

**DOCUMENTATIE TEMA 2**

**SIMULATOR DE COZI**

**Realizat de: Voaideș Marian**

**Grupa: 30229**

**Profesor Laborator Asistent Antal Marcel**

Cuprins

1. Cerinte Functionale

2. Obiective

2.1 Obiectiv Principal

2.2 Obiective Secundare

3. Analiza probelemei

4. Proiectare

4.1 Diagrama de clase

4.2 Algoritmi

4.3 GUI

5. Implementare

6. Testare

7. Concluzii si Dezvoltari Ulterioare

8. Bibliografie

1. Cerinte Functionale

Obiectivul acestei teme se refera la crearea si implementarea unui simulator de cozi cu ajutorul limbajului de programare Java, acest simulator fiind bazat pe strategia de introducere a unui client in cadrul cozii la coada cu timpul de asteptare cel mai mic. In cee ace priveste datele de intrare utilizatorul simulatorului de cozi este rugat sa introduca de la tastatura un set de date de intrare pentru functionarea in mod normal al acestui proiect, aceste date de intrare sunt numarul clientilor ce vor fi generate in mod aleatoriu pentru simulare, numarul cozilor la care se vor aseza clientii din simulare, perioada de simulare, valoarea minima pentru timpul de sosire al clientului, valoarea maxima pentru timpul de sosire al clientului, timpul minim pentru efectuarea servirii clientului si timpul maxim pentru realizarea servirii clientului. Cerinta presupunea de asemenea inchiderea si deschiderea automata a cozilor, astfel incat la inceputul simularii toate cozile sa fie inchise, acestea urmand sa fie deschise in momentul in care primul client se aseaza la coada, iar apoi sa se inchida din nou in momentul in care nu mai sunt client ce trebuie sa fie procesati. In cadrul proiectului trebuia sa existe un “simulator de cozi” acesta avand rolul de a eficientiza procesul de simulare, in asa fel incat fiecare client sa fie directionat catre coada ce are timpul de asteptare cel mai mic dintre cozile existente. La finalul executiei simulatorului de cozi, rezultatele simularii trebuie sa fie afisate intr-un fisier text, iar la finalul lui se vor afisa ora de varf din cadrul simularii, timpul mediu de asteptare al clientilor la cozile generate si timpul mediu de servire al clientilor din cadrul simularii.

2. Obiective

**2.1 Obiectivul principal**

Cozile reprezinta o structura de date abstractata ce pot fi utilizate in numeroase scenarii, acestea avand scopul de a modela diferite domenii din lumea reala. Obiectivul principal al unei cozi este acela de a oferi un loc in interiorul unei cozi pentru un anumit „client” ce urmeaza sa astepte o perioada de timp inainte de a primi un „serviciu”. Fiecare coada poate fi privita ca un grup dintre casa de marcat si clientii ce sunt asezati la coada, acesta corespondenta poate fi vazuta prin prisma unui anumit aspect, fiecarei cozi i se atribuie clientii ce trebuie sa fie procesati, dar clientul este atribuit fiecarei cozi prin intermediul principiului ca timpul de asteptare in interiorul unei anumite cozi sa fie minim. Acest proiect are scopul de a simula o serie de clienti ce vor ajunge la cate o coda la un moment dat de timp, dupa sosire acestia trebuie sa astepte sa le vina randul in cadrul cozii, iar dupa ce ajung sa inceapa sa fie serviti si isi termina operatia de servire acestia trebuie sa iasa din interiorul cozii. Pentru efectuarea acestei operatii avem nevoie sa cunoastem momentul in care fiecare client soseste la coada si timpul de care are nevoie pentru a fi servit. In ceea ce priveste cozile din cadrul simularii este necesar sa cunoastem pentru fiecare dintre ele cat timp trebuie sa astepte un nou client ce a sosit la coada, pana in momentul in care clientul va fi servit. Acest timp poate varia in functie de doi parametri ci anume de numarul clientilor de la coada respectiva dar si de timpii acestora de servire.

