MINISTERUL EDUCAȚIEI ȘI CERCETĂRII ȘTIINȚIFICE

UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

FACULTATEA DE AUTOMATICĂ ŞI CALCULATOARE
DEPARTAMENTUL CALCULATOARE

Documentatie tema 2

la disciplina

Tehnici de programare

Queue management sistem

Ardeleanu Madalin-Florin

An academic : 2018 – 2019

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

Cuprins

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

1. Obiectivul temei

Cerinta:

Objective Design and implement a simulation application aiming to analyze queuing based systems for determining and minimizing clients' waiting time. Description Queues are commonly used to model real world domains. The main objective of a queue is to provide a place for a "client" to wait before receiving a "service". The management of queue based systems is interested in minimizing the time amount their "clients" are waiting in queues before they are served. One way to minimize the waiting time is to add more servers, i.e. more queues in the system (each queue is considered as having an associated processor) but this approach increases the costs of the service supplier. When a new server is added the waiting customers will be evenly distributed to all current available queues. The application should simulate a series of clients arriving for service, entering queues, waiting, being served and finally leaving the queue. It tracks the time the customers spend waiting in queues and outputs the average waiting time. To calculate waiting time we need to know the arrival time, finish time and service time. The arrival time and the service time depend on the individual clients – when they show up and how much service they need. The finish time depends on the number of queues, the number of clients in the queue and their service needs.

Input data: - Minimum and maximum interval of arriving time between customers;

-Minimum and maximum service time;

- Number of queues;

- Simulation interval;
- Other information you may consider necessary;

Minimal output:

- The average of waiting time, service time and empty queue time for 1, 2 and 3 queues for the simulation interval and for a specified interval (other useful information may be also considered);
- Log of events and main system data;
- Queue evolution;
- Peak hour for the simulation interval;

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

Pentru implementarea aplicatiei cerute, vom analiza cerintele date, iar ulterior vom analiza/studia tipul de programare “Programare Orientata pe Obiecte”, deoarece conceptele acestui tip de programare stau la baza implementarii acestei aplicatii. bazandu-ne pe aceste concepte, putem incepe dezvoltarea aplicatiei. Mai mult decat atat, vom folosi si timere, threaduri pentru a realiza concurenta.

Un scenariu in care ar putea fi folosit acest tip de aplicatie: existand un hiper-market, avand mai multe cozi, deoarece este populat in majoritatea timpului de un numar ridicat de persoane, fiecare client care isi termina cumparaturile va cauta sa se aseze la coada unde va astepta cel mai putin, acest lucru va depinde de numarul de clienti din coada respectiva si timpul de servire al fiecaruia. Astfel se putea calcula timpul total de asteptare pentru fiecare coada in momentul respectiv, clientul urmand sa fie introdus in coada unde va astepta cel mai putin pana va ajunge la casa,

ceea ce este echivalent cu introducerea acestuia in coada care are totalWaitingTime-ul cel mai mic.

Presupunem ca managerul magazinului tine o evidenta legata de eficienta casierilor.

Acest tip de aplicatie este folosit de utilizatorii care doresc realizarea simularea timpilor de asteptare pentru clienti la casa, eficienta casierilor, si mai multe aspecte legate de timpii de servire si finalizare.

3. Proiectare

➢ Decizii de proiectare

Pe plan intern, pentru a implementa aplicatia, am hotarat ca fiind potrivit ca un client sa aiba un nume, un timp de venire in coada si unul de servire, ambele fiind generate random, totodata, fiecare client are un timp de finalizare(cand iese de coada) si un timp de asteptare(pana ajunge la casa), fiecare dintre acesti 2 timp este calculat in momentul introducerii clientului in coada.

Fiecare coada din magazin poate sa contina unul sau mai multi clienti, la un anumit timp.

Asignand fiecarei cozi de clienti cate un thread, reusim astfel sa procesam clientii din fiecare coada simultan. Mai mult decat atat, folosind un timer, putem face acest lucru in timp real.

