DOCUMENTATIE

TEMA *2*

Queues Management Application

NUME STUDENT: Chindris Alexandra - Maria

GRUPA: 30226

# CUPRINS

[1. Obiectivul temei 3](#_Toc95297885)

[2. Analiza problemei, modelare, scenarii, cazuri de utilizare 3](#_Toc95297886)

[3. Proiectare 5](#_Toc95297887)

[4. Implementare 8](#_Toc95297888)

[5. Rezultate 12](#_Toc95297889)

[6. Concluzii 16](#_Toc95297890)

[7. Bibliografie 16](#_Toc95297891)

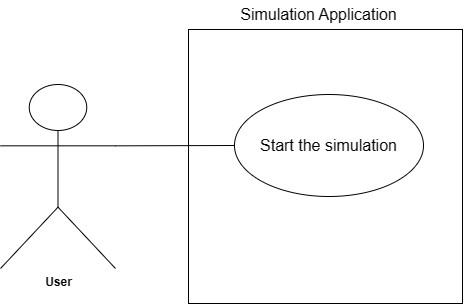
# Obiectivul temei

* Obiectivul principal : Proiectarea și implementarea unei aplicații care vizează managementul unei cozi prin simularea unei serii de N clienți care sosesc pentru servire, intră în Q cozi, așteaptă, sunt serviți și în cele din urmă părăsesc cozile, cât și calcularea timpului mediu de așteptare, a timpului mediu de servire și a orei de vârf.
* Obiectivele secundare:
  + Analizarea problemei și identificarea cerințelor - determinam cerintele functionale si cele non-functionale. [Capitolul 2]
  + Proiectarea aplicației de simulare - determinam clasele de care avem nevoie impreuna cu atributele si metodele lor si le organizam in pachete. Determinam legaturile dintre pachete si clase. [Capitolul 3]
  + Implementarea aplicației de simulare – implementam clasele, cat si algoritmii de aduagarea in coada. [Capitolul 4]
  + Testarea aplicației de simulare [Capitolul 5]

# Analiza problemei, modelare, scenarii, cazuri de utilizare

* Cerințe funcționale:
  + Aplicația trebuie să permită utilizatorului să introduce datele cu care se efectuează simularea.
  + Aplicația trebuie să permită utilizatorului să pornească simularea.
  + Aplicația trebuie să afișeze în timp real evoluția cozilor.
* Cerințe non-funcționale:
  + Aplicația trebuie sa fie ușor de folosit de utilizator.
  + Aplicația trebuie să precizeze ce date se doresc introduce de utilizator.

*Diagrama use case*:



**Use case:** Start the simulation

**Primary actor:** User

**Main Success Scenario:**

1. Utilizatorul insereaza valorile pentru: numărul de clienți, numărul de cozi, intervalul de simulare, timpul minim și timpul maxim pentru preluarea clientului, timpul minim și timpul maxim pentru servisrea clientului și selectează strategia conform căreia dorește sa se realizeze simularea: coada cu timpul cel mai scurt sau coada cea mai scurtă.
2. Utilizatorul selecteaza butonul de „Start simulare”.
3. Aplicația afișează în timp real evoluția cozilor, iar la finalul simulării afișează timpul mediu de așteptare, timpul mediu de servire și ora de vârf.

**Alternative Sequence:**

* 1. Utilizatorul introduce date invalide pentru simulare. De exemplu, numere negative sau intervale de timp care nu respectă condiția de interval

(adică [a, b] => a < b).

* 1. Aplicația anunță utilizatorul că datele introduse nu sunt valide și trebuie reintroduse. De asemenea, aplicația îi dă și câteva sugestii în legătură cu problema datelor introduce de acesta.

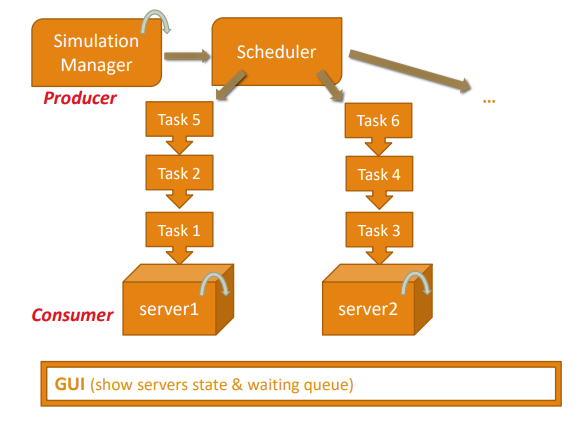
2.1) Utilizatorul nu introduce toate datele necesare simulării.

