QUEUE SIMULATOR/SIMULAREA COZILOR UNUI MAGAZIN

Stefan Tudor

Grupa 30229

Profesor laborator: Mitrea Dan

Cuprins:

1.Cerinte functionale………………………………………………

2.Obiective………………………………………………………...

3.Proiectare………………………………………………………..

4.Implementare……………………………………………………

5.Rezultate………………………………………………………...

6.Concluzii………………………………………………………..

7.Bibliografie……………………………………………………..

**1.CERINTE FUNCTIONALE**

In aceasta tema se cere implementarea unui simulator de cozi ale unui magazin(Queue Simulator).Se foloseste metoda multithreading (un thread/coada) pentru procesarea clientilor magazinului.Se va folosi un thread principal care este responsabil de pornirea celorlalte thread-uri.

Programul se va rula din linia de comanda cu formatul:

**java -jar <calea spre fisierul.jar/fisierul .jar> In-TestX.txt Out-TestX.txt** ( X poate sa aiba valoarea 1,2 sau 3 avand in vedere ca se vor testa 3 situatii in care se poate afla magazinul).

Din fisierele In-TestX.txt se vor citi datele necesare simularii magazinului, iar in Out-TestX.txt se va descrie situatia magazinului. Acest lucru este realizat prin descrierea cozilor (a starii acestora) si a starii clientilor( timpul la care acestia ajung la coada si timpul lor propriu de procesare, care scade in timp ce clientul este procesat de coada).

Vor fi generati N clienti aleatori care vor fi adaugati la numarul de cozi existente in functie de timpul de ajungere (arrivalTime). Prin procesarea acestora timpul prorpiu de procesare va scadea cu o unitate.Cand acest “processTime” va ajunge la 0, clientul respective va fi scos din coada facand loc pentru un nou client.

**2.OBIECTIVE**

Obiectivul principal al acestei teme este de a implementa un simulator de cozi ale unui magazin prin folosirea thread-urilor.

***Obiectivele secundare*** si atingerea lor realizeaza indeplinirea obiectivului principal.

Aceste obiective secundare sunt:

* Intelegerea functionarii thread-urilor
* Alegerea structurilor de date necesare
* Impartirea claselor
* Dezvoltarea unor anumiti algoritmi ce pot ajuta la o buna functionare a programului
* Implementarea unei solutii
* Testarea programului

**3.PROIECTARE**

Ca si prim pas al proiectarii am ales sa imi impart proiectul in acest fel: O clasa principala (SimManager), 3 clase secundare: Queue,Scheduler,Client si o clasa auxiliara pentru citirea din fisiere: ReadFile.

**Clasele** programului au fost proiectate astfel:

Clasa “Client” reprezinta clientul unui magazine cu atributele sale ID, arrivalTime si processTime.

Clasa “Queue” reprezinta o coada de magazine care contine o structura de date de tipul Collections : LinkedBlockingQueue<Client> la care se vor adauga client in cazul in care acestia au arrivalTime egal cu timpul curent al simulatorului.

Clasa “Scheduler” decide atunci cand un client trebuie sa fie introdus in coada cu timpul minim de asteptare.De asemenea, aceasta clasa “Scheduler” contine o structura de date de tipul Collections: ArrayList<Queue> in care sunt pastrate toate cozile magazinului, impreuna cu clientii acestora. Tot prin intermediul acestei clase se pornesc thread-urile reprezentate de cozi ( clasa “Queue” implementeaza interfata “Runnable”).

Clasa “SimManager”, clasa principala a acestui program, face posibila functionarea corecta a implementarii. In interiorul acestei clase sunt generati ce N client aleatori si de asemenea vor fi pornite cozile ( thread-urile). Initial acestea sunt considerate inchise pana ce primul client va fi adaugat. Aceasta clasa imbina functionalitatile claselor secundare cat si a celei auxiliare. Metodele si modul in care clasa “SimManager” realizeaza cele mentionate anterior vor fi descrise la capitolul de “Implementare” al acestui proiect.

