DOCUMENTATIE TEMA 2

ROTARU IONUT

Seria B, gr. 302210

**CUPRINS**

1. Obiectivul temei. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .3
2. Analiza problemei, scenario, cazuri de utilizare . . . . . .. 4
3. Decizii de proiectare. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 4
4. Implementare. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 5
5. Rezultate. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 8
6. Concluzii. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 9

1. **Obiectivul temei**

Cerinta temei este crearea unui proiect care implementeaza o simulare a unui sistem de client si cozi. Numarul de client, numarul de cozi, durata de simulare, timpii de sosire si timpii de procesare a fiecarui client se citesc dintr-un fisier, iar date precum starea si continutul cozilor, numarul de clienti in asteptare, timpul la care se afla simularea se scriu intr-un alt fisier.

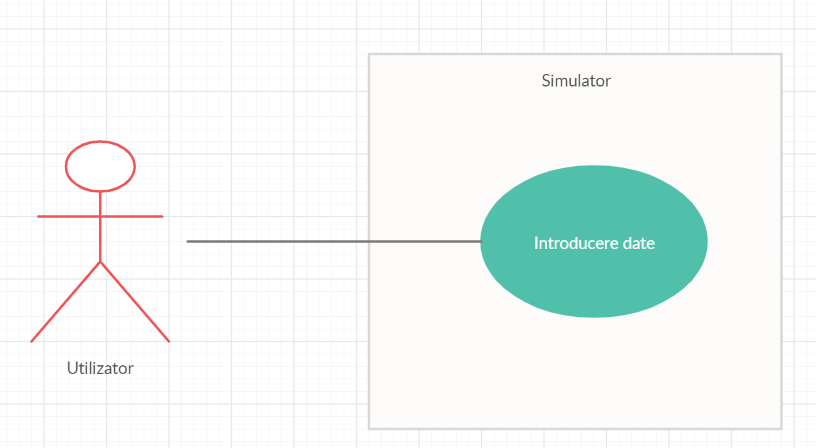
Proiectul trebuie sa poata fi accesat din terminal, iar numele fisierelor cu date de intrare, respective iesire sunt trimisi ca argumente in linia de comanda.

Implementarea trebuie facuta astfel incat fiecare client este asezat la coada cu cel mai mic timp de astepare. Simularea se termina fie cand nu mai exista clienti in cozi sau in asteptare, fie cand se termina timpul alocat simularii.

De asemene, se cere si afisarea timpuli mediu de asteptare.

|  |  |  |
| --- | --- | --- |
| Obiectiv secundar | Descriere | Capitol |
| Utilizarea diagramelor si descrierilor use-case | Se prezinta cerintele functionale si cazurile de utilizare | 2 |
| Decizii de proiectare, folosirea structurilor de date, diagrame UML, proiectare de clase, | Pasii urmati inainte de a trece la implementarea efectiva a problemei | 3 |
| Impartirea pe clase | Programul este structurat pe clase dupa anumite criterii | 3 |
| Dezvoltarea algoritmilor | Pentru fiecare operatie principala sau secundara s-a dezvoltat un algoritm | 4 |
| Implementare simulare | Folosirea thread-urilor | 4 |
| Testarea tuturor operatiilor | Verificarea corectitudinii simularii si a datelor de iesire | 5 |

1. **Analiza problemei, modelare, scenarii, cazuri de utilizare**



Use case: Pornire simulare:

• Primary Actor: Student

• Main Success Scenario:

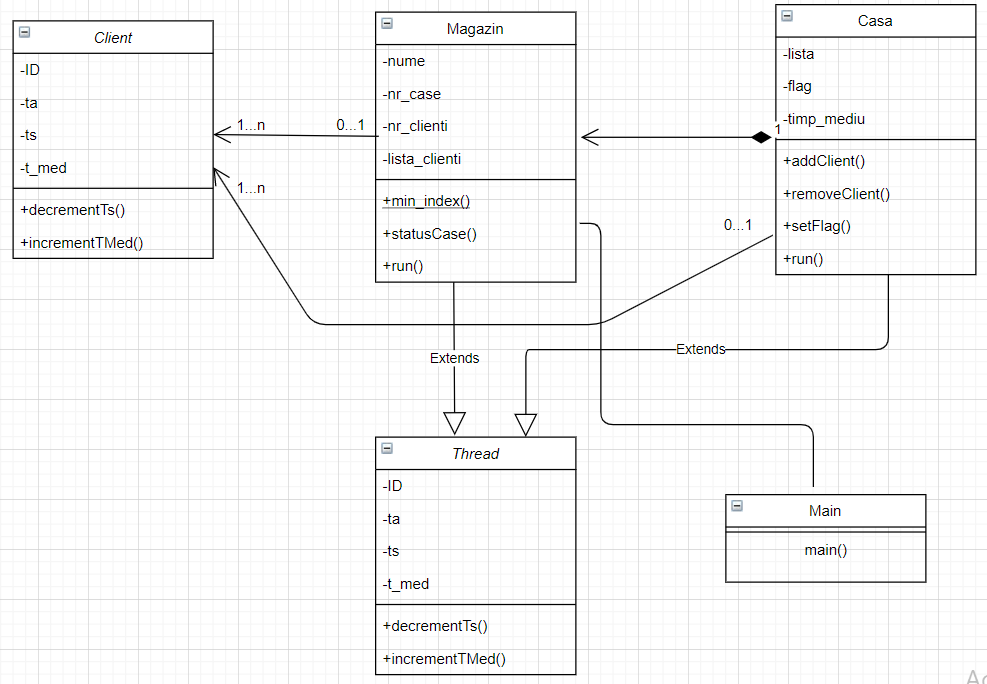
1. Utilizatorul introduce datele simularii;
2. Simulare porneste afisand pe ecran un timer;
3. La fiecare pas se scriu date in fisierul de iesire;
4. Simularea se opreste cand se termina timpul alocat/s-au procesat toti clientii;
5. Rezultatul simularii poate fi citit din fisierul de iesire

**3.Decizii de proiectare, structure de date, clase, algoritmi**

Modul de functionare a simularii este asemanator unui magazin cu mai multe case care se inchid si deschid, in funcie de numarul de client, iar clientii aleg sa stea la coada cu timpul de asteptare cel mai mic. Pentru a fi posibil acest lucru in prgramarea orientate pe obiect, se folosesc thread-uri. Acestea permit functionarea simultana a mai multor fire de lucru si au posibilitatea de intrerupere (sleep), ceea ce este de folos intr-o simulare in timp.

Thread-urile reprezenta cozile, care vor fi controlate de un alt thread, magazinul, care va asigura si partea de timer a sistemului.

Asadar, cele patru clase create sunt Client, Coada, Magazin si Main. Diagrama UML de clase este urmatoarea :



**4.Implementare**

**a)** **Clasa Client:**

In aceasta clasa se stocheaza informatii despre fiecare client: ID, timpul de sosire (ta), timpul de procesare (ts) si timpul total de asteptat la coada ( t\_med ). Fiecare client are informatii generate aleator, dar respecta limitele din fisierul de intrare:

Ta\_min <= ta <= ta\_max

Ts\_min <= ts <= ts\_max

ID-ul nu este genereat aleator pentru a nu se repeta, ci reprezinta un index care porneste de la 0 si se incrementeaza atunci cand un nou client trebuie creat.

Majoritatea metodelor din aceasta clasa sunt de tip get, fiind o clasa destinata stocarii informatiilor. Alte metode sunt:

-decrementTs(): scade cu o unitate variabila ts. Aceasta metoda este apelata atunci cand clientul se afla pe prima pozitie a cozii, adica este procesat, iar timerul se incrementeaza.

-incrementMed(): incrementeaza timpul de asteptare al clientuluicuo unitate. Aceasta metoda este apelata atat cand clientul este procesat, cat si atunci cand se afla la coada, dar nu este primul.

