Universitatea Tehnica din Cluj-Napoca

Facultatea de Automatica si Calculatoare

Sectia Calculatoare si Tehnologia Informatiei

Documentatie pentru aplicatie de generare a serverelor de clienti

Student: Timotei Molcut

Cluj-Napoca

4 Aprilie 2019

Cuprins

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

Obiectivul temei

Principalul obiectiv al temei este de a crea o simulare in care mai multi clienti sunt procesati pe rand in mai multe servere. Practic, simularea aceasta consta dintr-un thread mare care maniuleaza mai multe threaduri mai mici, numite servere. Datele simularii pot fi introduse de la inceput de catre utilizator.

Pentru a atinge acest obiectiv a fost nevoie de o logica de control concurenta(primul obiectiv secundar). Aceasta logica concurenta se refera la felul in care threadurile trebuie sa functioneze. Adica, la anumite momente de timp fiecare thread trebuie sa intre in pauza, lasand restul threadurilor sa functioneze. Se numeste logica concurenta deoarece threadurile sunt prin definitie fire de executie care „lucreaza in paralel”.

Un alt aspect secundar reprezinta conceperea interfetei grafice si a legaturii sale cu simularea. Pentru a putea seta simularea din interfata este posibila introducerea parametrilor simularii in prima fereastra. Aceasta contine un buton care deschide o alta fereastra cu simularea propriu-zisa, iar pentru a afla statisticile, la finalul simularii se apasa un buton care va deschide fereastra finala.

Obiectivul secundar final a fost modelarea conceptelor de Client, Server si Simulator. Am ajuns la concluzia ca un Server detine mai multi Clienti, ia simulatorul detine mai multe Servere. Fiecare dintre aceste obiective va fi discutat amanuntit in urmatoarele randuri.

Analiza problemei, modelare, scenarii, cazuri de utilizare

Analizand problema de fata, trebuie sa determinam organizarea obiectuala a fiecarui concept de care avem nevoie.

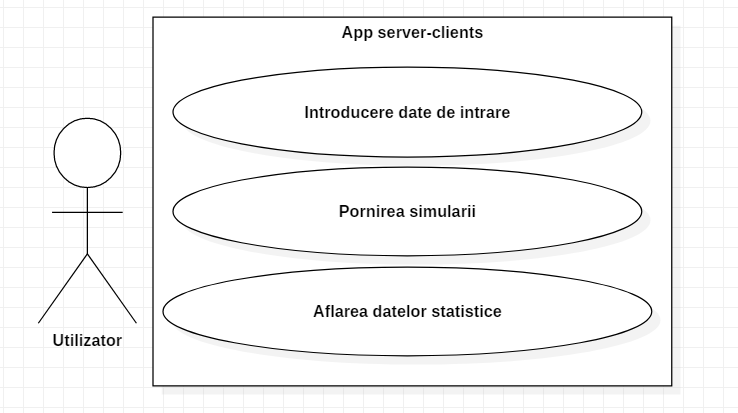
Conceptul de Client poate fi descris in felul urmator: Clientul este reprezentat de orice fel de obiect care cere indeplinirea unor cerinte(de catre altcineva). De exemplu, putem vorbi de clientul de la o casa de marcat din super-market sau, intr-un context mai abstract, clientul poate fi un utilizator web care doreste sa acceseze un site. In cazult de fata cerinta Clientului este de a i se acorda un timp „random” de asteptare.

De cealalta parte, Serverul este reprezentat ca un servitor(de unde ii vine si numele). Acest servitor trebuie sa indeplineasca cerinta data de client, adica, in cazul de fata, sa poata astepta atat cat cere clientul.

Pana aici putem modela problema folosint conceptele de Client si Servitor, dar se va putea ridica intrebarea: „cine va simula trecerea clientilor?”. Asadar, este nevoie de definirea a inca unui concept obiectual si anume Simulatorul. Acest Simulator poate fi descris ca o baza de date a Clientilor si Serverelor de la un anumit moment. Adica, Simulatorul tine evidenta la fiecare secunda despre cum arata Serverele din componenta sa si pentru fiecare servere, care este ordinea de procesare pentru Clienti.

