

Tehnici de Programare Fundamentale

-Queues Simulator-

Documentatie

Nume: Souca Tania-Carina

Grupa: 30223

1. Obiectivul temei:

Obiectivul temei il reprezinta proiectarea si implementarea unei aplicatii ce gestioneaza un numar dat de cozi cu clienti ce vor sa foloseasca un server.

2.Analiza problemei:

Clientul citeste datele dintr-un fisier txt sau din interfata grafica. Sistemul verifica daca datele coincid cu cele introduse de problema.

In caz afirmativ, se simuleaza evolutia in timp a cozilor dupa generarea clientilor si asignarea la o coada. Prima data se vor deschide toate serverele, dupa care clientii se vor aseza pe rand pana se vor epuiza toate casele. In momentul in care toate casele sunt pline, clientii vor astepta sa se actualizeze timpul minim de procesare pentru a stii la care dintre servere sa se asigneze mai departe.

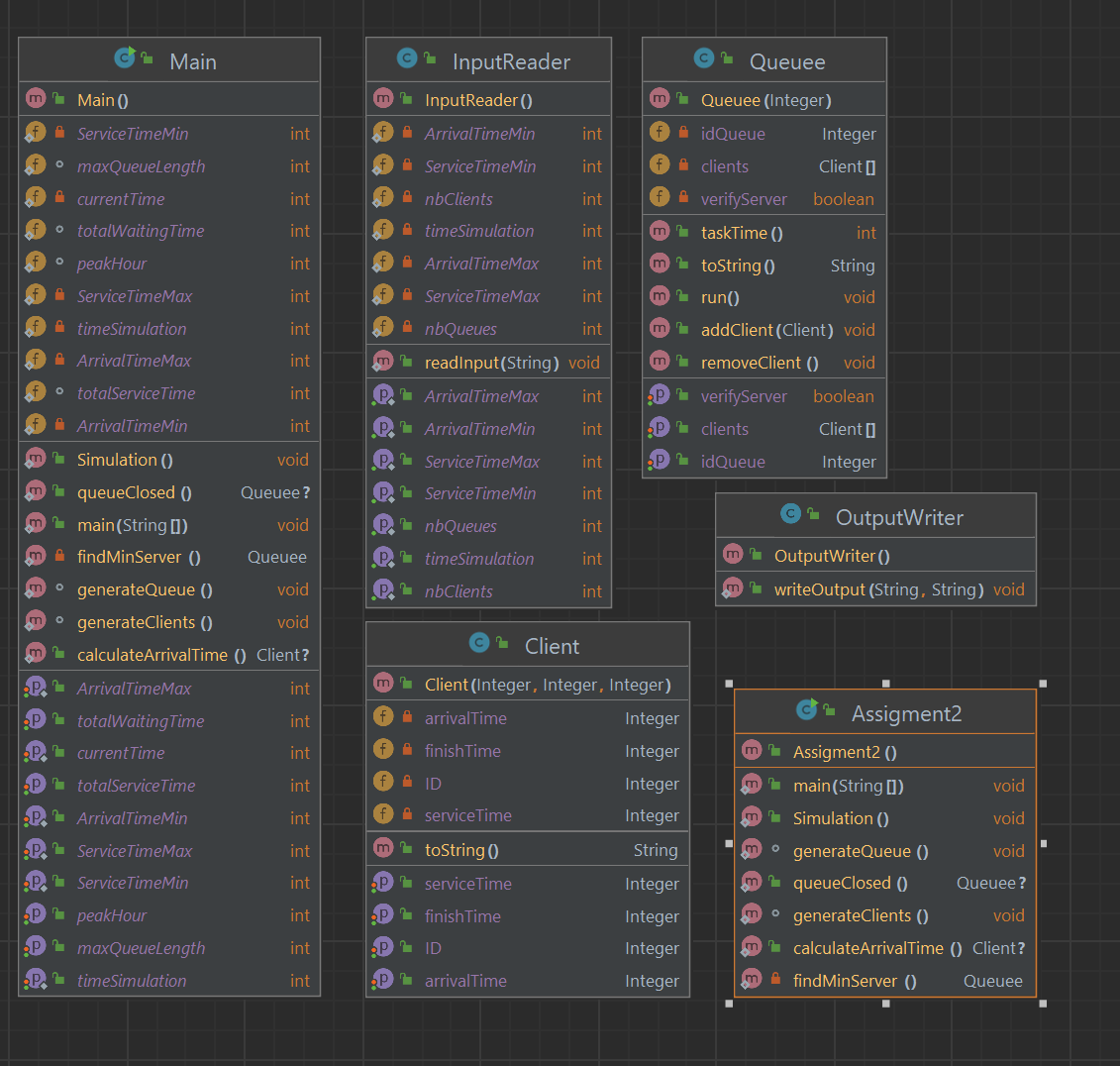
In acelasi timp, sunt afisate date despre generarea clientilor si despre parcursul lor in timp.

