MINISTERUL EDUCAȚIEI NAȚIONALE
UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

Facultatea de Automatica si Calculatoare

Tema 2:
Implementarea unei aplicatii
de simulare a cozilor de asteptare

Disciplina: Tehnici de programare

Student:
Coman Vasile
An II
Grupa 30221

Profesor coordonator:
Antal Marcel
Profesor curs:
Ioan Salomie

Cuprins

1. Obiectivul temei

Obiectivul principal al temei este acela de a implementa o aplicatie de simulare a cozilor de astepare cu scopul determinarii si minimizarii timpului de asteptare.
Obiective secundare:
-implementarea clientilor cu timpii de sosire si de procesare;
-crearea serverelor
-implemenatrea unui planificator si in functie de strategie de distribuire aleasa clientii sa fie distribuiti la unul dintre servere;
-implementarea unui manager al aplicatie care sa preia toate datele legate numarul de servere, numarul de clienti, timpi si sa apeleze mecanismele de simulare descrise si in acelasi timp ca coordoneze afisarea pe o interfata grafica a datelor dorite.

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

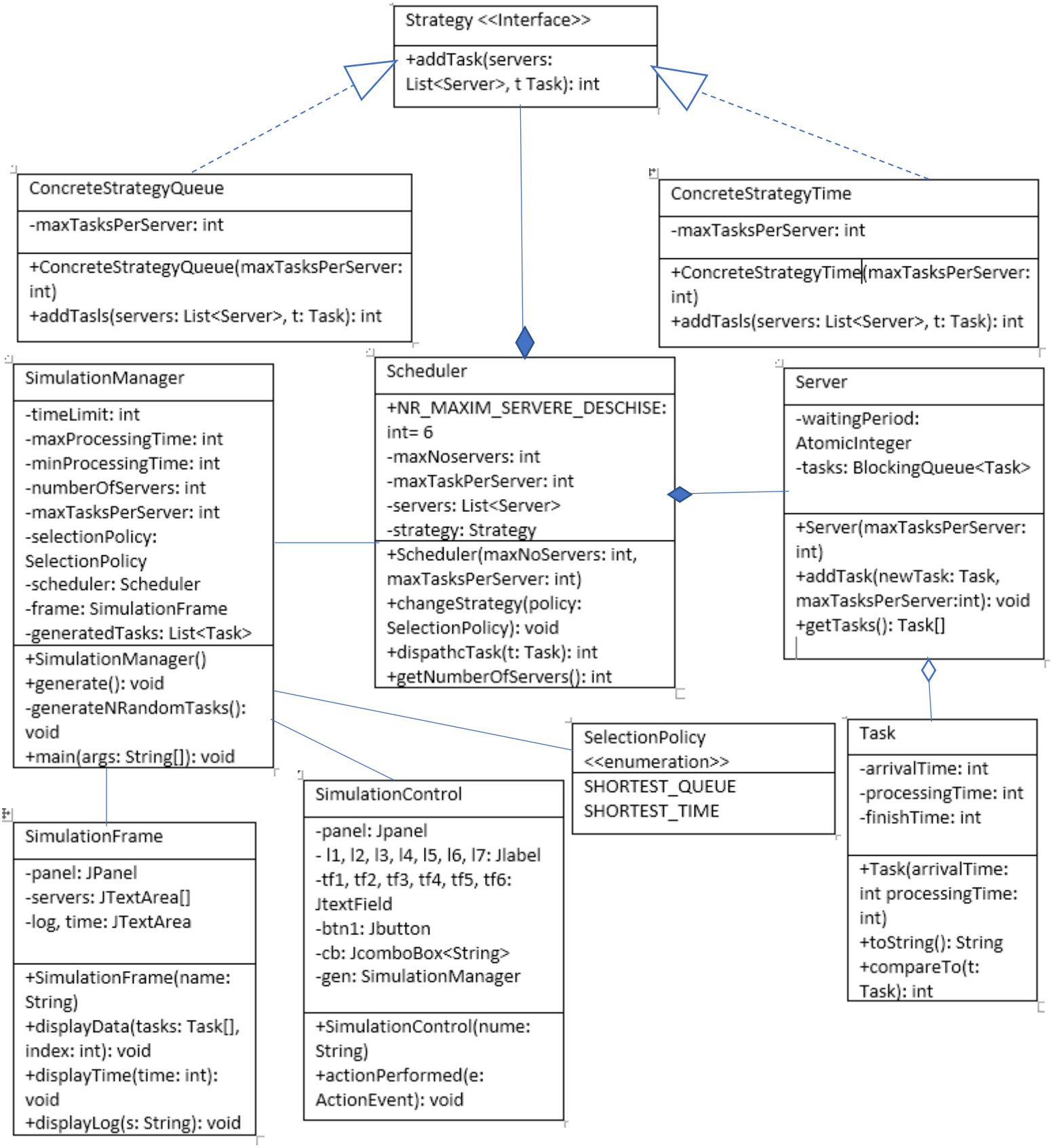
Coada este un container bazat pe principiul first-in first-out (FIFO). Adică elementele pot fi adăugate în coadă în orice moment, dar numai cel mai vechi (cel de la bază) element poate fi eliminat în orice moment (se extrage cea mai veche informaţie adăugată). Mai simplu, elementele sunt adăugate în spate şi eliminate din faţă.
Cerinte si functionalitati:
-avem de adaugat elemente, generate random in functie de un interval dat, intr-o coada in functie de strategie aleasa;
-strategie aleasa poate sa fie de tipul cea mai scurta coada sau de tipul cel mai scurt timp de asteptare;
-deschiderea unui nou server in cazul in care toate celelalte servere sunt pline, insa nu pot fi deschise mai multe servere decat un numar maxim dat, numarul serverelor initiale trebuie sa fie mai mic decat acest maxim;
-procesarea cozilor adica eliminare din coada a taskului daca timpul de procesare pentru acel task a ajuns la 0 si prelucrarea urmatorului element;
-continuarea simularii pana cand toate taskurile au fost procesate;
-afisarea in timp real al cozilor;
-afisarea unui istoric cu toate actiunile petrecute.

