**DOCUMENTATIE TEMA 2**

**Nume prenume Tigarean Lucian**

**Grupa 302210**

**Profesor Laborator Assist Antal Marcel**

Contents

[1. Cerinte Functionale 3](#_Toc476131445)

[2. Obiective 3](#_Toc476131446)

[2.1. Obiectiv Principal: 3](#_Toc476131447)

[2.2. Obective Secundare: 3](#_Toc476131448)

[3. Analiza Problemei 3](#_Toc476131449)

[4. Proiectare 3](#_Toc476131450)

[4.1. Structuri de date 3](#_Toc476131451)

[4.2. Diagrama de clase 3](#_Toc476131452)

[4.3. Algoritmi 3](#_Toc476131453)

[5. Implementare 4](#_Toc476131454)

[6. Testare 4](#_Toc476131455)

[7. Concluzii si Dezvoltari Ulterioare 4](#_Toc476131456)

[8. Bibliografie 4](#_Toc476131457)

# Cerinte Functionale

Dezvoltarea si implementarea unei aplicatii cu scopul de a analiza sistemele de cozi pentru determinarea si minimizarea timpului de asteptare al clientiilor.

Aplicatia trebuie sa permita intoducerea unor parametrii precum ar fi: numarul clientiilor, intervalul de simulare, numarul coziilor precum si date prin care se pot genera random clienti(timpul minim si maxim de sosire in coada, timpul minim si maxim de procesare a clientului cand ajunge la punctul de procesare)

Aplicatia trebuie sa ofere informatii in timp real despre evolutia coziilor, jurnalul de evenimente precum si timpul mediu de asteptare si ora de varf.

# Obiective

## Obiectiv Principal:

Implementarea unei aplicatii prin care putem urmari evolutia unei cozi prin intoducerea unor parametri. Pentru realizarea acestei aplicatii, procesarea simultana si independenta a cozilor, s-au folosit Thread-urile.

## Obective Secundare:

|  |  |  |
| --- | --- | --- |
| **Obiectiv Secundar** | **Descriere** | **Capitol** |
| Dezvoltarea de use case-uri si scenarii | Sunt descrise use case-uri pentru pornirea unei simulari prin interfata grafica | 3 |
| Alegerea structurilor de date | Alegerea unei structuri de date convenabile atat din punct de vedere al eficientei cat si a memoriei ocupate | 4 |
| Impartirea pe clase | Impartirea pe clase:Client, Coada, SimulatorManager | 4 |
| Dezvoltarea algoritmilor | Dezvolatrea de algoritmi pentru calcularea timpului mediu de asteptare si a orei de varf | 4 |
| Implementarea solutiei | Implementarea solutiei folosind obictele Client si Coada gestionate de obiectul SimulatorManager | 5 |
| Testare | Testarea realizata pe parcursul proiectarii | 6 |

# Analiza Problemei

Diagrama use case pentru pornirea unei simulari:

1. Utilizatorul introduce timpul minim si maxim de sosire in coada
2. Utilizatorul introduce timpul minim si maxim de procesare
3. Utilizatorul intoduce numarul coziilor
4. Utilizatorul introduce numarul clientiilor
5. Utilizatorul introduce intervalul de simulare
6. Utilizatorul apasa pe butonul Start pentru a porni o simulare
7. Se afiseaza rezultatele care sunt preluate si analizate de utilizator

# Proiectare

## Structuri de date

Structura de date folosita este Vector din pachetul java.util. Vector implementeaza interfata List care contine liste(secvente) de elemente indexate. Listele pot contine duplicate si permit un control precis asupra pozitiei unui element prin intermediul indexului acelui element. Clasa Vector are metode implementate care ne permite sa lucram mai usor, de exemplu pentru a adauga un element in lista folosim metoda add(Object a), pentru accesare indexata folosim metoda get(int index), pentru a afla numarul de elemente din lista folosim metoda size(), pentru stergere putem folosi atat metoda remove(Object a) cat si remove(int index) care sterge obiectul dat, respectiv sterge obiectul de pe pozitia data.

