

**Tehnici de programare**

**Documentatie Tema2**

Queues Simulator

Muresan Ioana Danina

Grupa 30226

Cuprins:

1.Obiectivul temei

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

3.Proiectare

4.Implementare

5.Rezultate

6.Concluzii

7.Bibliografie

**1. Obiectivul temei**

Cerinta proiectului a fost: Propuneti si implementati o aplicatie de simulare pentru a analiza sistemele bazate pe cozi si pentru a reduce timpul de asteptare al clientilor. Obiectivul temei este de a implementa o aplicatie cu interfata grafica pentru a fi folosita ca simulator de cozi. Acest simulator este implementat folosind limbajul de programare Java si conceptele de programare orientata obiect. 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 . In esenta, primul client care se aseaza la coada va fi primul client servit. De asemenea clientii se pot aseza doar la sfarsitul cozii. Sistemul trebuie sa simuleze sosirea unui set de clienti pentru a fi serviti, intrarea in coada , asteptarea , servirea si parasirea cozii. Datele pe care se bazeaza simularea sunt citite dintr-un fisier, iar la sfarsit trebuie specificate , intr-un alt fisier , ordinea in care au intrat in cozi clientii , iar la fiecare moment de timp este specificat timpul ramas pana acesta va parasi coada

Ca obiective secundare am putea mentiona :

· Descrierea unei clase “ Task “ ;

· Descrierea unei clase “ Queue ” , care contine un sir de client ;

· Descrierea unei clase “ SimulationManager “ , unde se citesc din fisier toate datele necesare implementarii aplicatiei ;

· Descrierea unei clase “ Server “ , unde sunt prelucrate cozile ;

· 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-un fisier ;

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

**· Analiza**

Problema consta in realizarea unui algoritm de impartire a clientilor la cozile care pot sa existe la un moment dat. Algoritmul consta in realizarea unor cozi cu timp de asteptare minim, si in functie de timpul se sosire. Astfel se va alege coada ceea mai eficienta pentru fiecare client. Ca iesire se va simula in timp real cu ajutorul threadurilor un sistem de gestiune al cozilor, in care se va evidentia modul si ordinea in care clientii se vor aseza la o anumita coada. Totodata se va afisa timpul necesar pentru golirea coziilor si timpul de asteptare total al cozilor.

Fiecare client este identificat unic printr-un idClient pentru a se putea face distinctie intre acestea,la generarea idului s-a folosit timerul implementat in limbajul java, precum si casele sunt identificate printr-un identificator unic. Intervalul timpului de sosire si a celui de servire sunt introduse din interfata, pentru ca dupa timpul de servire si timpul de sosire a fiecarui client sa fie generat random in intervalele date. Conceptul de thread (fir de execuție) definește cea mai mică unitate de procesare ce poate fi programată spre execuție de către [sistemul de operare](http://ro.wikipedia.org/wiki/Sistem_de_operare) . Este folosit în programare pentru a eficientiza execuția programelor , executând porțiuni distincte de cod [în paralel](http://ro.wikipedia.org/wiki/Calcul_paralel) în interiorul aceluiași [proces](http://ro.wikipedia.org/w/index.php?title=Proces_(sisteme_de_operare)&action=edit&redlink=1) . Câteodata însă , aceste portiuni de cod care constituie corpul threadurilor , nu sunt complet independente și în anumite momente ale execuției , se poate întampla ca un thread să trebuiască să aștepte execuția unor instructiuni din alt thread , pentru a putea continua execuția propriilor instrucțiuni . Această tehnică prin care un thread asteaptă execuția altor threaduri înainte de a continua propria execuție , se numește sincronizarea threadurilor.

Diagram

Description automatically generated

**· Modelarea**

Prima problema cu care ne care ne confruntam consta in stocarea informatit . Pentru aceasta , incepem cu cea mai mica parte a programului ,si anume , “ Taskul ” . 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 ” , “ eliminarea “ clientilor .