Use-case
Avem o aplicatie capabila sa simuleze in timp real un sistem de gestionare a cozilor de asteptare utilizatorul putand sa introduca:
-intervalul de timp in care vor fi generate taskurile;
-timpul minim si timpul maxim de procesare;
-numarul de servere sau cozi initiale, alte servere vor fi deschise automat daca este nevoie pe parcursul simularii;
-numarul de taskuri care vor fi generate in acel interval de timp;
-si numarul maxim de taskuri pe care un server le poate avea la un moment dat.
In schimb utilizatorului i se vor oferi date prin intermediul interfetei grafice:

-i se vor afisa toate serverele deschise si taskurile continute de acestea si evolutia lor de la timpul 0 pana la timpul necesar procesarii tuturor taskurilor;
-se va afisa taskurile care intra, cele cara ies alaturi de timpii lor, daca s-a deschis o noua coada, timpul mediu de astepate si ora de varf;
- se va afisa timpul simularii.
Asumptii facute:
-se prespune ca fiecare coada are un numar maxim de taskuri pe care le poate avea;
-se presupune ca utilizatorul introduce un numar de servere mai mic sau cel putin egal decat numarul de servere maxim admis de aplicatie care este o constanta aleasa de noi;
-se presupune ca utilizator introduce doar date valide adica numere pozitive si ca timpul maxim de procesare este mai mare decat timpul minim;
-se presupune ca datele introduse nu vor duce la situatia cand toate serverele sun pline.

