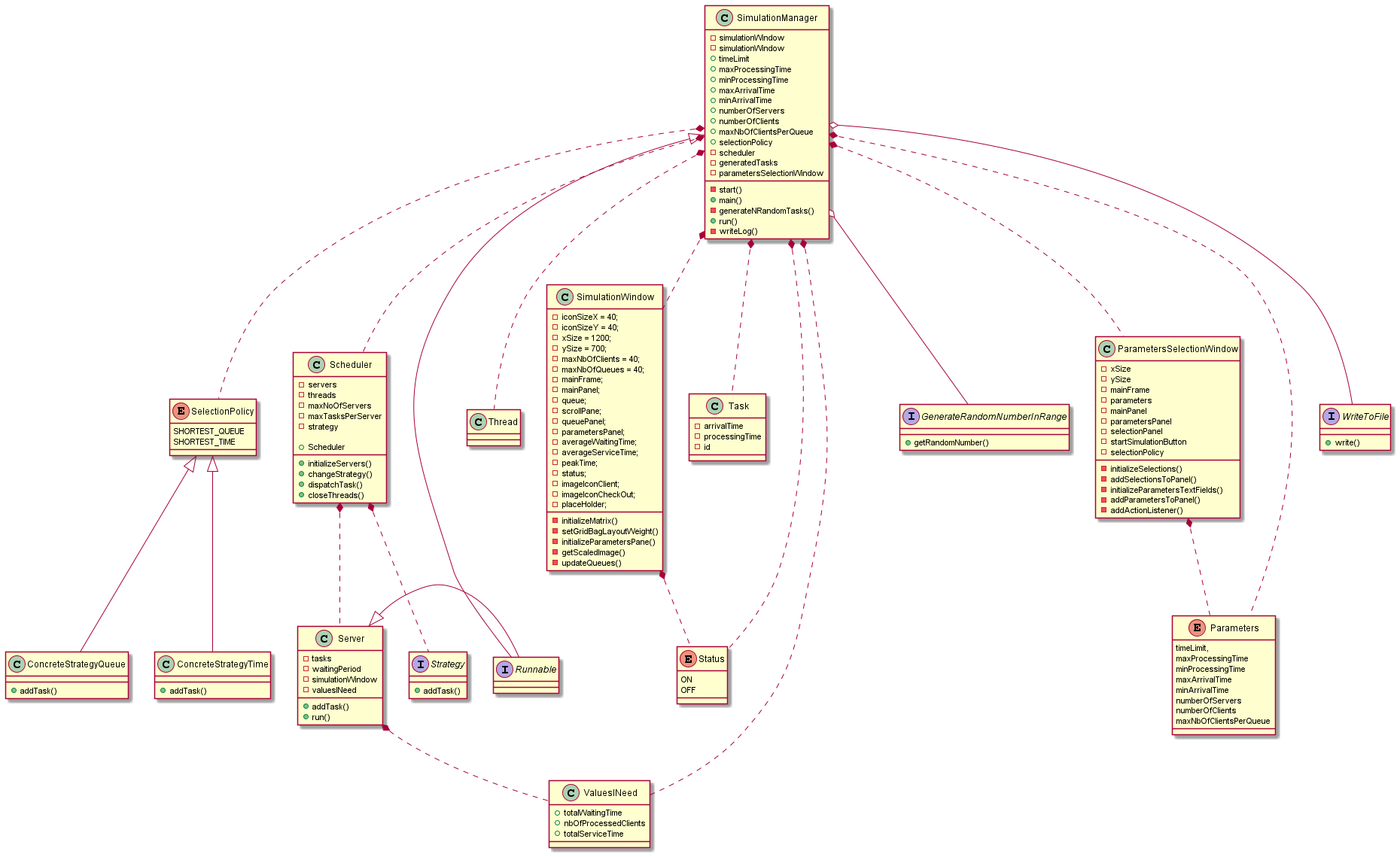
Pentru retinerea unei cozi am folosit BlockingQueue. Aceasta suporta operatii care asteapta ca coada sa fie nenula cand recupereaza un element si care asteapta ca in coada sa fie loc inainte de a adauga un element.