La finalul simularii se vor afisa rezultatele despre aceasta simulare fie intr-un fisier txt fie in interfata grafica.

3.Proiectare:

În acest capitol, vom prezenta pe larg notiunile de proiectare luate în concordanță cu paradigma programării orientate pe obiecte, vom expune diagramele UML de clase, structura ierarhică a proiectului și strucuturile de date folosite.

Aplicația este structurată în mai multe clase, cele principale fiind clasele: Client, Queue, InputReader, OutputWriter si Main, in cazul in care vrem sa folosim fisierele txt, iar pentru folosirea interfetei grafice avem in plus clasa Assigment2. Sunt implementate atât structurile de date în care stocăm informația despre clienți, respectiv cozi cat si despre generearea și repartizarea acestora în simulare.



**Clasa Client** are atribute pentru id-ul clientului, arrivalTime-ul, serviceTime-ul si finishTime-ul,toate fiind niste numere intregi. Metodele implementate în această clasă sunt gettere-le si settere-le tuturor atributelor precum si metoda toString,care are rolul de a aranja atributele clientului.

**Clasa Queue** are atribute pentru verificarea casei, fiind o valoare de tip boolean, pentru id-ul cozii, valoarea fiind de tip intreg si totodata mai contine si un vector de tip client .

* Metoda addClient(Client newClient) realizeaza adaugarea unui client intr-o coada.
* Metoda removeClient() stergere primului element din coada.
* Metoda run() verifica daca exista clienti in coada, daca exista il gaseste pe primul dupa care il va servi si apoi il va elimina din coada.
* Metoda taskTime() indeplineste rolul de a calcula timpul de procesare al tuturor clientilor din coada.
* Metoda toString() indeplineste rolul de a afisa informatiile referiroare la coada.

**Clasa Main** reprezinta thread-ul principal in aceasta aplicatie, urmat de inca doua thread-uri acestea fiind cele 2 cozi evidentiate in enunt si are la baza urmatoarele metode:

* Metoda generateClients() care are rolul de a genera un anumit numar de clienti pe baza a niste timpi(arrivalTime,serviceTime) dati random.
* Metoda generateQueues() care are rolul de a genera un anumit numar de cozi(numarul fiind specificat in enunt).
* Metoda ArrivalTimeMinim() indeplineste rolul de a calcula cel mai mic arrivalTime al clientilor care se gasesc in lista de asteptare.
* Metoda queueClosed() cauta daca exista vreo casa inchisa si in caz afirmativ o va returna, iar in cazul in care nu o gaseste va returna null.
* Metoda findMinServer() cauta in lista de cozi casa care pune in evidenta cel mai mic timp de procesare, coada la care clientul va avea de asteptat cel mai putin.
* Metoda Simulation() este cea mai importanta metoda din acest program intrucat ea coordoneaza intreaga aplicatie, ea urmareste ca toti clientii sa fie plasati la locul lor.

**Clasa InputReader** si **OutputWriter** ne ajuta sa citim datele, respective sa scriem rezultatul intr-un fisier.

Daca dorim sa folosim interfata grafica vom folosi **Clasa Assigment2** unde vom introduce datele de intrare intextField-urile speciale, iar outputul se va afisa dupa apasarea butonului “OK”.

4.Rezultate

Întrucât în acest proiect clienții sunt generați aleator, vom prezenta un caz de utilizare al aplicației.