3. Proiectare (decizii de proiectare, diagrame UML, structuri de date, proiectare clase, interfete, relatii, packages, algoritmi, interfata utilizator, modul de tratare a erorilor)

Am ales sa impart proiectul in 8 clase, o interfata si un enum:
-clasa Task definita prin timp de procesare, timp de sosire si timp de terminare;
-clasa Server definita printr-o coada de taskuri si o perioada de asteptare pentru fiecare coada;
-clasa Scheduler definita printr-o lista de servere si de tipul de strategie care adauga taskul in serverul disponibil;
-interfata Strategy care contine metoda neimplementata de a adauga un task;
-enumul SelectionPolicy care contine tipurile de strategii disponibile;
-clasele ConcreteStrategyQueue si ConcreteStrategyTime care implementeaza interfata Strategy si implementeaza metoda de adaugare a unui task in functie de strategia dorita;
-clasa SimulationManager care controleaza interfata grafica;
-clasa SimulationFrame care afiseaza rezultatele;
-clasa SimulationControl care preia datele de la utilizator.


Strategy <<Interface>>
+addTask(servers: List<Server>, t Task): int
ConcreteStrategyQueue
-maxTasksPerServer: int
+ConcreteStrategyQueue(maxTasksPerServer: int)
+addTasls(servers: List<Server>, t: Task): int
ConcreteStrategyTime
-maxTasksPerServer: int
+ConcreteStrategyTime(maxTasksPerServer: int)
+addTasls(servers: List<Server>, t: Task): int
SimulationManager
-timeLimit: int
-maxProcessingTime: int
-minProcessingTime: int
-numberOfServers: int
-maxTasksPerServer: int
-selectionPolicy: SelectionPolicy
-scheduler: Scheduler
-frame: SimulationFrame
-generatedTasks: List<Task>
+SimulationManager()
+generate(): void
-generateNRandomTasks(): void
+main(args: String[]): void
Scheduler
+NR_MAXIM_SERVERE_DESCHISE: int= 6
-maxNoservers: int
-maxTaskPerServer: int
-servers: List<Server>
-strategy: Strategy
+Scheduler(maxNoServers: int, maxTasksPerServer: int)
+changeStrategy(policy: SelectionPolicy): void
+dispathcTask(t: Task): int
+getNumberOfServers(): int
Server
-waitingPeriod: AtomicInteger
-tasks: BlockingQueue<Task>
+Server(maxTasksPerServer: int)
+addTask(newTask: Task, maxTasksPerServer:int): void
+getTasks(): Task[]
SelectionPolicy <<enumeration>>
SHORTEST_QUEUE
SHORTEST_TIME
Task
-arrivalTime: int
-processingTime: int
-finishTime: int
+Task(arrivalTime: int processingTime: int)
+toString(): String
+compareTo(t: Task): int
SimulationFrame
-panel: JPanel
-servers: JTextArea[]
-log, time: JTextArea
+SimulationFrame(name: String)
+displayData(tasks: Task[], index: int): void
+displayTime(time: int): void
+displayLog(s: String): void
SimulationControl
-panel: Jpanel
- l1, l2, l3, l4, l5, l6, l7: Jlabel
-tf1, tf2, tf3, tf4, tf5, tf6: JtextField
-btn1: Jbutton
-cb: JcomboBox<String>
-gen: SimulationManager
+SimulationControl(nume: String)
+actionPerformed(e: ActionEvent): void

4. Implementare

Clasa Task defineste un client care este caracterizat prin timpu de sosire(arrivalTime), timp de procesare(processingTime) si timp de inchiere(finishTime). Toate 3 campurile sunt de tip int. Constructorul primeste 2 parametri de tip int care reprezinta timpul de sosire si cel de procesare iar timpul de terminare va reprezentat suma acestora la care vom adauga perioada de astepare(waitingPeriod) a serverului la care va fi plasat taskul.

Metode implementate:

-settere si gettere pentru variabilele instanta;

-am suprascris metoda toString pentru a afisa un fiecare task care se afla in coada alaturi de timpul de sosire si cel de procesare;

-de asemenea clasa Task implementeaza interfata Comparable si am suprascri metoda compareTo pentru a putea sorta un array de taskuri.