Utilizarea acestei aplicatii necesita introducerea datelor principale ale simularii, acestea fiind: timpul simualrii, intervalul pentru asteptare intre clienti(cu minimum si maximum), intervalul de procesare al unui client(cu minimum si maximum), numarul de clienti si numarul de servere(cozi). Dupa introducerea acestor date, apasand un buton se va putea deschide fereastra simularii. Avand a doua fereastra deschisa, cand utilizatorul doreste poate porni simularea apasand pe un buton de start. La finalul simularii, va aparea un buton pentru afisarea statisticilor. Apasand butonul respectiv se va deschide o noua fereastra care va specifica anumite statistici cum ar fi: timpul mediu de asteptare pentu fiecare server, momentul de timp cand a fost serverul mai aglomerat(peak hour).

Asadar, cerintele functionale ale acestei aplicatii sunt: introducerea datelor de intrare, apasarea butoanelor de generare a simularii si de pornire a simularii si in final aflarea statisticilor simularii. Aceste cerinte pot fi descrise in diagrama UML de tip use-case, de mai jos.



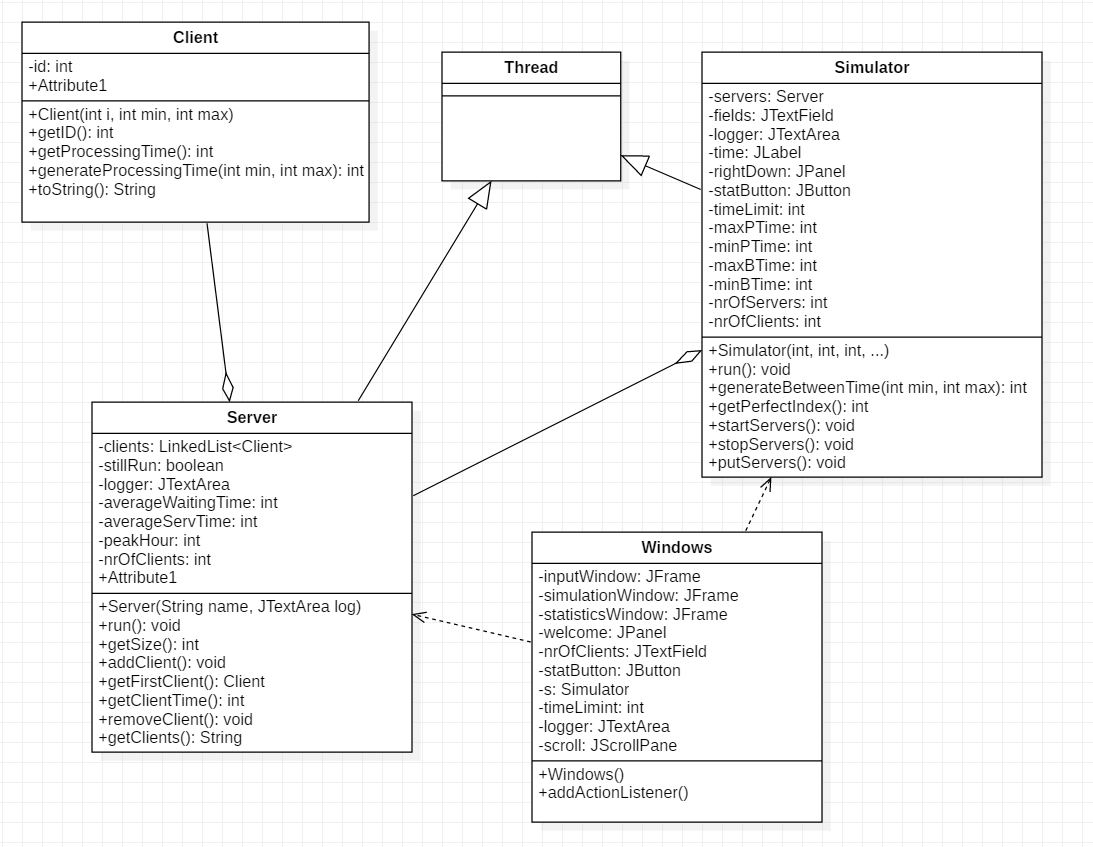
Proiectare

Asa cum am precizat deja, conceptul de Server contine mai multi Clienti care trebuie sa fie procesati intr-o structura de tip coada(primul venit, primul servit). Astfel, am folosit o lista LinkedList<Client>. Asadar se va crea o relatie de agregare intre Server si Client. Fiind practic un thread, Serverul este de fapt un derivat al clasei Thread din Java. Astfel, el trebuie sa implementeze metoda run() pe care o voi descrie pe larg in capitolul urmator (implementare). Pe langa aceasta Serverul mai are un flag numit „stillRun” care practic determina daca Serverul mai trebuie sa ruleze sau nu. Acest flag va fi manipulat in Simulator. Pentru a determina statistici Serverul mai contine anumite atribute care memoreaza numarul maxim de clienti si valorile perioadelor de timp mediu de asteptare si servire, precum si peak hour.