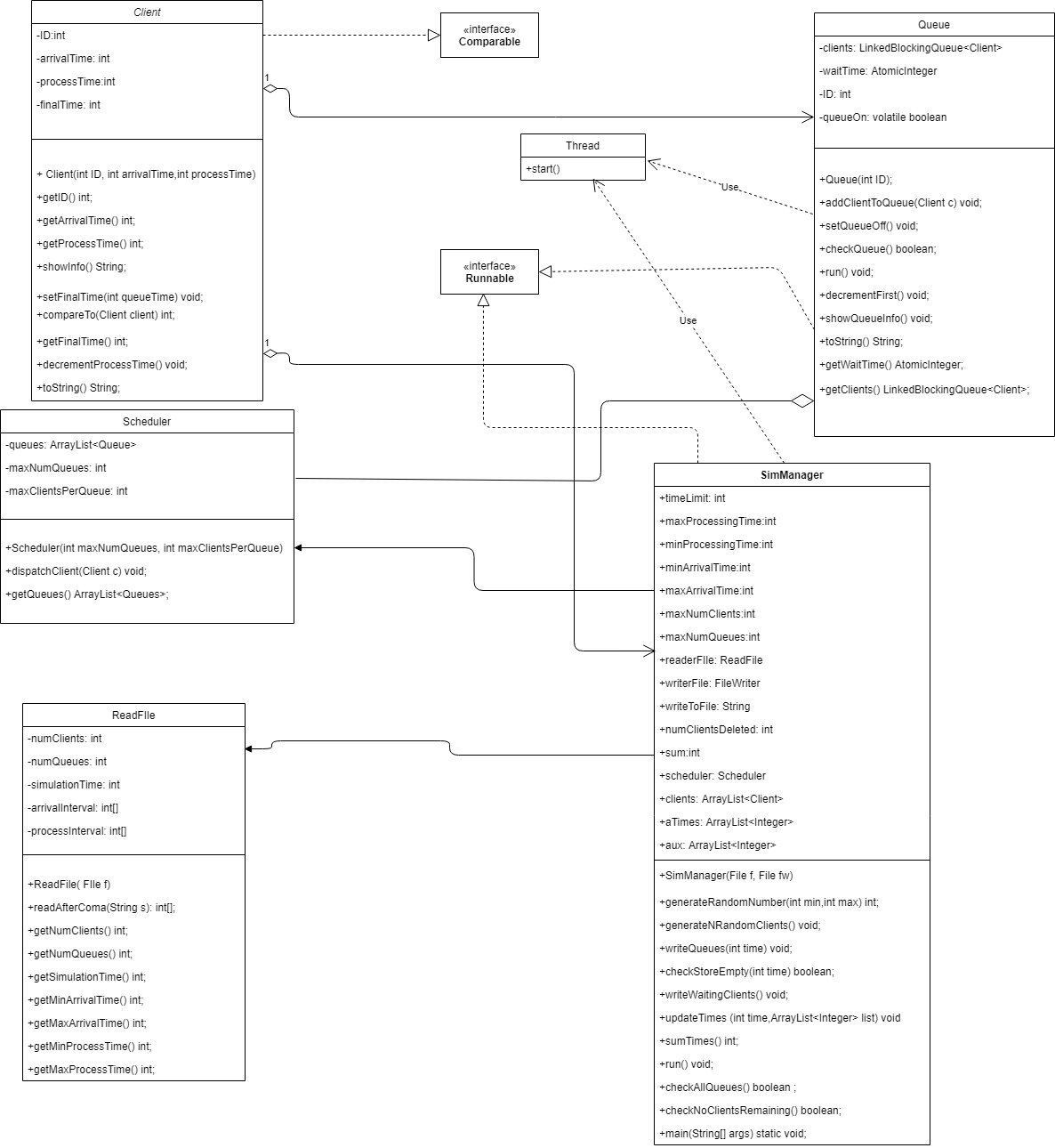
Clasa “ReadFile” este, dupa cum a fost mentionat anterior, o clasa auxiliara.Aceasta face posibila citirea datelor unui magazin din fisierele cu format In-TestX.txt .