Acest BlockingQueue este folosit impreuna cu alte atribute pentru a crea clasa Server careia ii este atribuit un thread pe care poate rula.

Am folosit o structura Task care retine timpul la care un client poate incepe sa stea la coada, timpul necesat pentru ca acesta sa fie procesat, respectil id-ul unic al acestuia.

Algoritmi folositi:

In view pentru a afisa grafic starea curenta a cozilor am creat o matrice de JComponents care sa tina minta toate label-urile (fiecare client/checkout are un label cu un icon atasat), matricea are ca linii nr maxim de cozi si ca coloane nr de clienti. In metoda care updateaza coada grafica toate aceste label-uri sunt facute invizibile, iar dupa se verifica starea serverelor si se fac vizibile primele label-uri de pe fiecare linie astfel incat nr de label-uri vizibile pe fiecare coada corespunde cu numarul de clienti din fiecare coada.



UML Class Diagram

**4. Implementare**

Pachetul policy:

* Strategy = este o interfata care defineste strategia aleasa de clienti pentru a alege coada la care se vor pune
  + addTask() = metoda pentru a adauga un task la unul dintre server
* StrategyPolicy = enumeratie ce are valorile SHORTEST\_QUEUE si SHORTEST\_TIME, valori ce reprezinta modul de de alegere al metodei de asezare la coada
* ConcreteStrategyQueue = implementeaza strategy si reprezenta solutia in care un client se aseaza la coada cu numarul cel mai mic clienti
  + addTask() = metoda care intereaza prin servere si il alege pe cel cu numar minim de clienti
* ConcreteStrategyTime = implemeanta strategy si reprezenta solutia in care un client se aseaza la coada unde timpul pe care ar trebui sa-l astepte este minim, timpul pe care este asteptat este calculat ca suma timpilor de procesare al tuturor clientilor ce asteapta plus timpul ramas de procesare al clientului de la casa

Pachetul simulation:

* Scheduler = se ocupa de simularea propriu-zisa, alegerea clientilor ce ajung la cozi, pozitionarea lor in functie de strategie etc.
  + initializeServer() = se initializeaza servers si threads, lista de serevere, respectiv lista de thread-uri asociate, dupa care sunt initializare elementele lor, iar fiecare thread in parte este pornit
  + changeStrategy() = in functie de policy-ul primit se alege strategia noua
  + dispatchTask() = adauga un task folosind strategia aleasa
  + getServers() = getter pentru lista de servere
* Server = structura de date ce retine o coada, date despre ea si care este data unui thread
  + addTask() = adauga un task la coada si creste perioada de asteptare pentru coada curenta
  + getWaitingPeriod() = getter pentru waitingPeriod
  + getNbOfWaitingTasks() = returneaza dimensiunea BlockingQueue-ului tasks
  + getTasks() = getter pentru BlockingQueue tasks
  + getValuesINeeed() = retuneaza atributul valuesIneed
  + run() = metoda chemata de threadul pe care serverul curent ruleaza, atat timp cat thread-ul nu este intrerupt se verifica varful cozii, iar daca acesta nu este null task-ul curent este procesat si atribului ValueINeed este updatat, se creste timpul de procesare cu durata de procesare a clientului curent, se creste numarul de clienti procesati cu 1, iar timpul de asteptare se creste cu nr de clienti inafara de cel procesat
* SimulationManager = centralizeaza toate clasele ce au rol in simulare
  + start() = metoda ce creeaza fereastra de simulare si care porneste thread-ul principal
  + main() = se creeaza un obiect de tipul SimulationManager si se instantiaza thread-ul principal
  + generateNRandomTasks() = se creeaza numberOfClienti thread-uri cu valori arrivalTime si processingTime in intervalele specifice citite ca parametrii
  + run() = metoda rulata de thread-ul principal unde are loc simularea, se creeaza un while in care avem un timp curent ce creste pana la limita de timp, se verifica daca este vreun task ce are arrivalTime egal cu currentTime, daca sunt egale task-ul este dat mai departe procesarii
  + writeLog() = scrie un log pentru un timp specificat, metoda este chemata la fiecare incremantare a variabila currentTime
* Task = clasa ce retine un client, arrivalTime, processingTime si id
  + setters and getters + toString method
* ValuesINeed = stuctura de date ce retine date necesare pentru calcularea pe fiecare server a timpilor medii

