DOCUMENTAȚIE

TEMA 2

NUME STUDENT: Vlase Rafaella

GRUPA: 30225

# CUPRINS

[1. Obiectivul temei 3](#_Toc95297885)

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

[3. Proiectare 5](#_Toc95297887)

[4. Implementare 6](#_Toc95297888)

[5. Rezultate 10](#_Toc95297889)

[6. Concluzii 11](#_Toc95297890)

[7. Bibliografie 11](#_Toc95297891)

# Obiectivul temei

Obiectivul principal al temei este implementarea unei aplicații Java de management a unor cozi de clienți, astfel încât clienții să fie adăugați la coada cu timpul minim de așteptare.

În interfața grafică se pot insera: numărul de cozi, numărul de clienți, timpul minim și maxim la care va ajunge un client, timpul minim și maxim de servire, precum și timpul maxim de simulare.

Programul va genera un Log Of Events pentru datele inserate.

Obiective secundare: analiza problemei și identificarea cerințelor, modelarea și implementarea unei aplicații de simulare, testarea aplicației de simulare.

Am avut de implementat următoarele:

Interfața grafică

Managerul simulării

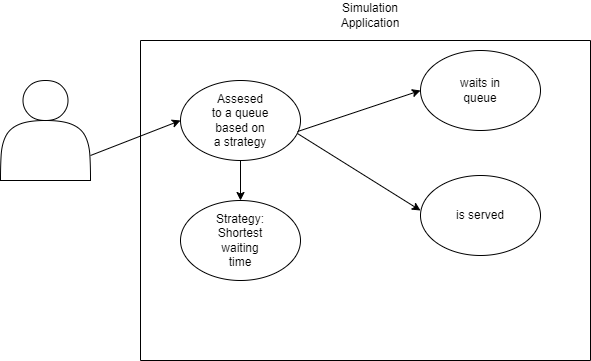
Cozile

Clienții

Controller-ul

Alte clase și funcții ajutătoare

# Analiza problemei, modelare, scenarii, cazuri de utilizare



Analiza:

Programul adaugă în fiecare secundă de simulare clienții care ajung în coada cu timpul de așteptare cel mai scurt, urmând să îi scoată din coadă după ce au fost serviți.

Modelare:

Clasa Task reprezintă clienții, cu atributele specifice: ID-ul fiecăruia, timpul la care ajung în coadă și timpul de servire.

Clasa Server reprezintă cozile și conține coada în care sunt înșirați clienții, precum și timpul de așteptare.

Scenarii:

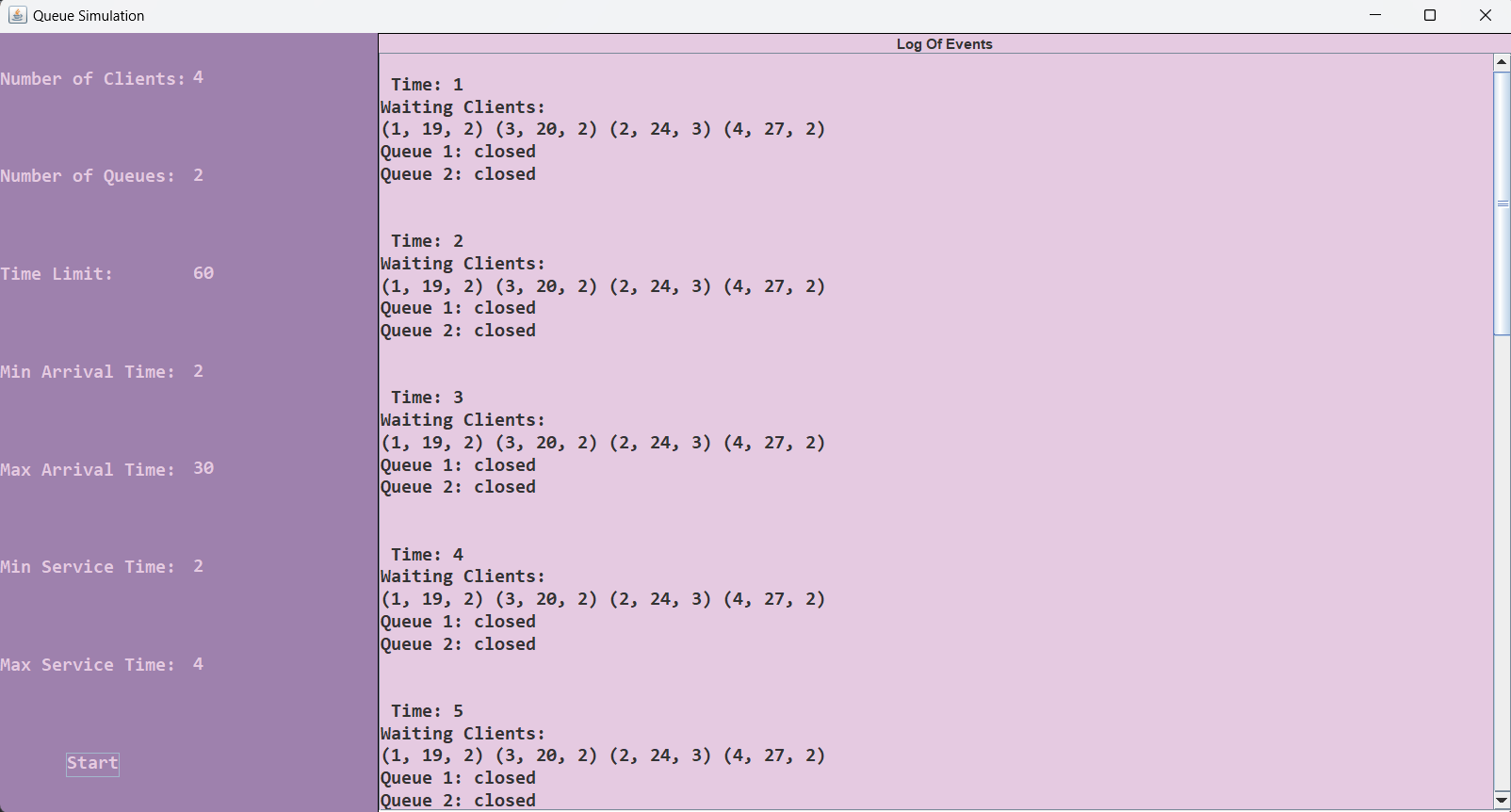
Programul se oprește atunci când toate cozile sunt goale după servirea tuturor clienților sau dacă se ajunge la limita timpului de simulare fără golirea tuturor cozilor.

Timpul minim de servire trebuie să fie mai mic sau egal cu timpul maxim de servire.

Timpul minim de ajuns la coadă trebuie să fie mai mic sau egal cu timpul maxim de ajuns la coadă.

