**DOCUMENTATIE TEMA 2**

**SIMULATOR DE COZI**

**Nume prenume: Calugar Gabriel Catalin**

**Grupa: 30229**

**Profesor Laborator Assist Antal Marcel**

Cuprins

[1. Cerinte Functionale 3](#_Toc476131445)

[2. Obiective 3](#_Toc476131446)

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

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

[3. Analiza Problemei 4](#_Toc476131449)

[4. Proiectare 4](#_Toc476131450)

[4.1. Structuri de date 4](#_Toc476131451)

[4.2. Diagrama de clase 4](#_Toc476131452)

[4.3. Algoritmi](#_Toc476131453) 6

[5. Implementare](#_Toc476131454) 6

[6. Testare](#_Toc476131455) 7

[7. Concluzii si Dezvoltari Ulterioare](#_Toc476131456) 9

[8. Bibliografie](#_Toc476131457) 9

# Cerinte Functionale

Obiectivul acestei teme a constat in proiectarea si implementarea in limbajul de programare Java a unui simulator de cozi. Din punct de vedere al datelor de intrare, utilizatorul este rugat sa introduca anumite date pentru efectuarea simularii, acestea sunt: introducerea numarului clientilor, introducerea numarului cozilor, introducerea valorii timpului maxim (pe parcursul caruia se va desfasura simularea), introducerea unor valori minime si maxime referitoare la timpii de sosire si de servire ale clientului, aceste intervale reprezentand setul de constrangeri in cadrul generarii clientilor. Generarea acestora se face in mod aleator, mai spefic timpul de sosire si de servire sunt generate in mod aleator. Cerinta presupunea de asemenea inchiderea si deschiderea automata a cozilor, astfel incat la inceputul simularii toate cozile sa fie inchise, acestea urmand sa fie deschise in momentul in care primul client se aseaza la coada, iar apoi sa se inchida din nou in momentul in care nu mai sunt client ce trebuie sa fie procesati. In cadrul proiectului trebuia sa existe un “simulator de cozi” acesta avand rolul de a eficientiza procesul de simulare, in asa fel incat fiecare client sa fie directionat catre coada ce are timpul de asteptare cel mai mic. La finalul executiei programului, rezultatele simularii trebuie sa fie afisate intr-un fisier text, iar la finalul lui se vor afisa ora de varf in cadrul simularii, timpul mediu de asteptare la cozi si timpul mediu de servire.

# Obiective

## Obiectiv Principal:

Cozile reprezinta o structura de date ce sunt folosite in numeroase scenarii, acestea avand scopul de a modela domenii din lumea reala. Obiectivul principal al unei cozi este acela de a oferi un loc pentru un „client” ce urmeaza sa astepte o perioada de timp inainte de a primi un „serviciu”. Fiecare coada poate fi privita ca un grup dintre casa de marcat si clientii ce sunt asezati la coada, acesta corespondenta poate fi interpretata dupa urmatorul aspect, fiecarei cozi i se atribuie clientii ce trebuie sa fie procesati, dar clientul este atribuit fiecarei cozi din prisma principiului ca timpul de asteptare sa fie minim. Acest proiect are scopul de a simula o serie de clienti ce vor ajunge la cate o coda la un moment dat de timp, dupa sosire acestia trebuie sa astepte sa le vina randul, iar dupa ce sunt serviti acestia vor iesi din coada. Pentru efectuarea acestei operatii avem nevoie sa cunoastem momentul in care fiecare client soseste la coada si timpul de care are nevoie pentru a fi servit. In privinta cozilor este necesar sa cunoastem la pentru fiecare dintre ele cat timp trebuie sa astepte un nou client ce a sosit la coada, pana in momentul in care va fi servit. Acest timp poate varia in functie de numarul clientilor de la coada dar si de timpii acestora de servire.

## Obective Secundare:

|  |  |  |
| --- | --- | --- |
| **Obiectiv Secundar** | **Descriere** | **Capitol** |
| Dezvoltarea de use case-uri si scenarii | S-a realizat o diagrama cazurilor de utilizare pentru operatia de adunare. | 3 |
| Alegerea structurilor de date | ArrayBlockingQueue a fost utilizat pentru retinerea clientilor, List a fost utilizat pentru retinerea cozilor. | 4 |
| Impartirea pe clase | Controller, View, Scheduler, Server, Task, SimulationManager, Main. | 4 |
| Dezvoltarea algoritmilor | S-au implementat algoritmi pentru cautarea minimului si a maximului. | 4 |
| Implementarea solutiei | Implementarea propiu zisa a operatiilor s-a efectuat in clasa SimulationManager | 5 |
| Testare | Testarea simulatorului s-a efectuat prin rulari repetate si verificarea rezultatelor afisate cu cele asteptate. | 6 |

