

MINISTERUL EDUCAȚIEI NAȚIONALE



UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

FACULTATEA DE AUTOMATICĂ ȘI CALCULATOARE

DEPARTAMENTUL CALCULATOARE

DOCUMENTATIE

la disciplina

Tehnici de Programare

QUEUES SIMULATOR

Irimus Ileana-Maria, grupa 30223

An academic 2019 – 2020

1. Obiectivul temei	2
1.1 Obiective secundare	2
2. Analiza problemei, modelare, scenarii, cazuri de utilizare	2
3. Proiectare (diagrame UML, structuri de date, proiectare clase, packages)	4
4. Implementare.....	5
5. Rezultate	7
6. Concluzii	8
7. Bibliografie	8

1. Obiectivul temei

Obiectivul principal al acestei teme este proiectarea si implementarea unei aplicatii de simulare care vizeaza analiza sistemelor bazate pe cozi pentru determinarea si minimizarea timpului de asteptare al clientilor.

1.1 Obiective secundare

Modelarea in clase si pachete – presupune impartirea proiectului in anumite pachete fiecare pachet fiind mai departe fragmentat in clase cu denumiri sugestive. Acest obiectiv va fi prezentat in detaliu in capitolul 3.

Alegerea structurilor de date - vor fi prezentate structurile utilizate in realizarea proiectului. Acest obiectiv va fi prezentat in detaliu in capitolul 3.

Implementarea claselor – se va descrie fiecare clasa cu metodele si campurile corespunzatoare fiecareia, dar si implementarea interfetei utilizator. Acest obiectiv va fi prezentat in detaliu in capitolul 4.

Testarea - se vor prezenta scenariile pentru testare din cmd. Acest obiectiv va fi prezentat in detaliu in capitolul 5.

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

Cerinte functionale

Aceasta tema presupune atat lucrul cu fisiere text de intrare/iesire date din linia de comanda ca parametri prin tabloul unidimensional args, structuri de date precum Queue si cel mai important: threaduri.

Conceptul de programare multithreaded provine de la cuvantul thread-ul cuvânt care înseamnă “fir” si, deoarece exista mai multe threaduri, prin urmare, multithreadingul este posibil. Aproape toate sistemele de operare accepta conceptul de thread. Un thread poate fi considerat ca fiind calea luata pentru executia unui program. In mod implicit, Java prezinta un thread care ruleaza mereu, care este in threadul main() si este creat in mod intentionat doar de JVM. Cu alte cuvinte, puteti defini threadul ca mai multe sarcini ce coexista simultan intr-un singur proces. Threadurile Java sunt deasemenea denumite si procese “light-weight”. "Threadurile Java sunt obiecte ca orice alte obiecte Java. Threadurile sunt instante ale clasei java.lang.Thread, sau cazuri de subclase. Pe langa faptul ca sunt obiecte, threadurile Java pot executa cod.". Pentru a defini un thread este nevoie ca acesta sa fie extins si mai apoi ca atat interfata Runnable cat si metodele sale sa fie implementate. Fiecare thread suprascrie metoda run () din interfata Runnable, metoda ce va fi executata pe parcursul ciclului de viata a threadului.

Threadurile sunt avantajoase si prezinta numeroase caracteristici:

- ajuta un program sa utilizeze sistemele multiprocessor;
- ajuta la simplificarea modelarii;
- creeaza, de asemenea, o interfata de utilizare mai eficienta;
- face ca programul sa efectueze activitati de fundal asincrone;
- exista intr-un proces (fiecare process are cel putin un singur thread);
- distribuie resurse (atat memorie cat si diverse fisiere);

Asadar pentru gestionarea clientilor din aceasta aplicatie vom utiliza notiunea de coada. Cozile sunt utilizate in mod obisnuit pentru modelarea domeniilor din lumea reala. Principalul obiectiv al unei cozi este sa furnizeze un loc pentru ca un „client” sa astepte inainte de a primi un „serviciu”. Gestionarea bazata pe coada are drept interes minimizarea timpul in care „clientii” lor asteapta la coada inainte ca acestia sa fie serviti. Un mod de a minimiza timpul de asteptare se poate realiza prin adaugarea mai multor servere, adica mai multe cozi in sistem (fiecare coada este considerata ca avand un procesor asociat), dar aceasta abordare creste costurile furnizorului de servicii. Cand se adauga un server(coada) nou, clientii in aasteptare vor fi uniform distribuiti la toate cozile disponibile curente.

