MINISTERUL EDUCAȚIEI ȘI CERCETĂRII ȘTIINȚIFICE

UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

FACULTATEA DE AUTOMATICĂ ŞI CALCULATOARE
CATEDRA CALCULATOARE

FUNDAMENTAL PROGRAMMING TECHNIQUES

ASSIGNMENT 2

QUEUES MANAGEMENT APPLICATION USING THREADS AND SYNCHRONIZATION MECHANISMS

Moldovan Claudia
Grupa 30224
An universitar 2021-2022

MINISTERUL EDUCAȚIEI ȘI CERCETĂRII ȘTIINȚIFICE
UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA
FACULTATEA DE AUTOMATICĂ ŞI CALCULATOARE
CATEDRA CALCULATOARE

1.Cerință

Proiectați și implementați o aplicație de gestionare a cozilor care atribuie clienții la cozi astfel încât timpul de așteptare să fie minimizat.
Cozile sunt utilizate în mod obișnuit pentru a modela domeniile lumii reale. Obiectivul principal al unei cozi este de a oferi un loc pentru ca un „client" să aștepte înainte de a primi un „serviciu". Managementul sistemelor bazate pe cozi este interesat să minimizeze timpul pe care „clienții" lor îl așteaptă în cozi înainte de a fi serviți. O modalitate de a minimiza timpul de așteptare este să adăugați mai multe servere, adică mai multe cozi în sistem (fiecare coadă este considerată ca având un procesor asociat), dar această abordare crește costurile furnizorului de servicii.

Aplicația de gestionare a cozilor ar trebui să simuleze (prin definirea unui timp de simulare $tsimulation$) o serie de N clienți care sosesc la service, care intră în Q cozi, așteaptă, sunt serviți și în final părăsesc cozile. Toți clienții sunt generați la pornirea simularii și sunt caracterizați de trei parametri: ID (un număr între 1 și N), $tarrival$ (timp de simulare când sunt gata să intre în coadă) și $t$service intervalul clientul; adică timpul de așteptare când clientul se află în fața cozii). Aplicația urmărește timpul total petrecut de fiecare client în cozi și calculează timpul mediu de așteptare. Fiecare client este adăugat la coadă cu timpul minim de așteptare atunci când timpul său $tarrival$ este mai mare sau egal cu timpul de simulare ($tarrival$ ❑ $tsimulation$).

Următoarele date ar trebui considerate date de intrare pentru aplicație care ar trebui să fie introduse de utilizator în interfața de utilizator a aplicației:
- Numar de clienti (N);
- Numărul de cozi (Q);
- Interval de simulare ($tMAX$); $simulation$
- Ora minimă și maximă de sosire ($tMIN\ arrival$)
- Timp de service minim și maxim ($tMIN\ service$)

2.Teorie necesara

Pentru a realiza acest proiect a fost necesara aprofundarea cunostintelor despre thread-uri.
Un thread este un fir de execuție într-un program. Mașina virtuală Java permite unei aplicații să aibă mai multe fire de execuție care rulează simultan.
Fiecare fir are o prioritate. Firele cu prioritate mai mare sunt executate de preferință față de firele cu prioritate mai mică. Fiecare fir poate fi sau nu marcat ca demon. Când codul care rulează într-un fir de execuție creează un nou obiect Thread, noul fir de execuție are prioritatea setată inițial

egală cu prioritatea firului de creare a execuției și este un fir de execuție demon dacă și numai dacă firul de execuție care creează este un demon.

Când o mașină virtuală Java pornește, există de obicei un singur fir non-daemon (care numește de obicei metoda numită principal a unei clase desemnate). Mașina virtuală Java continuă să execute fire până când apare oricare dintre următoarele:

Metoda de ieșire a clasei Runtime a fost apelată și managerul de securitate a permis ca operația de ieșire să aibă loc.
Toate firele de execuție care nu sunt fire daemon au murit, fie prin întoarcerea de la apelul la metoda run, fie prin lansarea unei excepții care se propagă dincolo de metoda run.
Există două moduri de a crea un nou fir de execuție. Una este să declari o clasă ca fiind o subclasă a Thread. Această subclasă ar trebui să suprascrie metoda de rulare a clasei Thread. O instanță a subclasei poate fi apoi alocată și pornită.

