DOCUMENTATIE TEMA2

SIMULATOR DE COZI

Baleanu Sorina-Diana

Grupa 302210

Profesor laborator: Stoica Anda

# Cuprins:

1.Obiectivul temei

2.Analiza problemei

3.Proiectare

4.Implementare

5.Rezultate

6.Concluzii

7.Bibliografie

# 1.Obiective

## 1.1.Obiectivul principal

Obiectivul principal al acestui proiect este realizarea unui aplicatii care consta in proiectarea și implementarea unei aplicații de simulare care vizează analiza sistemelor bazate pe cozi pentru determinarea și minimizarea timpului de așteptare al clienților.

## 1.2.Obiectivele secundare

Obiectivele secundare constau in pasii urmati pentru atingerea obiectvelor principale.

|  |  |  |
| --- | --- | --- |
| Obiectiv secundar | Descriere | Capitol |
| Dezvoltarea de scenarii si cazuri de utilizare | In computere, un scenariu este o naratiune a interactiunilor dintre roluri ( actori in UML ) si sistem. In cazul nostru rolul - un utilizator al aplicatiei.  În inginerie software și sisteme, un caz de utilizare este o listă de acțiuni sau pași de eveniment care definesc de obicei interacțiunile dintre un rol (cunoscut în Unified Modeling Language (UML) ca actor) și un sistem pentru a atinge un obiectiv. Actorul poate fi un sistem uman sau alt sistem extern. | 2 |
| Structuri de date | Alegerea structurilor de date care au fost folosite pentru indeplinirea obiectivului principal | 2 |
| Proiectare clase | Alegerea claselor si a relatiilor intre ele | 3 |
| Algoritmi | Algoritimi folositi pentru implementarea proiectului | 3 |
| Implementarea solutiei | Se va descrie fiecare clasa alaturi de toate metodele implementate in aceasta | 4 |
| Testare | Testarea corectitudinii programului prin simularea a mai multor cazuri care difera in timp de simulare, numar de clineti si numar de cozi, de asemenea realizarea mai multor simulari pe aceleasi date deoarece timpii de procesare si sosire a clientilor sunt generati aleator. | 5 |

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

2.1.Analiza problemei

Proiectarea și implementarea unui sistem eficient de distribuire a clientilor in cozi,acest sistem modeleaza un domeniu din lumea reala si are ca scop minimizarea timpului de asteptare a clinetilor pana in momentul in care acesti pot fi serviti.

Operatiile care sunt efectuate de program sunt:

* Generarea a n clienti cu timp de ajungere si timpul de procesare aleator
* Generarea a unui numar Q de cozi
* Simularea servirii celor N clienti in cele Q cozi
* Timpul total in care fiecare coada proceseaza clientii care i sunt atribuiti
* Timpul mediu de procesare al clientilor de catre fiecare coada

2.2 .Scenarii si cazuri de utilizare

Descrierea functionalitatii programului

Pasul 1:Sunt citite din fisier datele corespunzatoare simularii, adica numarul de clineti, numarul de cozi , timpul minim si maxim de sosire al unui client, timpul minim si timpul maxim de procesare al unui client, timpul de simulare. In cazul in care calea fisierului nu este valida, se va arunca o exceptie si pe ecran se va afisa un mesaj de eroare.

Pasul 2 :Se vor genera cei N clienti, fiecaruia atribuindu-i-se un timp de sosoire si procesare aleator, care corespunde cu limitele impuse de programul in cauza. Clientii sunt apoi ordonati in functie de timpul de sosire in ordine crescatoare.

Pasul 3:Simularea porneste, iar in momentul in care timpul curent este egal cu timpul de sosire al unui client, acest client va fi introdus in una din cozi in functie de strategia curenta, strategia poate sa fie in functie de timpul total de asteptare al cozii sau in functie de numarul de clienti care sunt in coada.