Aplicatia simuleaza (prin definirea unui timp de simulare *tsimulation*) o serie de N clienti care sosesc pentru a indeplini serviciul, intrarea in cozi Q, asteptarea in coada pentru a fi servit si, in final, parasirea cozilor. Toti clientii sunt generati aleator la inceperea simularii si sunt caracterizati de trei parametri, acestia fiind: ID(un numar intre 1 si N), *tarrival* (timpul de simulare cand sunt gata sa mearga la coada, adica ora la care clientul a terminat cumparaturile) si *tserve* (intervalul de timp sau durata necesara pentru a ca respectivul client sa fie servit de catre casier, adica timpul de asteptare cat timp clientul se afla in fata cozii). Aceasta aplicatie

urmaresteste timpul total petrecut de catre fiecare client la cozi si calculeaza timpul mediu de asteptare. Fiecare client este adaugat la coada cu un timp minim de asteptare cand timpul sau *tarrival* este egal cu timpul de simulare ($tarrival \geq tsimulation$). Iata un exemplu de date considerate date de intrare citite dintr-un fisier text pentru aplicatie dat ca si argument in metoda main() din linia de comanda:

- numarul de clienti (N);
- numarul de cozi (Q);
- intervalul de simulare (*tsimulation MAX*);
- ora minima si maxima de sosire ($tarrival MIN \leq tarrival \leq tarrival MAX$);
- timp de serviciu minim si maxim ($tserve MIN \leq tserve \leq tserve MAX$);

Astfel, un numar de threaduri Q vor fi lansate pentru a prelucra in paralel clientii. Iar un alt thread va fi lansat pentru a mentine timpul de simulare *tsimulation* si pentru a distribui fiecare client i la coada cu cel mai mic timp de asteptare in momentul in care $tarrival i = tsimulation$. Iesirea aplicatiei este reprezentata de un fisier text care contine atat un jurnal al executiei aplicatiei cat si timpul mediu de asteptare al clientilor pentru a fi serviti si mai apoi pentru a

parasi magazinul. Rezultatul simulării descrie starea grupului de clienti in asteptare si cozile, cat timp timpul de simulare *tsimulation* creste de la 0 la *tsimulation MAX*.

Simularea are loc astfel: clientii sunt generate aleator, insa ordonati in functie de timpul de sosire tarrival, iar timpul simulării incepe de la 0. Timpul de sosire al fiecarui client ne va spune la ce moment din simulare va intra in coada, iar timpul de service ne va spune cat va dura pana cand el este procesat si poate pleca. Aceleasi reguli se aplica pentru toti clientii. In ceea ce priveste cozile, acestea vor fi inchise (closed) cat timp niciunul dintre clienti nu asteapta la coada. Simularea este incheiata atunci cand nu mai exista clienti in coada de asteptare sau la cozile de serviciu sau *tsimulation > tsimulationMAX*. Timpul mediu de asteptare este calculat si anexat la fisier.