Pachetul utils:

* GenerateRandomNumberInRange = genereaza un numar aleator in intervalul specificat
  + gerRandomNumber() = pentru 2 valori returneaza un numar aleator in interval
* Parameters = enumeratie ce retine parametrii de intrare specificati de utilizaot
  + timeLimit, maxProcessingTime, minProcessingTime, maxArrivalTime, minArrivalTime, numberOfServers, numberOfClients, maxNbOfClientsPerQueue
* Status = enumratie ce retine starea pornita sau oprita (termianta) a simularii
  + ON, OFF
* WriteToFile = interfata ce scrie in fisierul log.txt
  + write() = se creeaza un nou file numit "log.txt" in cazul in care nu exista unul deja, dupa se creeaza un FileWriter cu append si se scrie string-ul primit ca input

Pachetul view:

* ParametersSelectionWindow = clasa pentru fereastra din interfata grafica in care utilizatorul alege parametrii de simulare
  + Avem 8 campuri unde se poate introduce text, fiecare cu un border cu numele specific.
  + Avem un dropdown meniu pentru a alege strategia clientilor pentru gasirea cozii unde vor merge
  + Avem un buton "Start" care porneste simularea si deschide o alta fereastra unde se afla interfata grafica in timp real a simularii
* SimulationWindow = clasa pentru fereastra unde se va afisa simularea in timp real
  + initializeMatrix() = initializeaza o matrice de tip JComponent cu label-uri care au in interior fie o imagine cu o casa sau cu un client, este nevoie de aceasta matrice deoacare componentele grafice exista mereu ( in memorie exista toate cozile cu nr maxim de clienti ), componentele sunt facute vizibile sau invizibile
  + setGridConstraints() = seteaza restrictiile elementelor din panoul principal
  + initializeParametsPane() = initializeaza label-urile folosite pentru afisarea timpiilor medii si a orei de varf
  + getScaledImage() = redimeansioneaza o imagine primita dupa latimea si inaltimea date ca parametrii
  + updateQueues() = primeste ca parametru o lista de servere, face ca toate componentele sa fie invizibile, dupa care pentru fiecare coada, in functie de numarul de clienti din ea, faci vizibile un numar egal de componente pe linia respectiva

Interfata grafica:

Fereastra **Select Parameters** este prima fereastra care apare la deschiderea aplicatiei. Aceasta are 3 parti principale:

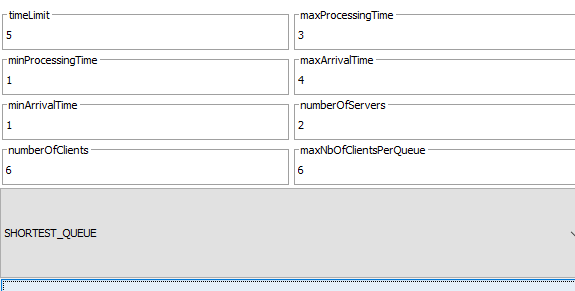
* Partea superiora in care se alfa 8 casete de text intitulate: timeLimit, maxProcessingTime, minProcessingTime, maxArrivalTime, minArrivalTime, numberOfServers, numberOfClients, maxNbOfClientsPerQueue.
  + timeLimit = valoarea introdusa semnifica timpul maxim pe care il va asteapta aplicatia inainte de a opri simularea si functionarea cozilor
  + maxProcessingTime = timpul maxim de procesare aleator ce poate fi alocat unui client generat
  + minProcessingTime = timpul minim de procesare aleator ce poate fi alocat unui client generat
  + maxArrivalTime = timpul maxim la care poate ajunge un client pentru a intra intr-o coada
  + minArrivalTime = timpul minim la care poate ajunge un client pentru a intra intr-o coada
  + numberOfServers = numarul de cozi disponibile
  + numberOfClients = numarul total de clienti ce trebuie procesati
  + maxNbOfClientsPerQueue = numarul de maxim de clienti ce pot astepta la o coada la un anumit moment in timp
* Partea din mijloc este un meniu de unde se poate alege intre 2 strategii : SHORTEST\_QUEUE, SHORTEST\_TIME
  + Clientii vor alege coada la care se duc cand le vine randul in functie de strategia aleasa
* Butonul "Start Simulation"
  + Porneste simularea coziilor si deschide fereastra in care este prezentata grafic