Pasul 4:Clientii sunt procesati de fiecare coada in parte, iar in momentul in care un client din coada are timpul de procesare 0, acesta se considera servit si este eliminat din coada, urmatorul client dupa acesta fiind acum procesat de coada.

Pasul 5:Simularea se incheie in momentul in care timpul de simulare se scurge sau in momentul in care toti clientii au fost serviti cu succes. La finalul simularii se va calcula media de asteptare a clientilor pentru fiecare coada, cat si media timpului de asteptare pentru toti clientii generati, urmand ca datele rezultate in urma simularii sa fie scrise in fiserul de iesire

# 3.Proiectare

3.1.Decizii de proiectare

O importanta decizie de proiectare a fost utilizarea StringBuilder. Astfel pentru screierea rezultatului in fiser se va proceda in felul urmator: metoda run va adauga textul care se doreste a fi inserat in fiserul de iesire in obicetul de tip StringBuilder, iar la finalul metodei textul construit cu ajutorul acestei structuri de date va fi scris in fisierul de iesire. Aceasta decizie de proiectare ne va asigura ca nu vor exista desincronizari din cauza timpului mare care este necesar pentru accesarea spatiului de pe disk, unde se aflta fisierul de iesirire.

3.2.Diagrame UML

Unified Modeling Language (prescurtat UML) este un limbaj standard pentru descrierea de modele si specificatii software. Diagrama de clase UML Este folosită pentru reprezentarea vizuală a claselor și a interdependențelor, taxionomiei și a relațiilor de multiplicitate dintre ele. Diagramele de clasă sunt folosite și pentru reprezentarea concretă a unor instanțe de clasă, așadar obiecte și a legăturilor concrete dintre acestea.

A screenshot of text

Description automatically generated

3.3.Structuri de date

Una dintre structurile principale folosite in acest proiect este un CopyOnWriteArrayList de Task-uri in clasa SimulationManager. Pentru aceasta structura de date când utilizăm oricare dintre metodele de modificare - cum ar fi add () sau remove () - întregul conținut al CopyOnWriteArrayList și atunci când se întâmplă modificări simultane. este copiat într-o copie internă. Datorită acestui fapt , se poate itera peste listă într-un mod sigur. Prin alegerea acestei structuri de date, metoda run() a clasei SimulationManager functioneaza corespunzator,fiind posibila eliminarea unui Task in timpul iterarii, cand timpul de sosire al unui client este egal cu timpul curent.

O alta structura utilizata este LinkedBlockingQueue, . Această coadă comandă elemente FIFO (first-in-first-out). Capul cozii este acel element care a fost pe coadă cel mai mult timp. Coada cozii este acel element care a fost pe coadă în cel mai scurt timp. Elementele noi sunt introduse la coada cozii, iar operațiunile de preluare a cozii obțin elemente în capul cozii. Cozile legate de obicei au un randament mai mare decât cozile bazate pe tablă, dar performanțe mai puțin previzibile în majoritatea aplicațiilor concomitente. Utilizarea aceste structurii pentru a inmagazina obicetele de tip Task in clasa Server ne asigura ca procesul de MultiThreading este safe.

3.4.Proiectare clase si relatii intre clase

In realizarea cerintelor am folosit urmatoarele clase:

Clasa Task:

Aceasta clasa defineste un client, in alte cuvinte o sarcina care sa fie procesata de o coada. Fiecare instanta a clasei task este definita de un id, timp de procesare si timp de sosire.

Clasa Server:

Aceasta clasa reprezinta o coada, are in componenta sa o lista de sarcini care trebuiesc a fi procesate, de asemenea in aceasta clasa este implementata metoda run(), se va instantia un thread cu un server doar in momentul in care se adauga pentru prima data un element in Server, in cazul in care coada ramane fara elemente de tip Task, threadul care a fost pornit este oprit, urmand a se porni alt thread al severului respectiv doar in cazul in care un alt element de tip Task este adaugat in Server cu scopul de a se procesa de acesta.