# Analiza Problemei

In cadrul diagramei de cazuri vom avea drept actor principal si unic utilizatorul simulatorului de cozi. Preconditiile necesare sunt introducerea valorilor necesare efectuarii simularii de catre utilizator. Post conditiile se refera la realizarea cu succes a simularii si afisarea rezultatelor in timp real in cadrul interfetei, iar rezultatul total va fi stocat intr-un fisier text ce va fi suprascris la fiecare noua rulare a unei simulari. Modul normal de desfasurare al acestui eveniment consta in introducerea corecta a datelor de intrare, selectarea butonului de start si apasarea butonului de submit si afisarea rezultatelor corespunzatoare datelor. Pe de alta parte pot exista si alte moduri e functionare, acestea constau in neintroducerea unei valori numerice intr-un camp sau mai multe, fapt ce va duce la functionarea necorespunzatoare a programului, sau neselectarea butonului de start, fapt ce va duce la neactivarea simulatorului, chiar daca au fost introduse date corecte in toate campurile.

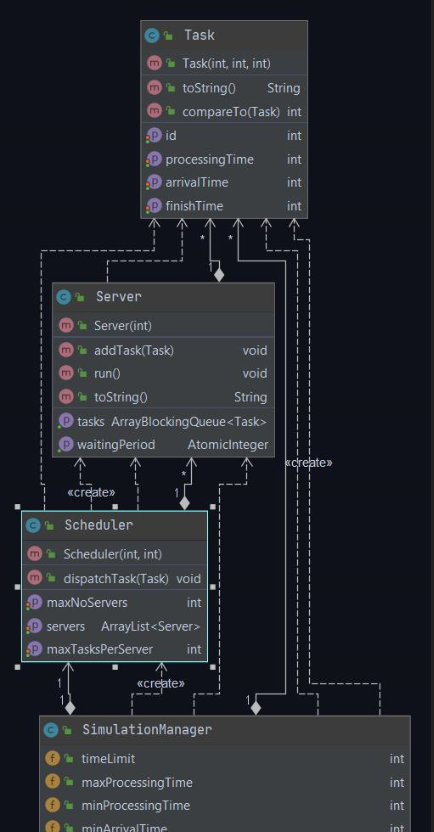
# Proiectare

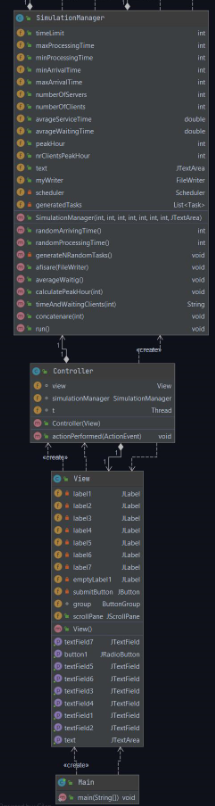
## Structuri de date

Structurile de date ce au fost folosite in cadrul acestei teme sunt: ArrayBlockingQueue, List. ArrayBlockingQueue a fost utilizata doar in cadrul clasei Server pentru retinerea clientilor ce ajung la coada, iar structura List a fost utilizata in cadrul mai multor clase pentru a se retine Serverele (cozile), dar si pentru retinerea clientilor ce au fost generati la inceputul simularii.

## Diagrama de clase

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.





Clasa Main este clasa in care spunem programului ce sa execute, in acesta clasa se realizeaza deschiderea interfetei grafice.

Clasa View este clasa in care in care se construieste partea vizuala a proiectului.

Clasa Controller este clasa de granita dintre partea grafica si programul in sine, unde se petrece interactiunea utilizator-sistem de calcul.

## Algoritmi

In cadrul acestei teme au fost implementati algoritmii de cautare a minimului si maximului. Algoritmul de cautare al minimului din interiorul clasei Scheduler a fost implementat cu scopul de a gasi timpul minim de asteptare pentru cozile din simulare. Algoritmul de cautare al maximulului din interiorul clasei SimulationManager a fost implementat cu scopul de a gasi numarul maxim de clienti de la cozi si ora de varf cand are loc acest eveniment.

# Implementare

**Clasa Task:** Acesta clasa contine structura unui client, acesta avand urmatoarele caracteristici: un id pentru identificare, un timp de sosire la coada, un timp de servire si un timp de terminare. In cadrul acestei clase se afla metode accesoare si mutatoare pentru preluarea si setarea proprietatilor, un constructor, metoda de toString pentru afisarea clientilor, iar la finalul clasei se afla implementata o metoda de comparare.

