**DOCUMENTATIE**

**TEMA 2**

**NUME STUDENT: Barbu Andrada Corina**

**Grupa: 30224**

Contents

[1. Cerinte Functionale 3](#_Toc133197718)

[2. Obiective 3](#_Toc133197719)

[2.1. Obiectiv Principal: 3](#_Toc133197720)

[2.2. Obective Secundare: 3](#_Toc133197721)

[3. Analiza problemei, modelare, scenarii, cazuri de utilizare 4](#_Toc133197722)

[4. Proiectare 4](#_Toc133197723)

[4.1. Structuri de date 4](#_Toc133197724)

[4.2. Diagrama de clase 5](#_Toc133197725)

[4.3. Algoritmi 5](#_Toc133197726)

[5. Implementare 6](#_Toc133197727)

[6.Testari 9](#_Toc133197728)

[7.Concluzii si Dezvoltari Ulterioare 13](#_Toc133197729)

[7.1 Dezvoltari ulterioare 13](#_Toc133197730)

[8.Bibliografie 14](#_Toc133197731)

# Cerinte Functionale

Trebuie sa implementam o aplicatie care aseaza clienti la coada in asa fel incat fiecare client sa astepte cat mai putin la coada la care urmeaza sa fie asezat. Totodata este nevoie ca fiecare coada sa fie reprezentata de un thread, astfel incat fiecare dintre cozi sa mearga in acelasi timp(in timp real), avand un numar maxim de client pe coada si o limita de timp.

* Date de intrare:
  + Numarul de clienti
  + Numarul de cozi disponibile
  + Minimul si maximul timpului de sosire al clientilor
  + Minimul si maximul timpului de servire al clientilor
  + Limita de timp
  + Alte informatii pe care le considerati necesare
* Date de iesire:
  + Timpul mediu de asteptare per coada
  + Timpul mediu de servire
  + Ora de varf pentru fiecare casa
  + Jurnalul de evenimente aparute in timpul simularii la fiecare moment de timp

# Obiective

## Obiectiv Principal:

Obiectivul principal al proiectului este constuirea si dezvoltarea unei aplicatii care realizeaza simularea in timp real al unor cozi si case de marcat in functie de timpul de asteptare. Pentru usurarea utilizatorilor aplicatiei am realizat o interfata grafica usor de folosit. Ea permite introducerea datelor necesare si vizualizarea tuturor evenimentelor petrecute pe fiecare coada(casa de marcat) si chiar si evolutia timpului.

## Obective Secundare:

Pentru indeplinirea obiectivul principal a fost necesara in primul rand preocuparea asupra unor obiective secundare, cum ar fi: alegerea unor structure de date potrivite pentru cerinta noastra, impartirea pe clase, dezvoltarea unor algoritmi, implementarea solutiei alese si in cele din urma, testarea efectiva a proiectului.

# Analiza problemei, modelare, scenarii, cazuri de utilizare

Cel mai important pas inainte de inceperea proiectului a fost stabilirea unor lucruri de baza, adicaa practice a unei fundatii a proiectului. . Din cerinta problemei ne dam seama ca trebuie sa ne axam pe gestiunea cozilor de asteptare astfel incat fiecare client sa astepte cat mai putin la coada. Deci o problema a fost indicarea fiecarui client coada la care trebuie sa se aseze, astfel incat sa aiba timpul de asteptare cat mai mic.

O alta etapa a fost modelarea proiectulu, strangerea de informatii dar si arhitectura prin care putem indeplinii cerintele proiectului. Deci in cele din urma am descoperit ca vom avea nevoie de un obicet de tip coada. Totodata am avut nevoie de strangerea datelor fiecarui client. Desigur la fel ca si in viata reala, exista mai multe cozi la un magazine, deci automat vom fi obligati sa avem si un obiect Server care va avea numarul de clienti de pe fiecare coada. Vom avea nevoie si de un Scheduler care va continue lista de cozi si totodata si lista de thread-uri pentru fiecare coada si nu in ultimul rand si de SimulationManager care va face legatura intre celelalte clase si totodata va genera random clinentii care vor urma sa fie bagati in coada.