Clasa Server implementeaza interfata Runnable deoarece pentru fiecare coada creata vom crea un nou fir de executie(thread). Variabilele instanta a clasei Server sunt:

- waitingPeriod care reprezinta perioada de asteptare a unui task in momentul in care intra in coada pana cand este procesat si scos din coada si este de tip AtomicInteger pentru a evita incosistenta cand aceasta valoare este accesata de mai multe threaduri;

-tasks care reprezinta taskurile care sunt asociate acelui server de tip BlockingQueue fiecare coada avand un numar maxim de elemente pe care le poate contine acest lucru fiind transmis ca si parametru in constructor.

Metode implementate:

-metoda addTask adauga un task in coada daca nu este depasita capacitatea acesteia, seteaza timpul de terminare ca suma din timp de sosire si timp de procesare a taskului si timp de asteptare al cozii si creste timpul de asteptare al cozii cu timpul de procesare;

-in metoda run se ia din coada primul element(head) si timpul acestuia de procesare iar cat timp timpul de procesare este diferit de 0 stopam trehadul pentru o secunda apoi decrementam waintingPeriod si processingTime pana ajunge la 0, dupa care eliminam elementul din coada;

-metoda getTask returneaza un array de taskuri.

SelectionPolicy este un enum care contine enumerarea strategiilor de distribuire a taskurilor la cozi: SHORTEST_QUEUE, SHORTEST_TIME.

Interfata Strategy contine metoda neimplementata addTask.

Clasa ConcreteStrategyQueue implementeaza interfata Strategy si descrie metoda addTask care primeste ca si parametru lista de servere si taskul care trebuie distribuit. Serverului cu cele mai putine elemente ii va fi distribuit taskul transmis. Aceasta metoda va returna si indexul serverului lucru util care ne ajuta la afisarea in interfata.

Clasa ConcreteStrategyQueue implementeaza interfata Strategy si descrie metoda addTask. Taskul transmis ca si parametru va fi adaguat in serverul care are cel mai mic timp de asteptare. De asemenea se va returna si indexul serverului la care a fost distribuit taskul pentru a fi de folos la afisarea in interfata.

Clasa Scheduler are rolui unui planificator. El contine o constanta care reprezinta numarul maxim admis de servere care se pot deschide, deoare exista un numar inital de servere dat de utilizator dar in cazul in care aceste servere vor fi pline planificatorul va avea dreptul sa deschida si alte servere pana cand numarul acesta este mai mic sau egal cu acea constanta.

Variabilele instanta:

-initialNoServer care reprezinta numarul initial de servere deshise dat de catre utilizator;

-maxTasksPerServer care reprezinta capacitatea fiecarui server;

-strategy care este de tipul interfetei Strategy;

-o lista de servere;

Construtorul initializeaza numarul initial de servere, numarul maxim de taskuri pe servere si creeaza o lista de servere iar pentru fiecare server se creaza un thread si se porneste fiecare thread.

Metode implementate:

-metoda changeStrategy alege una din cele 2 strategii implementate in funcite de datele date de utilizator.

- metoda dispatchTask verifica daca toate serverele sunt pline iar daca da se creeaza alt thread acest lucru fiind posibil pana cand numarul thredurile atinge constanta de numar maxim admis de threaduri dupa care se apeleaza metoda de adaugare task care in functie de strategia aleasa ba distribui taskul la una dintre cozi, Metoda returneaza si indicele cozii.

-metoda getServers returneaza o lista cu toate serverele create si getNumberOfServers numarul acestora.

Clasa SimulationFrame extinde clasa JFrame si aici este construita interfata grafica care va afisa rezultatele. Constructorul din aceasta clasa primeste ca si argument un string care va reprezenta numele ferestrei. Se va construi un JTextArea pentru fiecare server, un JTextArea pentru a afisa istoricul actiunilor de intrare, iesire, timpul mediu si ora de varf si un JTextArea pentru a se afisa timpul. In metoda displayData se va

seta sa fie vizibil acel JTextArea si se vor parcurge taskurile din serverul transmis si se vor adauga in JTextArea. La fel se intampla si in metodele displayLog si displayTime care vor afisa istoricul si timpul. Toate aceste 3 metode apeleaza repaint si revalidate pentru a actualiza fiecare modificare facuta.

