TEHNICI DE PROGRAMARE FUNDAMENTALE
ASSIGNMENT 2

QUEUES SIMULATOR

DOCUMENTATIE

Bakk Cosmin-Robert
Grupa 30227

Cuprins

1. Obiectivul temei

Obiectivul acestei teme de laborator este de a implementa un simulator de cozi. Pentru a simula o coada, un utilizator trebuie sa creeze si sa apeleze printr-o linie de comanda aplicata unei arhive .jar un fisier text de intrare care continte urmatoarele date: numarul de clienti. numarul de cozi, intervalul de timp pentru simulare, timpul minim si maxim de ajungere a unui client la coada si timpul minim si maxim de procesare pentru fiecare client. In urma introducerii acestor date, se vor crea cozile si campurile clientilor, fiecare avand un ID unic, un timp de ajungere si un timp de procesare. Fiecare client va ajunge la coada dupa un anumit timp, iar apoi va trebui pus la una dintre cozi. Strategia prin care determinam la care dintre cozi va fi pus este una in care timpul de asteptare este mai importat decat numarul de clienti de la o anumita coada, adica vom alege coada la care ar avea de asteptat timpul minim, indiferent de numarul de clienti de la acea coada.

Obiectivele secundare sunt: respectarea paradigmei programarii orientate pe obiect, folosirea threadurilor si a metodelor asociate clasei Thread, imbunatatirea cunostiintelor de folosire a functiilor de citire/scriere intr-un fisier text si crearea unei arhive .jar pentru portabilitate si testarea mai usoara a programului.

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

Pentru a genera clientii dupa ce am citit datele din unul din fisierul de intrare, am folosit metoda nextInt a clasei Random, iar apoi am pornit thread-ul asociat clasei care se va ocupa de administrarea cozilor si a clientilor.

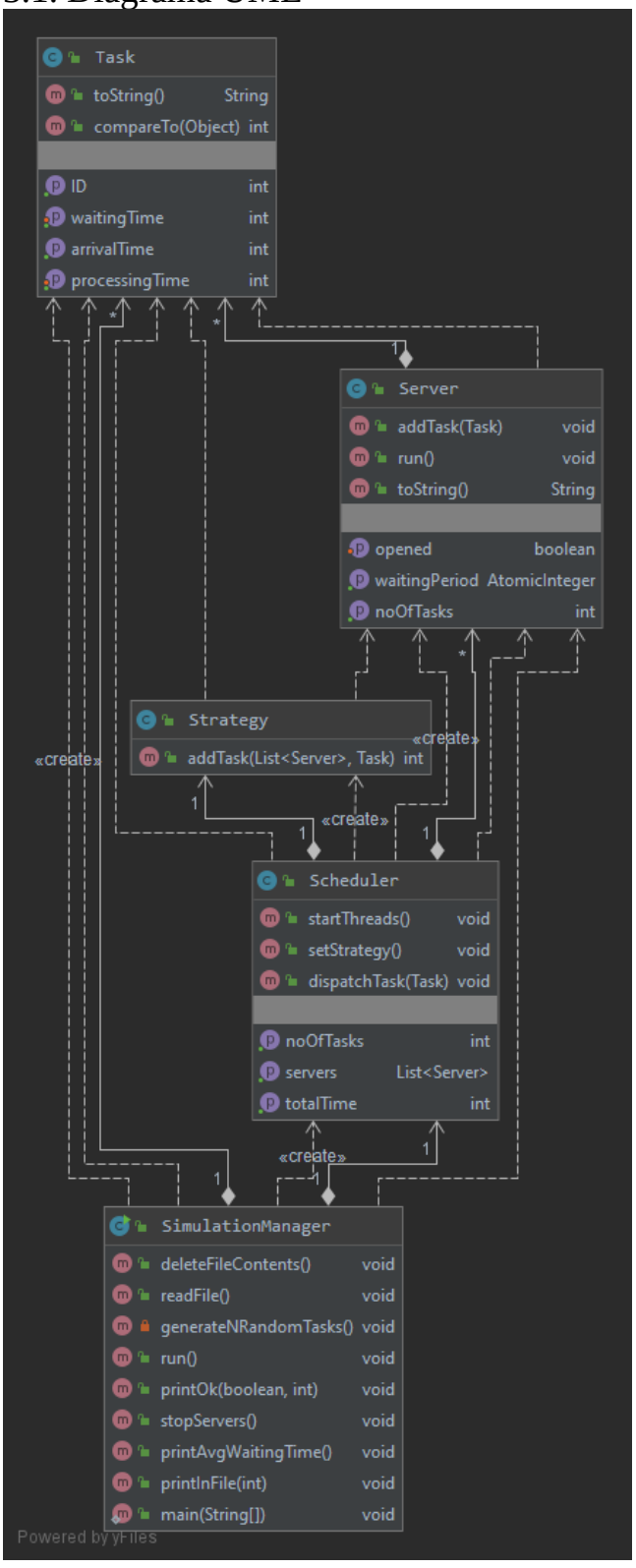
Cerintele sunt ca utilizatorul sa creeze fisiere.txt care respecta formatul de mai jos. Rezultatele simularii vor fi puse intr-un fisier text de output, cu numele dat de catre utilizator.