2.2) Aplicația anunță utilizatorului că datele introduce de acesta sunt

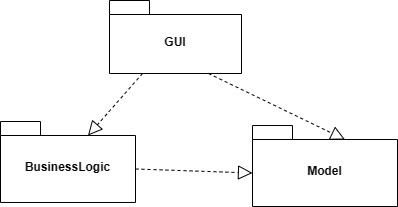
incomplete și trebuie reintroduse.

# Proiectare

*Design – Conceptual Architecture:*

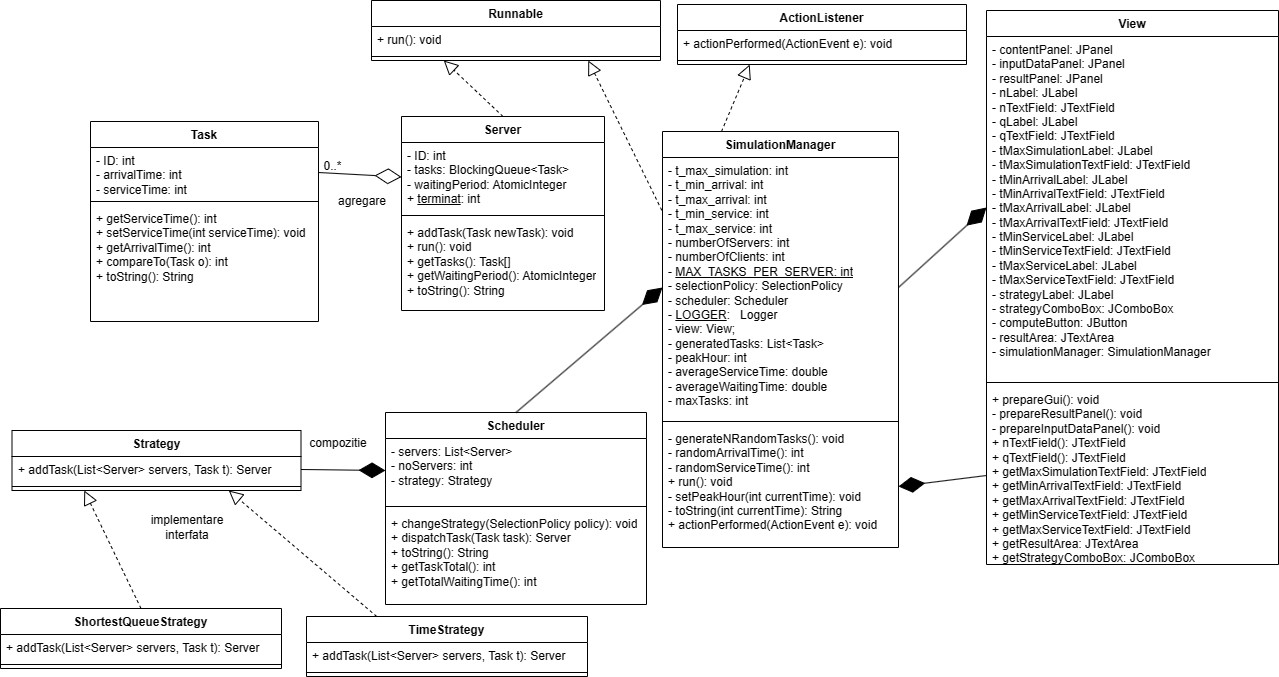


*Diagrama de pachete:*



*(relatie de dependenta)*

*Diagrama de clase:*

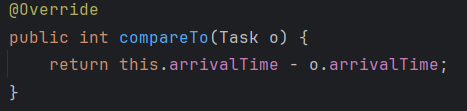


* + - ShortestQueueStrategy si TimeStrategi implementeaza interfata Strategy => relatie de implementare
    - Strategy este o parte a lui Scheduler, au aceasi perioada de viata => relatie de compozitie
    - Scheuler este parte a lui SimulationManager => relatie de compozitie
    - SimulationManager este parte a lui View si invers => relatie de compozitie
    - SimulationManager implementeaza interfata Runnable si ActionListener, iar Server implementeaza interfata Runnable => relatie de de implementare
    - Server poate contine unul sau mai multe obiecte de clasa Task, dar acestea pot exista si independent => relatie de agregare
* Ca structuri de date am folosit BlockingQueue<Task>, implementata de Array BlockingQueue<Task>. De asemenea, am folosit variabile atomice de tipul AtomicInteger si o enumerare pentru tipul strategiei care are doua variante: SHORTEST\_QUEUE, SHORTEST\_TIME.