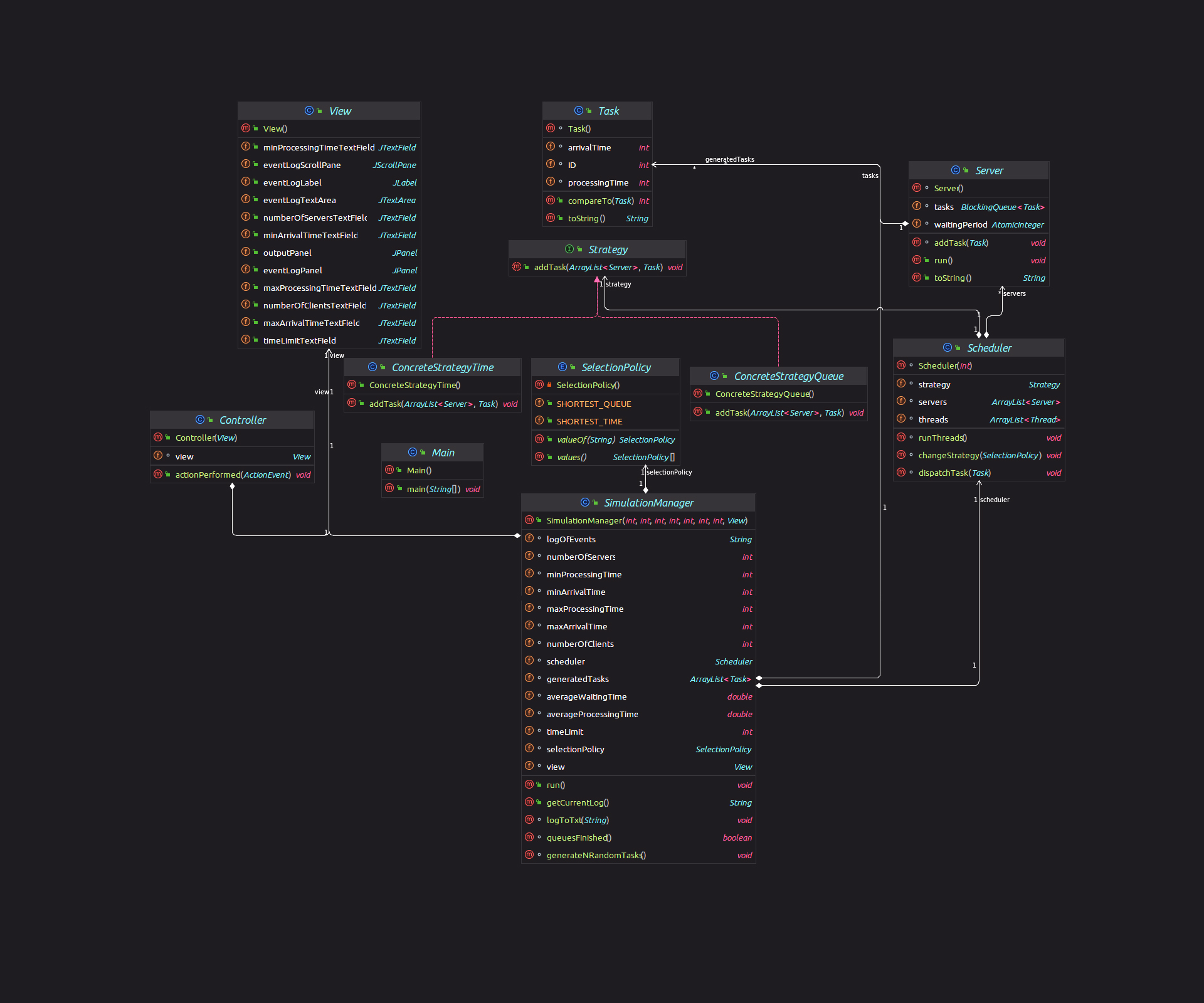
Use cases:

În interfața grafică pot fi inserate datele în câmpurile din partea stângă a ferestrei. După inserare utilizatorul poate apăsa pe butonul “Start”, după care va porni simularea. Log-ul de evenimente va fi afișat în chenarul din partea dreaptă a ferestrei.

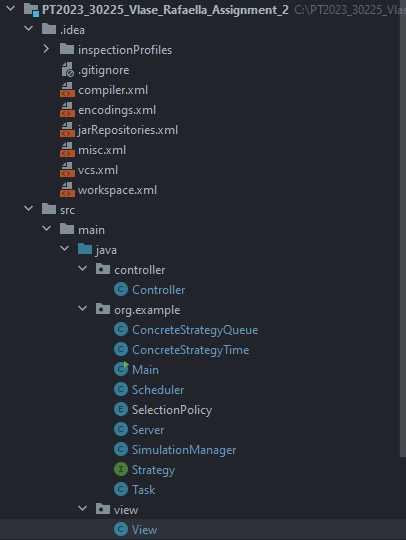


# Proiectare

Diagrama UML:



Ierarhia de fișiere a proiectului:



În clasa Main este apelat constructorul clasei View, instanțiind interfața grafică.

# Implementare

Clasa ConcreteStrategyQueue

Metoda "addTask" este suprascrisă din interfața "Strategy" și îndeplinește următoarea funcție: adaugă un client la serverul cu cea mai scurtă coadă. Metoda caută serverul cu cel mai mic număr de clienți din coadă și adaugă clientul la acesta.