Descriere use-case: utilizatorul deschide o fereastra Command Prompt in folderul proiectului si ruleaza comanda java -jar PT2020_30227_Cosmin-Robert_Bakk_Assignment_2.jar in.txt out.txt, unde in.txt este fisierul .txt creat de catre utilizator, care contine urmatoarele:

<numarul de clienti>
<numarul de cozi>
<timpul de simulare>
<timpul de ajungere minim>,<timpul de ajungere maxim>
<timpul de procesare minim>,<timpul de procesare maxim>

3. Proiectare

3.1. Diagrama UML



3.2. Proiectare clase

In cele ce urmeaza, termenul “task” este echivalent termenului “client”, iar termenul “server” este echivalent termenului “coada”.

Clasa Task: implementeaza Comparable. Aceasta clasa defineste conceptul de client, acesta fiind caracterizat printr-un ID, specific fiecarui client, un timp de ajungere de tipul intreg, arrivalTime, un timp de procesare de tipul intreg, processingTime si un timp de asteptare de tipul intreg, waitingTime. Campul arrivalTime reprezinta timpul dupa care clientul ajunge la coada, processingTime reprezinta timpul necesar pentru procesarea clientului cand acesta este in capatul cozii, iar waitingTime este timpul total pe care l-a asteptat in coada, inclusiv procesarea.

Clasa Server: implementeaza Runnable. Fiecarui obiect de tip Server i se va asocia un thread. Aceasta clasa defineste conceptul de coada, este caracterizat de o lista de clienti de tipul BlockingQueue, tasks, un timp de asteptare in acea coada la un moment dat de tipul AtomicInteger, waitingPeriod, si un boolean opened, care marcheaza faptul ca respectiva coada este inchisa sau deschisa, threadul asocial acesteia fiind activ sau inactiv.

Clasa Strategy: aceasta clasa reprezinta strategia aleasa pentru a pune fiecare client la coada si se va ocupa de acest lucru printr-o singura metoda, addTask.

Clasa Scheduler: aceasta clasa se ocupa cu alegerea strategiei de distributie a clientilor (in cazul de fata a fost implementata doar o strategie), cu instantierea cozilor si cu pornirea thread-urilor asociate fiecarei cozi. Clasa Scheduler contine o lista de cozi, servers, un vector de thread-uri, threads, numarul maxim de cozi, maxNoServers, numarul maxim de clienti pe coada, maxTasksPerServer, o instanta a clasei Strategy, strategy, si un intreg care reprezinta timpul total de asteptare a clientilor, totalTime.