Pe parcursul proiectarii mi-am dat seama ca notiunea de timp este foarte importanta, ea reprezentand una dintre caracteristicile principale. Astfel m-am folosit de firele de lucru (thread-uri) si de cuvantul cheie “synchronized”. Cu ajutorul lor putem sincroniza si garanta accesul unui singur thread, dar doar la un anumit timp. . Doar cu aceste lucruri putem sa realizam aplicatia, deoarece avem nevoie ca timpii generali de astepare sa fie modificati constant in functie de timpul de inceput al simularii, dar si de timpul current al simularii.

# Proiectare

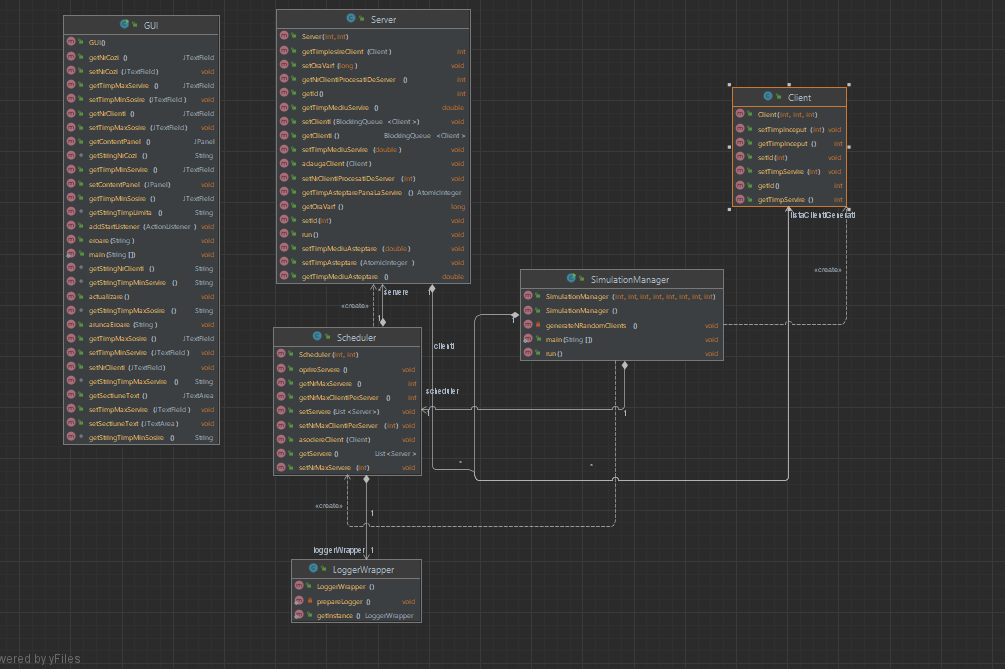
## Structuri de date

La baza proiectului nostru se afla 2 structuri de date foarte importante care ne ajuta sa gestionam datele furnizate de utilizator, si anume List si BlockingQueue. In proiect, aceaste structuri au rolul de a stoca cozile de asteptare puse la dispozitie, resprectiv clientii unei cozi. Astfel, structura List ne ajuta totodata sa implementam si carecteristicile unei cozi de asteptare, cum ar fi id-ul, numarul de clienti, timpul de asteptare(de tip AtomicInteger), timpul mediu de servire si de asteptare, ora de varf si chiar si structura BlockingQueue in care sunt stocati clientii cozi. De cealalta parte, BlockingQueue ne ajuta sa implementam caracteristicile unui client(timpul de intrare in coada, timpul de procesare si id-ul). Cele doua structure sunt asemanatoare in ceea ce priveste capabilitatile concurentei, dar diferenta dintre cele doua consta in faptul ca BlockingQueue este ’thread safe’.

