***Queues Simulator***

***Materie: Tehinici de Programare***

***Elev: Apostol Ovidiu-Catalin***

***Grupa: 30223,semigrupa 1***

***Specializare: CTI RO, an 2***

*Cuprins*

[*Obiectivul temei* 2](#_Toc35361509)

[*Analiza problemei* 2](#_Toc35361510)

[*Proiectare* 4](#_Toc35361511)

[*Implementare* 5](#_Toc35361512)

[*Rezultate* 8](#_Toc35361513)

[Concluzii 11](#_Toc35361514)

[*Bibliografie* 11](#_Toc35361515)

# *Obiectivul temei*

Principalul obiectiv al acestei teme as dobandirea cunostiintelor cu privire la programarea concurenta prin simularea unui sistem de gestionare a unor cozi care servesc niste clienti.Se va lucra cu cozi(BlockedQueues) ,array-uri , liste , threaduri , programare concurent , citit si scris in fisiere text .

Pentru a indeplini aceste obiective va trebui sa impartim problema in subprobleme mai mic :

* Transpunearea cozilor si clientilor din lumea reala sub forma unor clase abstracte (respectiv server si task ) ;
* Creeare unui sistem care sa gestioneze interactiunile din clienti si cozi .
* Crereare unui sistem care sa simuleze aceasta interactiune .
* Generarea a N clienti impreuna cu arrival time-ul si processingTime-ul lor cuprinse intre doua marje .
* Deschiderea a doua fisiere pentru citirea constrangerilor simularii si respectiv pentru scrierea la fiecare perioada de timp a starii cozilor si respectiv clientilor ce vor ajunge.

# *Analiza problemei*

Pentru a putea avea o buna viziune a acestei probleme si a o rezolva intr-un mod optim trebuie mai intai sa ne gandim care sunt actiunele pe care o sa le faca “clientii “ . Cand un client ajunge in magazin acesta doreste sa isi rezolve task-ul cat mai rapid , deci strategia de proiectare v-a fi una bazat pe Timp! Pentru a iesi din magazin intr-un timp optim , clientul v-a verifica toate cozile si se va aseza la coada la care urmeaza sa astepte un timp minim .

Acum pentru a analiza structura de BlockedQueue , este o coada care are capacitatea de a bloca threadul in caz ca aceasta este plina si incearca sa se adauge in ea , sau daca este goala si se doreste a se scoate din ea . Structura unei cozi este usor de inteles : FIFO( first In , first Out) deci clienti sunt adaugati la coada (sfarsitul) cozii si sunt scosi de la capul cozii .

Acum trebuie sa ne gandim la cum vom reprezenta clientii . Informatiile de care avem nevoie de la acestia sunt ora/ timpul la care clientul ajunge in magazin , si actiunea pe care o are de efectuat ( respectiv timpul necesar efectuarii acestei actiuni ) . Deci vom abstractiza clientul la o clasa Task care va avea ca principale atribute Processing Time-ul si Arrival Time-ul.

Pentru a avea o implementare mai eficienta a acestei probleme vom lansa mai multe threaduri , fiecare thread se va ocupa de o singura coada (acestea lucrand in paralel ) . Dar avand in vedere ca acestea lucreza in parele trebuie sa avem grija ca in caz ca mai mutle threaduri acceseaza aceeasi resursa, accesarea sa se realizeze Thread Safe .

***Modelare***

Un prim pas in modelarea acestei probleme il reprezinta parcurgerea template-urilor puse la dispozitie de profesor pentru o mai usoara intelegere a abstractizarii acestei probleme (transpunerea ei in cod ) . Dupa care v-a trebui sa completam campurile pe care le avem de completat , si sa adaugam diferite metode , argumente proprii in functie de metoda de implementare aleasa de fiecare .

Pentru citirea si scrierea din fisier vom da ca si argumente din lina de comanda numele fisierelor.

***Utilizare (use-case)***

Pentru a folosi aplicatia , o vom apela din linia de comanda dupa generarea fisierului . Jar. Vom avea 2 argumente pe care le vom da : primul argument va fi numele fisierului de intrare , iar al doilea argument numele fisierului de iesire .

Astfel dupa apel se va genera simularea pe pasi in fisierul dat ca parametrul 2 avand ca si constrageri datele citite din parametrul 1 .