Vom citi inițial valori din fisierul de intrare , dupa care vom pune in evidenta faptul ca aceasta aplicatie functioneaza pe baza anumitor timpi si anume: trebuie sa gasim o valoare random care se afla in intervalul de arrivalTime, precum si o valoare care se afla in intervalul de serviceTime. Asadar, pe baza celei mai mici valorii de arrivalTime, clientii nostrii vor fi asezati la coada pentru a fi serviti.

In cele din urma clientii urmatori se vor aseza la casa cu cel mai mic timp de procesare,metoda bazata pe serviceTime.

Aceasta aplicatie va continua atata timp cat timpul curent va fii mai mic decat timpul de simulare citit din fisier.

Asadar pentru acest input :

4 nbClients  
2 nbQueues  
15 SimTime  
2,15 MinMaxArrivalTime  
2,4 MinMaxServiceTime

Vom avea acest output ca exemplu:

Time: 1  
Waiting clients: [(2, 8, 3), (1, 9, 4), (4, 10, 3), (3, 11, 4)]  
Queue1: CLOSED!  
Queue2: CLOSED!  
Time: 2  
Waiting clients: [(2, 8, 3), (1, 9, 4), (4, 10, 3), (3, 11, 4)]  
Queue1: CLOSED!  
Queue2: CLOSED!  
Time: 3  
Waiting clients: [(2, 8, 3), (1, 9, 4), (4, 10, 3), (3, 11, 4)]  
Queue1: CLOSED!  
Queue2: CLOSED!  
Time: 4  
Waiting clients: [(2, 8, 3), (1, 9, 4), (4, 10, 3), (3, 11, 4)]  
Queue1: CLOSED!  
Queue2: CLOSED!  
Time: 5  
Waiting clients: [(2, 8, 3), (1, 9, 4), (4, 10, 3), (3, 11, 4)]  
Queue1: CLOSED!  
Queue2: CLOSED!  
Time: 6  
Waiting clients: [(2, 8, 3), (1, 9, 4), (4, 10, 3), (3, 11, 4)]  
Queue1: CLOSED!  
Queue2: CLOSED!  
Time: 7  
Waiting clients: [(2, 8, 3), (1, 9, 4), (4, 10, 3), (3, 11, 4)]  
Queue1: CLOSED!  
Queue2: CLOSED!  
Time: 8  
Waiting clients: [(1, 9, 4), (4, 10, 3), (3, 11, 4)]  
Queue1: (2, 8, 3)  
Queue2: CLOSED!  
Time: 9  
Waiting clients: [(4, 10, 3), (3, 11, 4)]  
Queue1: (2, 8, 2)  
Queue2: (1, 9, 4)  
Time: 10  
Waiting clients: [(3, 11, 4)]  
Queue1: (2, 8, 1)  
Queue2: (1, 9, 3)(4, 10, 3)  
Time: 11  
Waiting clients: []  
Queue1:   
Queue2: (1, 9, 2)(4, 10, 3)(3, 11, 4)  
Time: 12  
Waiting clients: []  
Queue1:   
Queue2: (1, 9, 1)(4, 10, 3)(3, 11, 4)  
Time: 13  
Waiting clients: []  
Queue1:   
Queue2: (4, 10, 3)(3, 11, 4)  
Time: 14  
Waiting clients: []  
Queue1:   
Queue2: (4, 10, 2)(3, 11, 4)  
Time: 15  
Waiting clients: []  
Queue1:   
Queue2: (4, 10, 1)(3, 11, 4)

5.Concluzii

In concluzie ,aceasta tema intitulata “QUEUES SIMULATOR” si prezentată impune cunoașterea și aplicarea noțiunilor din paradigma programării orientate pe obiect. Așadar, rezolvarea cerinței permite aprofundarea și dezvoltarea capacităților de a scrie cod Java, identificarea cauzelor și tratarea problemelor ce pot apărea în procesul de implementare.

Această temă este importantă deoarece utilizează concepte din programarea concurentă și anume lucrul cu firele de execuție ce permit programelor să execute mai multe secvențe de cod în același timp. Totodată, această problemă stă la baza modelării proceselor din viața reală și rezolvarea acesteia oferă un avantaj atât în evidențierea trăsăturilor programării concurente, cât și în abstractizarea cât mai eficientă a problemelor din viața reală pentru implementare software.

6.Bibiliografie

<https://www.geeksforgeeks.org/java/>