Fereastra **simulation** are starea curenta a simularii:

* 3 casete cu text:
  + Average waiting time = reprezinta media timpului asteptat de un client pana in momentul curent al simularii
  + Average service time = reprezinta media timpului petrecut in procesare a clientilor ce au fost procesati
  + Peak time = timpul la care numarul total de clienti in cozi a fost maxim pana la momentul curent
  + Status = arata daca simularea este terminata sau nu
* O coloana de checkouts care reprezinta locul unde sunt procesati clientii, fiecare este numerotata de la 0 la numarul de cozi – 1
* Pentru fiecare linie (fiecare checkout) este o lista de clienti ce asteapta care este updatata o dat ala 1 unitate de timp, care este in cazul acesta o secunda

**5. Rezultate**

O simulare unde timpul a fost marit la 10s, de la 1 s, deci fiecare client va fi procesat intr-o secunda in simulare, dar valoarea timpului mediu pentru procesare va ramane corect. Datele sunt :

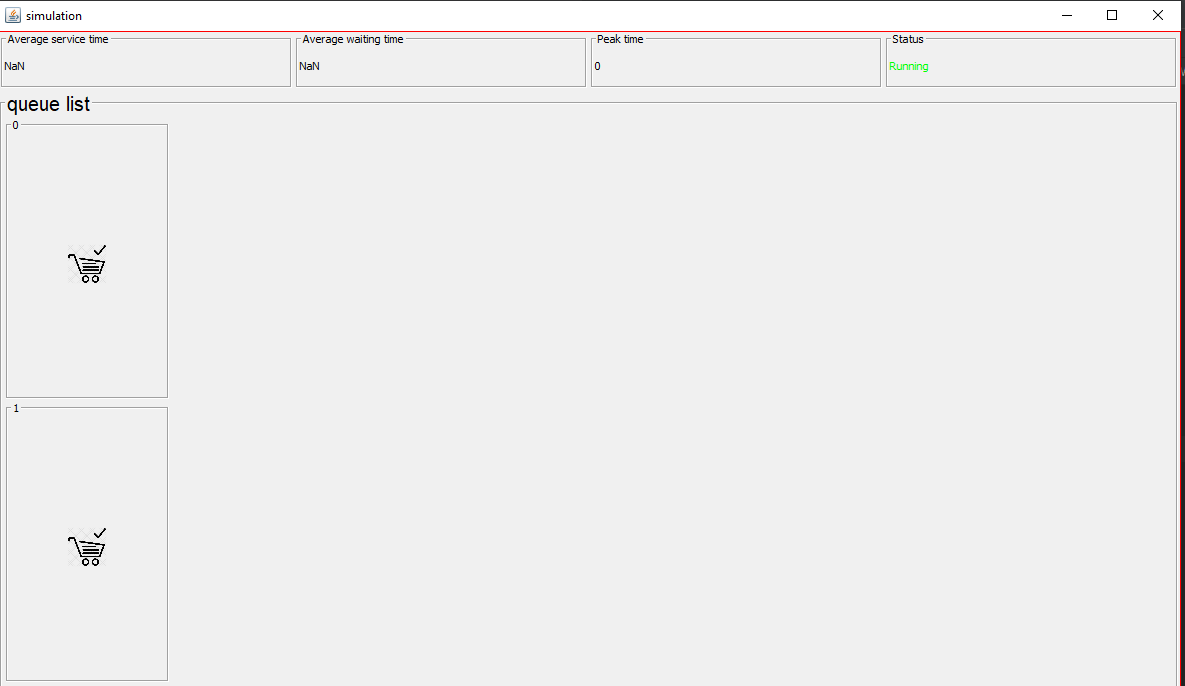


Time: 0

Waiting List: (1,1,1) (5,1,2) (2,2,2) (4,2,1) (3,3,2) (6,3,2)