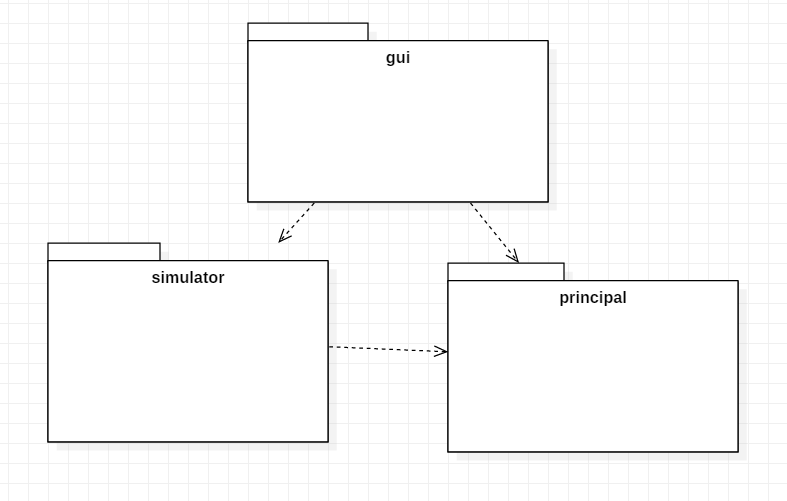
Intorcandu-ne la clasa Client, putem sa il descriem prin atributele de ID, care va fi unic, si timpul de procesare care va fi ales la intamplare intre doua valori introduse de utilizator. Dupa cum am spus, acest timp de procesare determina cat de mult timp va aloca serverul clientului respectiv(atunci cand clientul poate fi servit, adica este in capul cozii).

Tinand cont ca este nevoie de o simulare, am declarat clasa Simulator. Dupa cum am mai spus, aceasta clasa contine toate datele introduse de utilizator. Printre acestea specific din nou ca va contine un tablou de Server, cu dimensiunea maxima introdusa de catre utilizator. Din aceasta informatie se poate determina relatia de agregare dintre Simulator si Server. Sa nu uitam faptul ca si Simulatorul este un derivat al clasei Thread. Asadar, se va crea o relatie de mostenire intre Simulator si Thread.

Interfata grafica, fara de care ar fi mai grea observarea simularii este un avantaj pentru utilizator. Practic, dupa cum am mai zis, aceasta interfata va afisa dinamic trei ferestre: prima, de intrare, a doua, de simulare si a treia, de statistici. Fiecare fereastra contine mai multe JLabel sau JTextField. Pe aceasta interfata am numit-o sugestiv Windows. Dupa cum am mai spus, clasa de fata(Windows) contine un Simulator care la randul sau contine mai multe servere. Acest Simulator va infuenta foarte mult functionarea clasei Windows. De aici se poate determina o relatie de dependenta intre Windows si Simulator, respectiv Server. Toate delatiile de mai sus pot fi urmarite in diagrama UML de clase de mai jos.