Clasa ConcreteStrategyTime

Metoda "addTask" este suprascrisă din interfața "Strategy" și îndeplinește următoarea funcție: adaugă un client la serverul cu cel mai mic timp de așteptare. Metoda caută serverul cu cel mai scurt timp de așteptare și adaugă clientul la acesta.

Clasa Main

În clasa Main este apelat constructorul clasei View, instanțiind interfața grafică.

Clasa Scheduler

Atributul servers este o listă de obiecte de tip Server, iar atributul threads este o listă de fire de execuție (Thread) corespunzătoare fiecărui server. Atributul strategy este un obiect de tipul unei clase care implementează un anumit algoritm de planificare.

Constructorul Scheduler primește ca parametru numărul de servere și creează câte un obiect de tip Server pentru fiecare server, stocându-le în lista servers, și un fir de execuție corespunzător fiecărui server, stocându-le în lista threads.

Metoda changeStrategy primește ca parametru o politică de selecție a serverului și setează algoritmul de planificare corespunzător acelei politici, creând un nou obiect de tipul unei clase care implementează acel algoritm de planificare.

Metoda dispatchTask primește ca parametru un client (task) și îl planifică pe unul dintre servere, folosind algoritmul de planificare curent.

Metoda runThreads pornește toate firele de execuție corespunzătoare serverelor.

Clasa SelectionPolicy

Clasa SelectionPolicy este o enumerare (enum) care conține două constante: SHORTEST\_QUEUE și SHORTEST\_TIME. Această clasă este utilizată în clasa Scheduler pentru a specifica politica de selecție a serverelor în funcție de criteriile date: timpul de așteptare sau numărul de clienți în așteptare.

Clasa Server

Clasa Server este o clasă care implementează un server capabil să primească clienți (tasks) și să îi proceseze. Atributul tasks este o coadă blocaj care conține sarcinile primite de server, iar atributul waitingPeriod este un contor atomic care calculează timpul total de așteptare al tuturor clienților din coadă.

Constructorul clasei creează o coadă blocantă și inițializează contorul atomic waitingPeriod cu valoarea 0.

Metoda addTask primește ca parametru un client (task), îl adaugă în coada tasks și incrementază waitingPeriod cu timpul necesar servirii.

Metoda run este suprascrisă din interfața Runnable și implementează logica de procesare a clienților din coadă. Metoda verifică dacă coada tasks nu este goală. Dacă nu este goală, decrementăm timpul necesar servirii clientului curent, decrementăm waitingPeriod și verificăm dacă clientul a fost servit. Dacă da, eliminăm clientul din coadă. Dacă coada este goală, serverul va aștepta în bucla infinită pentru a primi un nou client.

Metoda toString este suprascrisă pentru a returna o reprezentare sub formă de șir de caractere a clienților aflați în coada tasks.

Clasa SimulationManager

Clasa SimulationManager are următoarele atribute:

timeLimit: un întreg ce reprezintă timpul maxim de simulare

maxProcessingTime: un întreg ce reprezintă timpul maxim de procesare al unui client

minProcessingTime: un întreg ce reprezintă timpul minim de procesare al unui client

maxArrivalTime: un întreg ce reprezintă momentul maxim de timp la care un client ajunge la coadă

minArrivalTime: un întreg ce reprezintă momentul minim de timp la care un client ajunge la coadă

numberOfServers: un întreg ce reprezintă numărul de cozi disponibile

numberOfClients: un întreg ce reprezintă numărul de clienți

averageProcessingTime: un număr real ce reprezintă timpul mediu de servire al clienților

averageWaitingTime: un număr real ce reprezintă timpul mediu de așteptare al clienților în coadă

logOfEvents: un șir de caractere ce reprezintă jurnalul evenimentelor

selectionPolicy: o variabilă de tip SelectionPolicy ce reprezintă politica de alegere a cozii

view: o instanță a clasei View pentru a afișa simularea

scheduler: o instanță a clasei Scheduler pentru a gestiona coada

generatedTasks: o listă de instanțe ale clasei Task, care reprezintă clienții care așteaptă să fie serviți

