DOCUMENTATIE TEMA 2

SIMULAREA COZILOR

Student: Moldovan Teodora
Grupa: 30225
Facultatea de Automatica si Calculatoare
Profesor Laborator : Antal Marcel

Cuprins

1. Obiective

1.1. Obiectiv Principal

Aceasta tema presupune realizarea si implementarea unei aplicatii care sa aiba ca scop analiza sistemelor bazate pe cozi pentru a determina si minimiza timpul de asteptare al clientiilor.

Pentru realizare aplicatiei si a simularii sunt necesare cateva date de intrare cum ar fi intervalul de timp in care poate fi procesat un client (valoare minima si maxima), intervalul de simulare si numarul de cozi care sunt disponibile pe perioada acestui interval.

1.2. Obective Secundare

Obiectiv Secundar	Descriere	Capitol
Analiza problemei si presupuneri	Se analizeaza diferite moduri de abordare a problemei propuse.	2
Alegerea structurilor de date	Prezentarea structurii de date alese si a argumentelor	3 (3.1)
Impartirea pe clase	Diagrama de clase si descriere a functionalitatii claselor	3 (3.2), 4
Dezvoltarea algoritmilor	Propunerea unor metode de implementare a a algoritmilor pentru operatiile propuse	3
Implementarea solutiei	Detalii referitoare la implementare in urma analizei realizate	4
Testare	Rezultatele catorva teste realizate pentru verificarea corectitudinii	5

2. Analiza Problemei

Utilizatorul trebuie sa poata introduce datele pe care le doreste pentru simulare (limitele – minim si maxim – a timpului de procesare pentru un client, numarul de cozi, durata simularii). De asemenea, aceste valori ar trebui sa reprezinte numere. Pentru intervalul de simulare si limitele timpului de procesare se vor introduce valori astfel incat o unitate sa reprezite o secunda. In cazul in care nu se introduc aceste valori sau se introduc valori care nu sunt numerice, aplicatia ar trebui sa afiseze un mesaj de eroare, care sa instiinteze utilizatorul ca nu a introdus corect datele, fara sa treaca la pasul urmator al rularii programului.

La momentul rularii aplicatiei, aceasta va trebui sa afiseze pasii de simulare, mai exact momentul cand un client ajunge la coada sau este procesat. Pentru usurinta in urmarirea corectitudinii simularii, ar trebui afisat si timpul curent al simularii.

In partea de simulare se va genera o lista de clienti care apoi vor fi distribuiti in ordinea timpului de sosire la cate o coada, in functie de strategia aleasa pentru simulare.

Pentru strategii se vor considera doua cazuri. In primul caz, clientii vor fi distribuiti la cozi in functie de timpul de asteptare la coada respectiva, adica un client nou sosit va fi distribuit la coada care are cel mai mic timp de asteptare. In al doilea caz, clientul care este nou sosit va fi distribuit la coada care are cei mai putini clienti in asteptare.

Cu scopul de a putea compara rezultatele simularii in mai multe cazuri, la terminarea acesteia se vor afisa date despre timpul mediu de asteptare, timpul mediu de procesare, timpul la care coziile au fost cele mai pline si timpul total in care coziile au fost goale.

Multithreading-ul reprezinta executarea concurenta a mai multor threaduri, aparent in acelasi timp. Problema gestiunii cozilor este una de o complexitatea destul de ridicata deoarece pot fi aplicate o serie extrem de larga de operatii asupra acestora.

3. Proiectare

3.1. Structuri de date

Una dintre primele probleme aparute in faza de proiectare presupune alegerea unor structuri de date corespunzatoare modului de lucru cu clasele si stocarii anumitor valori pe perioada efectuarii operatiilor dorite.

In implementarea acestei probleme s-a ales utilizarea a doua structuri de date principale: ArrayList si ArrayBlockingQueue.

Un ArrayList este o colectie simpla care poate stoca orice tip de obiect, fapt ce permite utilizarea ArrayList-ului pentru memorarea clientiilor generati in partea de simulare. Acest lucru este necesar, deoarece clientii trebuie stocati pe perioada dintre timpul la care acestia sunt generate si pana sunt introdusi in coada. In momentul in care timpul de sosire al clientului si timpul simularii sunt egale, se va adauga clientul la o coada, in conformitate cu strategia de distributie aleasa, iar apoi va fi eliminat din lista pentru a nu se introduce un client de mai multe ori.