Aplicatia de fata este organizata in trei pachete. Primul este numit „principal”. Dupa cum spune si numele aici sunt clasele principale Client si Server fara de care simularea nu ar putea functiona. Al doilea pachet l-am numit „simulator” deoarece contine clasa cu acelasi nume. Pana aici se determina o relatie de dependenta intre pachetul simulator si pachetul principal. Al treilea si ultimul pachet este numit „gui”, de la acronimul pentru Graphic User Interface. Practic, dupa cum spune si numele, aici se afla clasa Windows(interfata), care are rol de view si de controller al aplicatiei. In final se vor determina relatiile de dependenta intre pachetul gui si celelalte doua pachete. Aceste relatii sunt descrise de diagrama UML de pachete de mai jos.



Implementare

Acest capitol este cel mai substantial deoarece determinarea implementarii a reprezentat cel mai costisitor aspect al acestei aplicatii. Pentru inceput voi descrie implementare Clientului.

Dupa cum am precizat deja, Clientul nu are decat un ID, generat unic in Simulator si un timp de procesare, de asemenea, procesat in Simulator, care va lua o valoare random din intervalul minProcessingTime si maxProcessingTime. Aceasta valoare random o determin in metoda statica numita generateProcessingTime(int min, int max). In aceasta metoda folosesc un algoritm propriu de aflare a unui numar random. Inmultesc Math.random() cu valoarea intervalului Math.abs(max - min), dupa care fac inmultiri si impartiri, iar la final voi returna restul, random%=Math.abs(max - min) adunat cu capatul min al intervalului. Aceasta metoda o apelez in constructorul Clientului.

In continuare voi descrie implementarea unui Server. Dupa cum am mai zis, Serverul contine o coada de Clienti, adica LinkedList<Client>, un flag pentru pornirea/oprirea rularii, numit stillRun si, ca detaliu paralel, Serverul mai trebuie sa specifice intr-un logger de tip JTextArea ce se petrece in timpul rularii sale. In constructorul sau Serverul primeste un nume(ex: Server #0), care il va distinge in cadrul Simulatorului. Deoarece structura LinkedList<> nu are o sincronizare interna precum BlockingQueue, este necesar ca metodele ce acceseaza aceasta lista a clientilor sa fie sincronizate(synchronized), iar metodele de accesare sunt urmatoarele: getSize() care returneaza lungimea listei la un moment de timp, addClient(Client c) care adauga parametrul Client c in lista, la final, getFirstClient() care va returna primul client din lista, getClientTime() care va returna timpul de procesare al primului client din lista, removeClient() care va sterge primul client din lista(metoda echivalenta cu pop()) si in final metoda de afisare a listei de clienti, respectiv getClients().