**Pachetele** folosite pentru implementare acestui proiect sunt “java.util” si “java.io”.

**Interfetele** folosite in acest proiect sunt “Comparable” si “Runnable”.

**Diagrama UML:**

In continuare va fi prezentata diagrama UML a acestui proiect. In aceasta vor fi prezentate relatiile dintre clase si interfete.



**4.IMPLEMENTARE**

In aceasta parte vor fi descrise implementarile claselor si a metodelor acestora.Clasele ce vor fi descrise sunt: “SimManager”,”Client”,”Queue”,Scheduler” si “ReadFile”.

Clasa **“Client”:**

* Majoritatea metodelor acestei clase sunt metode de tip getter si setter, insa apar metode precum :
  + - * Metoda compareTo(Client client)
      * Metoda decrementProcessTime() in care se decrementeaza timpul de procesare al clientului respectiv.

Clasa **“Queue”:**

* Metoda **addClientToQueue(Client c)**: adauga un client la LinkedBlockignQueue<Client> si seteaza variabila volatila queueOn pe true.De asemenea se incrementeaza timpul de asteptare cu timpul de procesare al clientului adaugat.
* Metoda **checkQueue()** verifica daca LinkedBlockingQueue<Client> este goala, in caz afirmativ aceasta returneaza o variabila booleana cu valoarea “true”.
* Metoda **run()** va rula odata ce thread-ul este pornit.In aceasta metoda se afla o un loop “while” care ruleaza cat timp variabila volatila “queueOn” este “true”.In interior se verifica daca coada respectiva este goala.In caz afirmativ aceasta variabila va fi setata pe false.In caz contrar, thread-ul va fi pus pe “sleep” pentru un timp egal cu cat ii va lua primului client din coada sa fie procesat.Timpul de asteptare va creste cu timpul de procesare al primului client.
* Metoda **decrementFirst()** va decrementa timpul de procesare al primului client aflat la coada.
* Metoda **toString()** verifica daca exista clienti in coada.Acest lucru se face prin verificarea capului cozii.In caz ca acesta este “NULL” , atunci String-ul “result” va lua valoarea : “Queue” + ID-ul cozii + “closed”.In caz contrar se va scrie in “result” ID-ul cozii impreuna cu clientii care se afla la aceasta.

Clasa **“Scheduler”** :

* Constructorul clasei : **Scheduler(int maxNumQueues,int maxClientsPerQueue)**: in acest constructor se vor crea maxNumQueues cozi si se vor adauga la ArrayList<Queue>.Apoi se va crea cate un thread pentru fiecare coada adaugata si se va porni acel thread.
* Metoda **dispatchClient(Client c)** : primeste ca parametru un obiect de tipul Client.Se va parcurge ArrayList<Queue> si se va cauta coada cu timpul minim de asteptare.Odata ce s-a gasit aceasta coada, clientul pe care dorim sa il adaugam, va fi introdus in respectiva coada.

Clasa **“SimManager”**:

* In interiorul acestei clase va avea loc simularea prorpiu-zisa a magazinului.Constructorul acestei clase **SimManager(File f,File fw)** primeste ca parametrii doua fisiere: unul pentru citire si unul pentru scriere.In interior vor fi generate “maxNumClients” clienti si vor fi adaugati la un ArrayList<Client>.De asemenea vor fi parcurse cozile si se vor porni thread-urile reprezentate de catre cozi.La final se va folosi un ArrayList<Integer> pentru a memora “arrivalTime” al fiecarui client.Aceasta structura de date va fi folosita ulterior pentru calcularea timpului mediu de asteptare in magazin.
* Metoda **writeQueues(int time):** prin intermediul acestei metode se vor scrie starile cozilor la un timp T al simularii.Se parcurg cozile din “Scheduler” si ca prim pas vor fi scrise intr-un String starile acestora.Apoi se va verifica daca exista elemente in capul cozii.In caz afirmativ se va decrementa timpul de procesare al primului client si se va verifica de asemenea daca timpul de procesare al acestuia a ajuns la 0.Daca s-a ajuns la 0, atunci numarul clientilor ce vor fi stersi din list ava creste cu o unitate.Mai departe se va face un update la timpii din lista in care s-au pus timpii la care au ajuns clientii in magazin.In cele din urma clientul va fi eliminat din LinkedBlockingQueue<Client>, facand loc pentru clientul urmator.
* Metoda **checkStoreEmpty(int time)** : verifica daca toate cozile sunt goale si daca magazinul respective mai are sau nu clienti.In cazul in care se confirma cele doua conditii, se va calcula timpul mediu de asteptare.Acest lucru se face prin impartirea rezultatului returnat de functia **sumTimes(),** care aduna valorile din lista **aTimes** , la numarul de clienti care au fost procesati/stersi din lista de clienti (numClientsDeleted).
* Metoda **updateTimes(int time,ArrayList<Integer> list) :** se va parcurge lista cu timpii “arrivalTime” si fiecare timp va fi egal cu rezultatul scaderii dintre timpul curent si timpul respectiv din lista, adunat mai apoi cu o unitate de timp.Rezultatul acestei operatii va fi apoi adaugat intr-o list auxiliara.
* Metoda **generateNRandomClients()** are in interior un loop “for” care porneste de la 0 pana la maxNumClients. In acest loop se va crea cate un client, care mai apoi va fi adaugat la lista de clienti.In aceasta metoda este folosita si metoda **generateRandomNumber(int min, int max)**, care returneaza o valoarea intreaga aflata in intervalul [min,max] . Aceasta metoda este folosita pentru a genera timpii “arrivalTime” si “processTime” in mod aleator.La final lista de clienti la care au fost adaugati respectivii maxNumClients aleatori va fi sortata in functie de timpul la care acestia ajung la coada.
* Metoda **writeWaitingClients():** va scrie in String-ul folosit pentru scrierea in fisier clientii care inca sunt in starea de asteptare.
* Metoda **checkAllQueues()** foloseste in interiorul acesteia metoda **checkQueue()** pentru a verifica daca fiecare coada are clienti in interior.
* Metoda **checkNoClientsRemaining()** parcurge lista de clienti si verifica daca mai exista sau nu elemente in aceasta.Daca nu se gaseste niciun element atunci magazinul va fi considerat ca fiind gol.Ambele functii returneaza valori de tip Boolean.
* Metoda **run()** : aceasta metoda face posibila simularea magazinului.In metoda se gaseste un loop “while” care va rula atat timp cat timpul curent este mai mic decat limita maxima de timp ( tsimulation) sau pana cand acesta este interrupt de conditia care verifica daca magazinul este gol sau nu.In “while” se va parcurge lista de clienti prin intermediul unui iterator.Atunci cand s-a gasit un client cu timpul “arrivalTime” egal cu timpul curent, clientul respective va fi eliminat din coada.In acelasi timp acesta este memorat in LinkedBlockingQueue<Client> al cozii cu timp minim de asteptare, prin intermediul metodei **dispatchClient(Client c)** al **Scheduler-ului**.Dupa ce a fost parcursa lista,verificarile au fost facute si cozile de asemenea au fost construite(au fost adaugati clienti).Se vor scrie in fisier starile cozilor la momentul T al simularii prin intermediul metodei **writeQueues** care primeste ca parametru timpul curent al simularii.In continuare se va face o verificare a listei de clienti.Dorim sa vedem daca sunt clienti in asteptare/care nu au timpul “arrivalTime” egal cu timpul curent de simulare.In cazul in care raspunsul este afirmativ, atunci va fi apelata metoda **writeTheWaitingClients,** care va scrie in fisier clientii care asteapta sa fie adaugati la coada.Mai departe se va verifica daca au mai ramas clienti in magazin.Daca nu au ramas atunci se va da un “break” acestui loop “while”.Daca au ramas, atunci timpul curent va creste cu o unitate.Dupa ce a fost incrementat timpul curent al simularii, thread-ul responsabil de simularea programului va fi pus pe “sleep” timp de o secunda.