3. Proiectare (diagrame UML, structuri de date, proiectare clase, packages)

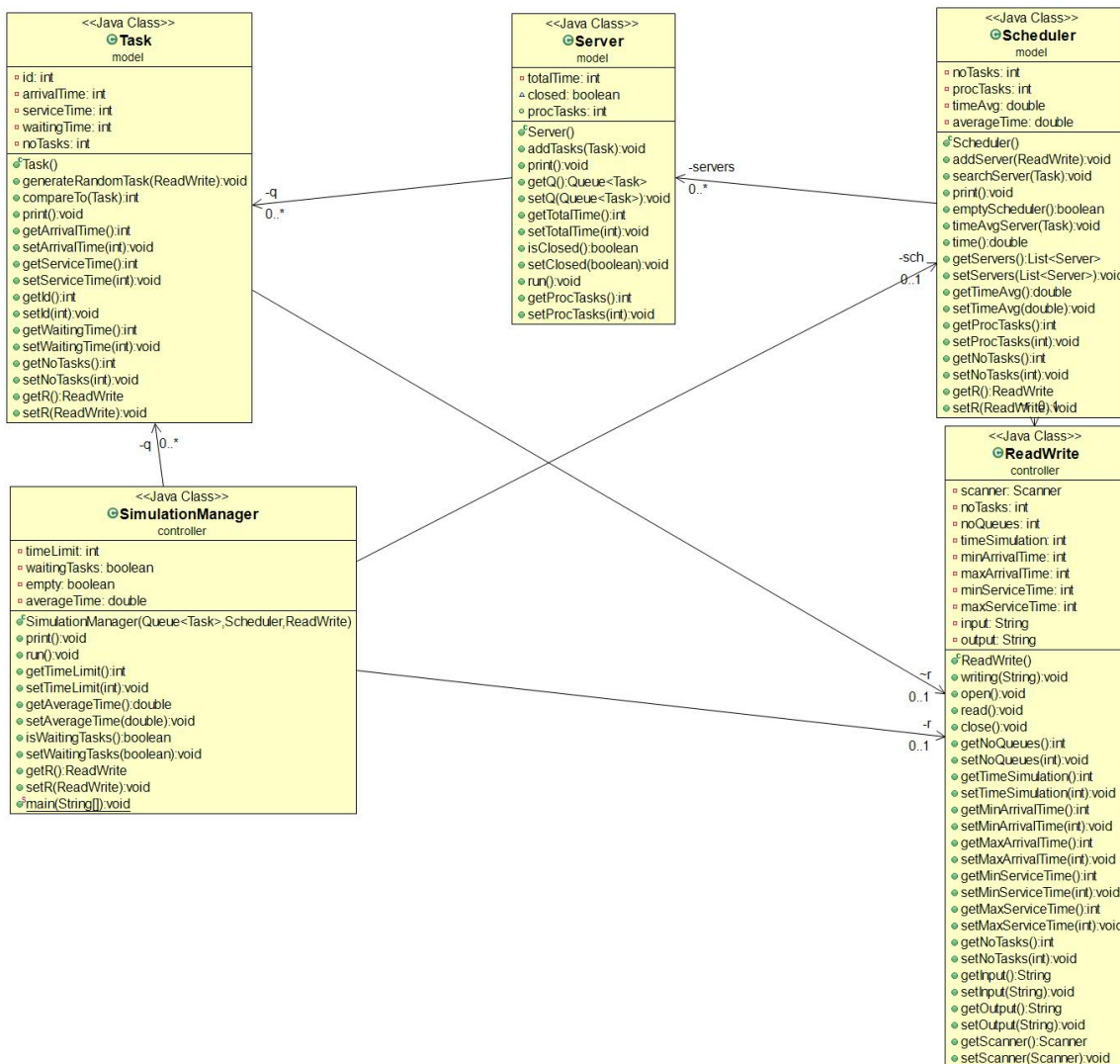


Diagrama UML de clasa

Aceasta aplicatie este proiectata dupa conventia specifica programarii orientate pe obiect, avand un pachet model si unul controller, logica programului fiind impartita in doua elemente interconectate. Scopul acestei divizari este de a separa reprezentarea interna de modul/informatia prezentata utilizatorului. Astfel, modelul gestioneaza direct datele de intrare ale utilizatorului primite de la controller. Controllerul primeste informatia, o valideaza, iar mai apoi o transmite modelului. In acest caz modelul este reprezentat de clasele Task, Server si Scheduler; controllerul este descris de alte doua clase ReadWrite si SimulationManager.