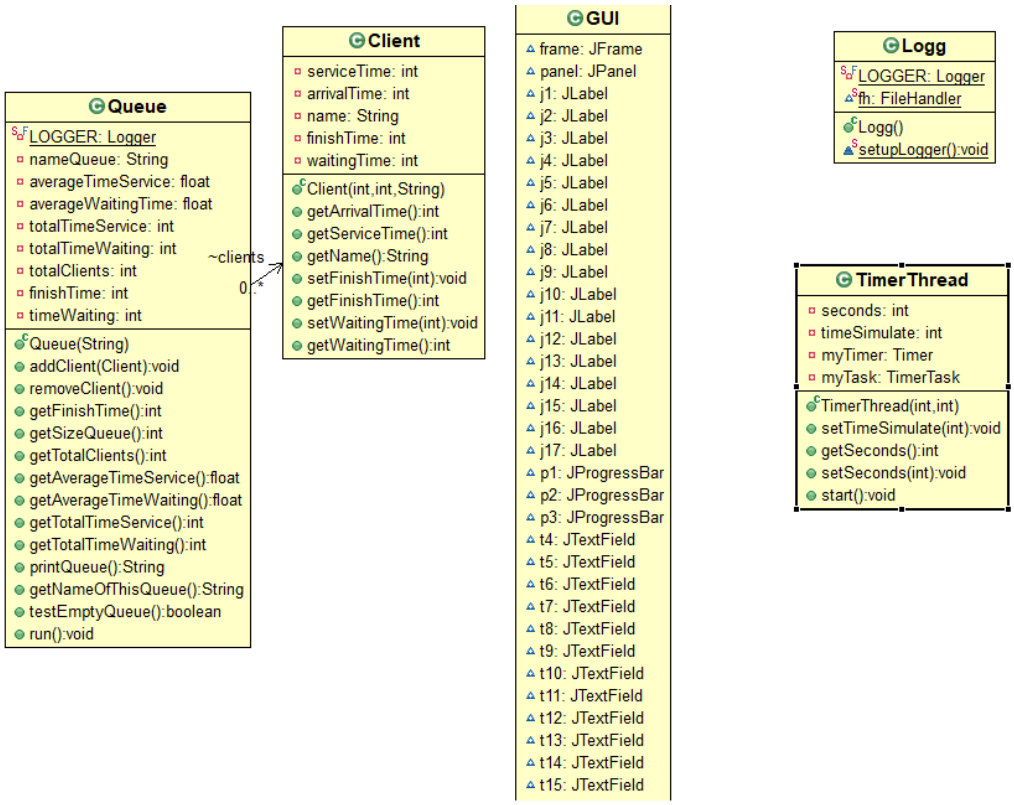
Clientii sositi la procesare, vor fi aseazati automat la casa care are timpul total de asteptare cel mai scurt in momentul respectiv.

Pe plan extern, interfata utilizator afiseaza in timp real situatia de la fiecare casa de marcat cu ajutorul unei animatii realizare cu un JprogressBar.

De asemenea , din interfata utilizator se fac setarile aplicatiei: timpul de rulare al aplicatiei, intervalul de generare al clientilor, intervalul timpulului lor de servire etc.

➢ **Diagrame UML**

UML este notaţia internaţională standard pentru analiza şi proiectarea orientată pe obiecte. Diagramele UML de clase sunt folosite in modelarea orientata obiect pentru a descrie structura statica a sistemului, modului in care este el structurat. Ofera o notatie grafica pentru reprezentarea: claselor - entitati ce au caracteristici comune relatiilor - relatiile dintre doua sau mai multe clase , Reprezentarea UML a claselor.



➢ **Structuri de date**

Am utilizat structuri de date de tip ArrayList, BlockingQueue si threaduri. O alternativa excelenta la tablourile unidimensionale precum : int numeVector = new int[100] este clasa ArrayList. Avantajele majore ale acestei clase este utilizarea mai buna a memoriei si administrarea mai usoara a elementelor listei.

De asemenea, un ArrayList poate contine Obiecte.

Adaugarea elementelor in lista se face foarte usor, prin apelareametodei .add. Primul element adaugat va ocupa pozitia 0, al doilea element adaugat va ocupa pozitia 1 si asa mai departe.

