**DOCUMENTATIE TEMA 2**

**SIMULAREA MODULUI DE FUNCTIONARE A COZILOR**

**Nume prenume: Dragomir Mihai-Robert**

**Grupa: 302210**

**Profesor Laborator Assist Mitrea Dan**

Contents

[1. Cerinte Functionale 3](#_Toc68571756)

[2. Obiective 3](#_Toc68571757)

[2.1. Obiectiv Principal: 3](#_Toc68571758)

[2.2. Obective Secundare: 3](#_Toc68571759)

[3. Analiza Problemei 4](#_Toc68571760)

[4. Proiectare 5](#_Toc68571761)

[4.1. Structuri de date 5](#_Toc68571762)

[4.2. Diagrama de clase 5](#_Toc68571763)

[4.3. Algoritmi 7](#_Toc68571764)

[5. Implementare 7](#_Toc68571765)

[6. Exemplificare 8](#_Toc68571766)

[7. Concluzii si Dezvoltari Ulterioare 8](#_Toc68571767)

[8. Bibliografie 9](#_Toc68571768)

# Cerinte Functionale

Dezvoltati o aplicatie-simulator, ce urmareste analiza sistemelor functionale bazate pe structurile de date de tip coada, care sa permita urmatoarele functionalitati:

* Permite plasarea unui client intr-o coada, in care acesta asteapta pana in momentul in care este “servit”.
* Permite, prin sistemul proiectat, dispersia clientilor in cozi, astfel incat timpul de asteptare al acestora, din momentul in care intra in coada si pana cand sunt serviti, este minim.
* Permite simularea unui cadru propice utilizarii unei structure de data de tip coada, prin definirea unui timp de simulare, a unei serii de N clienti care ajung pentru a fi serviti si intra in acest scop in Q cozi, unde asteapta, sunt serviti si finalmente parasesc coada.
* Permite utilizatorului sa ofere un cadru bine definit privind modul de desfasurare al aplicatiei-simulator, prin introducerea datelor in interfata grafica.
* Permite utilizatorului sa vizualizeze dintr-o perspectiva comprehensiva rezultatul simularii atat prin in cadrul unei ferestre noi in interfata grafica, cat si in cadrul unui fisier.
* Permite utilizatorului sa vizualieze in timp real evolutia cozilor si finalmente, o serie de informatii relevante, cum ar fi timpul mediu de asteptare, timpul mediu de servire, ora de varf si numarul total al clientilor existenti in cozi la ora respectiva.
* Permite utilizatorului sa introduca in interfata grafica timpul pentru care trebuie rulata simularea, numarul de client ce urmeaza a fi repartizati in cozi, o arie in care sa evolueze timpul la care trebuie plasati la coada prin introducerea unui timp minim si a unui timp maxim de intrare in coada. Totodata, permite utilizatorului sa ofere o plaja in care sa evolueze timpul in care un client este servit, prin introducerea unui timp minim, respective maxim de servire. Nu in ultimul rand, permite introducerea de catre utilizator a numarului de cozi in care urmeaza sa fie dispusi clientii.

# Obiective

## Obiectiv Principal:

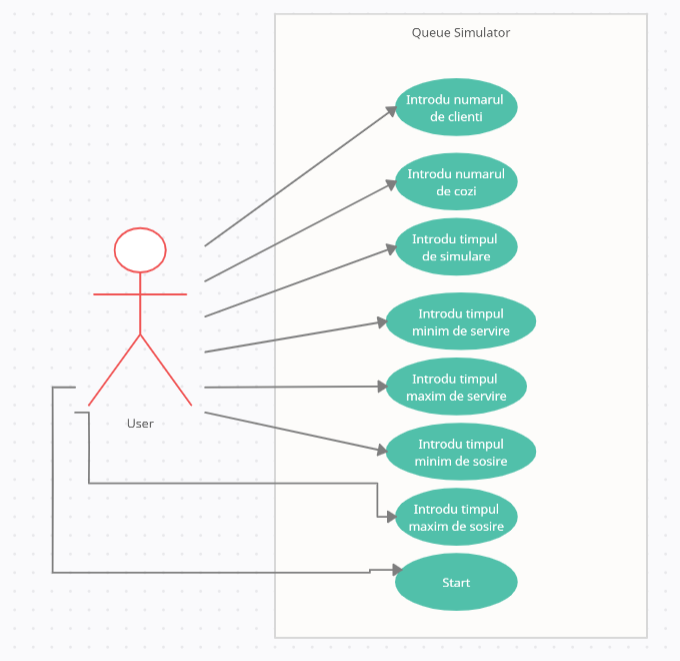
Propuneti, proiectati si implementati o aplicatie – simulator, care urmareste analiza functionalitatii sistemelor bazate pe structure de date de tip coada, prin determinarea si minimizarea timpului de asteptare al clientilor. Acesta beneficiaza de o interfata grafica dedicata, prin intermediul careia un utilizator poate introduce datele critice in vederea rularii simularii si poate sa vizualizeze rezultatul acesteia atat in cadrul interfetei, cat si in cadrul unui document text.