**Clasa Server:** Este clasa ce reprezinta coada la care se vor aseza clientii, acesta are drept proprietati o lista de clienti si o perioada de asteptare ce se va modifica pe tot parcursul derularii simularii. In interior exista un constructor, o metoda de toString pentru afisarea cozii in cadrul simularii, o metoda de adaugare unui client la coada, metode de accesare a proprietatilor clasei, si o metoda run, acesta ultima metoda este cea care creaza thread-ul specific pentru fiecare coada.

**Clasa Scheduler:** Reprezinta clasa in care se face atribuirea clientilor unei cozi conform strategiei alese (clientii se vor aseza la coada cu cel mai putin timp de asteptare). Acesta clasa are drept proprietati o lista de servere, un numar maxim de cozi ce vor fi create si un numar maxim de clienti ce pot fi atribuiti unei cozi. Aceste ultime doua proprietati vorproveni de la utilizator si au rol de constrangeri pentru ca acesta clasa sa nu construiasca o infinitate de cozi si de clienti. In acesta clasa avem un constructor, metode accesoare si mutatoare pentru proprietatile clasei si metoda de dispatchTask, acesta cautand minimul timpului de asteptare al fiecarei cozi si asezand clientul la coada cu cel mai mic timp de asteptare.

**Clasa SimulationManager:** Din punct de vedere al complexitatii acesta clasa este cea mai complexa, deoarece in interiorul ei se genreaza clientii, intr-un mod aleatoriu, ce vor fi utilizati in cadrul simularii. Avem un constructor, metode pentru generarea aleatorie a timpilor de sosire si servire, a clientilor, metode pentru afisarea rezultatelor in cadrul fisierului text, calcularea timpului mediu de asteptare, calcularea orei de varf, si a altor metode intermediare pentru a se afisa corespunzator rezultatele in timp real in interfata grafica si in fisierul text ce va fi fost generat la sfarsitul simularii.

Interfața grafică sau Graphical User Interface este 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. Pentru construirea acestei interfete grafice am folosit Java Swing. Interfata grafica cuprinde urmatoarele elemente:

**Frame**: acesta este „rama” in interiorul careia se adauga toate elementele de care avem nevoie pentru functionarea programului. Din punct de vedere practic este o fereastra ce poate fi inchisa prin apasarea butonului X din coltul dreapta sus dupa utilizare.

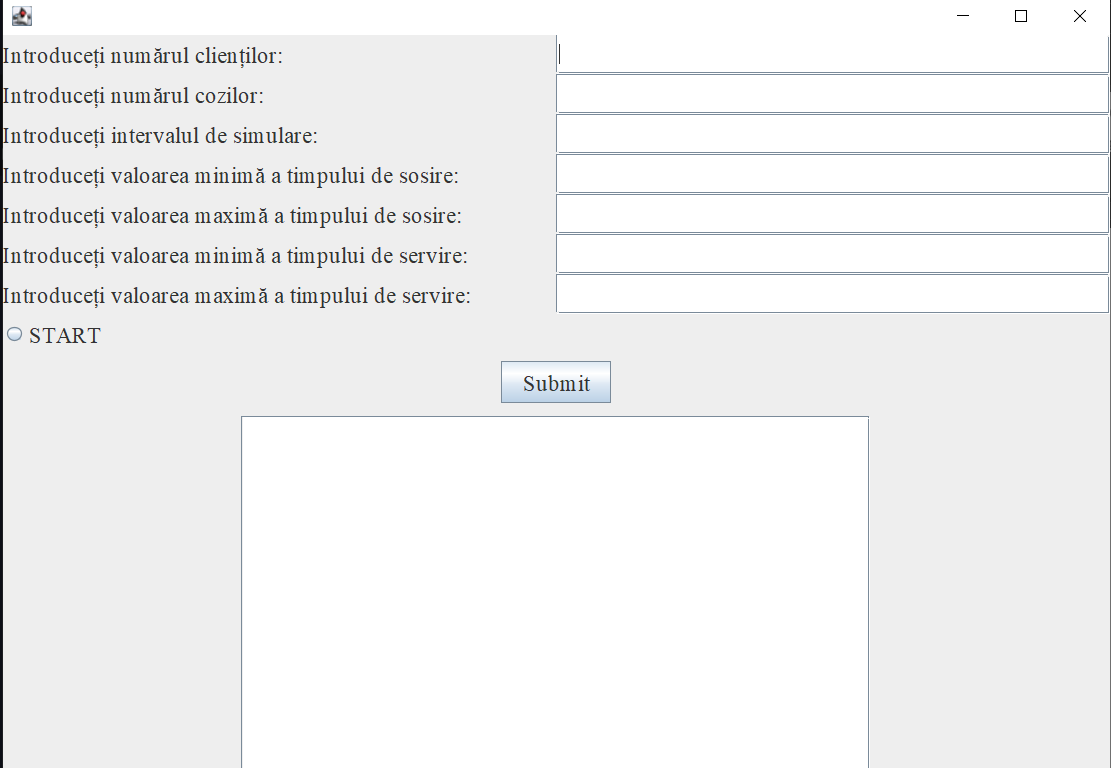
**Butoane:** Acestea sunt in numar de 2. Primul butoan are rolul de a selecta urmatorul eveniment ce are loc dupa apasarea butonului de submit, acestea formand un grup. Ultimul buton de „Submit” este cel care porneste simularea.