**List<Server> servere=new ArrayList<>(nrMaxServere);**

**BlockingQueue<Client> clienti;**

## Diagrama de clase



## Algoritmi

* Algoritmul clasei Server

Intreg algoritmul acestei clase consta in:

* O metoda sincronizata de adaugare a clientilor:
  + Adaugarea clientului nou in coada
  + Incrementarea numarului de clienti
  + Adaugarea timpului de procesare(servire) a clientului nou la timpul mediu de procesare a cozii
  + Actualizarea timpului mediu de asteptare la coada. Pentru fiecare client nou se calculeaza ca fiind diferenta dintre timpul la care va parasi coada clientul de dinainte si timpul la care a sosit in coada clientul nou; pentru aceste lucruri ne vom folosi si de timpul curent
* Algoritmul clasei Server:

In proiectul nostru intreg algoritmul acestei clase consta in:

* O metoda sincronizata cu ajutorul caruia putem distribui un client la coada unde trebuie sa astepte cel mai putin pentru a fi servit:
  + Se cauta in lista de servere(cozi) coada care are timpul de asteptare minim; dupa ce am identificat coada care are timpul de asteptare minim, punem clientul nou in acea coada prin apelul metodei public synchronized void adaugaraClient(Client c) din clasa Server al carui algoritm a fost descries mai sus

# Implementare

Proiectul contine la final 6 clase: Client, Server, Scheduler, SimulatorManager, LoggerWrapper si GUI.

* Clasa Client

Este una din clasele de baza ale proiectului nostru, ea fiind caracterizata de 3 variabile instanta: timpServire, timpInceput si id. Aceasta clasa contine un constructor cu 3 parametri, gett-ere(metode de preluare) si sett-ere(metode de modificare sau setare).

* Clasa Server

O alta clasa de baza din acest proiect, ea fiind caracterizata de: colectia de clienti(BlockingQueue<Client>clienti, id, timpAsteptare, ultimaActiune ,nrClientiProcesatiDeServer, timpMediuAsteptare, timpMediuServire, oraVarf.

Aceasta clasa implementeaza Runnable, contine un constructor cu parametrii(id-ul si nr maxim de clienti), gett-ere si sett-ere, dar si o suprascriere a metodei public void run() :

* Suprascrierea metodei run:
  + Am implementat algoritmii pentru calculul orei de varf, pentru calculul momentului cand coada este libera si pentru a afisa in jurnalul de activitate ce client paraseste coada
  + Pentru a calcula ora de varf:am tinut minte timpul de inceput al simularii, am verificat de fiecare data numarul de clienti din coada, retinand doar maximul acestuia si momentul de timp; dupa aflarea maximului am scazut din momentul de timp la care au fost cei mai multi clienti timpul de inceput

In aceasta clasa putem spune ca s-au realizat practic o parte din cerintele proiectului si anume: averagewaitingtime, averageservicetime, peakhour, empty queue time si o parte din jurnalul de evenimente aparute in timpul simularii.

Totodata am implementat metoda de adaugare a clientilor in coada, ea fiind de asemenea sincronizata.

* Clasa Scheduler

O alta clasa importanta este aceasta, avand o lista de servere, numraul maxim de servere, numarul maxim de client per server, o lista de thread-uri pentru servere si un loggerWrapper.

Ea continue un constructor cu doi parametrii, aici dandu-se start la thread-urile pentru fiecare coada in parte.