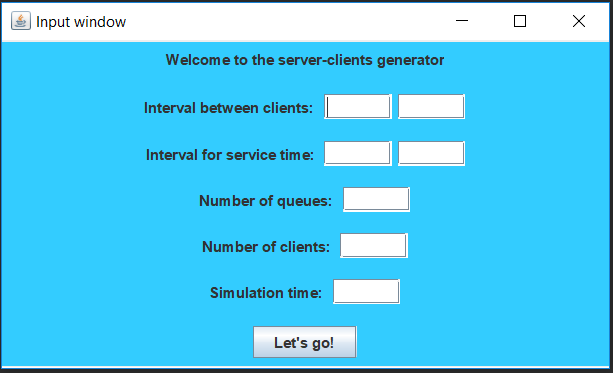
Deoarece Serverul este in sine un thread care nu mosteneste alta clasa, este de ajuns ca Server sa mosteneasca Thread, adica nu se va implementa interfata Runnable. Deoarece Server mosteneste Thread, acesta va specifica o noua functionare, adica va implementa metoda run(). Aceasta metoda este caracterizata de un while loop care are drept conditie flagul stillRun. Adica, cat timp stillRun este true tot ce este in metoda run, implicit in while loop, se va repeta. In acest while loop se verifica daca dimensiunea listei este mai mare decat 0, adica doar atunci are rost sa ne folosim de threadul respectiv. In acest caz se va returna primul client si timpul sau de procesare. Dupa aceasta, pentru a „simula” timpul de procesare al clientului serverul curent va intra in starea sleep pentru atatea secunde cat valoreaza timpul de procesat al clientului. Dupa aceasta pauza, se va scoate primul client din lista, adica clientul respectiv a fost deservit. Acest lucru se va specifica in logger-ul interfetei grafice. Asadar, pe scurt se poate spune ca serverul nu stie cand trebuie sa se opreasca. Acest lucru il va determina simulatorul. Dar momentul cand „se ia in calcul” serverul curent este atunci cand are cel putin un client de servit. Cand Simulatorul doreste sa se opreasca(adica simularea este gata), va putea sa seteze flagul stillRun pe false, iar serverul respectiv isi va termina activitatea. Pentru Server mai sunt si cateva metode auxiliare de setare si determinare a timpilor medii de asteptare, medii de servire si peak hour care vor fi apelate in timpul simularii, determinand in final statisticile.

Simulatorul este si el un thread. Asadar, este de ajuns sa se mosteneasca clasa Thread. Astfel, se va putea implementa metoda run(). In aceasta metoda, primul lucru este pornirea serverelor, adica setarea stillRun pe true pentru fiecare Server. Dupa aceasta, avand o variabila curTime simbolizand timpul curent al simularii, am scris un while loop care verifica ca timpul maxim de simulare sa nu fie intrecut de catre curTime. Intrand in while loop se va genera un index(perfecrIdx) care este indexul serverului cel mai avantajos, adica primul server cu cea mai scurta coada de asteptare. Dupa aceasta, avem o variabila id care creste si se verifica daca aceasta este maxim numarul de clienti dedicati pentru simulare. In caz pozitiv, se va crea un nou client pentru care se va genera un moment la care sa se adauge. Acest moment se determina random(cu acelasi algoritm de la clasa Client) intre valorile minBetweenTime si maxBetweenTime. Daca timpul generat este 0 clientul este adaugat in iteratia curenta, dar daca este mai mare se vor astepta atatea iteratii cat reprezinta timpul respectiv in secunde. Daca s-a adaugat un Client nou in cel mai avantajos Server se vor apdata campurile pentru statistici (averageWaitingTime, peak hour). De asemenea, se vor specifica in logger informatii despre adaugarea, respectiv cat timp se asteapta pentru Clientul urmator. La finalul iteratiei, pentru a se simula trecerea unei secunde la incrementarea lui curTime se va pune threadul Simulator pe sleep(1000). Repet: un Client este adaugat doar daca timpul de asteptare este zero altfel se va astepta pana cand trec atatea iteratii cate sunt necesare(cu alte cuvinte sleep(waitingTime)). La finalul simularii, chiar daca nu s-au introdus toti clientii sau nu au fost serviti toti, se vor finaliza datele de statistici, iar serverele se vor opri prin setarea stillRun = false.