3. Analiza problemei

Problema pusa in discutie consta in realizarea si implementarea eficienta si coerenta a unui simulator de cozi. In cadrul diagramei de cazuri vom avea drept actor principal si unic utilizatorul simulatorului de cozi. Preconditiile necesare pentru utilizarea corespunzatoare a simulatorului de cozi sunt introducerea valorilor necesare efectuarii simularii de catre utilizator. Conditiile de final se refera la realizarea cu succes a simularii si afisarea rezultatelor in timp real in cadrul interfetei, iar rezultatul total al simularii va fi stocat intr-un fisier text. Modul normal de desfasurare al acestui eveniment consta in introducerea corecta a datelor de intrare de catre utilizator, si in urma apasarii butonului de run se incepe executia propriu zisa a simularii, iar la finalul executiei utilizatorul poate apasa butonul de CE pentru a se goli text field-urile pentru ca utilizatorul sa poata introduca noi date de intrare pentru o noua simulare. Pe de alta parte pot aparea cazuri nefavorabile de utilizare, acestea constau in introducerea unor date gresite de la tastatura de catre utilizator (introducerea de litere nu de cifre, lasarea anumitor spatii goale, sau introducerea unor valori ce contravin logicii simulatorului, valorile minime sa fie mai mari decat cele maxime), in aceste cazuri programul simulatorului de cozi va functiona neconform asteptarilor.

4. Proiectare

**4.1 Diagrama de clase**

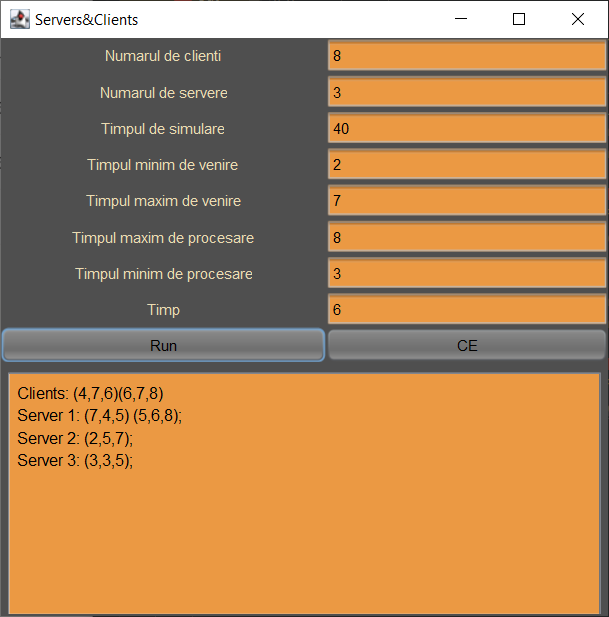
Diagrama UML (Unified Modeling Language) este o diagrama a carui rol este sa descrie relatiile intre clasele existente in proiect. Conceputa in faza initiala de implementarea in prealabil implementarii algoritmilor si sta ca suport in scrierea si crearea relatiilor dintre clase.

**4.2 Algoritmi**

Partea de algoritmica este implementata in clasele SimulationManager, Scheduler si in Server. In Simulation Manager dupa generarea celor n liste de clienti cu datele aleatorii si start-ul thread-urilor din servere in metoda run se va executa cod atata timp cat timpul de la inceputul executiei este mai mic decat timpul limita de executie. Daca se gaseste un task in lista celor n generate aleator care sa aiba timpul de executie egal sau mai mic (e tratat un caz exceptional in care mai multi clienti cu acelasi timp de sosire nu pot sa intre in toti in lista de asteptare la Servere). Dupa adaugarea in lista de asteptare a serverului cu cel mai mic timp de asteptare se task-urile vor fi sterse din lista de elemente generate aleator, vom incrementa contorul de timp si vom astepta o secunda. Functia WaitingTime calculeaza numarul total de secunde pe care un task trebuie sa il astepte pentru a intra in etapa de procesare, pentru aceasta vom insuma waitingPeriod-urile pentru fiecare task inainte de a se rula functia dispatchTask.. Functia calcPeakHour va masura ora de varf a serverelor, atunci cand numarul de clienti este maxim, comparand numarul de clienti in fiecare ora si actualizand varibila peakHour corespunzator. Functia finished va determina cand lista de cele n task-uri si lista task-urilor distribuite pe fiecare server s-au golit si incheia executia thread-ului ce ruleaza metoda run din clasa SimulationManager. Clasa Scheduler va asigura distribuirea corecta a task-urilor, masurand timpii de asteptare in fiecare server disponibil. Clasa Server va efectua procesarea efectiva a task-ului de fiecare server in parte.