**TextField:** reprezinta un spatiu dreptunghiular in interiorul caruia se pot introduce date de la tastatura, sau se pot afisa rezultate. textField1, textField2 sunt utilizate pentru introducerea numarului de clienti si cozi, textField3, textField4, textField5, textField6, textField7 sunt folosite pentru introducerea constrangerilor referitoare la timpul de sosire si de servire.

**Label:** Acest element de interfata grafica este o eticheta ce poate contine informatii, indicatii, etc., ce pot oferi suport utilizatorului referitor la modul de utilizare al programului. In acest caz este folosit pentru a indica unde se introduc datele.

In main-ul programului este instantiata clasa Interfata, deci de fiecare data cand se initialieaza clasa main se va instantia o noua interfata grafica.

Pentru butonul de submit s-a facut o metoda noua ce implica ActionEvent, astfel de fiecare data cand are loc o actiune informatia este transmisa la Controller. In interiorul acestei clase se realizeaza tokenizarea datelor de intrare si transformarea acestora in numere ce vor fi transmise clasei SimulationManger.



# Testare

In cadrul testarii proiectului au fost executate un set de teste pentru a se vedea comporamentul simulatorului de cozi in diferite scenarii de simulare, in interiorul setului de teste au fost testate si cele trei exemple din cadrul documentului ce contine cerintele pentru realizarea simulatorului de cozi. Pentru a se exemplifica functionarea acestui simulator de cozi vom folosi un exemplu ce contine un set cu valori de intrare ce tind sa fie destul de mici, la fel si timpul de simulare va avea o valoare destul de mica. Datele de intrare ce vor fi introduse in simulatorul de cozi vor fi:

Numarul clientilor ce vor fi generati pentru simulare: 4

Numarul cozilor ce vor fi utilizate in cadrul simularii: 2

Valoarea intervalului de simulare: 8

Valoarea minima pentru timpul de sosire al clientului: 2

Valoarea maxima pentru timpul de sosire al clientului: 5

Valoarea minima pentru timpul se servire al clientului: 1

Valoarea maxima pentru timpul de servire al clientului: 3

(1,4,1)

(2,4,3)

(3,5,1)

(4,5,2)

Time 0

Waiting clients: (1,4,1) (2,4,3) (3,5,1) (4,5,2)

Queue 1:

Queue 2:

Time 1

Waiting clients: (1,4,1) (2,4,3) (3,5,1) (4,5,2)

Queue 1:

Queue 2:

Time 2

Waiting clients: (1,4,1) (2,4,3) (3,5,1) (4,5,2)

Queue 1:

Queue 2:

Time 3

Waiting clients: (1,4,1) (2,4,3) (3,5,1) (4,5,2)

Queue 1:

Queue 2:

Time 4

Waiting clients: (3,5,1) (4,5,2)

Queue 1: (1,4,1)

Queue 2: (2,4,3)

Time 5

Waiting clients:

Queue 1: (3,5,1) (4,5,2)

Queue 2: (2,4,2)

Time 6

Waiting clients:

Queue 1: (4,5,2)

Queue 2: (2,4,1)

Time 7

Waiting clients:

Queue 1: (4,5,1)

Queue 2:

Average Waiting Time is: 0.25

Average Service Time is: 1.75

Peak Hour is: 5

# Concluzii si Dezvoltari Ulterioare

Prin urmare, in opinia mea consider ca acest proiect m-a ajutat sa aprofundez cunostiintele mele in limbajul de Java, implementarea paradigmelor programarii orientate pe obiect, realizarea unei interfete grafice. Posibilitatile de dezvoltare ale acestui simulator constau in afisarea unui mesaj de eroare in cazul in care utilizatorul nu introduce date corecte, afisarea intr-un mod mai interactiv a rezultatelor din cadrul simularii.

# Bibliografie

<https://www.w3schools.com/java/default.asp>

<https://www.javatpoint.com/java-swing>

Youtube

Wikipedia

Stack Overflow