Clasa SimulationManager: aceasta clasa este clasa main a proiectului. Campurile acestei clase sunt variabilele citite din fisier, timeLimit, maxProcessingTime, minProcessingTime, maxArrivalTime, minArrivalTime, numberOfServers, numberOfClients, numele fisierelor de input si de output, in si out, un obiect de tip Scheduler, scheduler, si o lista care va contine task-urile generate, generatedTasks. Aceasta clasa se ocupa cu citirea/scrierea din fisiere, si ii este asociat un thread, care are o variabila de timp locala si care se ocupa cu preluarea clientilor din lista de asteptare si trimiterea mai departe a acestora catre clasa Scheduler, pentru distribuire.

4. Implementare si testare
4.1. Metode

Clasa Task

public Task (int ID, int arrivalTime, int processingTime)
-constructorul clasei, primeste cele 3 variabile de instanta care sunt generate in clasa SimulationManager

public int getArrivalTime()
-returneaza arrivalTime-ul clientului

public int getProcessingTime()
-returneaza processingTime-ul clientului

public int getID()
-returneaza ID-ul clientului

public void setProcessingTime (int processingTime)
-seteaza processingTime-ul clientului

public int getWaitingTime()
-returneaza waitingTime-ul clientului

public void setWaitingTime(int waitingTime)
-seteaza waitingTime-ul clientului

public String toString()
-returneaza clientul sub forma unui string, pentru a putea fi scris in fisier
-este de forma (ID, arrivalTime, processingTime)

public int compareTo (Object x)
-compara doi clienti in functie de arrivalTime-ul lor

Clasa Server

public Server (int maxTasks)
-constructorul clasei, primeste numarul maxim de clienti din simulare si genereaza atatea cozi, iar waitingPeriod-ul este setat la 0

public void addTask (Task newTask)
-la primirea unui nou client, acesta e adaugat in BlockingQueue-ul cozii, iar timpul de asteptare pentru aceasta coada este crescut cu processingTime-ul noului client
-se apeleaza notify() pentru thread-ul curent

public void run()
-este metoda in care se intra la inceperea unui thread. Thread-ul este viu cat timp boolean-ul opened este true. Thread-ul este suspendat cu comanda wait() cat timp lista tasks este goala si opened este true. La adaugarea unui nou client in server si primirea notify-ului, thread-ul este scos de sub suspendare. Se citeste clientul din varful cozii, folosind metoda peek a BlockingQueue-ului. Pentru un timp egal cu processingTime-ul clientului, thread-ul intra in sleep, iar la fiecare secunda sunt decrementate processingTime-ul clientului si waitingPeriod-ul cozii. Dupa acest timp de asteptare, clientul este eliminat din coada.

public AtomicInteger getWaitingPeriod()
-returneaza waitingPeriod-ul cozii

public void setOpened(boolean opened)
-seteaza variabila opened a cozii

public int getNoOfTasks()
-returneaza numarul de clienti care se afla in coada, la un moment dat

public String toString()
-returneaza clientii cozii sub forma unui string, pentru a putea fi scrisi in fisier

Clasa Strategy

public int addTask(List<Server> servers, Task t)
-se declara la inceput un integer, acesta reprezentant waitingPeriod-ul primei cozi. Se parcurge apoi lista de cozi, pentru a gasi waitingTime-ul minim. Se mai parcurge inca o data lista, iar in momentul in care se gaseste o coada care are waitingTime-ul egal cu waitingTime-ul minim dintre cozi, se apeleaza metoda addTask pentru clientul primit ca parametru. Metoda returneaza timpul de asteptare a acelei cozi.

Clasa Scheduler

public Scheduler (int maxNoServers, int maxTasksPerServer)
-constructorul clasei, primeste numarul maxim de cozi si numarul maxim de clienti pe coada. Se apeleaza apoi metoda startThreads(), care va porni threadurile asociate cozilor

public void startThreads()
-se initializeaza vectorul de thread-uri threads si lista de cozi servers. Se adauga in lista servers maxNoServers cozi si se creaza un nou thread asociat acelei cozi, pentru care se apeleaza start(), ceea ce declanseaza executia metodei run() din acea coada

public void setStrategy ()
-se initializeaza o strategie

public void dispatchTask (Task t)
-clientul primit ca parametru se trimite mai departe pentru a putea fi adaugat unei noi cozi, respectand strategia aleasa. waitingTime-ul clientului este setat ca fiind suma dintre timpul de asteptare la coada si processingTime-ul sau. Variabila totalTime a Scheduler-ului este incrementata cu waitingTime-ul clientului.