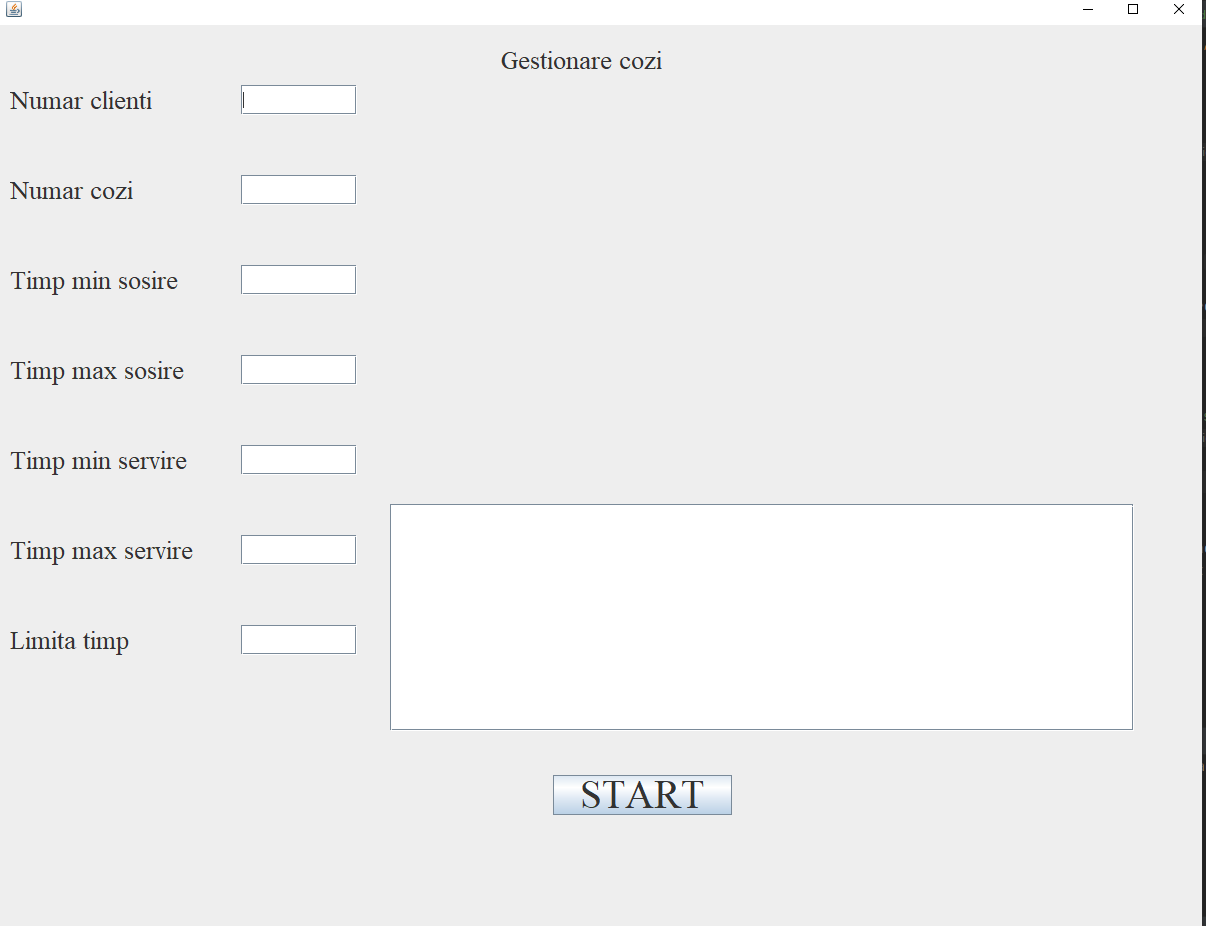
Aceasta clasa mai continue inca doua metode sincronizate, una de asociere a clientilor pe fiecare casa in functie de timpul minim de asteptare la coada si metoda sincronizata de oprire a serverelor(a thread-urilor) pentru ca la final sa putem afisa ora de varf, timpul mediu de servire si de asteptare si bineinteles cazul in care pe coada respective nu a fost niciun client.

In plus, aceasta mai continue metode de get-ere si sett-ere.

* Clasa SimulationManager
  + Clasa care ne genereaza clienti in functie de parametric trimisi in interfata grafica de catre utilizator. Aceasta are doi constructori, dar cel mai important este cel care are ca parametrii toate valorile introduce de utilizator in GUI. Tot aici se face generarea random a clientilor si pornirea thread-ului cozii respective. Totodata avem metoda de generare random a clientilor, unde se verifica daca numarul de clienti generati este mai mic decat numarul de clienti dat de utilizator in interfata, iar atat timp cat se intampla asta se adauga clientul cu datele sale(timpul de sosire si de servire) in lista.
  + Suprascrierea metodei public void run():
    - Se distribuie clientul nou creat intr-una din cozile stocate
    - Asa cum am zis mai sus, timpul curent va fi initial 0; astfel, primul client creat va avea timpul de sosire 0
    - Pentru a genera urmatorul client doar la timpul curent actualizat anterior, oprim thread-ul pentru o secunda(1000milisec)
* Clasa GUI

Aceasta clasa reprezinta implementarea propriu-zisa a interfetei grafice. Pentru realizarea acesteia am folosit 7 TextField-uri(toate pentru introducerea datelor necesare simularii), 1 buton(folosit pentru aexecuta aplicatia) si 1 TextArea(folosit pentru afisarea jurnalului de evenimente aparute in timpul simularii) La butonul folosit avem o metoda de adaugare a unui ascultator care apeleaza metoda caracteristica atunci cand este apasat din interfata. Astfel, atunci cand butonul este folosit se vor realiza urmatoarele lucruri:

* Se vor colecta informatiile introduse in TextField-uri din interfata si sunt convertite din String-uri in numere intregi
* Am creat un obiect de tip SimulationManager
* In try se creaza runnable, iar metoda public void run() este suprascrisa astfel:
  + In primul rand pentru a intrerupe acest thread avem nevoie de variabila timpCurent pe care o initializam la 0 cand thread-ul porneste
  + In aceasta metoda dorim sa actualizam interfata din 1000ms in 1000ms prin intermediul metodei public void actualizare() atat timp cat se doreste simularea aplicatiei; pentru a opri thread-ul la sfarsitul perioadei de simulare incrementam de fiecare data 1 secunda la variabila timpCurent despre care am discutat anterior
  + Se porneste totodata thread-ul la sfarsit pentru a se afisa informatia in timp real in acelasi timp atat in consola cat si in interfata.
* Tot aici avem o metoda care arunca o eroare in cazul in care inputul nu e correct
* In plus, aceasta clasa continue o clasa statica Logg care ne va ajuta impreuna cu LoggerWrapper la afisarea evenimentelor in GUI
* Clasa LoggerWrapper
* Ne folosim de ea pentru a putea afisa in GUI evenimentele care se petrec in proiectul nostru



# 6.Testari

Am testat cele 3 exemple de input-uri iar rezultatul pentru fiecare in parte a fost afisat in felul urmator:

**Test 1**

N = 4 Q = 2 𝑡𝑠𝑖𝑚𝑢𝑙𝑎𝑡𝑖𝑜𝑛 𝑀𝐴𝑋 = 60 seconds [𝑡𝑎𝑟𝑟𝑖𝑣𝑎𝑙 𝑀𝐼𝑁 ,𝑡𝑎𝑟𝑟𝑖𝑣𝑎𝑙 𝑀𝐴𝑋 ] = [2, 30] [𝑡𝑠𝑒𝑟𝑣𝑖𝑐𝑒 𝑀𝐼𝑁 ,𝑡𝑠𝑒𝑟𝑣𝑖𝑐𝑒 𝑀𝐴𝑋 ]= [2, 4]

**OUTPUT:**

**Time: 0**

**Waiting clients:**

**(0,24,3);**

**(1,14,2);**

**(2,29,2);**

**(3,28,3);**

**--------------------------acelasi output de la timpul 0 la timpul 13--------------------**

**Time: 14**

**Clientul 1 a ajuns la 14. S-a asezat la coada 0 si va astepta un timp de 0. Servirea clientului va dura 2. Acesta va iesi din coada la 16**

**Waiting clients:**

**(0,24,3);**

**(2,29,2);**

**(3,28,3);**

**Time: 15**

**Waiting clients:**

**(0,24,3);**

**(2,29,2);**

**(3,28,3);**

**Time: 16**

**La timpul 16 o sa iasa clientul 1 din coada 0**

**Waiting clients:**

**(0,24,3);**

**(2,29,2);**

**(3,28,3);**

**Time: 17**

**Waiting clients:**

**(0,24,3);**

**(2,29,2);**

**(3,28,3);**

**-------etc-------**

**Time: 24**

**Clientul 0 a ajuns la 24. S-a asezat la coada 0 si va astepta un timp de 0. Servirea clientului va dura 3. Acesta va iesi din coada la 27**

**Waiting clients:**

**(2,29,2);**

**(3,28,3);**

**Time: 25**

**Waiting clients:**

**(2,29,2);**

**(3,28,3);**

**Time: 26**

**Waiting clients:**

**(2,29,2);**

**(3,28,3);**

**La timpul 27 o sa iasa clientul 0 din coada 0**

**Time: 27**

**Waiting clients:**

**(2,29,2);**

**(3,28,3);**

**Time: 28**

**Clientul 3 a ajuns la 28. S-a asezat la coada 0 si va astepta un timp de 0. Servirea clientului va dura 3. Acesta va iesi din coada la 31**

**Waiting clients:**

**(2,29,2);**

**Time: 29**

**Clientul 2 a ajuns la 29. S-a asezat la coada 1 si va astepta un timp de 0. Servirea clientului va dura 2. Acesta va iesi din coada la 31**