* O alta interfata folosita ce nu apare in diagrame este Comparable<Task>.

# Implementare

Clasa Task

* + - Are un ID ce reprezinta identificatorul clientului, timpul acestuia de sosire in coada si timpul in care este servit
    - Implementeaza interfata Comparable<Task> pentru a putea sorta task-urile in ordine crescatoare in functie de timpul de sosire in coada.



Clasa Server

* + - Contine un ID identificator al acestuia, o coada de task-uri si timpul de asteptare pentru coada.
    - Implementeaza interfata Runnable, deci implementeaza metoda run() in care se decrementeaza timpul de servire pentru clientii din capul cozii la fiecare secunda pana cand acesta pleaca din coada si vine urmatorul.



Clasa ShortestQueueStrategy

* + - Implementeaza interfata Strategy, odata cu metoda addTask(List<Server> servers, Task t).
    - Metoda determina coada care are cei mai putin client alocati si adauga clientul la coada respectiva.

Clasa TimeStrategy

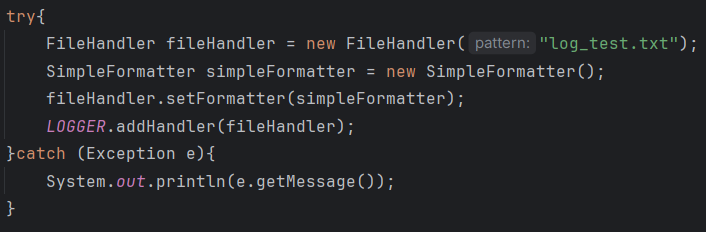
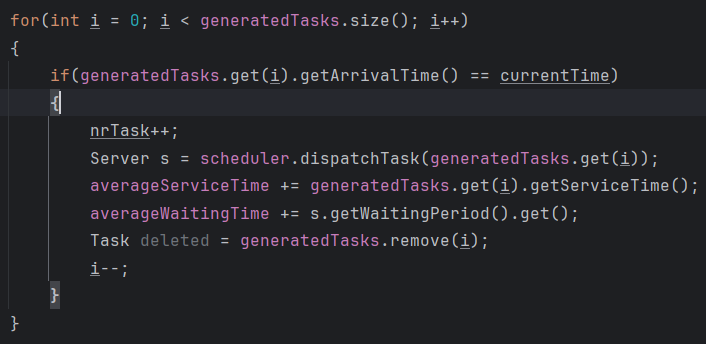
* + - Implementeaza interfata Strategy, odata cu metoda addTask(List<Server> servers, Task t).
    - Metoda determina coada care are cel mai mic timp de asteptare si adauga clientul la coada respectiva.