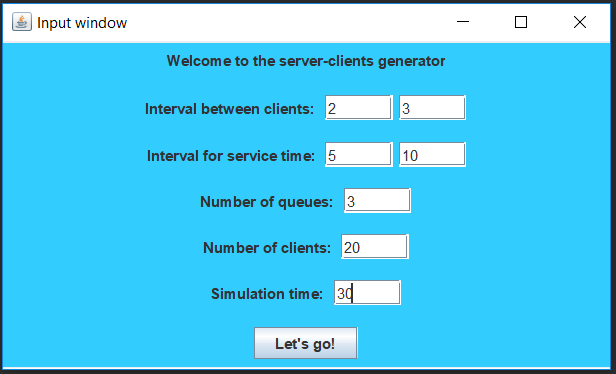
In final am ajuns la implementarea interfetei, adica clasa Windows. Aceasta are un inputWindow care contine mai multe JLabel si JTextField in care se vor introduce parametrii simularii. Daca cumva sunt incorecte(ex; se introduce un String cu litere), se va semnala printr-un mesaj de eroare. Prin apasarea butonului openSimulation fereastra curenta se inchide, dar nu inainte de a memora parametrii in variabile. Se deschide fereastra de simulare care contine in partea stanga numarul de server dorit, iar in partea dreapta se va afisa un logger si un buton de pornire al simularii. Prin apasarea acestiu buton se apeleaza cosntructorul Simulator, iar simularea incepe. Se afiseaza curTime, iar logger incepe sa afiseze ce se petrece in Simulator: se adauga clienti, se asteapta dupa clienti, se proceseaza clienti si se sterg clienti. La finalul simularii(si doar atunci) va aparea butonul pentru deschiderea ferestrei de statistici. Prin apasarea acestui buton, se inchide fereastra curenta, se deschide ultima fereastra(cu statistici) si se vor afisa rezultatele simularii: averageWaitingTime, averageServTime si peak hour, pentru fiecare server existent in simulare. In final, se poate inchide aplicatia.

Rezultate

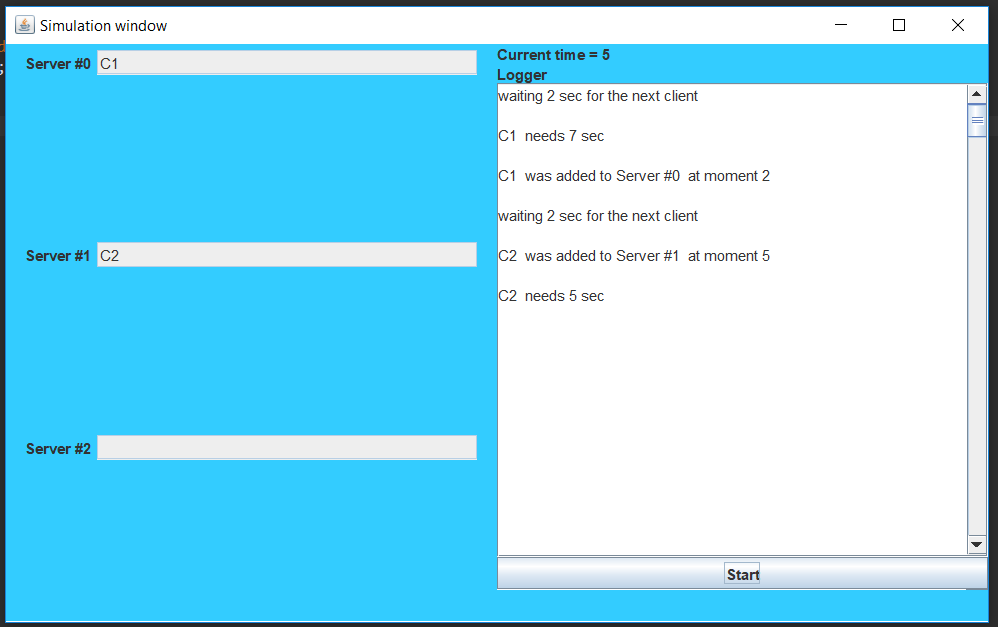
In clasa App.java se apeleaza constructorul Windows(), care porneste aplicatia. Aici aplicatia arata in felul urmator.