Queue 1:

Queue 2:

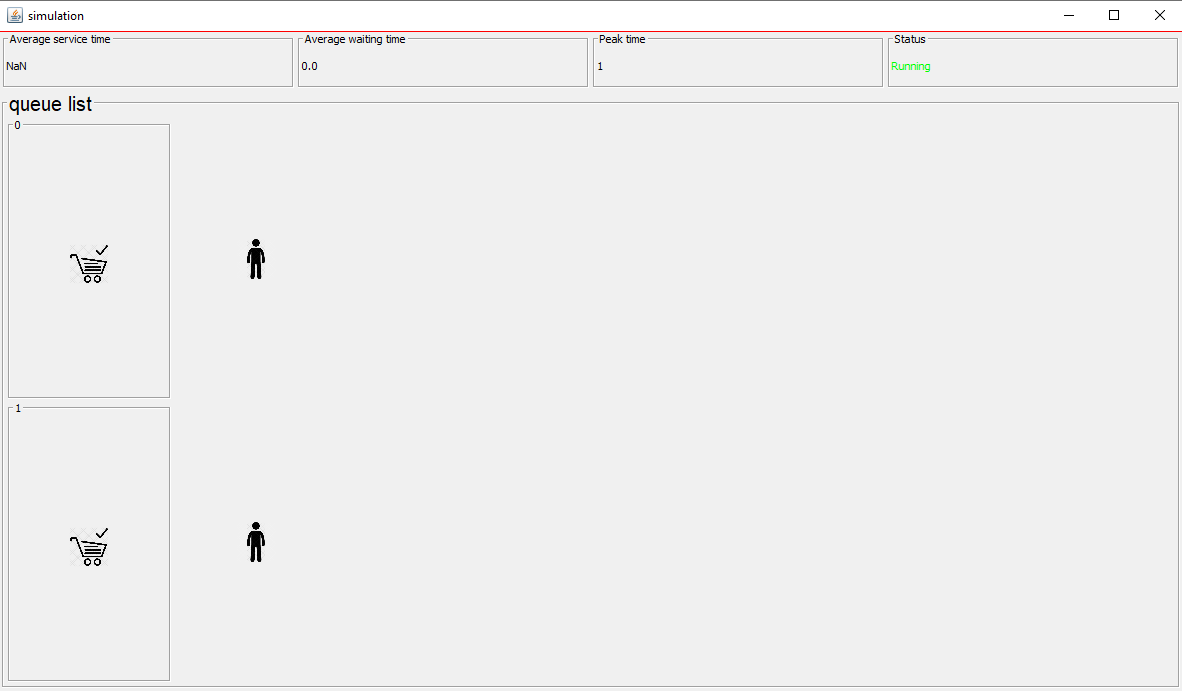


Time: 1

Waiting List: (2,2,2) (4,2,1) (3,3,2) (6,3,2)

Queue 1: (1,1,1)

Queue 2: (5,1,2)