Clasa Scheduler

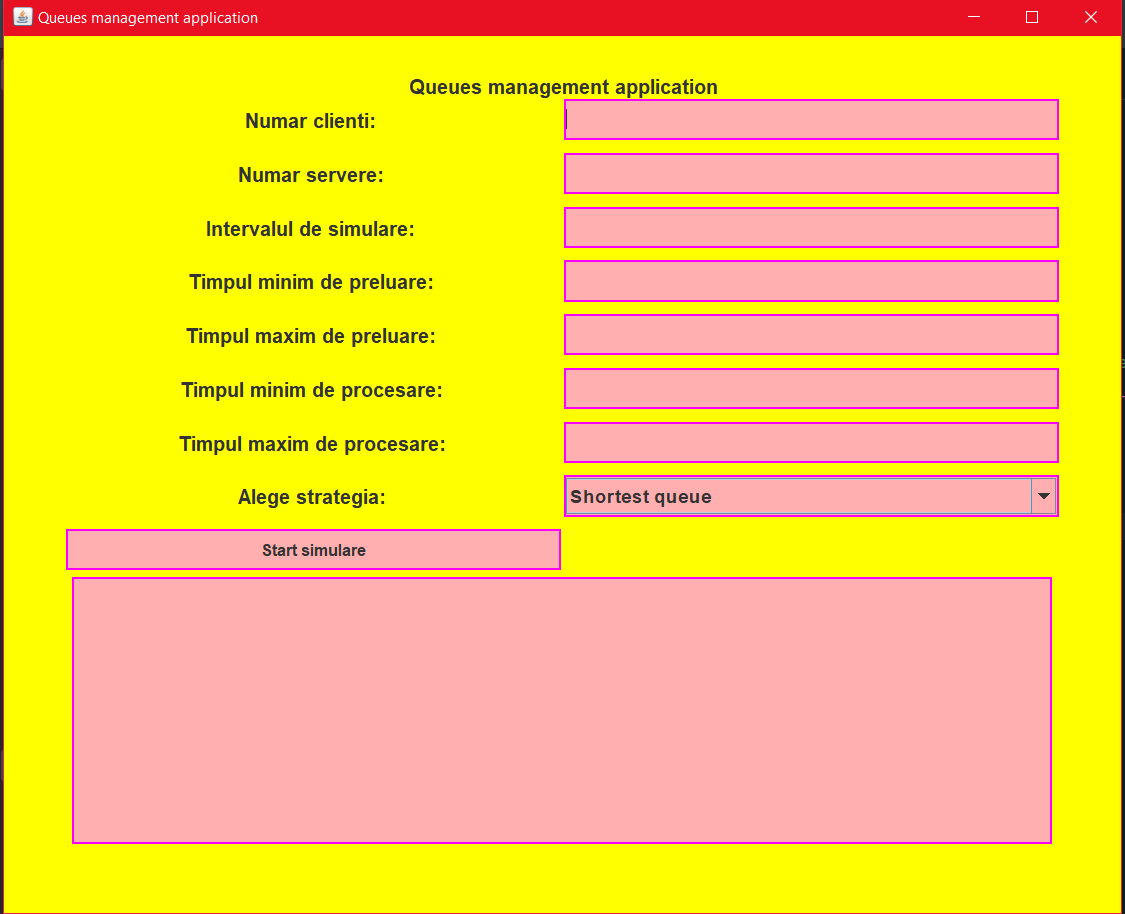
* + - Contine o lista cu server, numarul de servere si strategia conform careia se face adaugarea unui task la server.
    - Aceasta creeaza numarul noServers de servere si porneste un thread pentru fiecare server.
    - Metoda getTaskTotal() returneaza suma task-urilor din fiecare server (adica cate task-uri avem in server1 + cate task-uri avem in server2 + …etc).
    - Metoda getTotalWaitingTime() returneaza suma timpului de asteptare pentru fiecare coada.

Clasa SimulationManager

* + - Contine datele necesare pentru simulare, date introduse de utilizator in interfata.
    - In metoda generateNRandomTasks() ne generam random N client care au ID-ul, arrivalTime-ul si serviceTime-ul generate tot random, dar respectand conditiile impuse de datele de simulare.
    - Implementeaza interfata Runnable, deci si metoda run() in care folosim jurnalizarea pentru a scrie rezulatele simularii intr-un fisier si pentru fiecare timp din intervalul de simulare se adauga client in cozile corespunzatoare in functie de strategia aleasa. Pentru a putea prelua datele modificate, avand noua valoare, adormim thread-ul pentru 1.2 secunde.

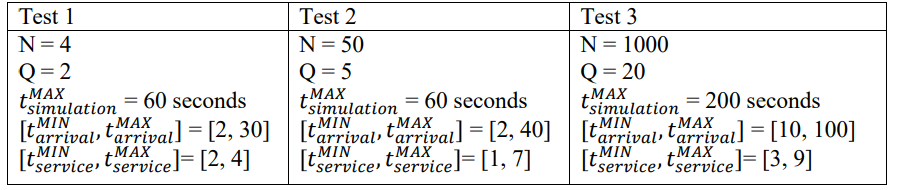


* + - Implementeaza interfata ActionListener, deci iplicit si metoda actionPerformed(ActionEvent e) pentru ca atunci cand utilizatorul apasa butonul de incepere a simularii, daca acestea sunt corecte si complete, se vor prelua si seta, iar mai apoi thread-ul isi poate incepe executia.