La parcurgerea listei cu ajutorul for-ului, ne-am folosit de metoda .size(),

metoda ce returneaza numarul de elemente al listei. Accesarea unui element se

face prin apelarea metodei .get().

Definitie BlockingQueue:

A Queue that additionally supports operations that wait

for the queue to become non-empty when retrieving an element, and wait for space to become available in the queue when storing an element.

Folosim acest tip de cozi pentru a nu genera erori daca la extragerea dintr-o coada, aceasta este goala, sau prea plina. Vom folosi peek() si remove().

➢ **Proiectare Clase**

In cadrul acestei aplicatii, intalnim cinci clase : Client, Queue, Logg, TimerThread, GUI si Test. Clasele Client, Queue si Test sunt clasele de baza ale aplicatiei.

Clasa Queue contine metoda de run() a threadurilor, iar aici se produce procesarea clientilor. Se extrage cate un client din BlockingQueue-ul de clienti, si se proceseaza in functie de timpul clientului de servire. In timpul procesarii, thread-ul cozii respective va fii pe sleep, in functie de timpul de servire al clientului. Dupa aceea, clientul respectiv este scos din Queue si se trece la procesarea urmatorului client.

In clasa Test, creeam cozile, threadurile, iar, in functie de un timer concurent, generam clienti care sunt introdusi in coada cu timpul de asteptare cel mai scurt. De asemenea, instantiem si un obiect al clasei TimerThread.

Cozile sunt primite din Test.

Clasa Logg ne ajuta sa tinem evidenta tuturor evenimentelor care au loc in magazin, in fiecare secunda, iar acestea vor fi afisare separat intr-un fisier txt, numit in cazul nostru file.

➢ **Algoritmi**

In continuare, vor fi explicati algoritmii de baza folositi in acest proiect:

• **run():**

Aceasta metoda din Queue reprezinta “treaba” pe care threadurile o executa. Cat timp cozile nu sunt goale, se va extrage cate un client din BlockingQueue-l cozii respective, il proceseaza(pune pe sleep threadul in functie de service time-ul clientului) si ulterior il scoate din coada. Astfel fiecare thread asignat unei case va procesa concurent fata de celelalte threaduri.

• **printQueue():**

Utilizam aceasta metoda din Coada pentru a afisa continutul cozii.

• **getToltalTimeWaiting():**

Folosim aceasta metoda pentru a avea acces la timpul total de asteptare pentru fiecare coada

• **getToltalTimeFinish():**

Folosim aceasta metoda pentru a avea acces la timpul total de finalizare pentru fiecare coada

• **getAverageTimeWaiting():**

Folosim aceasta metoda pentru a avea acces la timpul mediu de asteptare pentru fiecare coada

• **getAverageTimeService():**

Folosim aceasta metoda pentru a avea acces la timpul mediu de servire pentru fiecare coada

• **getTotalClients():**

Folosim aceasta metoda pentru a avea acces la numarul de clienti care au fost in fiecare coada

• **getSizeQueue():**

Folosim aceasta metoda pentru a avea acces la numarul de clienti din fiecare coada la momentul respectiv

• **getFinishTime():**

Folosim aceasta metoda pentru a avea acces la timpul de finalizare curent, folosit pentru generarea timpului de finalizare si asteptare pentru urmatorul client care intra in coada respectiva

• **removeClient():**

Scoate un client din coada

• **addClient():**

Adauga un client in coada

In clasa TimerThread am realizat un timer aproapiat de realitate, ce numara din secunda.

Genararea clientilor este realizata in clasa Test, unde mai putem gasii un array de string-uri care

va fi folosit pentru generarea unor nume random pentru clienti alaturi de timpul lor de intrare in coada si de cel de serire.

Tot aici mai avem si un ArrayList de threaduri de dimensiune 3, fiecare thread fiind folosit pe cate o coada, lucru ce determina “multithreading-ul”.

“Multithreading” înseamnă capacitatea unui program de a executa mai multe secvenţe de cod în acelaşi timp. “Multithreading” înseamnă capacitatea unui program de a executa mai multe secvenţe de cod în acelaşi timp.