## Obective Secundare:

|  |  |  |
| --- | --- | --- |
| **Obiectiv Secundar** | **Descriere** | **Capitol** |
| Dezvoltarea de use case-uri si scenario | Prezinta o reprezentare grafica a modului in care utilizatorul interactioneaza cu interfata grafica a sistemului de calcul, urmarind anumite spete si scenarii de functionare si utilizare. | 3 |
| Alegerea structurilor de date | Ofera explicatii cu privire la optiunea pentru anumite structuri de date in implementarea solutiei. | 4 |
| Impartirea pe clase | Urmareste din punct de vedere schematic modul de interactiune si asamblare al claselor in vederea functionarii corecte a aplicatiei. | 4 |
| Dezvoltarea algoritmilor | Aduce adnotari imperios necesare asupra modului in care sunt traduse in practica, sub forma algoritmica, anumite metode auxiliare sau fundamentale in implementarea simulatorului. | 4 |
| Implementarea solutiei | Realizeaza o introspectie asupra functionalitatii si a necesitatii fiecarei clase constituente. | 5 |
| Exemplificare | Presupune analiza unor date rezultate in urma simularii sistemului, pentru inputuri de diferite magnitudini. | 6 |

# Analiza Problemei

Diagrama de use case case:



Use Case: determina evolutia clientilor in cozi

Actor principal: user

Principalul scenariu

1. Utilizatorul insereaza in interfata grafica timpul de simulare, timpul minim si maxim de sosire, timpul minim si maxim de servire, numarul de clienti si numarul de cozi;

2. Utilizatorul apasa butonul de start;

3. Simulatorul prezinta in interfata grafica si in fisier modul in care evolueaza in timp real cozile de asteptare si in final ora de varf, timpul mediu de asteptare si timpul mediu de servire;

Scenariu alternativ

Date incorecte

-

Utilizatorul nu introduce in interfata grafica numere intregi strict pozitive.

-

Se realizeaza redirectionarea catre primul pas.

Scenariu alternativ II

Date incoerente

-

Utilizatorul introduce in interfata grafica un timp minim de sosire mai mare decat timpul maxim sau un timp minim de servire mai mare decat timpul maxim.

-

Se realizeaza redirectionarea catre primul pas.

# Proiectare

## Structuri de date

Structura de data fundamentala in vederea implementarii solutiei finale o constituie BlockingQueue-ul. Aceasta constituie o coada care se blocheaza atunci cand incerci sa realizezi operatia de dequeue (in cazul de fata poll) si coada este deja goala, sau daca incerci sa realizezi operatia de enqueue si aceasta este deja plina. O astfel de structura de date este utila si dezirabila in cazul utilizarii firelor de lucru care opereaza asupra sa. Astfel, un thread care incearca sa faca sa faca dequeue pe o coada goala este blocat pana cand un alt thread insereaza un nou element in coada. De asemenea, daca un thread incearca sa faca enqueue pe o coada plina, acesta va fi blocat pana cand alt fir de lucru va face spatiu in coada, fie realizand dequeue, sau golind-o in intregime. Asadar, BlockingQueue-ul devine fundamental pentru asigurarea unei operari sigure in cadrul simulatorului, permitand buna colaborare si coordonare intre thread-urile care opereaza pe cozi si cel care opereaza drept manager de simulare. Totodata, au fost utilizate structuri de date cu un impact mai redus asupra functionalitatii, dar care rezolva anumite probleme de stocare a datelor fie de la bun inceput, fie pe parcurs, cand este necesara prezervarea formei lor nealterate pentru a realiza operatii posterioare anumitor modificari. O astfel de structura de data, cu un caracter util incontestabil, este constituita de ArrayList.

## Diagrama de clase

Diagrama UML din randurile ce urmeaza descrie modul in care interactioneaza intre ele clasele utilizate in implementarea simulatorului de functionare a cozilor.