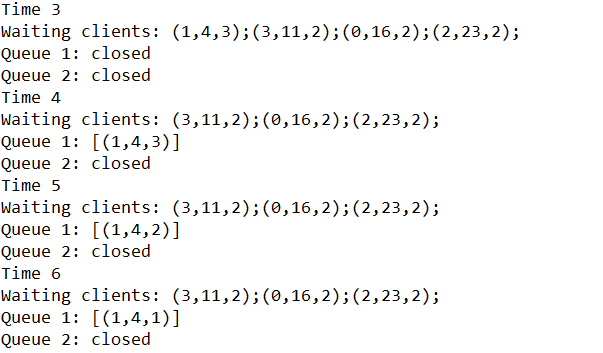
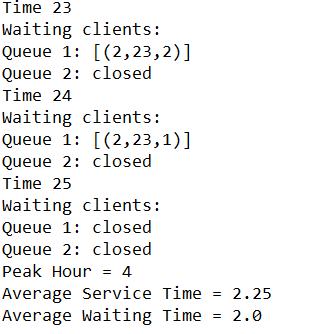
Clasa View

* + - Contine text field-uri, label-uri si un combo box pentru preluarea datelor de simulare si un text area pentru afisarea rezultatelor simularii in timp real.
    - Panoul principal este format din doua mai mici, unul pentru partea de date de intrare si unul pentru partea de afisare a rezulatatelor.

# Rezultate

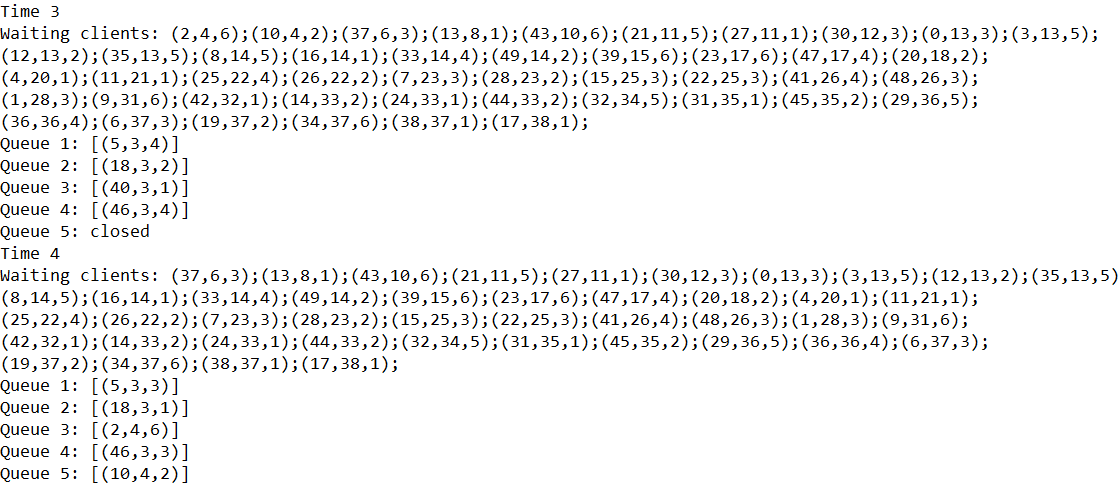


**TEST\_1**

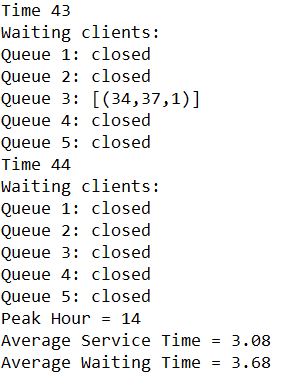
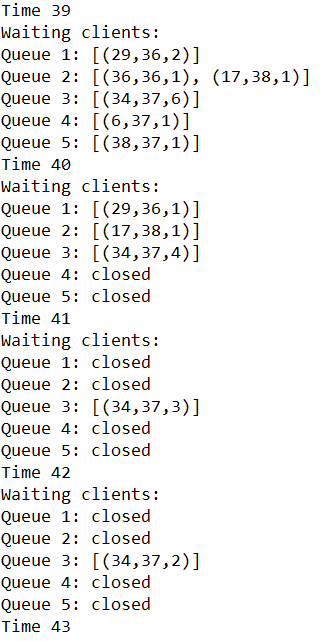
 

