

NUME : COSTEA PAULA MARIA

GRUPA : 302210

PROFESOR LABORATOR : DAN MITREA

**ASSIGNMENT 2**

**QUEUES SIMULATOR**

Cuprins

[1 . Obiective 3](#_Toc68470328)

[2 . Analiza problemei, modelare, scenarii, cazuri de utilizare 3](#_Toc68470329)

[3. Proiectarea 4](#_Toc68470330)

[4. Implementarea 6](#_Toc68470331)

[5. Rezultate 7](#_Toc68470332)

[6. Concluzii 7](#_Toc68470333)

[7. Bibliografie 7](#_Toc68470334)

# 1 . Obiective

Obiectivul principal al acestui proiect este de a descrie o simulare in timp real al unui sistem care lucreaza cu cozi pentru a determina si a minimiza timpul de asteptare al clientilor .

Printre obiectivele secundare ale acestei teme se numara:

* Descrierea unei clase “ Client “;
* Descrierea unei clase “ Coada ” , care contine un sir de clienti;
* Descrierea unei clase “ Informatie “ , unde se extrag toate datele necesare implementarii aplicatiei;
* Descrierea unei clase “ Eveniment “ , unde sunt prelucrate toate informatiile si sunt distribuiti clientii la coada potrivita;
* Crearea multi-threadurilor ( una pentru fiecare coada );
* Simularea ( cat de real posibil ) a timpului de sosire si de asteptare a clientilor
* Afisarea rezultatului simularii cozilor intr-o interfata ( si intr-un fisier, de asemenea );
* Calculul timpului mediu de asteptare al clientilor;
* Calculul timpului mediu de servire;
* Calculul orei de varf.

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

Analiza

Scopul principal al acestei probleme este acela de a dezvolta un program original care realizeaza o simulare in timp real a unui sistem bazat pe cozi, utilizand paradigmele Programarii Orientate pe Obiect . In primul rand, trebuie sa gasim un mod de a reprezenta toate datele semnificative si, de asemenea, un mod de a afisa rezultat intr-un mod simplu, atat pentru programator, cat si pentru utilizator. Pe masura ce incepem sa implementam simularea sistemului bazat pe cozi , vom realiza ca va deveni mai clar ca ne confruntam cu o situatie unde vom avea nevoie sa realizam anumite actiuni care se intampla in paralel .

Asadar, ne confruntam cu concurenta . Concurenta e abilitatea de a rula anumite programe sau anumite parti dintr-un program in paralel. Daca un task care se executa intr-o anumita perioada de timp, poate fi realizat asincron, sau in paralel, vom putea imbunatati cu acesta randamentul si interactivitatea programului .

Modelarea

Prima problema cu care ne confruntam consta in stocarea informatiei . Pentru aceasta , incepem cu cea mai mica parte a programului, si anume , “ Clientul ” . Dupa ce am rezolvat aceasta problema , am creat clasa “ Queue ” , care are , printre atributele sale , o lista de clienti . Aici simulam , de fapt “ sosirea “, ”asteptarea ” si “ eliminarea “ clientilor .

Ca o a doua problema , trebuie sa avem in vedere nevoia de concurenta a simularii programului . Pentru asta am utilizat threaduri ( fiecare coada are atribuit un thread ) .

Si , in final , a treia problema cu care ne confruntam ar fi gestionarea tuturor cozilor ,si , pentru asta , am implementat clasa “ Eveniment ” ( care are si ea un thread atribuit ) . Clasa “ Eveniment “ apeleaza clasa “ Informatie “ ,unde sunt prelucrate toate datele si sunt introduse in program informatiile necesare implementarii programului . In aceasta clasa , pe langa preluarea informatiilor, vom genera clientii si vom initializa toate cozile.

Scenariul

Daca programul ruleaza corect , utilizatorul ar trebui sa vada , intr-o interfata garfica ( dar si intr-un fisier, de asemenea ) modul in care intra clientii in cozi , timpul petrecut in coada, dar si clientii care asteapta . Iar, la finalul fisierului, va aparea timpul mediu de asteptare al clientilor, timpul mediu de servire si ora de varf.