Pentru realizarea acestei teme am decis sa utilizez Queue ca structura de date principala. Interfata Queue este disponibila in pachetul java.util si extinde interfata Collection. Colectia de cozi este utilizata pentru a retine elementele care urmeaza sa fie procesate si ofera diferite operatiuni, cum ar fi inserarea, eliminarea etc. din lista, adica urmeaza principiul FIFO sau First-In-First-Out. Fiind o interfata, coada are nevoie de o clasa concreta pentru declaratie, iar cele mai frecvente clase sunt PriorityQueue si LinkedList in Java. PriorityQueue este o implementare alternativa daca este necesara implementarea sigura a thread-ului. Cateva dintre caracteristicile importante ale cozii sunt:

- Queue este folosit pentru a insera elemente la sfarsitul cozii si pentru a le elimina de la inceputul cozii. Urmeaza conceptul FIFO;
- Java Queue accepta toate metodele de interfata de colectare, inclusiv inserarea, si stergerea;
- LinkedList, ArrayBlockingQueue si PriorityQueue sunt cele mai utilizate implementari;
- dacă orice operatie nula este efectuata pe BlockingQueues, NullPointerException este aruncat;
- BlockingQueues are implementari sigure in threaduri;
- toate cozile, cu exceptia celor de la Deques, accepta inserarea si eliminarea in coada si, respective, la capul cozii. Deques suporta introducerea si indepartarea elementelor la ambele capete.

4. Implementare

Proiectul are in componenta sa 2 pachete: model si controller.

Pachetul model contine cele trei clase Task, Server si Scheduler.

Clasa Task prezinta obiectul client, descris de un int id, reprezentand numarul de ordine al clientului, un int arrivalTime, reprezentand timpul de sosire in coada, un int serviceTime, reprezentand durata de asteptare a clientului in fata cozii, un int waitingTime, reprezentand timpul de asteptare al fiecarui client din coada, updatat in functie de numarul de client care se afla la rand inaintea sa, un int noTasks, reprezentand numarul total de client ce se afla in magazine si o instanta de tip ReadWrite pentru a genera datele din fisierul de intrare. Pe langa constructor, aceasta clasa include setters si getters pentru fiecare dintre campurile descrise mai sus, dar si functia de afisare print() ce va returna sub forma unui string datele despre fiecare client astfel: (id, arrivalTime, serviceTime). Tot in aceasta clasa este prezentata si functia compareTo utilizata mai tarziu pentru ordonarea clientilor in functie de

timpul de sosire in coada, dar si metoda generateRandomTask() care va initializa timpzii de sosire si de service a fiecarui client in mod aleator folosind functia Math.random() intr-un interval de numere citit din fisier.

Clasa Servers prezinta obiectul coada ca fiind de tip Queue<Task> q = new LinkedList<Task>(). Acest obiect este descris de un int totalTime, ce reprezinta timpul total de asteptare a clientilor si o variabila boolean closed initializata cu true, care ne spune daca coada este inchisa sau nu. Constructorul acestei clase este gol, deoarece in coada se vor adauga pe rand diversi clienti prin metoda de tip void numita addTasks(Task task), ce foloseste functia predefinita add, specifica Collection. Tot in aceasta metoda se va seta si timpul total, dar si timpul de asteptare al fiecarui client. De asemenea, aceasta clasa contine si setter si getter pentru lista de monoame, dar si o functie de printare prin parcurgerea cozii si apelarea functiei de printare din clasa Task. Cea mai importanta metoda din aceasta clasa este metoda run() specifica interfetei Runnable, utilizata pentru thread astfel: cat timp coada nu este goala, se extrage primul client, i se decrementeaza timpul de service, dar totodata si timpul total de asteptare, iar in cazul in care timpul de service este 0, clientul este indepartat din coada. In cazul in care coada este goala, variabila closed va fi setata ca true.