3.Descrierea problemei

1. Introducerea datelor cerute in interfata aplicatiei: numar clienti, numar cozi, timpul maxim de simulare, timpul minim, respectiv maxim al sosirii clientului si intervalul de timp cat ar putea dura un serviu.
2. Apasarea butonului start pentru inceperea simularii aplicatiei si asteptarea executarii
3. Afisarea rezultatelor in timp real in consola
4. Afisarea rezultatelor in fereastra noua si accesarea acestora de catre utilizator

4.Implementare

Am realizat acest proiect prin intermediul modelului MVC. Asadar acesta cuprinde package-urile urmatoare:
1.Package mode:cuprinde clasa client(id, timp sosire si timpul de servire) si clasa coada (timpul de asteptare, daca este ocupata sau nu si starea- inchisa/deschisa);
2.Package orar: cuprinde clasa simulare (generare aleatorie clienti si simulare program) si clasa orar (modul de atribuire a clientilor la coada);
3.Package view: cuprinde clasa afisare(implementeaza metode de afisare), clasa interfata (partea strict de interfata unde utilizatorul adauga in textfield-uri si se foloseste de butoane) si clasa controller (implementeaza listener-uri pentru fiecare buton);
4. Package MVC: clasa main

In cele din urma voi descrie comportamentul fiecarei clase in parte.

- Clasa Client

Implementeaza conceptele de baza specifice unui client, si anume sunt prezentate atributele: id, timeArrival (timpul de sosire), timeService (timpul necesar efectuarii unui anumit serviciu). O metoda importanta este cea toString ce ne va ajuta in viitor cand vom dori sa afisam un anumit client, aceasta fiind apelata de metoda toString din clasa Coada. Astfel, clientul va fi afisat sub forma: (id, timeArrival, timeService); . Exista si metoda compareTo ce ne va ajuta ulterior pentru a putea sorta clientii in ordinea timpului de sosire.

- Clasa Coada

Implementeaza conceptele specifice unei cozi de clienti, continand astfel urmatoarele atribute: coada de clienti (BlockingQueue), timpul de asteptare specific (AtomicInteger), respectiv statusul cozii respective. Putem schimba statusul unei cozi prin apelarea metodei schimbareStatus si de asemenea o putem afisa prin metoda toStrong care, dupa cum am specificat anterior, se bazeaza pe metoda toString din clasa Client.

Spre deosebire de clasele realizate pana in momentul de fata, aceasta implementeaza interfata Runnable ceea ce ne obliga sa implementam si metoda specifica acesteia: run(). Aceasta sugereaza in sistemul de cozi numarul de clienti disponibili. Se va lua clientul care se afla in varful acesteia si il va procesa cat timp timpul de servire al acestuia este mai mare ca 0. Prin procesare se intelege punerea thread-ului pe sleep timp de o secunda in mod repetat care reprezinta perioada de asteptare in timp ce clientul este servit. Dupa fiecare perioada de timp (1 secunda) se vor executa urmatoarele operatii: se decrementeaza timpul de servire al clientului ce se afla in curs de procesare, se decrementeaza waitingPeriod. Daca timpul de servire ajunge sa fie mai mic sau egal cu 1, clientul va fi dat afara din stiva, fiind scazut astfel si timpul de asteptare. In momentul in care nu mai ramanem cu niciun client in coada, statusul acesteia se va schimba (din true un false).

- Clasa Orar

Este caracterizata de lista de cozi si numarul acestora. Se vor adauga atat de multe thread-uri in functie de numarul de cozi.

In aceasta clasa se gaseste metoda cu ajutorul careia este atribuit un anumit client unei anumite cozi. Se cauta prima coada ce are atatusul false (e inchisa), se retine numarul acesteia, urmand daca numarul ei este mai mic deat numarul de cozi dat la intrare, sa se execute metoda run prezentata in clasa Coada. In caz contrar, se va merge pe ramaura a doua a if-ului, unde se stabileste coada unde va fi trimis clientul in momentul terminarii serviciului anterior, avand cel mai mic timp de asteptare.

- Clasa Simulare

Implementeaza conceptele care ajuta la simularea intregii aplicatii, fiind caracterizata de atributele: orar, interfata, lista de clienti, locul unde se afiseaza si lista de string ce vor fi ulterior afisate in noua fereastra, timpul mediu de efectuare a unui serviciu si timpul mediu de asteptare al clientilor.

Metoda generareClienti, ne va adauga in listta de clienti random, valorile atributelor clientilor: timpul de sosire si cel de servire fiind generate cu ajutorul Random.nextInt(), valorile acestora fiind cuprinse intre timpul minim si maxim de sosire, respectiv timpul minim si maxim de servire introdus de catre utilizator. Initial, toti au acelasi index, anum 0. Dupa generarea aleatoare a celor doua atribute, lista de clienti va fi sortata in functie de timpul de sosire si se vor adauga indexii specifici fiecarui client.