**b)** **Clasa Casa: . . .. . .**

Cum am specificat mai sus, aceasta clasa extinde clasa Thread. Variabilele folosite sunt;

-flag, de tip boolean (daca este setat pe true, threadul functioneaza, iar daca este setat pe false, threadul moare);

- timp\_mediu: timpul total de asteptare la aceasta casa in in tot timpul simularii; valoarea sa se obtine prin adunarea timpului mediu al clientului din capul cozii, inaine de a fi sters;

-lista: o variabila de tip Queue, in care se adauga clientii repartizati casei respective, tipul Queue fiind cel mai bun mod de reprezentare a unei cozi de magazine;

Metodele clasei Casa sunt:

-addClient(Client c):adauga un client in coada;

-removeClient(Client c):scoate clientul din capul cozii;

-lungime\_coada(): returneaza numarul de client din coada;

-setFlag(Boolean): seteaza flagul pe true sau pe false;

-getFlag(): returneaza starea flagului;

-getLista(): returneaza lista cu client dincoada;

-getMed(): returneaza timpul mediu de asteptare al casei;

Functionarea threadului:

Metoda run este principal metoda a acestei clase, fiind specifica threadului. Aici au loc operatiile de procesare a clientilor. Cat timp flagul este setat pe true, are loc urmatorul process: este scos clientul din capul cozii cu ajutorul metodei removeClient(); cat timp timpul lui de procesare este mai mare decat 0, ts al clientului este decrementat (metoda decrementTs()), iar pentru fiecare client din coada, timpul mediu de asteptare este incrementat (metoda incrementTmed()). Tot in aceasta bucla while este apelata metoda sleep(100), reprezentatnd incrementarea timerului. Atunci cand ts ajunge egal cu 0, clientul este scos definitive din coada, iar timpul mediu de asteptare al casei este actualizat, adaugandu-se timpul mediu de asteptare al clientului tocmai sters.

**c) Clasa Magazin**

Aceasta clasa controleaza modul de functionare a celorlalte doua: creeaza Clientii de care are nevoie si porneste sau opreste cozile.

Variabilele clasei sunt:

-casa: un vector de obiecte de tip Casa in care se stocheaza cozile necesare simularii;

-nr\_case: numarul de cozi alsimularii;

-ID: un indextrimis ca parametru la crearea clientilor;

-t\_sim: timpul de simulare;

-nr\_clienti: numarul de client pentru simulare;

-tamin, tamx, tsmin, tsmax: parametrii minimi si maximi pentru timpii de sosire si timpii de procesare a clientilor;

-lista\_clienti: un ArrayList in care se salveaza clientii creati;

-nume\_out: numele fisierului de iesire;

Metodele clasei Magazin:

-constructorul: primeste informatiile simularii din fisierul de intrare;

-min\_index(): cauta si returneaza casa la care sa fie trimis urmatorl client (cea cu timpul de asteptare cel mai mic);

-statusCase(): daca toate casele sunt goale, returneaza 0, altfel returneaza 1. Aceasta valoare reprezinta unul dintre criteriile de oprire a simularii;

-setNumeOut(String): seteaza numele fisierului de iesire;

Functionarea threadului:

La fel ca si la clasa Casa, metoda run() implementeaza algoritmul de functionare a threadului, prin urmatorii pasi:

- creaza fisierul de iesire cu numele corespunzator;

- creaza obiecte Client cu date aleatoare si salvarea lor in lista\_clienti;

- porneste casele si le seteaza flagul pe true (casa[i].start(), setFlag(true));

- porneste simularea folosind o bucla while care se opreste fie cand timpul simularii atinge limita, fie cand nu mai exista client de procesat, iar cozile sunt goale;

La fiecare iteratie din while:

- se scrie in fisier timpul la care a ajuns simularea: cariabila time;

- se parcurge lista de client si se cauta aceia care au timpul de sosire ta egal cu timpul la care a ajuns simularea. Daca se gaseste un astfel de client, se apeleaza metoda min\_index() pentru a afla ince coada trebuie plasat, este introdus in coada respectiva si este eliminate din lista de clienti a magazinului;

- se scrie in fisier lista clientilor care inca asteapta;

- se parcurege vectorul de case, iar acolo unde numarul de client este mai mare decat 0, se afiseaza lista cu client, iar unde numarul de client este egal cu 0, se scrie mesajul “closed”;