Ca o a doua problema , trebuie sa ne confruntam cu nevoia de concurenta a simularii programului . Pentru asta am utilizat threaduri ( fiecare coada are un thread ) .

Si , in final , a treia problema ar fi gestionarea tuturor cozilor ,si , pentru asta , am implementat clasa “ Server ” ( care este si ea un thread ) . Clasa “ Server “ apeleaza clasa “ SimulationManager “ ,unde sunt prelucrate din fisier si introduse in program informatiile necesare implementarii programului . In aceasta clasa , pe langa preluarea informaitiilor din fisier , vom genera clientii si vom initializa cozile .

**· Scenarii**

Daca programul ruleaza corect , utilizatorul ar trebui sa vada , intr – un fisier 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 .

**· Cazuri de utilizare**

La deschiderea fisierului unde sunt sunt trecute 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 curent ) iar ,apoi , apar 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 ( decizii de proiectare , diagrama UML , structuri de date , proiectare clase , interfete , relatii , packages , algoritmi , interfata utilizator )**

Unified Modeling Language sau UML pe scurt este un limbaj standard pentru descrierea de modele si specificatii pentru software. UML a fost la bază dezvoltat pentru reprezentarea complexității programelor orientate pe obiect, al căror fundament este structurarea programelor pe clase, și instanțele acestora ( numite și obiecte ). Cu toate acestea, datorită eficienței și clarității în reprezentarea unor elemente abstracte, UML este utilizat dincolo de domeniul IT. Așa se face că există aplicații ale UML-ului pentru management de proiecte, pentru business Process Design etc.

Text

Description automatically generated

**4.Implementare**

Clasele existente sunt:

**-Clasa Task**: contine informatii despre client : un id, un arrivingTime si un servingTime.

Contine si metode de getter si setter pentru fiecare atribut ,dar si 2 constructori : unul cu parametrii si unul fara parametrii .

**-Clasa Queue** : este o clasa care extinde Threads. Aceaasta contine o valoare de tip AtomicInteger numita waitingTime. Este de tip AtomicInteger deoarece contine un contor atomic (incrementAndGet(\_ ,etc ) care poate fi folosit de mai multe threaduri concurrent: De asemenea , clasa mai contine si un numberOfClients de tip int ,un timp de oprire (timpOprire) ,un t si o lista de clienti.

Metodele principale ale clasei sunt:

-*setttere si gettere* pentru fiecare atribut ,

- *addNewClient ( 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

- *stergereClient ( )* : - daca coada vrea sa stergem un client , insa coada este goala , trebuie sa asteptam ; daca nu este goala , pur si simplu stergem clientul din varful cozii si decrementam numarul de clienti .

-*startCoada() -*porneste un thread nou pentru acea coada

*-getSize() –* returneaza dimensiunea listei 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 Server:** contine atributele: numberOfClients , numberOfQueues ,simulationInterval ,currentTime ,minimumArrival ,maximumArrival ,minimumService ,maximumService ,queue ,averageWaitingTime si averageClients initializat cu 0.

Metodele principale ale clasei:

* Gettere si settere pentru fiecare atribut
* addTask() :
* -prPrint() ;
* -verificari() ;
* determinareAverageTime() ;
* toString() : care afiseaza rezultatele simulatii sistemului pe baza de cozi;

Ne creem un Obiect SimulationManager ,pentru a putea apela valorile pe care le-am citit din fisier . Intr-un “while“ :

- adaugam la coada clientii ce au timpul de sosire egal cu timpul current ;

- Dupa ce am facut asta, thread-ul trebuie sa “doarma “ pentru o secunda ;

- Incrementam timpul current ;

- Calculam timpul mediu de asteptare al clientilor .