**Scrierea** in fisier se realizeaza prin intermediul unei variabile instanta de tip FileWriter si prin intermediul unui String numit writeToFile care este initializat la inceput cu “”/String-ul null.La acest String se vor adauga rezultatele de tipul String de la metodele **writeTheWaitingClients** si **writeQueues**.De asemenea in fisier se va scrie si rezultatul calcularii timpului mediu de asteptare prin intermediul metodei **checkStoreEmpty()**.Dupa fiecare scriere se va apela metoda “flush” a clasei FileWriter pentru a elibera buffer-ul de scriere in fisier.

Clasa **“ReadFile” :** prin intermediul constructorului se acceseaza datele fisierului.Deoarece stim cum vor fi aranjate datele in fisier, putem avea o abordare simpla in legatura cu citirea datelor de pe liniile fisierului.Pentru primele 3 linii datele vor fi memorate printr-o simpla atribuire variabilelor instanta ce descriu numarul clientilor,numarul cozilor si durata simularii.Pentru liniile care descriu intervalele de timp pentru timpul “arrivalTime” si “processTime” capetele intervalelor sunt separate prin intermediul virgulei.Pentru extragerea intervalelor se cheama metoda **readAfterComa(String s)**.Aceasta metoda are rolul de parser.Primeste un String si parcurge acest String pana la intalnirea caracterului “,”.Odata ce a fost gasit tot ce este pe pozitiile anterioare ale virgulei va fi memorat pe prima pozitie a unui vector de tip “int” .Ce se afla dupa virgula va fi memorat pe pozitia a doua a vectorului de tip int.In final va fi returnat un vector de tip “int” de dimensiune 2, care contine capetele intervalului dorit. De asemenea in aceasta clase sunt prezente si metode de tip getter prin care sunt returnate valorile variabilelor instanta.

**5.TESTARE**

Ca mod de testare al acestui proiect/program sunt puse la dispozitie fisierele de testare In-TextX.txt si Out-TestX.txt( X are valori de la 1 la 3).In aceasta etapa de testare, se vor urmari urmatoarele aspecte:

* + Valorile, care au fost generate aleator pentru timpul “arrivalTime” si “processTime” sa fie cuprinse in intervalele mentionate in fisierul din care se citeste.
  + In fisierul de iesire Out-TestX.txt clienti generate trebuie sa fie plasati la cozi la momentul in care ajung( nici mai repede, nici mai tarziu).De asemenea pe masura ce timpul simularii trece, se va verifica decrementarea corecta a timpului de procesare ( o unitate scazuta/sec).
  + In caz ca sunt prea multi clienti generate si cozile nu au cum sa ii proceseze sau timpul se termina, acestia vor ramane neprocesati.

**6.CONCLUZII**

In concluzie, acest proiect a fost un mod bun de a ma acomoda cu ceea ce inseamna “Java Concurrency”, thread-urile si multithreading. Simularea cozilor ale unui magazin, dupa parerea mea, a reprezentat un exemplu foarte bun pentru ceea ce inseamna cele mentionate mai sus. Ca dezvoltare ulterioara m-am gandit la o interfata grafica prin care sa se decida ce cozi trebuie deschise si in ce moment acestea trebuie deschise.Mai mult se poate simula si plecarea unor clienti de la cozile prea pline la cele mai libere.

**7.BIBLIOGRAFIE**

<https://stackoverflow.com/>

<https://www.w3schools.com/>

<https://www.tutorialspoint.com/java/java_multithreading.htm>

<https://howtodoinjava.com/java-concurrency-tutorial/>