**4.3 GUI**

Interfata cu utilizatorul.



5. Implementare

Pentru implementarea simulatorului de cozi s-au folosit clasele Task, Server, SimulationManager, Scheduler, View, Controller si Main.

Clasa Task, reprezinta clientul in sau obiectul de lucru in simulatorul de cozi. Ca variabile de instanta clasa Task contine campurile idTask, identificatorul de Task, sau cheia prima ce defineste in mod unic fiecare Task, arrivalTime, ce reprezinta momentul de timp la care va fi introdusa in lista de asteptare a serverului, moment relativ la momentul de start al firelor de exectutie ( thread-urilor ), processingTime o valoare absoluta, ce reprezinta timpul efectiv de procesare al task-ului din momentul in care a ajuns sa fie in fata cozii de asteptare pentru serverul in care a fost distribuit. Si finishing time momentul in care task-ul a fost procesat cu succes, reprezinta un numar natural relativ la momentul de start al firelor de executie ( thread-urilor ). Timpul de procesare va scadea progresiv pe durata in care este pozitionat in fata cozii serverului pe care a fost distribuit task-ul. Clasa Task de asemenea contine accesorii si mutatorii necesari procesarii. Clasa Task implementeaza interfata Comparable prin suprascrierea metodei compareTo necesara ordonarii Task-urilor in functie de timpul de sosire. Metoda toString ce afiseaza intr-un mod sugestiv datele caracteristice fiecarui task in parte sub forma tuplei: (id-ul Taskului, timpul de sosire, timpul de procesare).

Clasa Server contine o lista cu obiecte de tipul Task, reprezentand clientii ce urmeaza sa fie procesati si un intreg ce reprezinta timpul de asteptare pe server. Pe langa accesorii necesari functia contine constructorul ce initalizeaza variabilele de instanta la valorile lor implicite (lista goala si waitingPeriod la zero ). De asemenea metoda printServer concateneaza fiecare rezultatul toString-ului implementat in clasa Task, apelat fiecarui task in parte. Ultima metoda ce reprezinta partea functionala a serverului, sau procesorul efectiv de clienti, este metoda run rulata de fiecare thread lansat in clasa Scheduler. In acesta metoda se va extrage primul client din lista de asteptare si se va astepta un numar de secunde egal cu processingTime-ul clientului respectiv, timp in care waitingPeriod-ul (timpul de ateptare pe server) va fi decrementat deasemenea, dupa care clientul va fi scos din lista de asteptare a serverului.

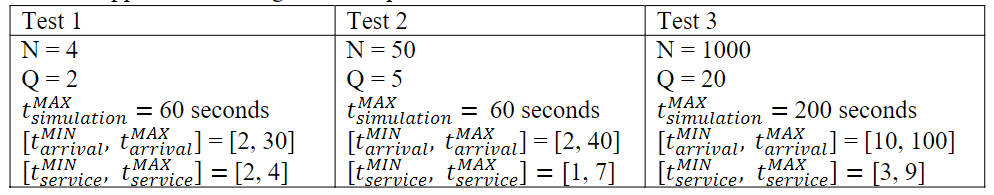
Clasa Scheduler, sau clasa planificator ea porneste executia thread-urilor pe fiecare server in parte si asigura distribuirea task-urilor fiecarui server. Clasa Scheduler contine variabilele de instanta servers, o lista de servere, reprezentand fiecare numarul de cozi de asteptare disponibile si campul maxTasksPerServer, reprezentand capacitatea maxima a fiecarei cozi in parte. Costructorul initializeaza lista de servere la valoarea implicita si porneste firul de executie al fiecarui thread. Pe langa accesori necesari functia Scheduler contine metoda dispatchTask ce cauta in lista serverelor serverul la care timpul de asteptare este minim. De asemenea verifica si ca serverul in care se va face distribuirea sa nu aiba capacitatea maxima depasita dupa introducerea task-ului ce trebuie distribuit.