O imagine care conține text

Descriere generată automat

## Algoritmi

In primul rand, coada, care aplica o politica de tip FIFO (primul venit – primul servit) constituie elementul primordial de modelare in practica a situatiei expuse in contextual simulatorului. Astfel clientul care ajunge initial intr-o coada va fi primul caruia i se va executa sarcina, fiind urmat dupa executie, in ordine cronologica de cei ajunsi dupa el in coada. Distribuirea clientilor in cozi se face dupa politica timpului de asteptare minimal. Astfel atributul de waitingPeriod, un AtomicInteger pe care opereaza atat serverele, cat si managerul de simulare defineste timpul de asteptare la fiecare moment pentru fiecare coada in parte. Acesta este decrementat de firul de lucru care opereaza pe serverul respective si incrementat in momentul adaugarii unui nou client in coda de catre managerul de simulare. Principiul prin care firul de lucru din SimulationManager determina coada in care urmeaza sa fie distrubuit un client presupune parcurgerea tuturor serverelor si optiunea pentru cel care are waitingTime-ul minim.

# Implementare

**Clasa Client.** Fiecare Client este caracterizat prin arrivalTime, timpul la care trebuie sa se aseze la coada, processingTime, timpul cat dureaza sa fie servit, finishTime, timpul la care acesta iese din coada, sau timpul la care se termina simularea in cazul in care acesta nu a fost in totalitate procesat pana in acel punct. De asemenea, finishTime-ul unui client este mentinut la -1 in cazul in care acesta nu este distribuit pana la finalul simularii in vreo coada. De asemenea un client este caracterizat printr-un cod de identificare unic (id). Pentru a asigura caracterul de unicitate, am optat pentru un atribut static (la nivel de clasa), idNumber, care la momentul instantierii unui Client nou se incrementeaza si da valoarea sa atributului de id. Nu in ultimul rand, suprascrierea metodei compareTo stabileste o relatie de ordine intre doua obiecte de tip client, bazata pe atributul lor de arrivalTime.

**Clasa Server.** Fiecare server modeleaza o anumita coada, reprezentata la nivelul clasei prin intermediu unui BlockingQueue de obiecte de tip Client. Din acest punct de vedere, fiecare coada are nevoie de un identifactor unic reprezentat prin intermediul atributului de id care se obtine in acelasi mod ca atributele de id pentru client. De asemenea, clasa Server implementeaza interfata Runnable, astfel ca prin suprascrierea metodei de run() este descrisa modalitatea in care firul de lucru opereaza pe coada. Principalul sa activitate se va desfasura in jurul waitingPeriod-ului, pe care il va decrementa la fiecare moment de timp, atata timp cat coada nu e vida, asigurand astfel in orice moment managerului de simulare o imagine reala asupra aglomerarii din server. Incrementarea, sau mai curand setarea noii valori a waitingPeriod-ului, inerenta introducerii unui nou client in coada, este realizata de catre firul de lucru principal. Atributul boolean exit determina un utilitar pentru managerul de simulare, fundamental prin setarea valorii sale de catre acesta pentru a opri executia firelor de lucru pe fiecare coada in parte. In absenta sa, un astfel de fir de lucru care opereaza pe un Server fie ar fi avut executia oprita in mod defectuos, fie ar fi rulat la infinit.

**Clasa Scheduler.** Este definita in principal de o lista de servere si introduce metodele necesare care urmeaza sa fie operate asupra servere-lor de managerul de simulare, in vederea unei gestiuni corecte a cozilor. La nivelul constructorului sau sun instantiate numberOfQueues fire de lucru si pornite, pentru a-si incepe executia in paralel, pe fiecare Server in parte. Metoda dispatchClient() trimite obiectul de tip Client, primit ca parametru, la coada cu waitingTime-ul minim.