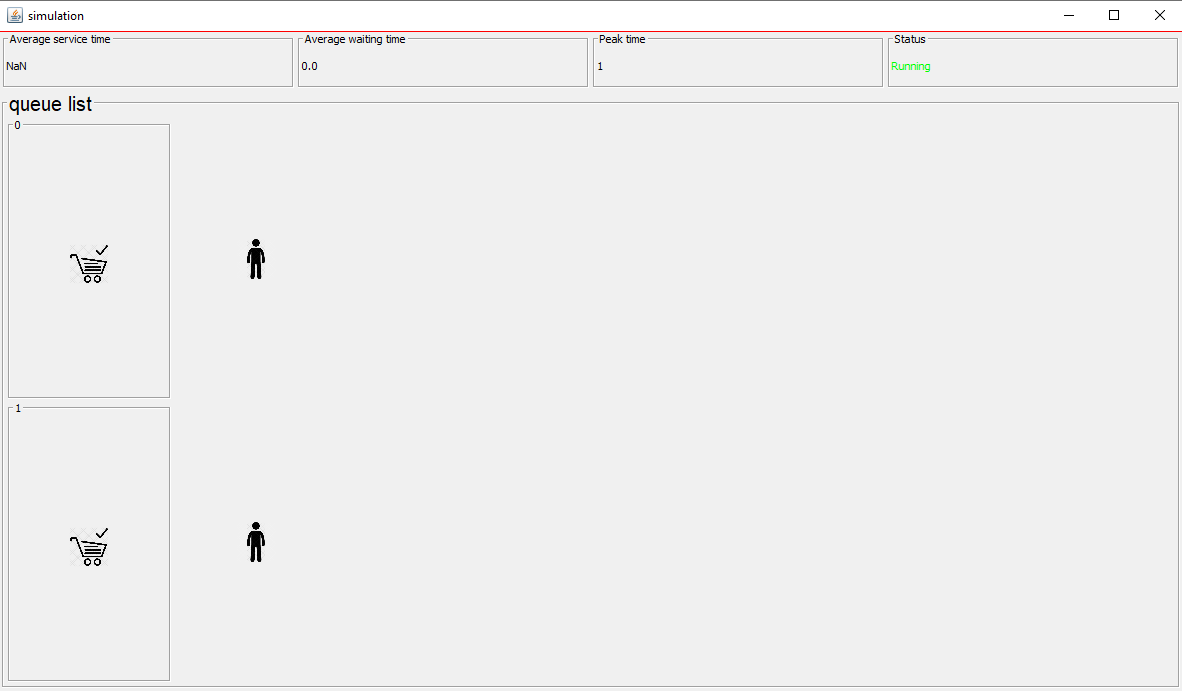


Time: 2

Waiting List: (3,3,2) (6,3,2)

Queue 1: (2,2,2)

Queue 2: (4,2,1)

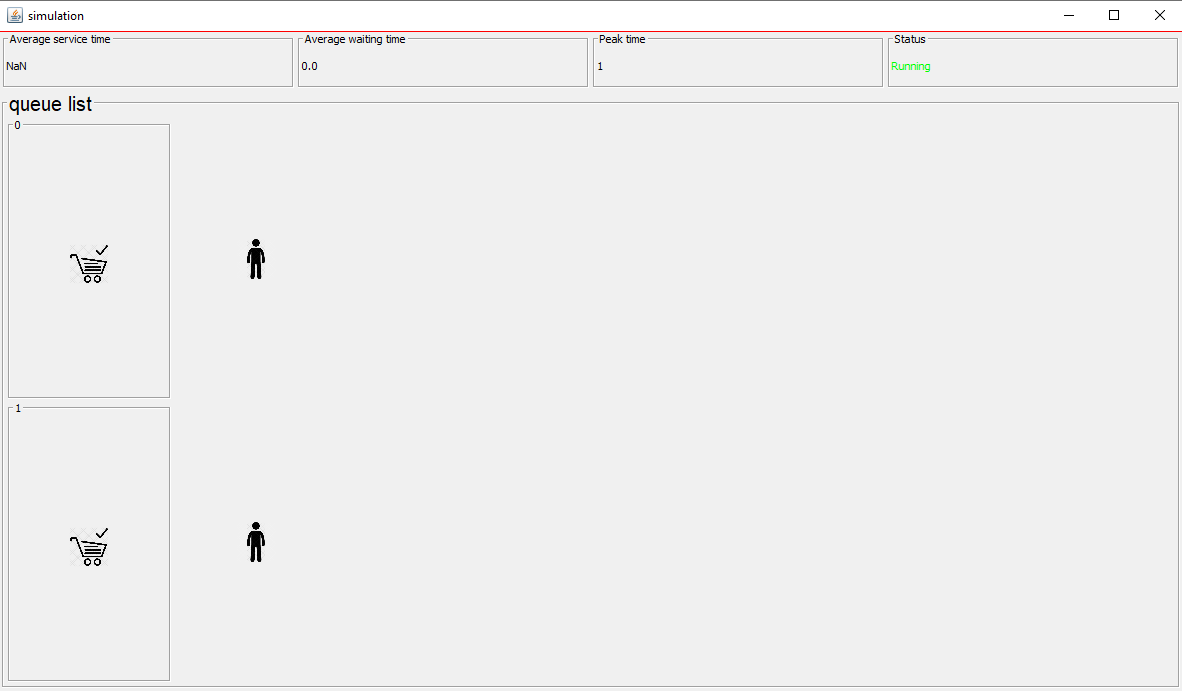


Time: 3

Waiting List:

Queue 1: (3,3,2)

Queue 2: (6,3,2)

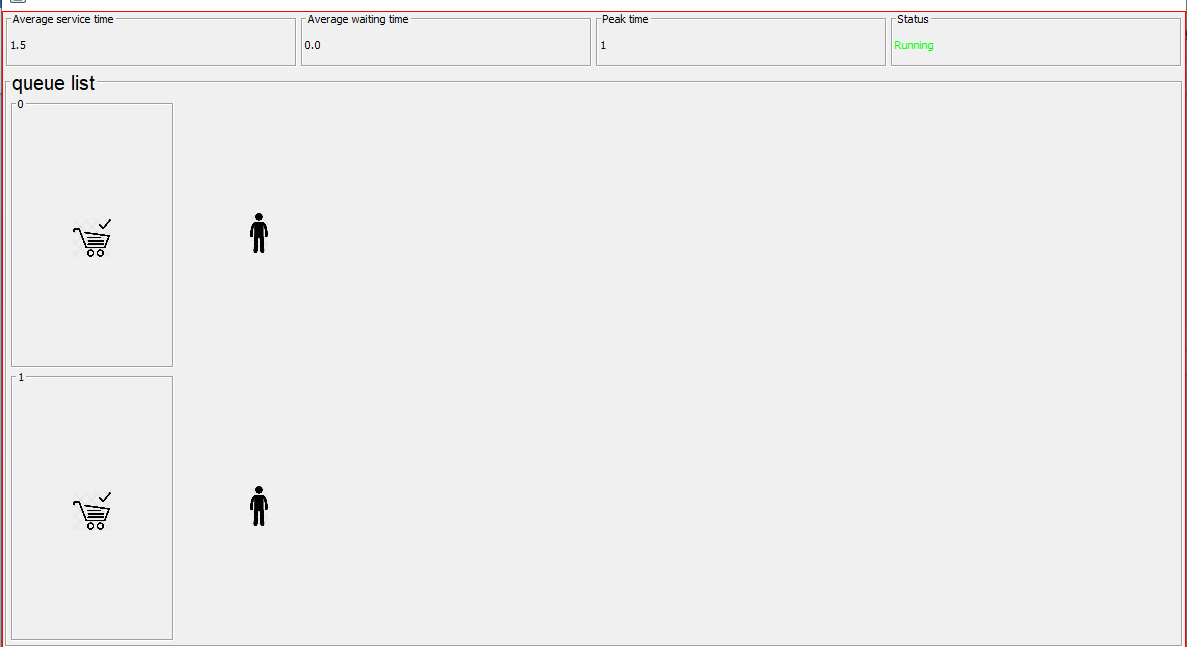


Time: 4

Waiting List:

Queue 1:

Queue 2:



**6. Concluzii**

Am realizat o simulare a mai multor cozi pentru mai multi clienti. Fiecare coada este procesata pe alt thread, valorile statistice fiind dupa reunite pe thread-ul principal pentru a fi afisat in interfata grafica. In acest proiect am invatat cum functioneaza un thread, si care este scopul sau. De asemenea am invatat cum sa pornesc si sa folosesc thread-uri in limbajul java, si cum interactioneaza acestea cu o interfata grafica.

**7. Bibliografie**

* stackoverflow.com
* www.baeldung.com
* https://www.w3schools.com/java/java\_threads.asp
* https://docs.oracle.com/javase/7/docs/api/java/lang/Thread.html
* https://docs.oracle.com/javase/tutorial/essential/concurrency/runthread.html
* https://www.javatpoint.com/multithreading-in-java