Clasa SimulationManager contine ca variabile de instanta intregi ce reprezinta timpul minim de sosire al unui task relativ la inceperea firului de executie (thread-ului), timpul maxim de sosire al task-ului relativ la inceperea firului de executie (thread-ului), timpul limita de simulare, timpul minim de procesare, timpul maxim de procesare, numarul de servere, numarul de clienti ce trebuie distribuiti si procesati, timpul de asteptare total, suma timpilor de asteptare la coada al fiecarui client, numarul de maxim de clienti la ora de varf, ora de varf, deasemenea clasa mai ai campurile scheduler ce reprezinta planificatorul de task-uri si servere, o lista ce contine clientii generati in mod aleator conform regulilor impuse si o instanta a clasei view asigurand coeziunea intre functionalitati si partea grafica. Constructorul clasei instantiaza scheduler-ul, timeLimit, numberofClients, numberofServers, maxProcessingTime, minProcessingTime, minArrTime, maxArrTime, view-ul si apoi apeleaza functia de generare aleatoare a task-urilor. Metoda de generare a taskurilor avand campurile instantiate in constructor, construieste task-uri cu id-urile de la 1 la numberofClients si atribuie un numar aleator de moment de sosire si timp de procesare conform regulilor impuse in ipoteza. Metoda finished verifica momentul de timp in care toti clientii din lista de clienti generarti aleator, au fost procesati si returneaza un boolean corespunzator, de asemea scrie in fisierul log timpul mediu de asteptare la cozile serverului si ora de varf a serverului. Metoda calcPeakHour primeste ca argument momentul de timp curent al simularii si cauta ora de varf al serverelor comparand numarul de clienti maxim gasiti anterior cu numarul de numarul de clienti ce se afla in cozile de asteptare al fiecarui server in momentul de timp currentTime de la inceperea simularii. Metoda WaitingTime insumeaza timpii de asteptare ai clientilor la cozile de asteptare ale fiecarui server, memorand rezultatul in variabila de instanta WaitingTime.. Metoda createOutput concateneaza rezultate ce urmeaza a fi transmise si afisate in interfata grafica. Metoda suprascrisa run proneste simularea, prin variabila de instanta scheduler prin care distribuie fiecarui server un task, contorizeaza timpii si apeleaza metodele amintite mai sus: calcpeakHour, createOutput, finsished, se incrementeaza contorul de timp curent al simularii si se suspenda executia firului de exectutie pe o durata de o secunda. Tot in metoda run se scrie in fisierul log datele relevante simularii prin intermediul metodei appendFile ce deschide fisierul „log.txt” si prin intermediul buffer-ului bw se scriu datele.

Clasa view contine elementele de grafica, interfata cu utilizatorul prorpiu-zisa textfield-urile, butoanele si jtextarea, ramand ca rolul de tampon intre partea functionala descrisa in clasele anterioare si clasa view sa il preia clasa Controller ce potrivit comenzii introduse de utilizator si datelor introduse de acesta sa porneasca simularea.

6. Testare

Pentru testare s-au executat simulari conform celor impuse in ipoteza.



Fireste pentru ultimul test nu se pot procesa toate task-urile, fiecare queue avand in medie cate 50 de task-uri si fiecare task durand in medie 6 secunde ar trebuie cel in cazul cel mai bun 220 de secunde, insa in cazul de fata executia incepe de la secunda 10.

7. Concluzii si dezvaltari ulterioare

Simulatorul de cozi este implementat conform convetiilor de programare POO, iar functionalitatea acestuia simuleaza principiul server-client. Proiectul m-a ajutat sa imi aprofundezi cunostintele de java si sa consolidez noile notiuni legate thread-uri.

Ca si posibilitati de dezvoltare ulterioara s-ar putea reclicla thread-urile din clasa scheduler facand ca programul sa poata rula pe date diferite fara a fi nevoie de inchidere.

8. Bibliografie

- ASSIGNMENT\_2\_SUPPORT\_PRESENTATION.pdf

* <https://stackoverflow.com/>
* -https://www.guru99.com/java-swing-gui.html