Cazuri de utilizare

La apasarea butonului „ GO ” din cadrul interfetei grafice, se citesc toate datele introduse de utilizator de la tastatura si sunt afisate rezultatele simularii aplicatiei ; va aparea , la fiecare moment de timp ( de la 0 pana la timpul maxim de simulare , introdus in fisierul de intrare ) lista clientilor , al caror timp de sosire nu corespunde cu timpul current , iar ,apoi , vor aparea toate cozile , urmate de clientii care au intrat in coada . In cazul in care o coada este goala , langa aceasta va aparea mesajul “ CLOSED ” .

# 3. Proiectarea

Pentru a respecta paradigmele Programarii Orientate pe Obiect , am impartit programul in mai multe clase.

* Prima clasa implementata – clasa “ Client “. Aceasta contine :
* 3 atribute ( id, tArrival, tService ) : Pentru toate aceste trei attribute au fost create setters si getters;
* Un constructor;
* Getters si setters pentru attribute;
* O metoda “ toString ( ) “
* Apoi , am implementat clasa “ Coada “ ( clasa de baza a acestui program). Clasa aceasta creeaza un thread , deci , implementeaza interfata “ Runnable “ . Aceasta clasa contine :
* O lista “ synchronizedList “ de client ( am ales aceasta lista , deoarece este sincrona si avem nevoie de o lista sincrona pentru a putea lucra cu threaduri, pentru a permite doar unui singur thread sa execute la un anumit timp bine definit);
* Un “ tAsteptare “ ( care este AtomicInteger , din acelasi motiv ca si mai sus – ca un contor atomic ( incrementAndGet ( ) , etc . ) care poate fi folosit de mai multe threaduri concurent);
* Un atribut – “ tInchidere “ – timpul fata de care ne raportam cand parcurgem coada;
* Un atribut “ temp “ – care tine evidenta timpului in coada;
* Un atribut “ nrClienti ” , care stocheaza numarul de client al cozii.

Avem , de asemenea , un constructor care are ca si parametru “ tInchidere “ , si , care seteaza de asemenea , atributul “tAsteptare “ pe 0 .

Ca si metode avem :

* “ removeClient ( ) “ – pentru stergerea unui client din coada;
* “ adaugaClient ( ) ” – pentru adaugarea unui client in coada;
* “ getSize ( ) “ – care ne returneaza dimensiunea cozii;
* O metoda “ run ( )” – care porneste threadul din coada;
* O metoda “ pornire ( ) “ – care creeaza un thread nou si il porneste;
* Urmatoarea clasa pe care o vom analiza este clasa “ Informatie “ ,care contine urmatoarele atribute :

- „ numarClienti “ : numarul de clienti care urmeaza sa fie repartizati in cozi;

- „ numarCozi ” : numarul de cozi disponibile in care vor fi repartizati clientii;

- „ tMaxSimulation ” : timpul de rulare al intergii simulari;

- „ tMinArrival ” : timpul minim de sosire al clientilor la coada;

- „ tMaxArrival ” : timpul maxim de sosire al clientilor la coada;

- „ tMinWaiting ” : timpul minim de asteptare al unui client la coada;

- „ tMaxWaiting ” : timpul maxim de asteptare al unui client la coada;

- „ clientiInAsteptare ” : o lista de clienti care sunt generati aleator;

- o lista de cozi , sincrona , pentru siguranta threadurilor;

- „ suma ” : suma timpilor de servire ai tuturor clientilor;

- „ sumah ” : suma timpilor de sosire ai tuturor clientilor.