Clasa SimulationManager are următoarele metode:

generateNRandomTasks(): generează o listă de clienți aleatorii, setând valori aleatoare pentru processingTime și arrivalTime și sortând lista după arrivalTime

queuesFinished(): verifică dacă toate cozile sunt goale și nu mai sunt clienți care să aștepte să fie serviți

run(): implementează metoda run() din interfața Runnable, ce se execută într-un fir de execuție separat și simulează comportamentul cozii în timp real. Metoda alege clienții care ajung la coadă în acel moment și îi distribuie în cozi. De asemenea, metoda actualizează interfața grafică și jurnalul de evenimente. La final, metoda afișează timpul mediu de procesare și de așteptare al clienților și scrie jurnalul de evenimente într-un fișier text.

getCurrentLog(): această metodă returnează un șir de caractere care conține informații despre starea curentă a simulatorului.

logToTxt(String logOfEvents): această metodă primește ca parametru un șir de caractere care conține informații despre evenimente și le scrie într-un fișier text denumit "Simulation Log.txt".

Clasa Task

Clasa Task are trei atribute: ID-ul clientului, timpul de sosire și timpul de servire.

Clasa implementează interfața Comparable și supraîncarcă metoda compareTo pentru a compara doi clienți după timpul lor de sosire.

În plus, clasa supraîncarcă și metoda toString pentru a returna un șir de caractere care conține informații despre client în formatul "(ID, timp de sosire, timp de servire)".

Clasa Controller

Clasa implementează interfața ActionListener și are un atribut de tip View. Constructorul primește un obiect de tip View ca parametru și inițializează atributul. Metoda actionPerformed este suprascrisă și verifică dacă butonul apăsat este "Start", apoi extrage valorile introduse de utilizator în câmpurile de text ale interfeței și creează un obiect de tip SimulationManager, care este apoi executat într-un fir de execuție separat. Clasa este responsabilă de gestionarea evenimentelor din interfața grafică și de inițierea simulării.

Clasa View

Clasa View este o extensie a clasei JFrame si are urmatoarele atribute:

numberOfServersTextField, numberOfClientsTextField, minArrivalTimeTextField, maxArrivalTimeTextField, minProcessingTimeTextField, maxProcessingTimeTextField, timeLimitTextField: toate acestea sunt obiecte JTextField folosite pentru a lua date de intrare de la utilizator.

eventLogPanel: este un obiect JPanel folosit pentru a stoca elementele necesare pentru a afișa jurnalul evenimentelor în interfața grafică.

eventLogLabel: este un obiect JLabel pentru etichetarea jurnalului evenimentelor.

eventLogTextArea: este un obiect JTextArea care va fi utilizat pentru a afișa jurnalul evenimentelor.