Tot un ArrayList este folosit si pentru a stoca toate coziile existente, deoarece numarul de cozi nu este unui fix (este dat de catre utilizator la inceput, prin intermediul interfetei utilizator).

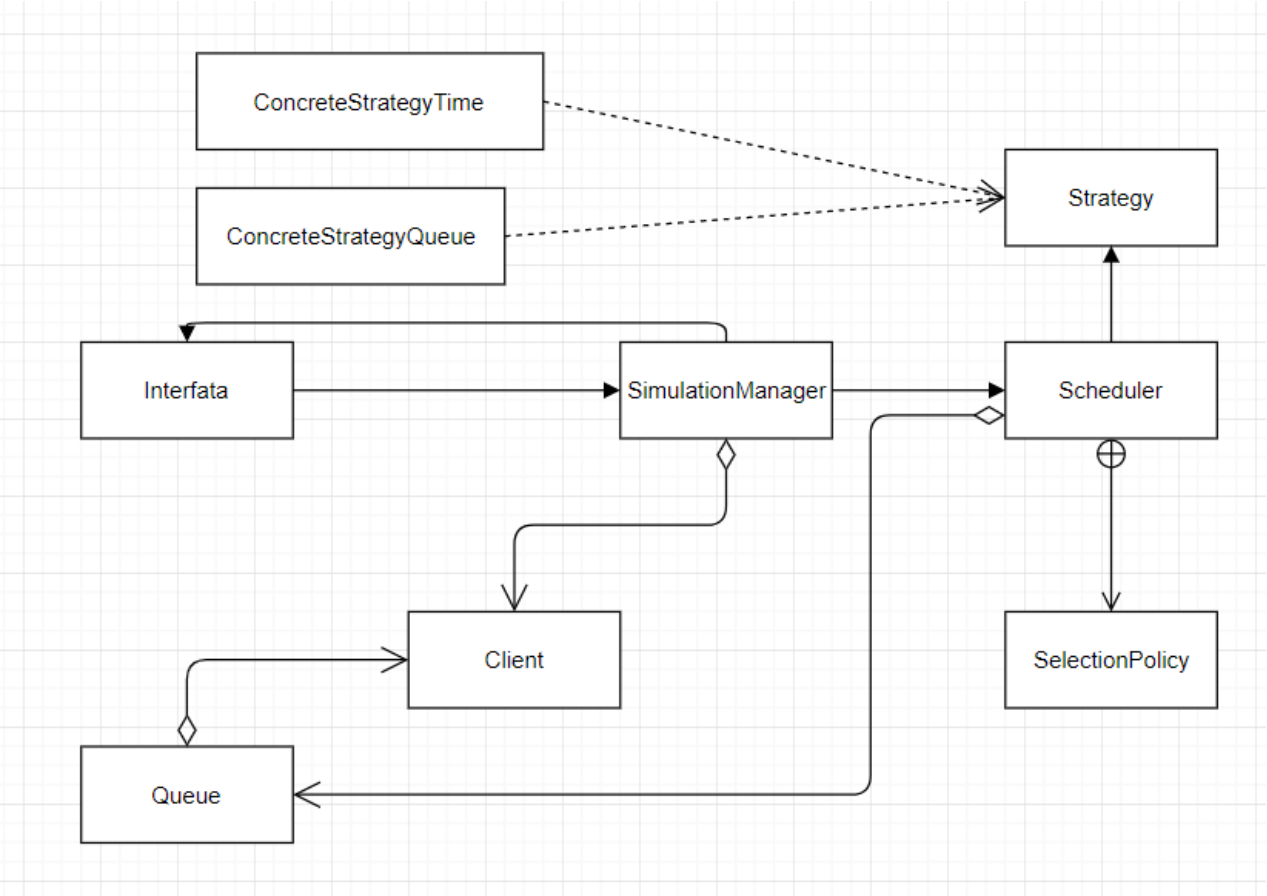
Aceasta clasa permite dimesiuni variabile ale sirului, crescand ca marime la adaugarea fiecarui element nou.

Parcurgerea unui ArrayList este mai usoara, deoarece se pot folosi bucle for-each. De asemenea, clasa ArrayList dispune de anumite metode care faciliteaza anumite operatii necesare pentru realizarea proiectului cum ar fi: add (Element e), indexOf (Element e), , get(int index), size(), isEmpty().