Clasa Scheduler prezinta lista tuturor cozilor ca fiind de tip List<Server> servers = new ArrayList<Server>(). Constructorul acestei clase este gol, deoarece in lista de cozi se vor adauga pe rand cozile prin metoda de tip void numita addServers, ce foloseste functia predefinita add, specifica Collection. Tot aici se face si adaugarea clientilor in coada cu timpul de asteptare minim in metoda searchServer(Task task). Printarea listei de cozi este implementata in functia void print() care in cazul in care coada nu este goala ii va afisa continutul (clientii), altfel se va afisa mesajul "closed". O alta metoda descrisa in aceasta clasa este cea de verificare daca lista de cozi este goala, adica daca niciun client nu asteapta pentru a fi servit in coada. Tot aici este calculate si timpul mediu de asteptare al clientilor dupa formula: timpul total de asteptare al clientilor procesati / numarul clientilor procesati.

Pachetul controller include clasele ReadWrite si SimulationManager.

Clasa ReadWrite realizeaza citirea si scrierea din/in fisierele date ca parametri din linia de comanda (args[0] si args[1]). Astfel, fisierul de intrare este deschis prin metoda open() si se citesc cele 5 linii prin metoda read(); mai apoi datele citite sunt convertite in intregi si sunt setati noTasks(numarul de client), noQueues(numarul de cozi), timeSimulation(durata simularii), minArrivalTime(timpul de sosire minim), maxArrivalTime(timpul de sosire maxim), minServiceTime(timpul de servire minim) si maxServiceTime(timpul de servire maxim). Toate aceste date sunt transmise mai departe prin getters. Scrierea in fisier se realizeaza cu ajutorul metodei writing(String s) folosind FileWriter si PrintWriter cu metoda specifica write().

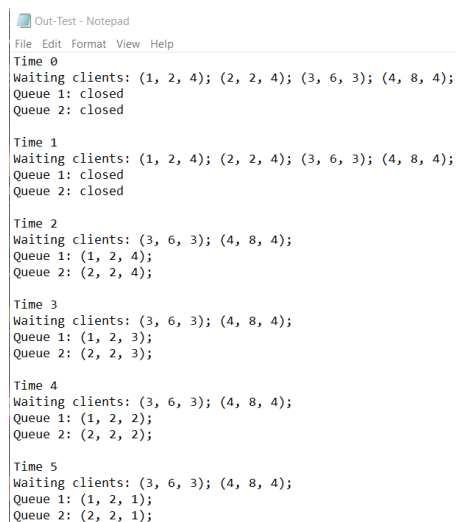
Clasa SimulationManager implementeaza Interfata Runnable si prezinta simularea aplicatiei. Astfel clasa este descrisa de un int timeLimit, reprezentand timpul maxim al simularii, o variabila de tip boolean waitingTasks, care indica daca mai sunt sau nu client in asteptare, un boolean empty, si un double averageTime, reprezentand timpul mediu de asteptare. Pe langa

metoda print, aceasta clasa implementeaza o metoda importanta pentru aceasta aplicatie: metoda run() astfel: cat timp timpul current al simularii este mai mic decat timpul maxim si ai sunt client care asteapta sa intre in coada sau cozile nu sunt goale, se verifica timpul de sosire al ficarui client din lista de asteptare; daca timpul sau de sosire in coada este egal cu timpul curent, clientul este adaugat in coada cu timpul de asteptare minim si este eliminat din lista de asteptare(a terminat de facut cumparaturile); apoi, pentru fiecare coada este pornit un thread; daca mai sunt client care asteapta (nu si-au terminat cumparaturile), variabila waitingTasks devinde adevarata, altfel se va afisa mesajul "none"; timpul current se incrementeaza, iar la final se va afisa timpul mediu de asteptare al clientilor. Am ales sa implementez metoda main in aceasta clasa, deoarece aici are loc si simularea. Astfel, in metoda statica main sunt instantiate clasele de citire/scriere din/in fisierele date, Scheduler, Task si SimulationManager si apelate functii de setters, add, initializarea clientilor.