Clasa SimulationManager controleaza afisarea datelor in interfata, genereaza si transmite taskurile spre a fi distribuite.

Variabilele instanta:

-timpul limita, timpul maxim si minim de procesare, numarul de servere, numarul de clienti si numarul maxim de clienti ce pot fi distribuiti pe un server de tip int;

-selectionPolicy de tip SelectionPolicy care va reprezenta strategia aleasa;

-un obiect de tip Scheduler, un obiec de tip SimulationFrame si o lista cu taskurile generate.

Metode implementate:

-metoda generate instantiaza obiectul de tip scheduler adica planificatorul, seteaza strategia si apeleaza metoda de generare a taskurilor;

-metoda generateNRandomTasks genereaza in mod aleator un numar de taskuri. Folosind un obiect de timp random generam pentru fiecare task timpul de sosire intre 0 si timpul limita si timpul de procesare intre minimul si maximul timpului de procesare apoi adaugam taskul in lista de taskuri si apelam metoda sort care sorteaza in mod crescator dupa timpul de sosire fapt implementat in metoda compareTo din clasa Task.

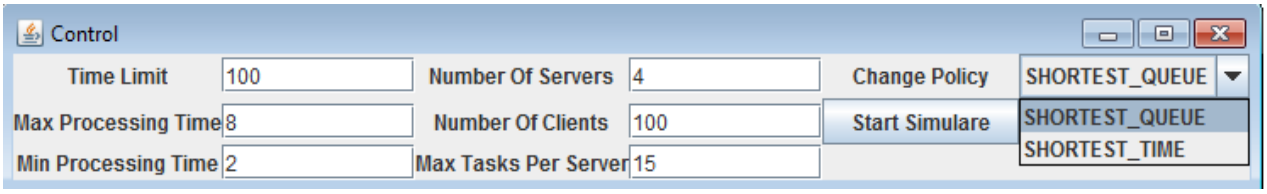
-metoda run mentine simularea atat timp cand timpul curent este mai mic decat cel limita sau listele nu sunt goale. Cu un iterator parcugem taskurile generate iar daca timpul de sosise al taskului este egal cu timpul curent apelam metoda dispatchTask pentru a distribui taskul in coada si apelam displayLog pentru afisa in interfa ca s-a introdus un task. Dupa ce toate taskurile au fost distribuite la timpul egal cu timpul curent afisam datele din fiecare server deschis. Daca timpul curent este egal cu timpul de terminare afisam cu ajutorul metodei displayLog ca avem un task iesit din coada. Apoi afisam timpul curent si incrementam timpul cu o unitate si stopam threadul pentru o secunda. La final putem afisa timpul mediu calculat ca si raport din suma perioadei de asteptare pe servere si numarul de clienti. Iar ora de varf reprezinta timpul in care s-a atins cel mai mare numar de taskuri aflate in stadiu de asteptare sau de procesare. In main este creat un obiect de tip SimulationControl care reprezinta prima interfata in care utilizatorul isi va alege datele de intrare pt simulare.

Clasa SimulationControl extinde clasa JFrame si implementeaza interfata ActionListener avand rolul de a permite utilizatorului sa aleaga timpul de simulare, timpul maxim si minim de procesare, numarul initial de servere, numarul de clienti, strategia aleasa si numarul maxim de taskuri ce pot fi distribuite pe un server. Am adaugat un ascultator la butonul de Start Simulare si am suprascris metoda