**…**

**TEST\_2**

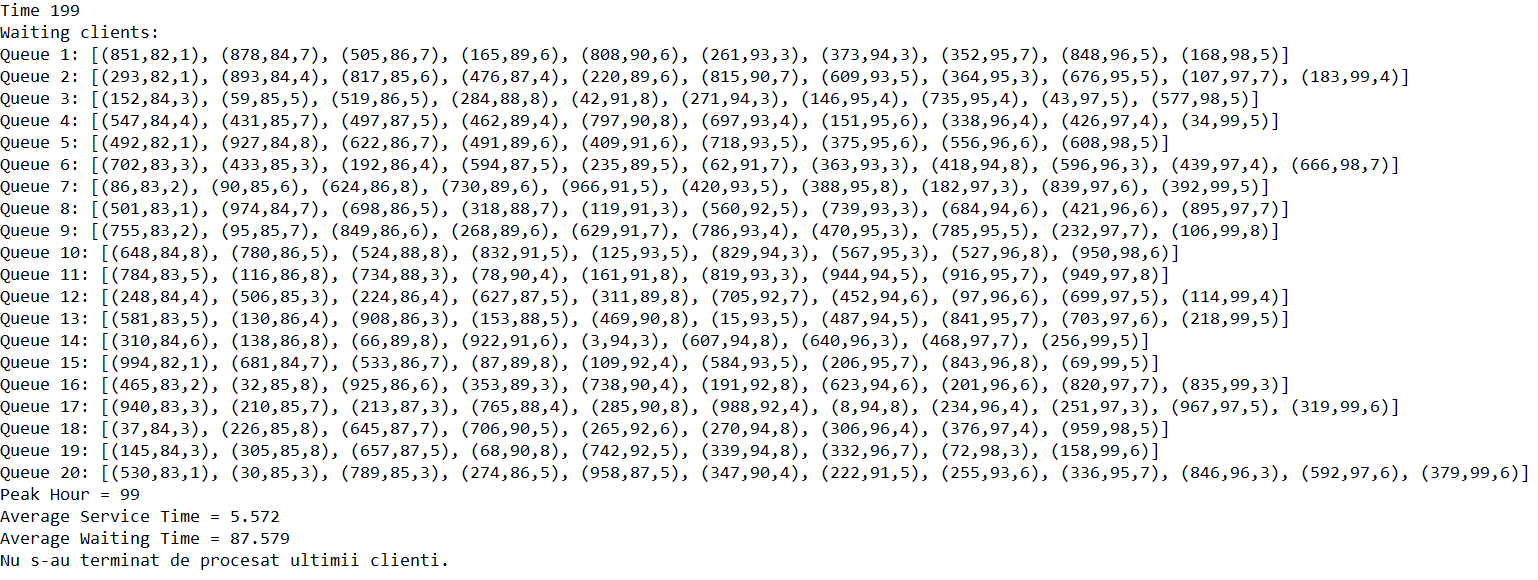


**….**

**TEST\_3**

**….**



! Rezultatul complet pentru cele 3 teste sunt atasate proiectului.

* + - In cazul in care intervalul de simulare ajunge la final si au mai ramas client in coada care nu au fost serviti se afiseaza un mesaj care sa anunte aceste lucru.
    - De asemenea, daca niciun client nu ajunge la coada pana se termina simularea sau ajung doar o parte in coada se semnaleaza la finalul simularii printr-un mesaj corespunzator.

# Concluzii

* + - *Posibile dezvoltari:* interfata poate sa fie imbunatatita si de asemenea sa se verifice ca utilizatorul nu introduce alte caractere in afara de cifre.

# Bibliografie

- [java.util.Timer.schedule() Method (tutorialspoint.com)](https://www.tutorialspoint.com/java/util/timer_schedule_period.htm)

- <http://www.tutorialspoint.com/java/util/timer_schedule_period.htm>

- [Java Thread Pool Example using Executors and ThreadPoolExecutor (javacodegeeks.com)](https://www.javacodegeeks.com/2013/01/java-thread-pool-example-using-executors-and-threadpoolexecutor.html)