Clasa Scheduler:

In clasa Scheduler se gaseste o lista de Servere. Aceasta clasa se ocupa cu creearea si gestionarea Serverlor, aici fiind creaate Serverele necesare simularii, de asemenea in aceasta clasa se alege strategia de distribuire a Task-urilor in Servere, in functie de timp sau in functie de numarul de Task-uri pe care le are fiecare coada.

Interfata Strategy:

Interfata Strategy descrie metoda abstracta de adaugare a unui Task intr-o coada, interfata este implementata de cele doua clase descrise mai jos ConcreteStrategyQueue si ConcreteStrategyTime.

Clasa ConcreteStrategyQueue:

Aceasta clasa implementeaza interfata Strategy si descrie metoda de adaugare a unui client in coada. Conform acestei strategii clientul va fi distribuit cozii care are minimul de clienti in asteptare.

Clasa ConcreteStrategyTime:

Aceasta clasa este de asemena o implemnetare a interfetei Strategy, in aceasta clasa metoda de adaugare de Task in Server va alege Serverul care are timpul minim de asteptare in momentul respectiv.

Clasa SimulationManager:

Aceasta clasa este de asemena o implementare a interfetei Runnable. Aceasta clasa se ocupa de citirea si screiera in fisier ,dar si de simularea propriu-zisa, metoda run ruleaza atata timp cat timpul de curent nu depaseste timpul de simulare precizat in fisierul de intrare. La fiecare secunda se cauta in lista de clienti generati un client care are timpul de sosire egal cu timpul curent, iar in cazul in care aceasta afirmatie este adevarata, clientul este atribuit unei cozi iar de aici mai departe el este procesat de Servarul caruia i-a fost atribuit. La finalul metodei run rezultatul se va scrie in fisierul de iesire.

3.5.Algoritmi

Adaugare unui Task:

In momentul in care se adauga un task intr un Server, se adauga task-ul in lista de Task-uri a Serverului, creste perioada de asteptare a cozii cu timpul de servire a Task-ului respectiv ,de asemenea creste si timpul total de servire a cozii, creste numarul de clienti care sunt serviti de coada si variabila isRunning devinde true asta insemnand ca Serverul are clienti de procesat.

Metoda run() din clasa Server:

Metoda run() din clasa Server va fi apelata in momentul in care Threadul creat pe baza obiectului Server va fi pornit. Aceasta metoda va rula atata timp cat coada este deschisa, adica are in ea clienti pe care trebuie sa ii proceseze. Algoritmul pe care il are la baza aceasta metoda executa urmatoarele operatii: mai intai va apela metoda peek() pe lista de Task-uri a serverului respectiv, ceea ce va permite manipularea Task-ului care se afla in capul structurii LinkedBlockingQueue fara a il elimina din structura, apoi va incerca sa intrerupa Threadul timp de o secunda(1000 de milisecunde), apeland metoda sleep(1000), daca intreruperea Threadului este executata cu succes metoda run continua prin decrementarea timpului de asteptare a cozii si decrementarea timpului de procesare al clientului ce se afla in capul cozii, la final se verifica daca timpul de procesare al clientului este egal cu 0, iar in cazul indelpliniiri aceste conditii, clientul este scos din coada, in cazul in care nu mai exista Task-uri in Server coada se va inchide pana la adaugarea de task-uri noi, astfel Threadul se va opri si se va creea unul nou in momentul in care un nou Task va fi adaugat.

Adaugarea unui Task (Strategie bazata pe timp):

Adaugarea unui task, cu strategia bazata pe timpul minim de asteptare are la baza urmatorul algroitm.Se va cauta in lista de Servere serverul care are timpul minim de asteptare. Dupa ce este gasit Serverul care are timpul cel mai mic de asteptare se verifica daca este activ sau nu, inainte de adaugarea task-ului in el, in cazul in care inainte de adaugarea task-ului Serverul este inactiv se va adauga task-ul si se va creaa un nou Thread care urmeaza sa fie pornit. In cazul in care inainte de adaugarea Task-ului Serverul este activ inseamna ca Threadul creeat pe baza Serverului respectiv este in curs de functionare, deci nu este necesara creerea unui nou Thread.