Ca si metode avem :

* un constructor : care extrage toate informatiile necesare executarea programului;
* O metoda “ getMaxWaitingTime “ – care returneaza valoarea maxima a timpului de asteptare pe care il pot avea clientii;
* O metoda “ getMinWaitingTime “ – care returneaza valoarea minima a timpului de asteptare pe care il pot avea clientii;
* O metoda “ gettMaxSimulation “ – care returneaza timpul maxim al unei simulari;
* O metoda “ getNumarClienti “ – care returneaza numarul total de clienti;
* O metoda “ getNumarCozi “ – care returneaza cozile;
* O metoda “ getClientiInAsteptare “ – care returneaza lista cu clientii care se afla in asteptare pentru a fi procesati;
* O metoda “ obtineCoadaMinima “ – care returneaza coada minima din lista de cozi ( coada cu cel mai mic timp de asteptare );
* O clasa “ Eveniment “ , care contine urmatoarele attribute :

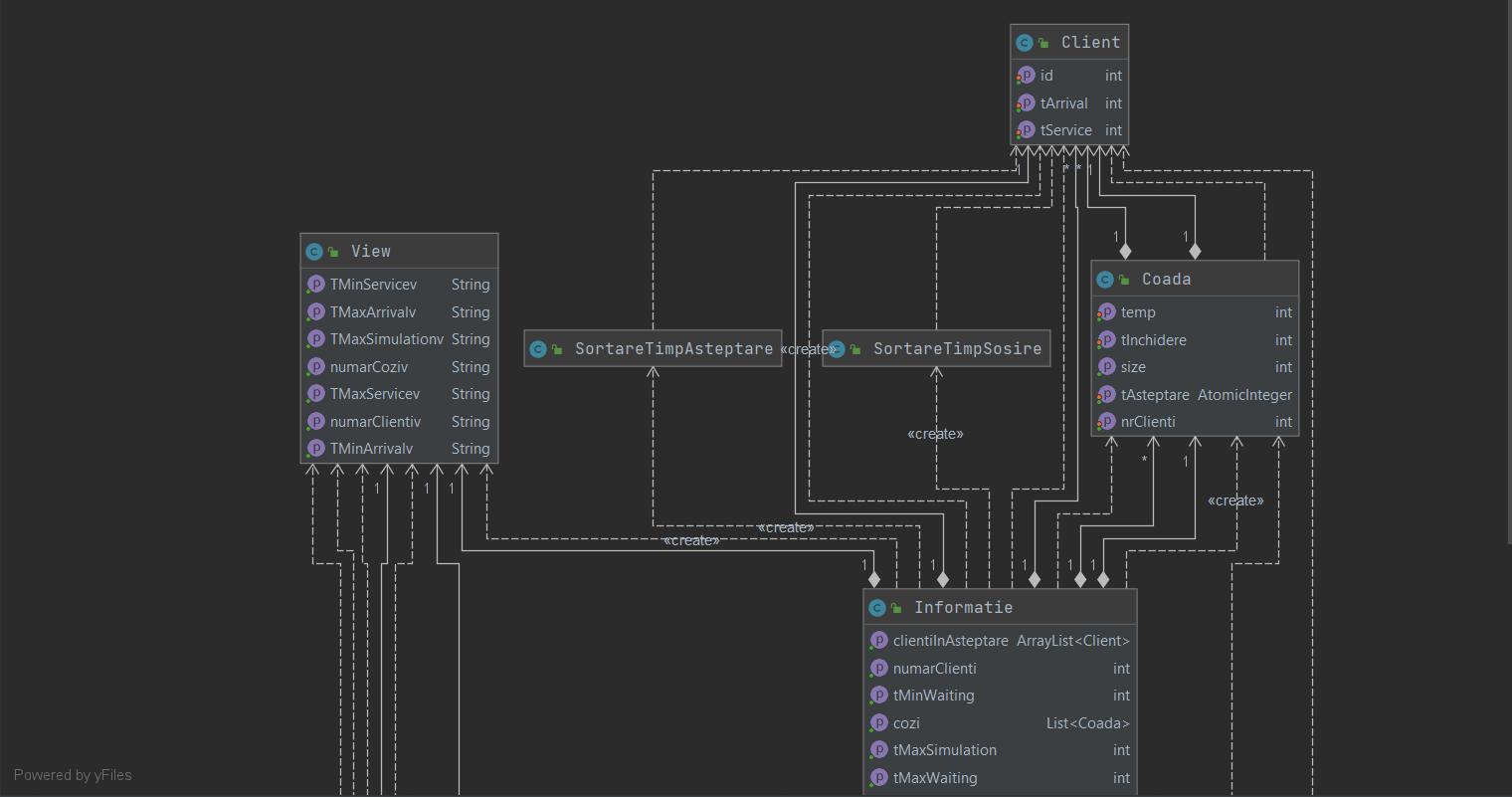
- “ timp “ : care retine timpul current ( de la 0 la timpul maxim de simulare a aplicatiei )