S-a ales aceasta structura de date doarece acest tip de lista poate fi sincronizata adica putem scrie metode sincronizate care nu permit modificarea listei doar in cadul executiei codului din metoda. Acest lucru usureaza lucrul cu thread-urile, unde se pot incerca accesarea simultana a listei ducand astfel la rezultate eronate.

SimulatorManager

-nrC: int

-text: JTextArea

-simResult: JTextArea

-logEvents: JTextArea

-simTime: int

-simTime: int

+SimulatorManager()

+generareClient(int,int,int,int,int,itn,int): void

+run(): void

+gestionareClient(Client): void

+peak(int): int

+startCozi: void

+stopCozi: void

+setRef(JTextArea, JTextArea, JTextArea): void

+AveWait(): void

## Diagrama de clase

Magazin

-pane3: JPanel

-pane1: JPanel

-pane2: JPanel

-mainPane: JPanel

-button1: JButton

.........................

etc.

+Magazin(String)

+actionPerformed(ActionEvent): void

+main(String[]): void

0..\*

-cozi

Coada

-index: int

-exit: boolean

-logEvents: String

-min: int

+Coada(int)

+run(): void

+stopThread(): void

+addClient(Client): void

+removeClient(): void

+getSize(): int

+getLogEvents(): String

+toString(): String

-copieLista

-magazin

-coada

0..\*

0..\*

0..\*

Client

-arrivalTime: int

-serviceTime: int

-idf\_number: int

-waitingTime: int

+client(int, int, int)

+getArrTime(): int

+getSerTime(): int

+setSerTime(): void

+setArrTime(int): void

+getID(): int

+getWTime(): int

+setWTime(): void

## Algoritmi

In acest proiect nu s-au folosit algoritmi speciali. S-au folosit niste algoritmi simpli pentru a calcula media timpului de asteptare si timpul in care coziile au fost cele mai aglomerate

# Implementare

Pentru realizarea acestui proiect de procesare a coziilor intr-un magazin s-au construit 3 clase plus clasa care descrie interfata cu utlizatorul.

Clasa **Client** contine atributele **arrivalTime** adica timpul de sosire, **serviceTime** (timpul de procesare), **idf\_number** (un atribut prin care putem identifica clientul), si **waitingTime** (atributul prin care se calculeaza timpul de asteptare al clientului in coada). Aceste atribute sunt de tipul private pentru a nu putea fi vizibile din alte clase (date personale ale clinetului).

Constructorul clasei seteaza cele 3 atribute arrivalTime, serviceTime si idf\_number atributul waitingTime urmand fi calculat si setat pe parcursul rularii. Metodele din aceasta clasa sunt de tipul getter-lor si setter-lor adica seteaza si returneaza valoarea atributelor pe parcursul procesarii.

Clasa **Coada** contine atributele **coada**(o lista de tip Vector care contine obiecte de tip Client, aceasta lista este folosita pe post de coada in care adaugam elemente la final si stergem primul element din lista), atributul **index** stocheaza un numar interg folosit ca identificator al listei, atributul boolean **exit** folosit pentru a putea controla exectuia Thread-ului, atibutul **logEvents** prin care stocam informatii despre procesarea cozii(introducerea in coada, stergerea din coada, precum si momentul in care s-au realizat aceste operatii), atributul **min** stocheaza informatii despre timp. Clasa coada extinde clasa Thread, in aceasta clasa suprascriem metoda run().

Metodele din aceasta clasa definesc comportarea si procesarea cozii. Constructorul seteaza indexul(identificatorul) cozii si adauga informatii in logEvents, metoda **run()** este o metoda suprascrisa a clasei Thread in care descriem codul pe care urmeaza sa il executam intr-un fir de lucru. Actiunea firului de lucru incepe la invocarea metodei run(); in interiorul acestei metode, intr- o bucla controlata de noi, scadem serviceTime-ul primului client si punem firul de lucru sa isi opreasca executia timp de 1000 ms prin invocarea metodei sleep(1000), Thread.sleep() este o metoda statica din clasa Thread care pune in pauza un fir de lucru din interiorul caruia se face invocarea metodei, metoda sleep() poate arunca o exceptie de tipul InterruptedException atunci cand un fir „adormit” este intrerupt. In momentul cand primul client are serviceTime egal cu 0 atunci il scoatem din coada.