Metoda principala, „cea mai importata", din intreg programul este run. Clasa Simulare, asemenea clasei Coada, implementeaza interfata Runnable. La inceputul metodei ,vom declara o variabila pentru a sti de fiecare data la ce moment de aflam, pentru a repeta intreg procesul pana cand aceasta variabila va ajunge la timpul de simulare dat de utilizator. In aceasta bucla, pentru fiecare coada se va rula metoda run. In momentul in care timpul simularii este egal cu timpul de sosire al unui client, clientul va fi pozitionat intr-o anumita coada si totodata si adaugat in lista clientilor ce vor urma sa fie extrasi din lista de clienti „In asteptare".

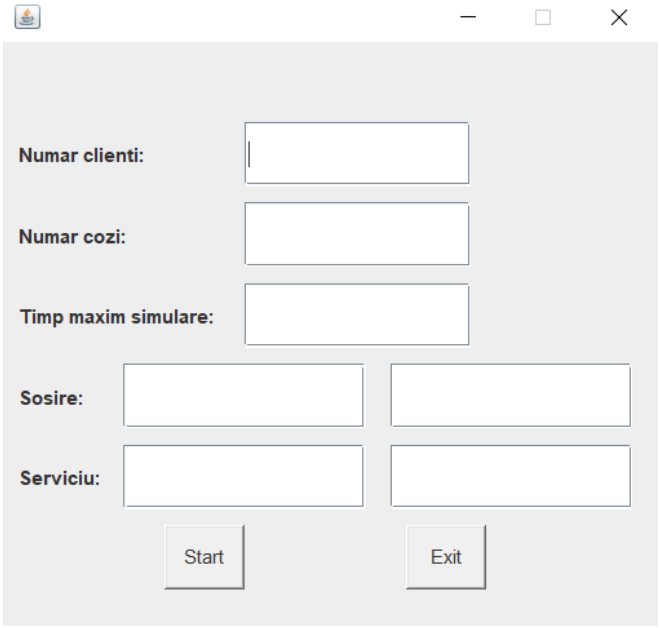
La fiecare perioada de timp se va itera peste lista de clienti si daca timpul de sosire al unui client este egal cu timpul curent de simulare acesta va fi adaugat in coada optima din acel moment cu ajutorul metodei atribuire din clasa Orar. Se vor printa toate valorile necesare, se va verifica daca mai exista clienti de procesat (se va iesi din while printr-un break daca toti au fost procesati) si la final se va pune sleep timp de 1 secunda thread-ul care reprezinta unitatea de timp necesara pentru a trece la urmatoarea perioada de timp.

- Clasa Afisare

Aceasta clasa ne ajuta sa printam rezultatele in fereastra deschisa in urma efectuarii simularii. Numele acesteia este dat prin intermediul unei variabile de tip String. In aceasta clasa, avem 2 metode: afisareFisier si transformString. Cea de-a doua metoda mentionata anterior, are ca paramentrii momentul, lista de clienti si lista de cozi. Astfel, vom putea afisa sub forma dorita, la fiecare moment de timp, parcursul simulatorului de cozi. Stringul rezultat in urma efectuarii acestei metode este adaugat in lista de stringuri din clasa Simulare. Se considera un string intreaga informatie de la un moment dat. Metoda afisareFisier, parcurge lista de stringuri formata in clasa Simulare prin apelarea dupa fiecare secventa a metodei transformString, scriind fiecare element in fereastra specificata.

- Clasa Interfata

Aceasta extinde JFrame. Aici sunt definite ca atribute private fiecare buton de care avem nevoie, fereastra propriu-zisa si textField-urile unde vom introducem parametrii simularii: numarul de clienti, numarul de cozi, timpul maxim de simulare, timpul minim, respectiv maxim de sosire si intervalul de timp in care se gasesc valorile duratei unui serviciu anume. Dupa ce setam dimensiunea, locul si numele fiecarui element, interfata va arata astfel:



Pentru fiecare buton din interfata este nevoie de ActionListener pentru a putea accesa butoanele si pentru a ne fi efectuate operatiile.

De asemenea in cazul in care in loc de numere reale vom pune litere o sa se afiseze mesajul “Date incorecte”.

- Clasa Controller

Are drept atribute Interfata I pentru a putea lega butoanele de ceea ce ar trebui sa faca. In constructorul acestei clase, sunt incluse si ActionListenerele pentru cele doua butoane: Exit si Start. In momentul in care este apasat Exit, se va iesi din aplicatie, pe cand in momentul in care se va apasa pe Start se va executa intregul program.