Al doilea tip de structura de date folosita este ArrayBlockingQueue. Se utilizeaza aceasta colectie, deoarece este o colectie concurenta , fapt ce contribuie la utilizarea sigura a threadurilor. Aceasta colectie este utilizata pentru a stoca clientii care se afla la un anumit moment intr-o anumita coada.

Colectia ArrayBlockingQueue ordoneaza elementele dupa principiul FIFO (First-In-First-Out). Primul element este cel care a apartinut cozii cea mai lunga perioada, iar ultimul element este ultimul care a fost introdus la coada. Acest principiu este strict necesar implementarii acestei aplicatii, deoarece este de asemenea si principalul mod de functionare a unei cozi din lumea reala. Incercarile de a adauga la o astfel de colectie care este plina sau de a extrage un element dintr-o coada goala, vor fi blocate. Metodele cele mai utilizate in proiect din implementarea acestei clase sunt : peek(), care permite extragerea anumitor informatii despre primul client din coada fara a-l extrage cu totul, si take() care extrage varful cozii (clientul care a fost adaugat de cea mai multa vreme).

3.2. Diagrama de clase



3.3. Algoritmi

La baza algoritmiilor utilizati sta principiul de lucru cu cozi si anume principiul FIFO (First-In-First-Out). Primul client care se adauga la coada este si primul care va iesi din aceasta, altfel spus, clientul care se afla in coada de cea mai lunga perioada este cel care va iesi primul. Ultimul client adaugat este cel care va iesi ultimul.

Pentru a putea decide la ce coada va intra un client este nevoie de stabilirea unei strategii. Doua astfel de strategii iau in considerare numarul de clienti la o coada sau timpul de asteptare la o coada. Daca se ia in considerare timpul de asteptare, clientul va fi adaugat la coada la care toti ceilalti clienti din fata au timpul insumat de procesare cel mai mic. Aceasta este coada care, din punct de vedere teoretic se va goli prima data. In celalalt caz, in care se ia in considerare numarul de clienti de la coada, clientul nou venit va fi adaugat la coada cea mai scurta, adica care are cel mai mic numar de clienti.

4. Implementare

Metoda de implementare aleasa contine 4 pachete:

- tema2_simulatingqueue: in care se gaseste clasa care contine interfata aplicatiei si controlul acesteia (aceasta contine si metoda main)
- strategie: contine o interfata Strategy, implementata de doua clase ConcreteStrategyTime si ConcreteStartegyQueue, fiecare continand o strategie de distribtie la cozi
- sistem: contine sistemul efectiv pe care il simulam: clasa Client, clasa Queue si clasa Scheduler
- simulation: contine clasa SimulationManager care este responsabila de generarea listei de clienti si implementarea proriu-zisa a simularii

Clasa Interfata reprezinta implementarea interfetei cu utilizatorul. In prima fereastra care apare, s-au folosit 5 campuri JTextField pentru ca utilizatorul sa poata introduce datele dorite:

- numarul total de clienti. Aceasta reprezinta numarul total de clienti care vor fi generati la inceputul simularii. Fiecare va avea un timp la care ajunge la coada si un timp de procesare.
- Intervalul de procesare(minim, maxim) reprezinta limitele timpului in care un client poate fi procesat
- Numarul de cozi
- Timpul de simulare

Datele trebuie introduse sub format numeric, iar in cazul in care campul respectiv reprezinta o perioada de timp se va considera o unitate ca fiind egala cu o secunda. Orice alt tip de data introdus, decat cel numeric, va genera un mesaj de eroare pentru a instiinta utilizatorul ca nu ii este permis sa introduca astfel datele. Executia nu se va continua pana cand datele nu sunt introduse corect. Pentru simplificare, campurile interfetei sunt initializate la anumite valori pentru a da o sugestie de utilizare.

Pentru alegerea strategiei dorite se folosesc doua butoane JRadioButton. Alegerea folosirii acestui tip de butoane rezulta din necesitatea ca o singura strategie sa fie luata in considerare pe parcursul unei simulari.

Pentru transmiterea semnalului de incepere a simularii s-a folosit butonul “Simulate”. Daca datele au fost introduse corect, se va incepe simularea si va aparea urmatoarea fereastra a aplicatiei. Aceasta va contine un numar de coloane egal cu numarul de cozi si un camp care va indica timpul curent al simularii.