Mai folosim si un ArrayList de intregi, unde vom stoca timpul de finalizare pentru fiecare client, acesta va fi folosit pentru stergerea clientiilor din coada, dupa urmatorul algoritm: Clientul ‘X’ a intrat in coada si isi va finaliza treaba dupa ‘y’ secunde, deci timpul sau de finalizare este y, daca timerul ajunge la secunda y, atunci clientul si-a terminat treaba si poate fi scos din coada.

Acest lucru face ca aplicatie sa simuleze un eveniment cat mai real precum in viata de zi cu zi.

Aceasta metoda reprezinta una din metodele principale ale proiectului. Cat timp ne incadram in intervalul de generare al clientilor introdus de utilizator, se vor genera clienti la un interval random de timp, interval dat tot de utilizator, la fel procedam si pentru timpul de servire al clientilor generati.

De asemenea, aici calculam si unele statistici care vor aparea la sfarsitul simularii in interfata grafica si anume: timpul mediu de servire al clientilor pentru fiecare coada, timpul mediu de asteptare al clientilor pentru fiecare coada si cate secunde este goala fiecare coada.

•actionPerformed():

Aceasta metoda din clasa Test o vom folosi pentru a afisa in timp real situatia din Magazin. Mai mult decat atat, dupa expirarea timpului de rulare al aplicatiei, vom afisa statisticile : peak time, average waiting time, queue empty time, etc.

➢ **Interfata utilizator**

In clasa GUI, avem constructia interfetei, butoanele necesare pentru inceperea aplicatiei si pentru preluarea datelor din text-box-uri, text-box-uri pentru citirea datelor de intrare si alte text-box-uri pentru afisarea statisticilor. De asemenea, avem si 3 JprogressBar-uri ce reprezinta evolutia celor 3 cozi in timp real, in timpul in care intra clientii in coada, bara de la ProgressBar incepe sa se umple putin cate putin iar odata ce clientii isi termina treaba si ies din coada, bara de la ProgressBar incepe sa scade, lucru destul de sugestiv si usor de priceput daca ma intrebari pe mine.

4. Implementare

➢ **Implementare**

La rularea aplicatiei, utilizatorul are acces la o interfata grafica care ii pune la dispozitie elementele necesare efectuarii simularii. El va putea introduce datele dorite in text-box-uri si anume: capatele intervalelor referitoare la timpul de venire al clientilor care este generat random, intervalul in care este cuprins timpul de servire al fiecarui client care este generat tot random si intervalul de simulare al aplicatiei.

Dupa aceea va apasa pe butonul denumit “Buton” pentru preluarea datelor de intrare si inceperea simularii.

In functie de datele de intrare, el va putea vizualiza in timp real continutul cozilor si alte date.

Timpul scurs pana la inceperea aplicatiei va putea fi vizualizat in consola.

5. Rezultate

Am obtinut astfel o implementare a cerintei proiectului , punand in practica simularea unui sistem de cozi, afisand in timp real continutul acestora si alte date utile unei statistici. Daca interfata este utilizata in mod corect, cum este specificat mai sus, rezultatele generate sunt corecte si utile pentru utilizator.

6. Concluzii, dezvoltari ulterioare

Realizand aceasta tema, am deprins o mai buna aptitudine in a rezolva algoritmi pe obiecte si am aprofundat concepte legate de structuri de date de tip BlockingQueue, cat si threaduri. De asemenea, am invatat cum functioneaza concurenta in Java. Ca o posibila dezvoltare ulterioara, aplicatiei ii mai pot fi aduse unele adaugiri/imbunatatiri. De exemplu:

- adaugarea unor animatii mult mai complexe reprezentand clientii
- afisarea in timp real a timpului de asteptare la cozi.

7. Bibliografie

https://moodle.cs.utcluj.ro/pluginfile.php/31664/mod_resource/content/2/POO10.pdf

https://moodle.cs.utcluj.ro/pluginfile.php/31667/mod_resource/content/2/POO11.pdf

https://stackoverflow.com/