public int getNoOfTasks()
-returneaza numarul de clienti care se afla in total in cozi

public List<Server> getServers()
-returneaza lista de servere

public int getTotalTime()
-returneaza totalTime-ul din scheduler, acesta fiind timpul total de asteptare a clientilor la cozi

Clasa SimulationManager
private SimulationManager(String in, String out)
-constructorul clasei, primeste cele doua String-uri reprezentand numele fisierelor de input si de output. Apeleaza apoi metodele deleteFileContents() si readFile(), dupa care instantiaza scheduler un nou Scheduler care primeste ca parametrii numberOfServers si numberOfClients. Ii seteaza strategia scheduler-ului, iar apoi apeleaza metoda generateNRandomTasks, care va genera clientii

public void deleteFileContents()
-sterge continutul anterior al fisierului de output

public void readFile()
-se parcurge fisierul de input folosind o instanta a clasei Scanner. Scanner-ul parcurge fiecare linie si o converteste la integer, ultimele doua linii fiind o exceptie, deoarece sunt de forma “a,b” – aici se va folosi un split pe string dupa caracterul virgula “,”

private void generateNRandomTasks()
-se declara o instanta a clasei Random, iar apoi se initializeaza lista generatedTasks cu numberOfClients clienti. Pentru fiecare client, se genereaza un arrivalTime intre limitele minArrivalTime si maxArrivalTime si un processingTime intre limitele minProcessingTime si maxProcessingTime, folosind metoda nextInt. Aceasta metoda primeste ca parametru diferenta dintre maxim si minim si genereaza un intreg intre 0 si acea diferenta. Acestui numar generat ii vom adauga minimul, rezultand astfel un intreg in intervalul (minim, maxim). Se sorteaza apoi lista generatedTasks, in functie de arrivalTime

public void run()
-este metoda in care se intra la inceperea thread-ului. Aceasta metoda are o lista de clienti, toRemove, care vor trebui sa fie stersi, si un timp local, currentTime, de tip intreg. Conditia de oprire a thread-ului este fie depasirea timeLimit-ului, fie atunci cand atat lista generatedTasks e goala, cat si numarul de clienti de la cozi este 0. In interiorul buclei while, se parcurg clientii listei generatedTasks si se verifica daca currentTime este egal cu arrivalTime-ul clientului. Daca se intampla asta, atunci clientul este trimis catre scheduler si este adaugat listei toRemove. Se sterg din generatedTasks toti clientii care apar in toRemove, apoi se apeleaza metoda printInFile, avand ca parametru currentTime, pentru a scrie in fisier situatia de la timpul currentTime a listei de asteptare si a cozilor. currentTime este apoi incrementat, iar thread-ul ia o pauza de o secunda. La iesirea din bucla while, se verifica daca lista generatedTasks mai are clienti sau daca mai sunt clienti in coada. In cazul acesta, timpul de simulare a fost prea mic pentru procesarea tuturor clientilor si se apeleaza metoda printOk, cu parametrii false si currentTime. Daca verificarea anterioara este falsa, inseamna ca lista generatedTasks e goala si nu mai sunt clienti la cozi si se apeleaza metoda printOk, cu parametrii true si currentTime. Se apeleaza apoi metoda printAvgWaitingTime, pentru a scrie in fisier timpul mediu de asteptare pentru un client. Se apeleaza apoi metoda stopServers, pentru a opri thread-urile asociate cozilor.

public void printOk(boolean ok, int currentTime)

-in functie de parametrul ok, scrie in fisier fie ca nu au fost procesati toti clientii, fie ca au fost cu totii procesati si cozile sunt goale

public void stopServers()

-pentru fiecare coada din scheduler seteaza campul opened la false, iar apoi apeleaza notify() pentru thread-ul acesteia

public void printAvgWaitingTime()