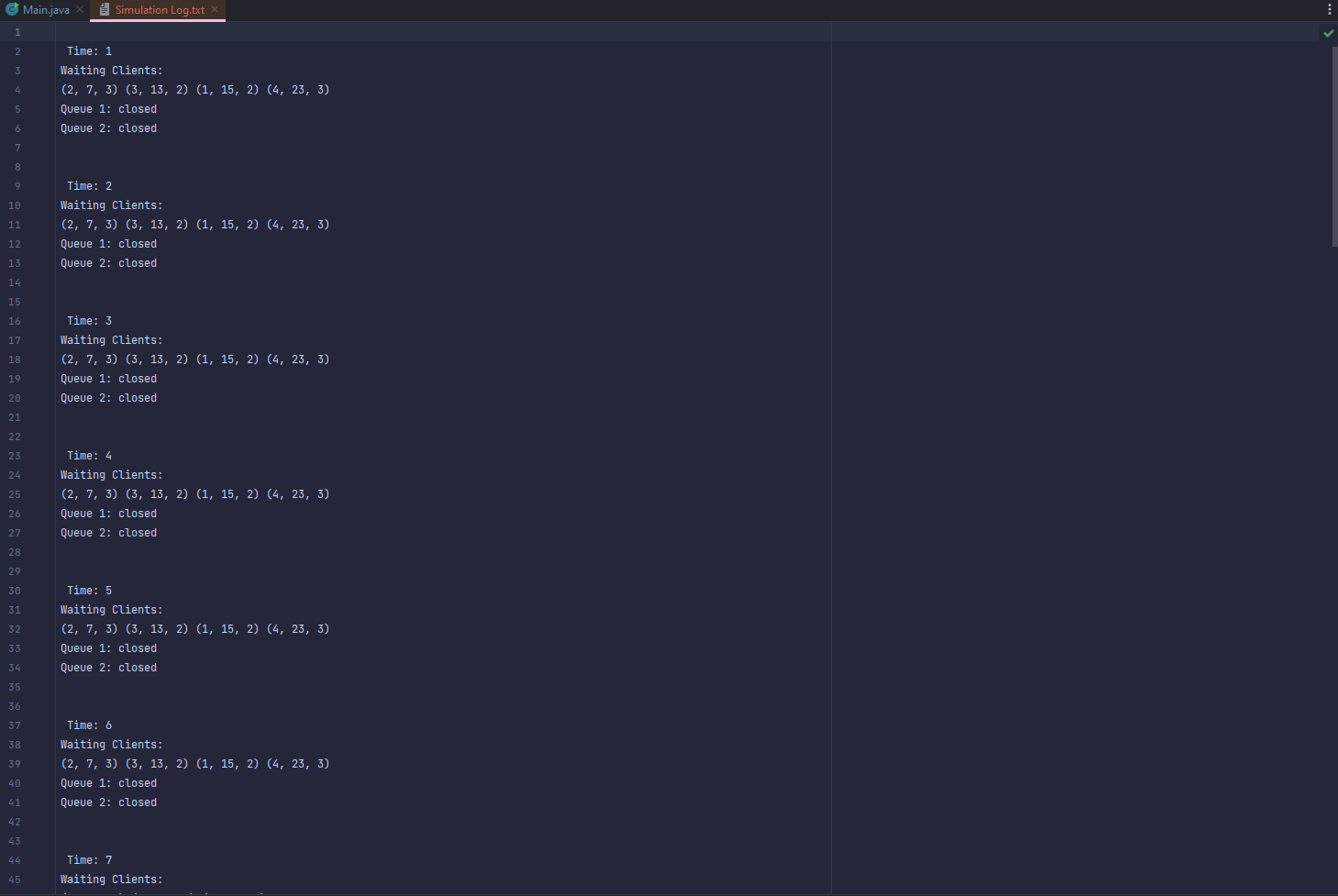
eventLogScrollPane: este un obiect JScrollPane pentru a permite utilizatorului să deruleze jurnalul evenimentelor.

outputPanel: este un obiect JPanel care va fi utilizat pentru a afișa rezultatele simulării cozii.

Clasa View are, de asemenea, un constructor fără argumente care inițializează toate obiectele menționate mai sus și setează aspectul interfeței grafice.

# Rezultate

Rezultatele testărilor pot fi găsite în fișierele text “Simulation Log.txt”, sau în aplicație, prin introducerea datelor în interfața grafică și rularea simulării.



# Concluzii

Cu ajutorul acestei teme am învățat cum să implementez o aplicație de tip management client-server și cum sa lucrez cu fire de execuție.

Ca dezvoltare ulterioară se poate implementa monitorizarea timpului de așteptare: Dacă clienții pot vedea timpul mediu de așteptare pentru a fi servit, aceștia pot decide să vină la altă oră sau să aleagă o altă locație. Acest lucru poate ajuta la gestionarea fluxului de clienți.

# Bibliografie

<https://www.tutorialspoint.com/java/util/timer_schedule_period.html>

<https://www.javacodegeeks.com/2013/01/java-thread-pool-example-using-executors-and-threadpoolexecutor.html>

<https://docs.oracle.com/javase/7/docs/>

<https://dsrl.eu/courses/pt/materials/PT2023_A2_S1.pdf>

<https://www.geeksforgeeks.org/introduction-to-java-swing/>