- Clasa Main

Conform structurii MVC, acest proiect a fost dezvoltat in jurul ei. Astfel, fiecare clasa se afla in pachetul sau corespunzator. Aceasta clasa instantiaza interfata, urmata de instantierea controllerului ceea ce face trimitere indirect la restul de clase si operatii pe care le-am folosit in dezolvatrea proiectului.

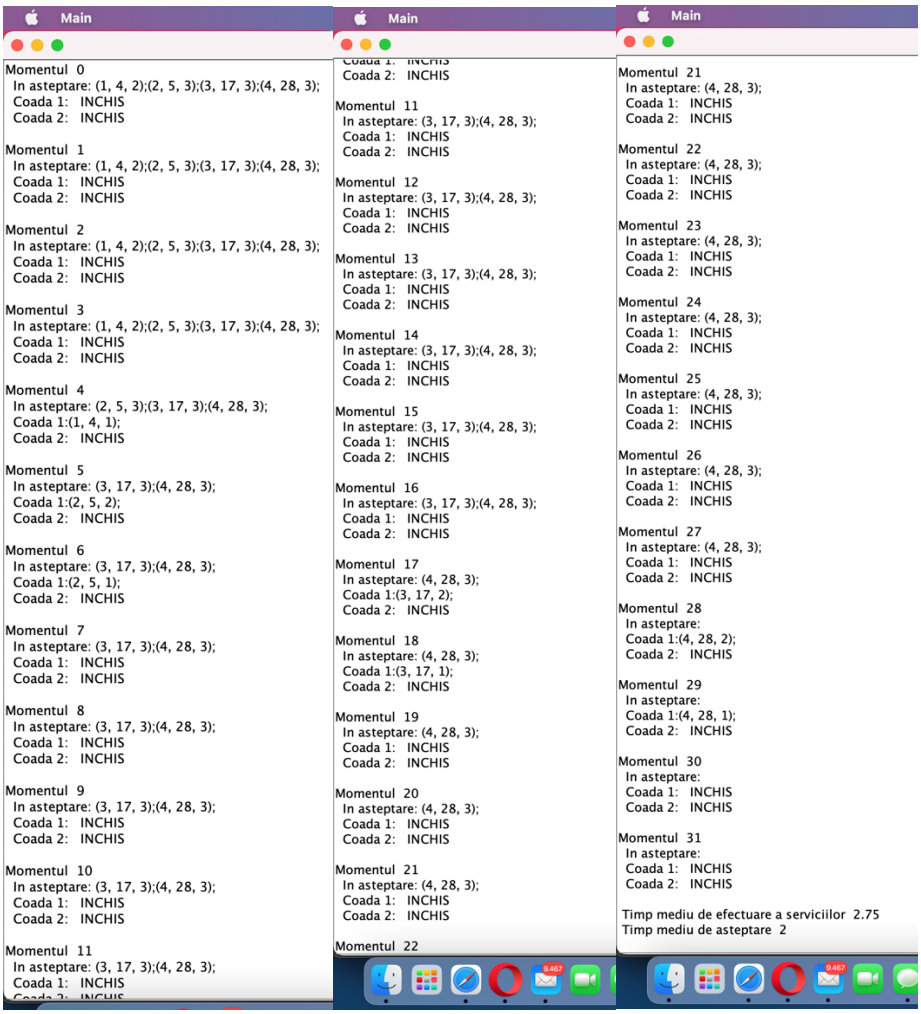
5.Teste

OBSERVATIE! Timpul de asteptare atunci cand este vorba de un timp lung sau de un numar mare de clienti/cozi depaseste timpul real.
OBSERVATIE! Programul urmareste doar strategia in care clientii sunt impartiti la coada unde timpul de asteptare este mai scurt.

Odata introduce datele penru test si apasat butonul de Start, prima data va incepe rularea programului in consola. In cazul in care clienti se epuizeaza mai repede decat timpul de simulare programul se va opri. La fel de intampla si in cazul in care timpul se termina inainte sa se termine clientii. In acest caz clientii vor ramane in lista de asteptare.

Voi da ca exmplu unul dintre cele 3 teste efectuate.