- se increenteaza timpul la care a ajuns simularea: time++l;

- se verifica daca toate casele sunt goale, un criteriu de oprire a simularii: statusCase();

- se seteaza flagul pe true pentru acele case care au macar un client: casa[j].setFlag(true);

- se apeleaza sleep(100);

- se seteaza flagul pe false pentru acele case care nu au niciun client in coada: casa[j].setFLag(false);

- bucla while se repeta cat timp sunt indeplinite carintele;

- se calculeaza timpul mediu de asteptare pentru clientii procesati si se scrie rezultatul la finalul fisierului de iesire. Daca simularea se termina cand se ajunge la limita de timp, dar mai exista client in lista de asteptare sau in cozi, timpul mediu de asteptare nu va fi influentat. Acesta este calculate doar pentru clientii care ajung sa fie scosi din coada;

1. **rezultate**

Datele simularii se intruduc cu ajutorul unui fisier de intrare. Acesta contine pe prima linie numarul de client, pe a doua linie numarul de cozi disponibile, pe a treia linie durata simularii,pe a patra linie timul minim si timpul maxim de sosire, iar pe ultima linie timpul minim si timpul maxim de procesare pentru un client.

Dupa ce sunt colectate datele, incepe simularea in timpul careia se creaza un fisier de iesire cu numele dat in linia de comanda. Acest fisier afiseaza date legate de timpul simularii, clientii in asteptare, continutul cozilor si timpul mediu de asteptare.

Un exemplu de fragment dintr-un fisier de iesie este urmatorul :

time: 3

Waiting clients:[ ( 0, 28, 3 ), ( 1, 28, 2 ), ( 3,7,3 ) ]

casa1: [ (2, 3 ,2 ) ]

casa2: closed

Se poate observa ca exista 3 clienti in asteptare (id 0,1 si 3 ), avand timpii de sosire sunt 28, 28 si 7. Casa 1 are un singur client care tocmai a sosit ( ta = 3 ) si care mai are de asteptat 2 unitati de timp. La urmatorul timer, casa 1 va afisa textul: [ (2, 3 ,1 ) ] . Ultima variabila a clientului (timpul de procesare) a scazut cu o unitate, iar cand va ajunge la 0 va fi sters din lista, deci nu va mai fi afisat.

La finalul fisierului de iesire se afiseaza si timpul mediu de asteptare al clientilor:

“timp mediu de asteptare: 2.5”

Rezultatul s-a obtinut astfel: ( 2 + 3 + 2 + 3 ) / 4 =10 / 4 = 2.5 . Pentru ca sunt 2 case disponibile, clientii care ajung la t = 28 sunt trimisi la cozi diferite (ambele goale atunci), deci nu exista timpi suplimentari de asteptare.

time: 28

Waiting clients:[]

casa1: [ ( 0, 28, 3 ) ]

casa2: [ ( 1, 28, 2 ) ]

O casa cu mai multi client in coada se afiseaza astfel:

Casa 3 : [ ( 24, 31, 1 ), ( 46, 32, 1 ), ( 34, 36, 5 ) ]

Daca simularea se termina pentru ca s-au procesat toti clientii, nu se mai afiseaza nimic dupa timerul la care s-a ajuns. Ultimul fragment afisat este cu toate casele inchise si lista de asteptare client goala. Daca simularea se termina datorita timpului insufficient, ultimul fragment afisat estecu starea in care s-a ajuns: client in asteptare sau in procesare. Timpul mediu de asteptare al lor nu se ia in considerare.

**6. Concluzie**

Lucrand la aceasta tema, am reusit sa inteleg o parte din functionalitatea thread-urilor si am reusit sa creez pentru prima data in java o simulare care depinde de timp. Consider ca acest timp de tema ajuta studentii sa inteleaga mai bine posibilitatile pe care le au atunci cand folosesc programarea orientate pe obiect. Personal, nu mi-a fost usor sa fac programul sa functioneze bine, dar ma bucur ca in final am reusit.

In concluzie, acum stiu mai multe despre java decat stiam acum 2 saptamani.