Metoda **stopThread ()** seteaza atributul boolean exit la valoarea true, astfel bucla while(!exit) din metoda run() se opreste iar firul de lucru isi termina executia.

Metoda **addClient(Client a)** este o metoda sincronizata, **sincroizarea metodelor** previne accesarea simultana a codului dintr-o metoda de mai multe thread-uri, sincronizarea se poate face folosind cuvantul cheie syncronized atat ca modificator al metodei cat si la inceputul unui bloc de cod. In interiorul acestei metode actualizam jurnalul de evenimente(atributul logEvents) prin adaugarea de informatii despre timpul si clientul intodus, introducerea clientului in coada prin apelearea metodei add().

Metoda **removeClient()** este o metoda sincronizata, in care stergem primul client din coada si actualizam informatii despre acel client in atributul logEvents.

Metoda **getSize()** si **getLogEvents()** sunt metode sincronizate care returneaza marimea cozii respectiv returneaza atributul cu jurnalul de evenimente logEvents.

Metoda **toString()** este o metoda sincronizata si suprascrisa pentru a afisa informatii in timp real despre evolutia coziilor

Clasa **SimulatorManager** este o clasa cu metode putin mai complexe deoarece trebuie sa gestioneze si sa simuleze comportamentul clientiilor. Aceasta clasa extinde clasa Thread, si suprascrie metoda run().

Atributele clasei sunt: **nrC** de tip int, in care se stocheaza numarul coziilor, un vector de cozi numit **cozi** , doua liste de tip Vector in care se stocheaza clientii generati de metoda generareClient, **magazin** si **copieLista**(o copie a listei magazin pentru a retine clientii care au fost in magazin si pentru calcule ulterioare), trei atribute de tip JtextArea **text**, **simResult**, **logEvents** in care se retin referinte catre casute de text din interfata grafica, atributul **simTime** in care salvam perioada de simulare.

Metodele clasei SimulatorManager sunt : **generareClient(int minArr, int maxArr, int minSer, int maxSer, int nrClienti, int nrCozi, int simTime)** metoda prin care se genereaza clienti random pe baza unor parametri primiti adica minimul si maximul timpului de sosire in coada, minimul si maximul timpului de procesare, numarul coziilor, numarul clientiilor precum si intervalul de simulare. In aceasta metoda se instantiaza un numar de cozi egal cun nrCozi, folosind pachetul java.util.Random se instantiaza un obiect de tipul Random, avand metoda nextInt() care returneaza aleator un numar din intervalul 0-nr.dat ca si parametru, astfel se creeaza un numar de nrClienti care sunt adaugati in lista magazin si in lista copieLista.

Metoda **run()** este o metoda suprascrisa a clasei Thread in care descriem codul pe care urmeaza sa il executam intr-un fir de lucru. Actiunea firului de lucru incepe la invocarea metodei run(). Variabilele din interiorul metodei run() sunt:sum(variabila folosita la calcularea orei de varf), contor(in care se salveza date despre timp), peakTime(timpul in care au fost cei mai multi clienti in cozi). Aceasta metoda se executa atat timp cat este atributul simTime este mai maire ca 0. Din interiorul metodei se apeleaza metoda startCozi() pentru a porni firul de executie a coziilor create. Se verifica daca mai sunt clienti in magazin la care li s-au terminat timpul de plimbare in magazin(adica arrivalTime este 0), in caz afirmativ se apeleaza metoda gestionareClient(Client a) care introduce clientul la una dintre coziile create. Tot in aceasta metoda se preiau informatii la un interval de 1000 ms(pauza creata de metoda statica sleep(1000)) despre statutul coziilor pentru a putea fi afisate in timp real pe interfata grafica.