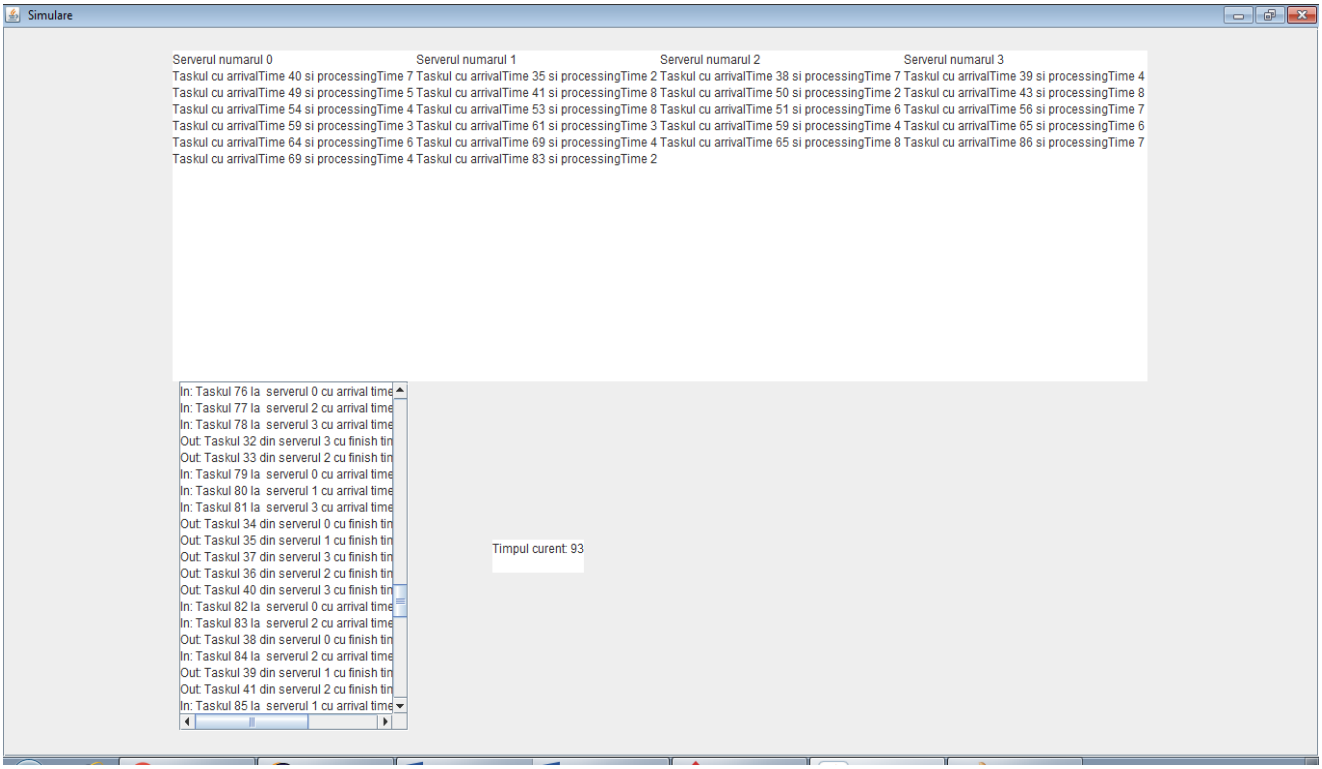
actionPerformed astfel incat atunci cand utilizatorul apasa Start Simulare sa fie setate datele de intrare prin intermediul setterelor si sa se genereze un obiect de tip SimulationManager care sa porneasca simularea.