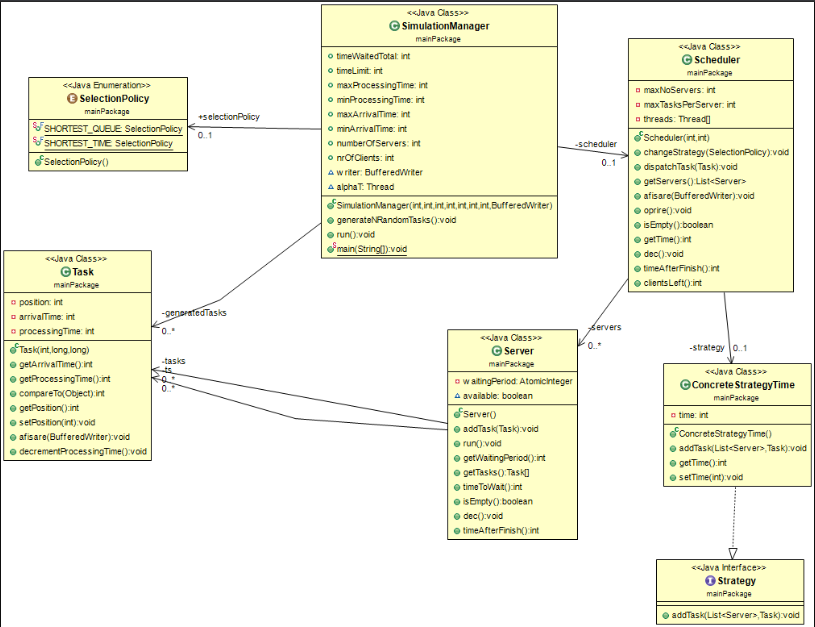
# *Proiectare*

***Decizii de proiectare***

Pentru proiectarea acestei teme vom urma urmatorii pasi :

* Mai intai vom transcrie template-urile de cod puse la dispozitie in slideurile despre programarea concurenta puse la dispozitie de catre profesor
* Apoi vom completa acele templetauri ajutandu-ne de comentariile care se afla in acesta .
* Dupa vom observa cum functioneaza programul si il vom modifica dupa propria viziune asupra problemei pentru o executie eficienta si pentru a putea calcula AvgWaitingTime-ul .

***Diagrama UML***



# *Implementare*

***Pachetul mainPackage***

***Clasa Server***

Aceasta clasa este clasa care va fi folosita pentru a genera Threadul . Are ca si argumente o coada si un AtomicInteger waiting Period care reprezinta numarul de clienti din coada la un moment dat .

Aceasta clasa este de fapt coada care va primi si scoate clienti .

Ca si metode avem :

***public Server ()***

* Acesta este constructorul clasei care initializeaza Queue-ul si waitingPeriod-ul .

***.***

***public void addTask(Task newTask)***

* Metoda care adauga un Task in in coada de taskuri si incrementeaza waitingPeriod.

***public boolean isEmpty()***

* Returneaza true in cazul in care lista este goala si false altfel

***public void dec()***

* Decrementeaza timpul de procesare al clientului din capul cozii , este apelata de simulation o data la o secunda.

***public int timeToWait()***

* Metoda timeToWait returneaza waiting time-ul unei cozii detinute de serverul respectiv . Este folosita cand dorim sa inseram un client in coada cu cel mai mic WaitingTime pentru a maximiza eficienta .

***public int timeAfterFinish()***

* Aceasta metoda este folosita pentru a verifica timpul total de asteptare a tuturor clientilor care se afla in coada ( de exemplu clientul 1 are t1, clientul 2 are t2+t1, clientul 3 are t3+t2+t1 si asa mai departe ) .
* Folosesc aceasta metoda pentru scade clienti neprocesati la finalul perioadei de timp .

***public void run()***

* Metoda run este metoda care v-a fi apelata la creeare unui Thread avand ca si parametru un Server .
* Aceasta metoda are o bucla infinita in care se verifica daca elementul din capul cozii are waitingTime-ul 0 , iar atunci acesta este eliminat din coada de Taskuri si se va decrementa waitingPeriod-ul care este de fapt numarul de clienti de la coada .

***public Task[] getTasks()***

* Returneaza un Array care cuprinde toate taskurile care se afla in coada la momentul respectiv.

***Clasa Scheduler***

Aceasta clasa are ca si argumente o lista de Servere, numarul maxim de servere , numarul maxim de Taskuri per Server si strategia aleasa. in cazul nostru avem o singura strategie cea care pune pe primul loc Timpul ! .

***public Scheduler(int maxNoServers, int maxTasksPerServer)***

* Acesta este constructorul clasei ce instantiaza un nr de maxNoServers de servere si creeaza iar apoi porneste Threadurile aferente pentru aceste servere

***public void dispatchTask(Task t)***

* Aceasta metoda adauga un task la unul din Threaduri , este trimis mai departe Taskul clasei ConcreteStrategyTime .

***public void afisare(BufferedWriter wr)***

* Aceasta metoda primeste ca si parametru un BufferedWriter care este de fapt referinta catre fisierul nostru in care scriem .
* Metoda noastra va scrie starea tuturor queueurilor la un moment dat cand este apelata de catre SimulationManager .

***public void oprire()***

* Aceasta metoda opreste toate Threadurile create pe baza listei de Servere .

***public void dec()***

* Metoda aceasta apeleaza metoda dec din clasa Server .

***public int clientsLeft()***

* Aceasta metoda returneaza numarul de clienti ramasi neserviti in cazul in care timpul depaseste timpul maxim de simulare .

***public int timeAfterFinish()***

* Aceasta metoda apeleaza meotda timeAfterFinish din clasa Server .

***Clasa Task***

In aceasta clasa avem ca si argumente principale arrival Time-ul , processing Time-ul dar si position , argument adaugat de mine pentru a vedea al catalea client este .