5. Rezultate

Testarea este realizata din Command Line prin comanda "java -jar Assignment2.jar <calea fisierului cu datele de intrare> <calea fisierului cu datele de iesire>".

Assignment2.jar este fisierul jar al acestui proiect. Un JAR (arhiva Java) este un format de fisier de pachete utilizat in mod obisnuit pentru a aglomera multe fisiere de clasa Java si metadata si resurse asociate (text, imagini etc.) intr-un singur fisier pentru a distribui software-ul aplicatiei sau bibliotecile pe platforma Java. In cuvinte simple, un fisier JAR este un fisier care contine versiunea comprimata a fisierelelor .class, fisiere audio, fisiere imagine sau directoare. Chiar si software-ul WinZip poate fi folosit pentru a extrage continutul unui .jar. Asadar, le puteti utiliza pentru sarcini precum compresia de date fara pierderi, arhivarea, decompresia si dezambalarea arhivei.



```
Out-Test - Notepad
File Edit Format View Help
Time 0
Waiting clients: (1, 2, 4); (2, 2, 4); (3, 6, 3); (4, 8, 4);
Queue 1: closed
Queue 2: closed

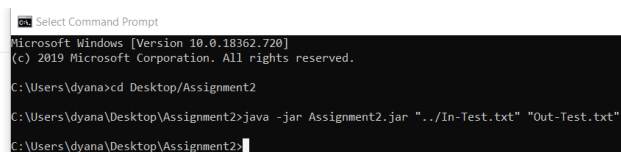
Time 1
Waiting clients: (1, 2, 4); (2, 2, 4); (3, 6, 3); (4, 8, 4);
Queue 1: closed
Queue 2: closed

Time 2
Waiting clients: (3, 6, 3); (4, 8, 4);
Queue 1: (1, 2, 4);
Queue 2: (2, 2, 4);

Time 3
Waiting clients: (3, 6, 3); (4, 8, 4);
Queue 1: (1, 2, 3);
Queue 2: (2, 2, 3);

Time 4
Waiting clients: (3, 6, 3); (4, 8, 4);
Queue 1: (1, 2, 2);
Queue 2: (2, 2, 2);

Time 5
Waiting clients: (3, 6, 3); (4, 8, 4);
Queue 1: (1, 2, 1);
Queue 2: (2, 2, 1);
```



```
Select Command Prompt
Microsoft Windows [Version 10.0.18362.720]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\dyana>cd Desktop/Assignment2

C:\Users\dyana\Desktop\Assignment2>java -jar Assignment2.jar ../In-Test.txt ../Out-Test.txt

C:\Users\dyana\Desktop\Assignment2>
```

6. Concluzii

In concluzie, aceasta aplicatie poate fi una foarte utila si eficienta in procesarea diverselor simulari care se bazeaza pe cozi cu scopul minimizarii timpului de asteptare al clientilor.

In urma realizarii acestei teme, am invat sa lucrez cu threaduri, dar si cu argumentele trimise din linia de comanda. O posibila dezvoltare ulterioara ar fi posibilitatea adaugarii mai multor cozi in momentul in care timpul mediu de asteptare al clientilor depaseste o anumita cifra sau daca se observa ca timpul acordat simularii nu este suficient.

7. Bibliografie

<https://mkyong.com/tutorials/junit-tutorials/>

<http://zetcode.com/tutorials/javaswingtutorial/>

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

<https://stackoverflow.com/>

<https://docs.oracle.com/javase/tutorial/essential/concurrency/index.html>