5. Rezultate

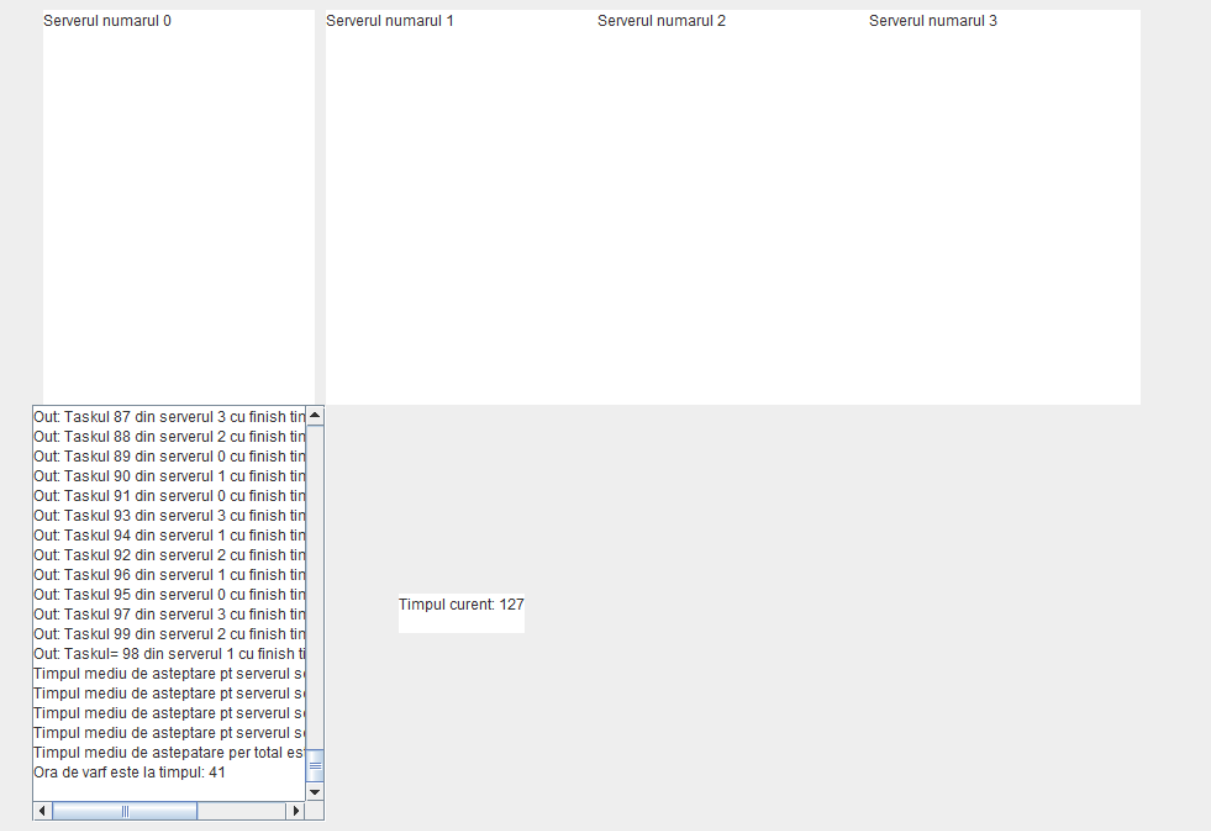
Cand rulam aplicatia utilizatorului i se va afisa o fereastra in care va putea introduce date.



Dupa ce a introdus datele in mod corect va putea apasa butonul Start Simulare care va porni simularea.



La final se va afisa tot in log file timpul mediu de asteptare si ora de varf.



6. Concluzii

In concluzie acest proiect implementeaza intr-un mod corect si eficient simularea unui sistem bazat pe cozi de asteptare minimizand timpul de asteptare.

La aceasta tema am invatat sa lucrez cu firele de executie(thredurile) si sa implementez o interfata care afiseaza modificarile in timp real.

Posibilitati de dezvoltare ulterioare:

- implementarea unor noi strategii pentru minimizarea timpului de procesare;
- aceasta aplicatie poate fi folosita ca si baza pentru proiecte mai mari care necesita implementarea cu ajutorul cozilor de asteptare.

7. Bibliografie

https://stackoverflow.com/
http://docs.oracle.com/javase/tutorial/uiswing/layout/visual.html
http://www.coned.utcluj.ro/~marcel99/PT/
http://www.coned.utcluj.ro/~salomie/PT_Lic/
http://www.tutorialspoint.com/java/util/timer_schedule_period.htm
https://www.javacodegeeks.com/2013/01/java-thread-pool-example-using-executors-and-threadpoolexecutor.html
http://ww1.javahash.com/