Ca si metode avem constructorul , getter setter si suprascrierea metodei CompareTo pentru a compara in obiectele de tipul Task in functie de Arrival Time .

***Clasa SimulationManager***

Ca si argumente ale acesti clase avem : timeLimit care este timpul maxim pe care il poate avea simulare ; max si min Processing time care este intervalul in care trebuie sa se afle processing time-ul nostru ; min max Arrival time este la fel intervalul de timp in care se situeaza Arrival time-ul fiecarui client ; numberOfServers – numarul de servere ; si numberOfClients numarul total de clienti pe care il vom genera .

Ca si metode avem : Constructorul care intializeaza toate variabilele de mai sus cu valorile primite din functia main . Alte metode :

***public void generateNRandomTasks()***

* Aceasta metoda genereaza nrOfClients taskuri care indeplinesc conditiile impuse de arrival time si processing time , si apoi le sorteaza si pune intr-o lista .

***public void run()***

* Aceasta este Metoda run care va fi apelata atunci cand se va porni Threadul avand ca si parametru o instanta a clasei SimulationManager .
* Aceasta este clasa principala care controleaza toate actiunile acestei simulari .
* Itereaza prin lista de Clients in asteptare atat timp cat currentTime-ul este mai mic decat time limit-ul simularii .
* Pentru fiecare Client daca arrivalTime-ul lui este >= currentTime-ul , acesta este adaugat intr-una din cozi ( coada in care waitingTime-ul este cel mai mic pentru o eficientizare a procesului de procesare ) .
* Dupa se va apela afisare , si sleep-ul de o o mie de milisecunde care face ca aceasta metoda de run sa se opreasca pentru o secunda
* Dupa terminarea procesarii tuturor clientilor sau dupa ce currentTime-ul depaseste timeLimite-ul metoda aceasta de run v-a calcula Average Waiting Time-ul si il va afisa , dupa care v-a inchide toate Threadurile deschise de aceasta si

***public static void main(String[] args) throws IOException***

* In main se vor deschide cele 2 fisiere pentru citire respectiv scriere , se va citit din fisierul de citire constrangerile simularii , se va instanta clasa SimulationManager si se va crea un Thread al acesteia care v-a fi pornit .

***public class ConcreteStrategyTime implements Strategy***

* Aceasta clasa are un singur atribut , time care ne va ajuta la calcularea timpului mediu de asteptare la coada.
* Are o singura metoda ( pe langa getter si setter ) , metoda de addTask care itereaza prin lista de servere si gaseste serverul al calui Queue are timpul de asteptare minim , si adauga la acel server Taskul t , dar si adauga timpul de asteptare a lui t la time , timpul total de asteptare .

***Alte clase :***

* ***Ca si alte componente avem SelectionPolicy care este o enumerare ( in care pana la urma am lasat un singur elemnt cel de ConcreteStrategyTime ) si Interfata Strategy cu metoda addTask .***

# *Rezultate*

Am reusit sa fac atat simulare pentru cele 3 fisiere , dar am si reusit sa gasesc o metoda Thread Safe pentru a computa Average Waiting Time-ul , atat in cazul in care programul se opreste dupa procesarea tuturor clientilor cat si in cazul in care programul se opreste deoarece a ajuns la timeLimit .

Ca si rezultate dupa testele pe cele 3 fisiere de intrare vom avea 3 fisiere de iesire in care avem un format de genul :

Time : t

Queue 1 : (a,b,c)..

…..

Queue n : (x,y,z)..

….

Iar la finalul fisierului dupa calculare se va scrie Average Waiting Time-ul care este media Timpului pe care fiecare client il petrece in “magazin “ adica la coada .

# Concluzii

# 

Consider ca in aceasta tema am invatat sa lucreaz mult mai bine cu listele si cozile din Java , am invatat ca e bine sa am un iterator care sa ma ajute sa iterez prin coada / lista . Am invatat ce este programarea concurenta , cum functioneaza aceasta . Cum functioneaza in paralel Threadurile pentru o executie mai rapida a programalui , cum pot fi acestea oprite sau puse pe pause pentru un anumit timp sau pana vor fi notificate de catre alt Thread . Cum sa fac Thread Safe folosirea unei resurse comune de catre mai multe Threaduri . Am invatat si sa citesc si sa scriu din fisiere , aici am testat mai multe metode si le-am ales pe cele care mi s-a parut cel mai usor de folosit si inteles . Si sa generez fisierul . Jar si sa Apelez programul din linia de comanda dandu-I argumentele fisiere.

Ca si posibilitate de Dezvoltare a problemei. Se poate gasi o metoda mai usoara de computare a timpului mediu de asteptare , poate creearea unei interfete grafice unde se va putea vizualiza mult mai bine aceasta simulare , si de ce nu creearea unui sistem de programari care genera programari la ore cat mai optime pentru ca clienti sa isi termine Taskul la o firma cat mai repede .

# *Bibliografie*

<https://www.youtube.com/watch?v=sa-TUpSx1JA>

<https://www.youtube.com/watch?v=OitW-vPd1Yc&t=447s>