N =4
Q =2
$tMAX$ = 60 seconds $simulation$
[$tMIN$, $tMAX$] = [2, 30] $arrival\ arrival$
[$tMIN$, $tMAX$]= [2, 4]


Main
Momentul 0
In asteptare: (1, 4, 2);(2, 5, 3);(3, 17, 3);(4, 28, 3);
Coada 1: INCHIS
Coada 2: INCHIS
Momentul 1
In asteptare: (1, 4, 2);(2, 5, 3);(3, 17, 3);(4, 28, 3);
Coada 1: INCHIS
Coada 2: INCHIS
Momentul 2
In asteptare: (1, 4, 2);(2, 5, 3);(3, 17, 3);(4, 28, 3);
Coada 1: INCHIS
Coada 2: INCHIS
Momentul 3
In asteptare: (1, 4, 2);(2, 5, 3);(3, 17, 3);(4, 28, 3);
Coada 1: INCHIS
Coada 2: INCHIS
Momentul 4
In asteptare: (2, 5, 3);(3, 17, 3);(4, 28, 3);
Coada 1:(1, 4, 1);
Coada 2: INCHIS
Momentul 5
In asteptare: (3, 17, 3);(4, 28, 3);
Coada 1:(2, 5, 2);
Coada 2: INCHIS
Momentul 6
In asteptare: (3, 17, 3);(4, 28, 3);
Coada 1:(2, 5, 1);
Coada 2: INCHIS
Momentul 7
In asteptare: (3, 17, 3);(4, 28, 3);
Coada 1: INCHIS
Coada 2: INCHIS
Momentul 8
In asteptare: (3, 17, 3);(4, 28, 3);
Coada 1: INCHIS
Coada 2: INCHIS
Momentul 9
In asteptare: (3, 17, 3);(4, 28, 3);
Coada 1: INCHIS
Coada 2: INCHIS
Momentul 10
In asteptare: (3, 17, 3);(4, 28, 3);
Coada 1: INCHIS
Coada 2: INCHIS
Momentul 11
In asteptare: (3, 17, 3);(4, 28, 3);
Coada 1: INCHIS
Main
Coada 2: INCHIS
Momentul 11
In asteptare: (3, 17, 3);(4, 28, 3);
Coada 1: INCHIS
Coada 2: INCHIS
Momentul 12
In asteptare: (3, 17, 3);(4, 28, 3);
Coada 1: INCHIS
Coada 2: INCHIS
Momentul 13
In asteptare: (3, 17, 3);(4, 28, 3);
Coada 1: INCHIS
Coada 2: INCHIS
Momentul 14
In asteptare: (3, 17, 3);(4, 28, 3);
Coada 1: INCHIS
Coada 2: INCHIS
Momentul 15
In asteptare: (3, 17, 3);(4, 28, 3);
Coada 1: INCHIS
Coada 2: INCHIS
Momentul 16
In asteptare: (3, 17, 3);(4, 28, 3);
Coada 1: INCHIS
Coada 2: INCHIS
Momentul 17
In asteptare: (4, 28, 3);
Coada 1:(3, 17, 2);
Coada 2: INCHIS
Momentul 18
In asteptare: (4, 28, 3);
Coada 1:(3, 17, 1);
Coada 2: INCHIS
Momentul 19
In asteptare: (4, 28, 3);
Coada 1: INCHIS
Coada 2: INCHIS
Momentul 20
In asteptare: (4, 28, 3);
Coada 1: INCHIS
Coada 2: INCHIS
Momentul 21
In asteptare: (4, 28, 3);
Coada 1: INCHIS
Coada 2: INCHIS
Momentul 22
Main
Momentul 21
In asteptare: (4, 28, 3);
Coada 1: INCHIS
Coada 2: INCHIS
Momentul 22
In asteptare: (4, 28, 3);
Coada 1: INCHIS
Coada 2: INCHIS
Momentul 23
In asteptare: (4, 28, 3);
Coada 1: INCHIS
Coada 2: INCHIS
Momentul 24
In asteptare: (4, 28, 3);
Coada 1: INCHIS
Coada 2: INCHIS
Momentul 25
In asteptare: (4, 28, 3);
Coada 1: INCHIS
Coada 2: INCHIS
Momentul 26
In asteptare: (4, 28, 3);
Coada 1: INCHIS
Coada 2: INCHIS
Momentul 27
In asteptare: (4, 28, 3);
Coada 1: INCHIS
Coada 2: INCHIS
Momentul 28
In asteptare:
Coada 1:(4, 28, 2);
Coada 2: INCHIS
Momentul 29
In asteptare:
Coada 1:(4, 28, 1);
Coada 2: INCHIS
Momentul 30
In asteptare:
Coada 1: INCHIS
Coada 2: INCHIS
Momentul 31
In asteptare:
Coada 1: INCHIS
Coada 2: INCHIS
Timp mediu de efectuare a serviciilor 2.75
Timp mediu de asteptare 2