-se calculeaza un numar real ca fiind raportul dintre totalTime al scheduler-ului si numberOfClients si se scrie in fisierul de output

public void printInFile(int currentTime)

-scrie in fisier, pentru timpul currentTime, toti clientii care sunt in lista de asteptare si situatia fiecarei cozi, daca aceasta e inchisa sau daca nu e, scrie toti clientii care se afla la acea coada

public static void main(String[] args)

-este functia main a clasei. Se intializeaza un obiect de tipul SimulationManager, gen, avand ca argumente args[0] si args[1], numele fisierului de intrare, respectiv numele fisierului de iesire. Se creeaza un thread nou, care este asociat obiectului gen, iar apoi este pornit

4.2. Testare

Pentru primul fisier, in-test-1.txt, datele de intrare sunt:

```
4
2
60
2,30
2,4
```

In urma executiei comenzii

```
java -jar PT2020_30227_Cosmin-Robert_Bakk_Assignment_2.jar in-test-1.txt out-test-1.txt
```

se va crea un nou fisier de output, out-test-1.txt, iar textul pentru ultimii 5 timpi ai programului este:

```
Time 14
Waiting clients: (1,15,3)
Queue 1: (4,12,1)
Queue 2: (3,14,3)

Time 15
Waiting clients:
Queue 1: (1,15,3)
Queue 2: (3,14,2)

Time 16
Waiting clients:
Queue 1: (1,15,2)
Queue 2: (3,14,1)

Time 17
Waiting clients:
Queue 1: (1,15,1)
Queue 2: closed

Time 18
Waiting clients:
Queue 1: closed
Queue 2: closed

All clients processed. No more clients in queues.
Average waiting time: 2.75
```

Pentru cel de-al doilea fisier, datele de intrare sunt:

```
50
5
60
2,40
1,7
```

In urma executiei comenzii

```
java -jar PT2020_30227_Cosmin-Robert_Bakk_Assignment_2.jar in-test-2.txt out-test-2.txt
```

se va crea un nou fisier de output, out-test-2.txt, iar textul pentru ultimii 3 timpi ai programului este:

```
Time 41
Waiting clients:
Queue 1: closed
Queue 2: (7,36,1)
Queue 3: (49,37,2)
Queue 4: (48,38,1)
Queue 5: (47,36,1)

Time 42
Waiting clients:
Queue 1: closed
Queue 2: closed
Queue 3: (49,37,1)
Queue 4: closed
Queue 5: closed

Time 43
Waiting clients:
Queue 1: closed
Queue 2: closed
Queue 3: closed
Queue 4: closed
Queue 5: closed

All clients processed. No more clients in queues.
Average waiting time: 5.14
```

Pentru cel de-al treilea fisier, datele de intrare sunt:

```
1000
20
200
10,100
3,9
```

In urma executiei comenzii

```
>java -jar PT2020_30227_Cosmin-Robert_Bakk_Assignment_2.jar in-test-3.txt out-test-3.txt
```

se va crea un nou fisier de output, out-test-3.txt, iar textul din finalul fisierului este:

```
Not all clients processed. Need more time.
Average waiting time: 94.653
```

Acest lucru se datoreaza faptului ca au trecut cele 200 de secunde alocate pentru rularea simularii, dupa care, chiar daca lista de asteptate era goala, cozile aveau inca clienti.

5. Concluzii

Acest proiect poate fi util nu doar pentru simularea unor clienti intr-o coada, ci si pentru simularea altor obiecte in cozi, in mod concurent, prin folosirea thread-urilor. Proiectul m-a ajutat sa dobandesc mai multe cunostiinte in programarea orientata pe obiect si am invatat conceptul de thread si lucrul cu arhivele .jar.

6. Bibliografie

1. http://coned.utcluj.ro/~salomie/PT_Lic/4_Lab/
2. BlockingQueue, AtomicInteger, Thread documentation
3. https://www.edureka.co/blog/java-wait-and-notify/
4. https://www.jetbrains.com/help/idea/packaging-a-module-into-a-jar-file.html
5. BlockingQueue, AtomicInteger, Thread documentation