Adaugarea unui Task(Strategia bazata pe numarul minim de clienti din coada):

Se procedeaza ca si in cazul adaugarii uni Task in functie de timpul minim de asteptare al cozii, insa in cazul acestei strategii se va parcurge lista cu Servere si se va gasi Serverul care are cei mai putini clienti de procesat in momentul in care se incearca adaugarea Task-ului.

Generarea de Taskuri:

Generarea de N Task-uri se va face utilizand un obiect de tip Random, care ne va ajuta sa stabilim pentru fiecare client un timp de sosire si procesare aleator , iar dupa stabilirea acestor doua valori se va creea un nou obiect de tip Task.

Dupa ce se genereaza N astfel de obiecte de tip Task, se va ordona lista de Task-uri crescator dupa timpul de sosire al clientiilor, cu ajutorul metodei sort din clasa Collections care functioneaza pe baza coparatorului din clasa Task.

Calcularea timpului mediu de procesare clienti:

Fiecare coada este responsabila de proprii ei clienti, astfel fiecare coada isi va calcula timpul mediu de asteptare, deoarece in fiecare Server este contorizat timpul total de asteptare al clientiilor si numarul total de clienti care au fost procesati de coada. La final se va realiza de asemenea si o medie a timpului de asteptare pentru toate coziile cu ajutorul mediilor calculate de feicare coada.

Metoda run() din clasa SimulationManager:

Aceasta metoda ruleaza atata timp cat timpul curent este mai mic decat timpul de simulare si cat timp mai exista clienti in asteptare sau cozi care proceseaza clinenti.

Se cauta in lista de clienti in asteptare daca exista un client care are timpul de sosire egal cu timpul de asteptare, in cazul in care un astfel de client este gasit el este distrbuit unui Server in functie de strategia abordata si este eliminat din lista cu clientii care sunt in asteptare. Se va adauga la textul ce trebuie scris in fisierul de iesire timpul curent, clientii in asteptare si serverele, Threadul se va intrerupe timp de o secunda, iar apoi se va incrementa timpul curent si se va relua procedeul. In momentul in care conditiile mentionate mai sus nu mai sunt indeplinite se va adauga la textul de iesire timpul mediu de procesare, iar apoi se va scrie textul in fisier

3.6.Date de intratre si date de iesire

In cazul acestui proiect datele de intrare pe baza carora se va face simularea se vor citi dintr-un fisier. Care respecta un anume format. Conform formatului datele de intrare sunt citite si apoi se construieste un obiect de tipul SimulationManager corespunzator datelor de inrtare.

Se va instantia un Thread cu obicetul SimulationManager creeat si apoi Threadul este pornit, la finalul metodei run se va executa automat scrierea in fisiererul corespunzator de iesire.

# 5.Implementare

Clasa Task:

In aceasta clasa au fost implementate urmatoarele metode:

* Metodele de get si set ale atributelor specifice Task-urilor
* Metoda compareTo- aceasta metoda defineste modul in care se vor copara Task-urile, acestea vor fi comparate in functie de arrivalTime
* Moetoda toString-este definit modul de afisare al clientilor, mai intai se afiseaza id-ul dupa care timpul de sosire al clientului si apoi timpul de procesare

Clasa Server:

In clasa Server au fost implementate urmatoarele metode:

* Metoda de get si set pentru Servere
* Metoda de addTask() -defineste modul in care se adauga un Task in server si cum influenteaza adaugarea Taskului atributele Server-ului
* Metoda de calculare a timpului mediu de asteptare al Serverului -calculeaza media de asteptare impartind timpul total de asteptare al clientiilor din Serverul respectiv la numarul total de clienti care au fost procesati in Server
* Metoda run()-se ocupa de procesarea unui Task
* Metoda toString-folosita pentru a afisa toate Task-urile existenete in Server, in cazul in care exista sau va afisa „closed” in cazul in care in momentul respectiv coada nu proceseaza niciun client

Clasa Scheduler:

In clasa Scheduler sunt definite operatiile prin urmatoarele metode:

* Metodele de get si set pentru Scheduler
* Metoda dispatchTask- apeleaza metoda de adugare a unui Task impementata in ConcreteStrategyTime si ConcreteStrategyQueue,este apelata metoda in functie de modul in care a fost instatiat atributul de Strategy specific Schedulerului
* Metoda stillRunning- verifica daca Serverele sunt inchise sau inca proceseaza informatii
* Metoda averageTime- afiseaza timpul mediu pentru fiecare coada si calculeaza media pentru toate cozile
* Metoda toString- se afiseaza toate Serverele creeate

Clasa ConcreteStrategyQueue:

Clasa ConcreteStrategyQueue avem implementata interfata Strategy

* Metoda addTask- implementeaza metoda abstracta din interfata Strategy si alege Serverul in care este adaugat Task-ul folosind o strategie bazata pe numarul clientilor din cozi.

Clasa ConcreteStrategyTime:

Clasa ConcreteStrategyTime avem implementata interfata Strategy

* Metoda addTask- implementeaza metoda abstracta din interfata Strategy si alege Serverul in care este adaugat Task-ul folosind o strategie bazata pe timpul minim de asteptare al coziilor

Clasa SimulationManager:

* Metoda generateNRandomTasks- genereaza un numar N de Task-uri cu timp de sosire aleator cat si cu tipmp de procesare aleator,aceasta metoda este apelata in constructorul clasei SimulationManager
* Metoda run- se apelaza in momentul in care se apeleaza metoda start pentru Threadul creat cu SimulationManagerul corespunzator simularii, de asemenea aceasta metoda construieste textul care este scris in fisierul de iesire
* Metoda writeOutputInFile- se ocupa cu scrierea in fisier a datelor obtinute in urma simularii.

# 6.Rezultate

Am realizat simulari cu diferite date de intrare, rezultatele sunt mai usor de observat pentru date de intrare cat mai mici.

Fiserele cu datele de iesire sunt atasate in proiect.

# 7.Concluzii

In concluzie realizarea acestui proiect a fost foarte utila , am invata despre concurenta in java , am folosit pentru prima data thread-urile intr-un proiect, modeland in acelasi timpu un program care are aplicabilitate in viata reala. De asemenea , am realizat pentru prima data un program care executa operatia de citire si scriere intr-un fisier in limbajul de programare Java.

O dezvoltare ulterioara care ar putea fi facuta ar putea fi creerea spre exemplu a unei cozi rapide, astfel clientii cu timpul de procesare mic sa nu fie nevoiti sa astepte dupa clientii cu timp de procesare foarte mare si in acest fel timpul mediu de procesare al tuturor clientilor ar putea sa scada semnificativ.

Alte dezvoltari ale aplicatiei ar putea fi realizarea unui mod de a schimba in timpul rularii strategia abordata calculand care strategie este mai avantajos de aplicat in timpul respectiv.

# 8.Bibliografie

* <https://www.geeksforgeeks.org/linkedblockingqueue-class-in-java/>
* <https://www.geeksforgeeks.org/stringbuilder-class-in-java-with-examples/>
* <https://stackoverflow.com/questions/1677194/dumping-a-java-stringbuilder-to-file>
* <http://coned.utcluj.ro/~salomie/PT_Lic/4_Lab/Assignment_2/Java_Concurrency.pdf>
* <https://www.tutorialspoint.com/java/java_multithreading.htm>