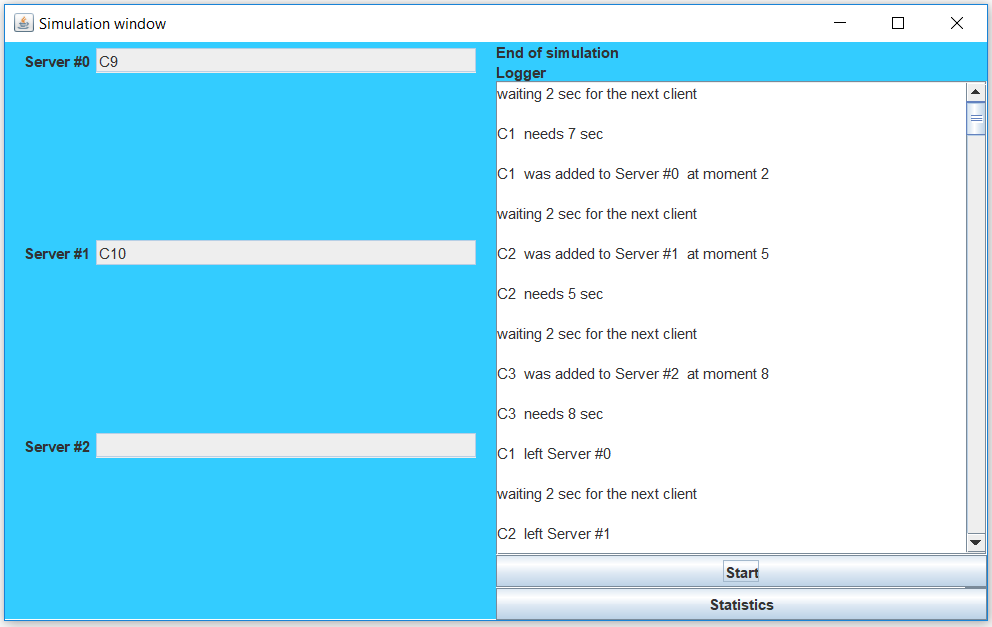
Urmeaza sa putem introduce niste date in interfata.



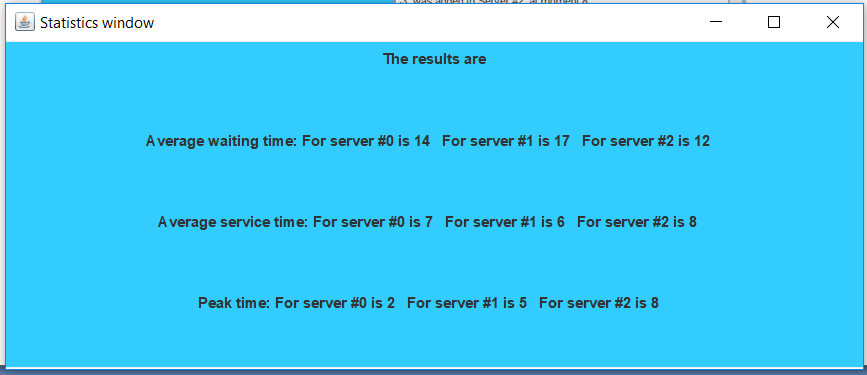
Dupa apasarea lui „Let’s go!” urmeaza sa se deschida a doua fereastra si daca apasam „Start”, simularea incepe.



La finalul simularii fereastra arata asa.



Apasand pe „Statistics” urmeaza...



De aici se poate opri aplicatia.

Concluzii

In concluzie, pot sa spun ca aceasta applicatia a fost interesanta si a reperzentat o adevarata provocare in a intelege modul de functionare al threadurilor si metode de a le manipula. A fost cu adevarat un beneficiu intelegerea de baza a executiei concurente pe care limbajul Java o ofera prin threaduri.

Imbunatatirile pe care le-as aduce acestei aplicatii ar fi: eficientizarea unor metode, introducerea de clienti manual si in mod dinamic in timpul simularii.

Bibliografie

* World Wide Web
* <https://en.wikipedia.org/wiki/Main_Page>
* <https://stackoverflow.com>
* <http://inf.ucv.ro/documents/tudori/laborator8_53.pdf>