* run() :

**Clasa SimulationManager**  – metodele principale ale acestei clase :

* Constructorul SimulationManager ( )
* Metoda generateClients ( … ) , unde , se genereaza aleator un timp de sosire – cuprins in acel interval pe care l-am citit din fisier , un timp de astepare – cuprins in acel interval pe care l-am citit din fisier . Prin intermediul acestor valori , o sa creem un client , caruia ii atribuim aceste atribute , impreuna cu id – ul unic .
* Metoda getCoadaBuna ( ) : 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 initializareCozi ( . . . ) : 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 startQueue ( ) .

**Clasa View-** Interfața grafică sau Graphical User Interface este o [interfață cu utilizatorul](https://ro.wikipedia.org/wiki/Interfață_cu_utilizatorul" \o "Interfață cu utilizatorul) bazată pe un sistem de afișaj ce utilizează elemente grafice. Interfața grafică este numit sistemul de afișaj grafic-vizual pe un ecran, situat funcțional între utilizator și dispozitive electronice. Folosim o interfata grafica User-Friendly cu scopul de a putea fi folosit acest simulator de cozi.

In aceasta clasa am scris codul pentru realizarea interfetei grafice, cod care cuprinde : modul in care am aranjat frame-ul , etichetele , un buton de start , casetele text , atribuindu-le tututor niste coordonate , pentru o vizibilitate mai buna;

**Clasa ViewSimulator -**interfata grafica pentru a afisa in timp real continutul fiecarei cozi,cum se modifica timul fiecarui client etc.

**5.Rezultate**

Pentru a demonstra functionalitatea programului,utilizatorul poate introduce date de intrare din interfata unde se vor afisa si rezultatele simularii . Mai jos avem un screen cu o rulare, un exemplu de utilizare care se desfasoara astfel:

Inainte de toate, se introduce datele de intrare, timpul de sosire minim si maxim, si timpul de servire minim si maxim intre care dorim ca fiecare client sa il aiba. Aceste valori sunt generate random, valori aflate intre cele introduse de noi.

Graphical user interface

Description automatically generated

**6. Concluzii**

In urma implementarii acestei teme , sa dobandit o experienta noua in ceea ce consta programarea , prin folosirea concepului de thread , lucru benefic pentru orice programator , s – au recapitulat elemente de interfata grafica incepand de la utilizarea butoanelor , a componentelor JTextField , JtextArea.

Ca o concluzie, acest proiect m-a ajutat sa imi consolidez cunostintele de programare orientata pe obiect dobandite in primul semestru si sa imi organizez munca in baza paradigmelor POO. Totodata, felul in care am structurat proiectul si am creat clasele si metodele m-a ajutat sa eficientizez codul din punct de vedere al lungimii si al usururintei intelegerii acestuia, lucruri vitale in cazul programelor complexe care necesita mai multe clase sau care presupun un numar mai mare de programatori.

La capitolul imbunatatiri si dezvoltari ulterioare intra foarte multe elemente . De exempul se poate imbunatati aspectul ferestrei , se pot adauga alte operatii cum ar fi deschiderea caselor si inchiderea caselor in timpul rularii programului , sau deschiderae respective inchiderea acestora in functie de numarul de client. Tot ca si imbunatatire ar fi crearea unei simulari vizuale in care sa fie prezentate casele si deplasarae clientilor in cozi sau incercarea de a rezolva proiectul prin algoritmi mai eficienti. Evidentierea si folosirea unor concepte ale programarii orientate pe obiect , introducerea de interfete si/ sau clase abstracete ar putea fi o imbunatatire a codului existent. De asemenea, imbunatatirea interfetei grafice, inchiderea si deschiderea caselor in functie de numarul de clienti, crearea unui algorit mai eficient pentru distribuirea clientilor la case, precum si aducerea de noi posibilitati pentru utilizator, cum ar fi selectarea timpului de rulare a simularii (1 secunda = 0.5 secunde -> simulare de 2 ori mai rapida pentru aceleasi date.

**7. Bibliografie**

1. Wikipedia

2. StackOverflow

3. Youtube.com - tutoriale