In momentul in care simTime a ajuns la valoarea 0 inseamna ca timpul de simulare s-a inchiat, se apeleaza metoda stopCozi() pentru a opri firele de executie, se apeleaza metoda aveWait() unde se calculeaza media timpului de asteptare in coada si se seteaza continutul casutelor de text din interfata grafica cu informatiile salvate de-a lungul rularii.

Metoda **gestionareClient(Client cl)** are rolul de a introduce clientul primit ca si parametru in coada cu marimea cea mai mica. In aceasta metoda se calculeaza marimea listelor si se adauga clientul in lista gasita.

Metoda **peakT(int sum)** calculeaza si returneaza numarul de clienti din toate coziile pentru a determina timpul in care coziile au fost cele mai aglomerate.

Metoda **startCozi()** porneste executia Threadurilor a fiecarei coada creata.

Metida **stopCozi()** opreste executia Threadurilor apeland pentru fiecare coada creata metoda stopThread() definita in clasa coada.

Metoda **setRef(JTextArea a, JTextArea b, JTextArea c)** este o metoda prin care se seteaza referintele din interfata grafica pentru a putea fi actualizat din interiorul clasei SimulatorManager.

Metoda aveWait() calculeaza media timpului de asteptare in coada si seteaza in prin referinta simResult informatiile calculate.

Clasa **Magazin** cuprinde elementele de interfata grafica. Clasa Magazin extinde clasa Jframe si implementeaza interfata ActionListener. Sunt definite 4 panouri cu un layout de tipul GridBagLayout(). Intr-un panou sunt definite campurile de text, pentru intoducerea parametriilor, si butonul start care incepe executia programului. Cele 3 panouri ramase sunt de tipul JtextArea unde sunt afisate informatii despre evolutia cozilor, jurnalul de evenimente si rezultatele simularii.

Metoda actionPerformed(ActionEvent arg0) este o metoda suprascrisa unde daca s-a intamplat o actiune asupra butonului se executa codul scris de noi din interiorul if-ului. In interiorul if-ului se creeaza un obiect de tipul SimulatorManager, se apeleaza metodele setRef pentru a trimite referintele in clasa SimulatorManager, se apleaza metoda generareClient cu parametrii din campurile de text si se apeleaza metoda start() pentru a porni Threadurile.

# Testare

Testarea s-a realizat pe parcursul dezvoltarii aplicatiei, urmarind comportamentul si rezultatele( comparate cu cele asteptate de noi) astfel s-au corectat unele erori si s-a optimizat programul.

# Concluzii si Dezvoltari Ulterioare

Realizand acest proiect am invatat sa implementez o clasa care extinde clasa Thread precum si descrierea metodei run(), m-am familiarizat cu implementarea unei interfete grafice precum si adaugarea de asculatori pe anumite elemente, mi-am imbunatatit lucrul cu lista Vector, am reusit sa sincronizez operatiile asupra listei Vector astfel incat doar un Thread poate aduce modificari asupra listei intr-un anumit interval de timp. Pe partea de interfata grafica am invatat sa adaug un scroollbar asupra unui panou.

Dezvoltarie ulterioare. Proiectul mai are nevoie de cateva imbunatatiri precum ar fi:

* O intefata mult mai interactiva cu elemente noi, care ar putea ilustra evolutia coziilor
* Un jurnal de eveniment mult mai detaliat si precis(indicand timpul exact cand au loc modificarile in coada)
* Pe partea de rezultate ale simularii ar mai putea fi adaugate rezultate noi de exemplu: timpii in care o coada nu a avut clienti.
* Managerul de simulari ar putea fi implementat pentru a verifica mult mai des evolutia coziilor, s-ar putea introduce atribute noi pentru simulare precum ar fi un atribut pentru modificarea vitezei de simulare.

# Bibliografie

https://dzone.com/articles/random-number-generation-in-java

http://inf.ucv.ro/documents/tudori/laborator8\_53.pdf

https://stackoverflow.com/questions/10177183/java-add-scroll-into-text-area