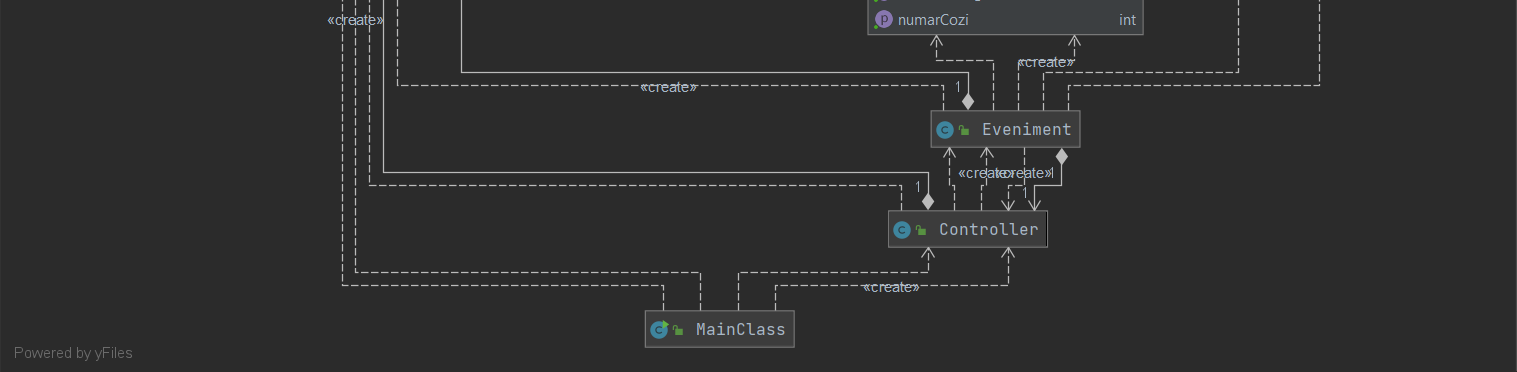
* “ thread “;
* “ timpMediuAsteptare “ : pentru a calcula timpul mediu de asteptare al clientilor
* “ semafor “

Ca si metode , avem : - O metoda “ rediretionareClient ( ) “ ( metoda folosita pentru a adauga un client intr –o coada – se alege coada minima prin apelarea metodei “ obtineCoadaMinima ( ) “ )

* O metoda “ afisareRezultate ( ) “ : unde se vor scrie in fisier rezultatele simularii dar si in interfata grafica;
* O metoda “ timpMediuCalculare ( ) “ : aceasta metoda calculeaza timpul mediu de asteptare;
* O metoda “ afisTimpMediu ( ) “;
* O metoda “ timpMediuServireAfisare ( ) “ : pentru afisarea timpului mediu de servire;
* O metoda “ peakHourAfisare ( ) “ : pentru afisarea orei de varf;
* O metoda “ run ( ) “.
* O clasa “ View “ : contine elementele de structura necesare pentru interfata grafica a proiectului. Aceasta face interactiunea cu computerul mai usoara si mai eficienta pentru utilizator.
* O clasa “ Controller ” : reprezinta clasa care traduce interacțiunile utilizatorului cu vederea în acțiuni pe care le va executa ‘modelul’. Tot in aceasta clasa se porneste si simularea.
* O clasa “ MainClass “ : aceasta clasa porneste interfata grafica.