**Waiting clients:**

**Time: 30**

**Waiting clients:**

**La timpul 31 o sa iasa clientul 3 din coada 0**

**La timpul 31 o sa iasa clientul 2 din coada 1**

**Time: 31**

**Waiting clients:**

**--------------etc---------**

**Time: 59**

**Waiting clients:**

**Coada 0 are timpul mediu de asteptare de 0.0**

**Coada 0 are timpul mediu de servire de 2.6666666666666665**

**Coada 0 are ora de varf 0**

**Coada 1 are timpul mediu de asteptare de 0.0**

**Coada 1 are timpul mediu de servire de 2.0**

**Coada 1 are ora de varf 0**

**Test 2**

N = 50 Q = 5 𝑡𝑠𝑖𝑚𝑢𝑙𝑎𝑡𝑖𝑜𝑛 𝑀𝐴𝑋 = 60 seconds [𝑡𝑎𝑟𝑟𝑖𝑣𝑎𝑙 𝑀𝐼𝑁 ,𝑡𝑎𝑟𝑟𝑖𝑣𝑎𝑙 𝑀𝐴𝑋 ] = [2, 40] [𝑡𝑠𝑒𝑟𝑣𝑖𝑐𝑒 𝑀𝐼𝑁 ,𝑡𝑠𝑒𝑟𝑣𝑖𝑐𝑒 𝑀𝐴𝑋 ]= [1, 7]

**OUTPUT:**

**Coada 0 are timpul mediu de asteptare de 3.3636363636363638**

**Coada 0 are timpul mediu de servire de 3.3636363636363638**

**Coada 0 are ora de varf 21**

**Coada 1 are timpul mediu de asteptare de 1.7**

**Coada 1 are timpul mediu de servire de 3.2**

**Coada 1 are ora de varf 27**

**Coada 2 are timpul mediu de asteptare de 2.5**

**Coada 2 are timpul mediu de servire de 4.0**

**Coada 2 are ora de varf 16**

**Coada 3 are timpul mediu de asteptare de 4.0**

**Coada 3 are timpul mediu de servire de 4.125**

**Coada 3 are ora de varf 15**

**Coada 4 are timpul mediu de asteptare de 5.153846153846154**

**Coada 4 are timpul mediu de servire de 3.0**

**Coada 4 are ora de varf 12**

**Test 3**

N = 1000 Q = 20 𝑡𝑠𝑖𝑚𝑢𝑙𝑎𝑡𝑖𝑜𝑛 𝑀𝐴𝑋 = 200 seconds [𝑡𝑎𝑟𝑟𝑖𝑣𝑎𝑙 𝑀𝐼𝑁 ,𝑡𝑎𝑟𝑟𝑖𝑣𝑎𝑙 𝑀𝐴𝑋 ] = [10, 100] [𝑡𝑠𝑒𝑟𝑣𝑖𝑐𝑒 𝑀𝐼𝑁 ,𝑡𝑠𝑒𝑟𝑣𝑖𝑐𝑒 𝑀𝐴𝑋 ]= [3, 9]

# 7.Concluzii si Dezvoltari Ulterioare

In cele din urma, datorita implementarii acesturi proiect consider ca mi-am imbunatatit abilitatile de scriere a codului in JAVA si totodata abilitatea de a imparti codul in mai multe clase, astfel incat sa fie cat mai lizibil.

De asemenea, am invatat ca daca este necasara rularea concurenta a instructiunilor si conditionarea acestora in functie de timp, sunt necesare thread-uri si metode sincronizate. Deci in cele din urma am descoperit cum functioneaza thread-urile si metodele sincronizate.

## 7.1 Dezvoltari ulterioare

* Imbunatarirea, infrumusetarea si dezvoltarea interfetei grafice
* Dezvoltarea si afisarea in timp real a cozilor si a modului in care sunt bagati si scosi din coada clientii

# 8.Bibliografie

<https://docs.oracle.com/javase/7/docs/api/overview-summary.html>

<http://stackoverflow.com/questions>

<https://examples.javacodegeeks.com/core-java/util/concurrent/linkedblockingqueue/java-util-concurrent-linkedblockingqueue-example/>