**Clasa SimulationManager.** Determina cadrul functional, optim de gestiune a aplicatiei. Metoda generateNRandomClients genereaza in mod randomizat numberOfClients valori atat pentru timpul de sosire, cat si pentru cel de procesare, intre valorile minime si maxime corespunzatoare, oferite de utilizator. Metoda decrement urmareste scoaterea unui Client din coada in momentul in care timpul sau de procesare devine egal cu 0 si il seteaza in mod corespunzator in functie de momentul in care trebuie sa se termine simularea raportat la momentul current, luand in calcul toate cazurile particular care pot interveni. De asemenea, in cadrul metodei este setat si finishTime-ul pentru fiecare Client. Metoda isEmpty() determina daca la un anumit moment de timp cozile sunt goale si fiecare client a fost atribuit unei cozi. Metoda verifica deci, daca toti Clientii au fost serviti in totalitate, pana in momentul atingerii timpului maxim de simulare, constituind deci un resort pentru oprirea simularii in aceasta speta, pentru a nu mai prezenta exhaustive date ce nu constituie interes. Metoda createClientsCopy salveaza fiecare Client generat intr-un ArrayList, pentru a putea prelucra pe acesta in momentul determinarii timpilor medii de asteptare si servire si a peakHour-ului. Clasa SimulationManager implementeaza totodata interfata Runnable, suprascriind in metoda run() modul in care firul de lucru principal gestioneaza intreaga simulare si interactioneaza cu firele de lucru adiacente servere-lor. Firul de lucru principal executa cat timp timpul current e mai mic decat timpul maxim de simulare si cat timp metoda isEmpty() nu semnaleaza o terminare premature a simularii. Avand in vedere ca ulterior generarii, clientii sunt sortati in ordine crescatoare dupa arrival time, cat timp pe pozitia principala in lista de client generati se situeaza un client cu arrivalTime-ul egal cu currentTime, atunci firul de lucru va creste numarul de clienti distribuiti in cozi si il va distribui si pe acesta la coada potrivita prin metoda dispatchClient() din Scheduler. De asemenea, inainte de a-l scoate din lista de client care nu au fost inca distribuiti, timpul sau de finalizare este setat drept ultimul timp de simulare, urmand a fi modificat ulterior daca va fi terminat de procesat pana la finalul simularii in coada in care a fost distribuit. Dupa ce thread-ul principal a adaugat toti clientii cu arrivalTime egal cu currentTime, apeleaza metoda de write() care afiseaza clientii in asteptare si clientii din cozi de la timpul current. De asemenea e apelata metoda decrement() explicate anterior, iar currentTime-ul este incrementat. Firul de lucru doarme apoi pentru o secunda. Dupa ce simularea propriu-zisa s-a incheiat, e apelata metoda computeFinalLogEvents() care primeste drept parametru lista initiala de clienti si care are drept scop calcularea timpilor medii de servire si de asteptare, dar si a orei de varf. Acestea din urma, dupa ce sunt calculate vor fi afisate.

**Clasele View si LogView** contureaza interfata grafica cu care interactioneaza utilizatorul. Clasa LogView contureaza fereastra secundara, care se deschide in momentul in care utilizatorul apasa butonul de start, dup ace a introdus in prealabil correct datele.

**Clasa Controller** face legatura dintre SimulationManager si clasele de view si adauga un ActionListener pe butonul de start. Totodata, la nivelul clasei Controller, prin intermediul metodei checkInput() este verificata corectitudinea datelor introduse de utilizator si preluate de Controller din interfata grafica, afisand mesaje de eroare corespunzatoare in cazul unor nepotriviri privind inputul.

# Exemplificare

In urma rularii datelor propuse in prezentarea temei se poate observa modul de evolutie al timpilor medii de asteptare si servire, dar si a orei de varf. Astfel, in urma rularii ultimului test, cand cozile sunt in majoritatea timpului aglomerate, ca urmare a numarului mare de clienti, se poate observa cum peakHour-ul se apropie de valoare mediana a timpului total de simulare, in cazul de fata de timpul 100. De asemenea, in cazul rularii primului exemplu, unde, ca urmare a numarului mic de clienti, cozile sunt in majoritatea timpului goale, iar clientii ajung la momente diferite de timp urmare a randomizarii, fiecare client este procesat pana la finalul simularii si in consecinta averageWaitingTime-ul tinde spre valoare de average a timpilor de procesare a clientilor, iar averageProcessingTime-ul la fel. Totodata, peakHour-ul ca urmare a rarefierii datelor in log-ul final tinde spre jumatate din numarul de clienti. Rezultatele aferente rularii celui de-al doilea exemplu constituie un pas intermediar intre cele doua rulari extreme, expuse anterior.

# Concluzii si Dezvoltari Ulterioare

Finalmente, simulatorul isi releva utilitatea intrucat modeleaza o posibila instanta a unui concept extrem de intalnit in viata reala, precum cozile. O posibila modalitate de dezvoltare ulterioara ar putea-o constitui oferirea posibilitatii utilizatorului de alegere a schimbarii strategiei de adaugare a clientilor in coada, prin implementarea unei euristici de tipul cozilor de lungime minima, in detrimental cozilor de waitingPeriod minim, intrucat in realitate, un client nu cunoaste perioada de asteptare si manifesta o apetenta catre coada mai scurta.

# Bibliografie

1. <http://tynerblain.com/blog/2007/04/09/sample-use-case-example/>
2. <https://www.w3schools.com/java/java_regex.asp>
3. <https://docs.oracle.com/javase/7/docs/api/java/util/TreeSet.html>
4. <https://www.inflectra.com/ideas/topic/use-cases.aspx>
5. <http://coned.utcluj.ro/~marcel99/PT2021/Model%20Documentatie/>
6. <http://tutorials.jenkov.com/java-concurrency/blocking-queues.html>
7. <https://www.w3schools.com/java/java_files_create.asp>