Diagrama UML





# 4. Implementarea

* Clasa “ Client “ :

Aceasta clasa are doar 3 atribute ,un constructor , setters si getters pentru fiercare atribut ( pentru a putea accesa si schimba atributele clientilor )

* Clasa “ Queue “ :

Metodele principale ale clasei “ Queue “ :

-adaugareClient ( Client c ) :- in cazul in care clientul nu e primul client din lista ,trebuie sa ii scadem timpul de asteptare cu 1, crestem numarul de clienti ale cozii respective, adaugam timpul de asteptare al clientului la timpul de asteptare global;

-removeClient ( ) : - daca se doreste eliminarea unui client, insa coada este goala, trebuie sa asteptam; daca nu este goala, stergem clientul din varful cozii si decrementam numarul de client .

-run ( ) : cat timp inca este timp sau inca avem client in lista de asteptare luam primul client din coada si , intr – un “ for “ ii decrementam timpul de asteptare , si , de asemenea , decrementam timpul de asteptare al cozii si asteptam o secunda . Cand timpul de asteptare s-a scurs , stergem clientul .

* Clasa “ Informatie “

Metodele principale ale clasei “ Informatie “ sunt :

-Constructorul “ Informatie ( ) “ , in care extragem informatiile cu care urmeaza sa lucram pe tot parcursul implementarii programului ;

-Metoda “ clientiRandom ( … ) , unde , se genereaza aleator un timp de sosire si un timp de astepare. Prin intermediul acestor valori , o sa creem un client ,caruia ii atribuim aceste attribute , impreuna cu id – ul unic .

-Metoda “ obtineCoadaMin ( ) “ : care returneaza pozitia cozii minime din lista de cozi . In cazul in care gaseste o coada goala , returneaza pozitia acesteia , in caz contrar , afla care e coada minima , prin aflarea cozii cu dimensiune minima . In caz ca sunt mai multe cozi de dimensiune minima , vom returna coada cu timpul de asteptare mai mic .

-Metoda “ initializare ( . . . ) “ : parcurgem un ciclu de la 0 la numarul de cozi pe care l-am citit din fisier si adaugam cate o coada in lista de cozi , iar apoi , pornim threadul acestei cozi , prin metoda “ pornire ( ) ” .

* Clasa “ Eveniment “ :

Metodele principale ale clasei acestei clase sunt :

-Metoda “ afisareRezultate ( . . . ) “ : care afiseaza rezultatele simularii sistemului pe baza de cozi intr-un fisier;

-Metoda “ Run ( ) “;

* Clasa “ Controller “ : aceasta este clasa care se ocupa de realizarea „evenimentelor” in momentul in care se apasa pe un anumit buton. Aici, pentru un ascultator adaugat se realizeaza un anumit eveniment al carei rezultat este afisat pe ecran.
* Clasa „ View ” : in aceasta clasa se afla toate componentele necesare in realizarea interfetei specifice cat si diferite metode, ca de exemplu, metode care ne ajuta sa culegem informatia din casete texte. Tot in aceasta clasa, se gaseste pentru butonul “ GO “ un ActionListener ( ).
* Clasa “ MainClass “ : aici este instantiat un obiect de tip “ Eveniment “ , si creem un nou thread cu acest obiect, pentru a-l putea porni ,urmand apoi sa porneasca simularea aplicatiei .

# 5. Rezultate

Rezultatele acestei aplicatii se regasesc in fisierele text incarcate , test1\_afisare.txt, test2\_afisare\_txt, respectiv test3\_afisare.txt.

# 6. Concluzii

Aceasta tema ne introduce notiunea de “ thread - uri “ , mai exact , in notiunea de “ multi - threading “ , fiindu-ne foarte util , pe viitor , sa stim cum sa gestionam acest tip de probleme . O imbunatatire a acestui proiect ar putea insemna o sincronizare mai buna .

# 7. Bibliografie

-https://stackoverflow.com/

- http://www.tutorialspoint.com/java/util/timer\_schedule\_period.htm

- http://www.javacodegeeks.com/2013/01/java-thread-pool-example-using-executors-andthreadpoolexecutor.html