La terminarea simularii, pe ecran se vor afisa date statistice despre simulare: timpul mediu de asteptare, timpul mediu de procesare, timpul la care a fost cel mai aglomerat si timpul cat cozile(toate) au fost goale(in secunde).

Clasele ConcreteStrategyTime si ConcreteStrategyQueue reprezinta cele doua strategii pentru care se poate executa simularea si pentru care s-au prezentat deja algoritmii. Ambele clase implementeaza interfata Strategy.

Clasa Client reprezinta clientii care vin si care trebuie distribuiti la cozi. Acestia au un timp la care ajung la coada si un timp de procesare.

Clasa Queue este o coada care contine o lista de clienti si care functioneaza pe principiul FIFO (s-a folosit ArrayBlockingQueue). Aceasta clasa implementeaza interfata Runnable , ceea ce indica faptul ca la crearea unei noi cozi aceasta va fi un thread care se va executa separat de celelalte.

Clasa Scheduler este responsabila pentru distribuirea clientiilor la cozi. Aceasta verifica care este strategia dorita si adauga in mod corepunzator clientul la o coada. Pentru aceasta , clasa contine un ArrayList cu toate coziile.

Clasa SimulationManager este responsabila pentru executarea simularii. La inceput se genereaza o lista de clienti, care vor fi ordonati dupa timpul de ajungere la coada. In functie de valoarea timpului curent al simularii si de timpul de ajungere a unui client, acesta va fi adaugat la coada. La atingerea timpului limita al simularii se verifica daca mai sunt clienti in cozi si se prelungeste timpul de simulare cu valoarea celui mai lung timp de asteptare dintre cele ale cozilor, astfel incat toti clientii care au ajuns intr-o coada sa fie procesati.

5. Testare si Rezultate

Pentru a facilita testarea rularii aplicatiei, campurile din interfata au fost presetate la niste valori implicite, avand atat rol orientativ pentru utilizator cat si rolul de a putea rula aplicatia spre exemplificare mai rapid.

Primul pas in testarea corectitudinii aplicatiei este verificarea daca datele de intrare sunt introduse corect de catre utilizator si apoi preluate corect pentru folosire.

	Date de intrare	Rezultat asteptat	Rezultat obtinut	PASS/FAIL
Numar clienti	20	20	20	PASS
Interval minim	1	1	1	PASS
Interval maxim	5	5	5	PASS
Numar cozi	3	3	3	PASS
Timp de simulare	20	20	20	PASS

Pentru verificarea ulterioara a functionarii sistemului de distribuitie la cozi si a evenimentelor care se petrec, fiecare coada este reprezentata in interfata grafica prin intermediul unei coloane. In partea de sus este afisat timpul curent de simulare sub formatul minute : secunde (chiar daca valoarea timpului de simulare s-a introdus in secunde). La introducerea unui client in coada, daca coada nu este goala, timpul de asteptare (afisat la baza fiecarei cozi; “Total Wait”) al cozii va creste (se aduna timpul de procesare al noului venit la timpul celor dinaintea sa).

6. Concluzii si Dezvoltari Ulterioare

In concluzie, aceasta aplicatie are rolul de a simula doua moduri in care se poate face distribuitia la un numar dat de cozi, avand aplicabilitate pentru situatii din lumea reala cum ar fi distributia clientiilor intr-un magazin la cozi pentru a minimiza timpul de asteptare al acestora sau distributia unor task-uri mai multor servere.

Aplicatia poate fi dezvoltata ulterior prin adaugarea unui alt algoritm mai eficient de distributie sau prin luarea in considerare ca la una dintre cozi cel care proceseaza informatia (serverul, vanzatorul) are o viteza diferita de ceilalti, informatia fiind procesata mai rapid la unele cozi decat la altele. O alta imbunatatire ar putea fi facuta la realizarea interfetei cu utilizatorul.

7. Bibliografie

https://docs.oracle.com/javase/7/docs/api/java/util/ArrayList.html

https://docs.oracle.com/javase/7/docs/api/java/util/concurrent/ArrayBlockingQueue.html

http://coned.utcluj.ro/~marcel99/PT/Tema%202/Java%20Concurrency.pdf