DOCUMENTATIE

TEMA *2*

NUME STUDENT: Tîrcă Mihnea

GRUPA: 30224

# CUPRINS

[1. Obiectivul temei 3](#_Toc95297885)

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

[3. Proiectare 3](#_Toc95297887)

[4. Implementare 3](#_Toc95297888)

[5. Rezultate 3](#_Toc95297889)

[6. Concluzii 3](#_Toc95297890)

[7. Bibliografie 3](#_Toc95297891)

# Obiectivul temei

Obiectivul acestei teme este de a implementa o aplicatie de management a unor cozi, astfel incat clientii sa fie adaugati in coada cu cel mai mic timp de asteptare. Acest program va fi implementat intr-o interfata grafica, unde datele de intrare pot fi inserate (numarul de cozi, numarul de clienti, timpul minim si maxim la care ajunge un client, timpul minim si maxim de procesare si timpul maxim de simulare). Dupa inserarea datelor de intrare, programul va genera date despre fiecare client cu parametrii inserati.  
  
Au trebuit implementate:

Interfata grafica

Managerul simularii, care contine datele de intrare si este responsabil cu mutarea clientilor din lista de asteptare in una dintre cozi, la fiecare moment de timp

Cozile

Controller-ul, care preia datele din interfata grafica

Clientii

Diferite functii ajutatoare

# Analiza problemei, modelare, scenarii, cazuri de utilizare

**Analiza:**

Programul trebuie, in fiecare secunda de simulare, sa adauge clientii care ajung, in coada cu timpul de asteptare cel mai scurt si de asemenea sa scoata din cozi clientii care au terminat.

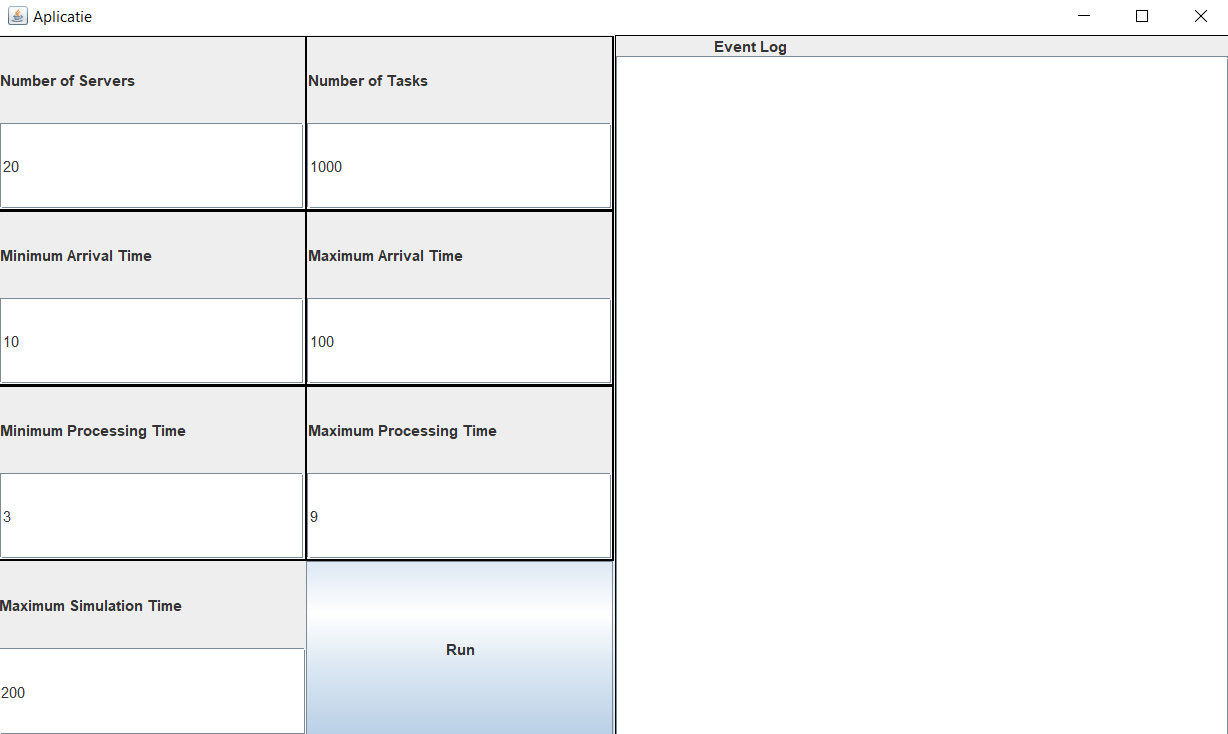
**Modelare:**

Structura de date „Task” reprezinta clientii. Aceasta contine atributele specifice (id-ul, timpul in care ajunge si timpul de procesare).  
  
Structura de date „Server” reprezinta cozile. Aceasta contine efectiv coada cu clienti, cat si timpul de asteptare total.

**Scenarii:**

Daca se ajunge la timpul de simulare fara terminarea efectiva a tuturor cozilor, programul se opreste. Timpul minim de procesare trebuie sa fie mai mic sau egal cu timpul maxim de procesare. La fel este valabil si in cazul timpului de ajungere. In cazul in care input-ul nu este valid, utilizatorul va fi atentionat.

**Cazuri de utilizare**

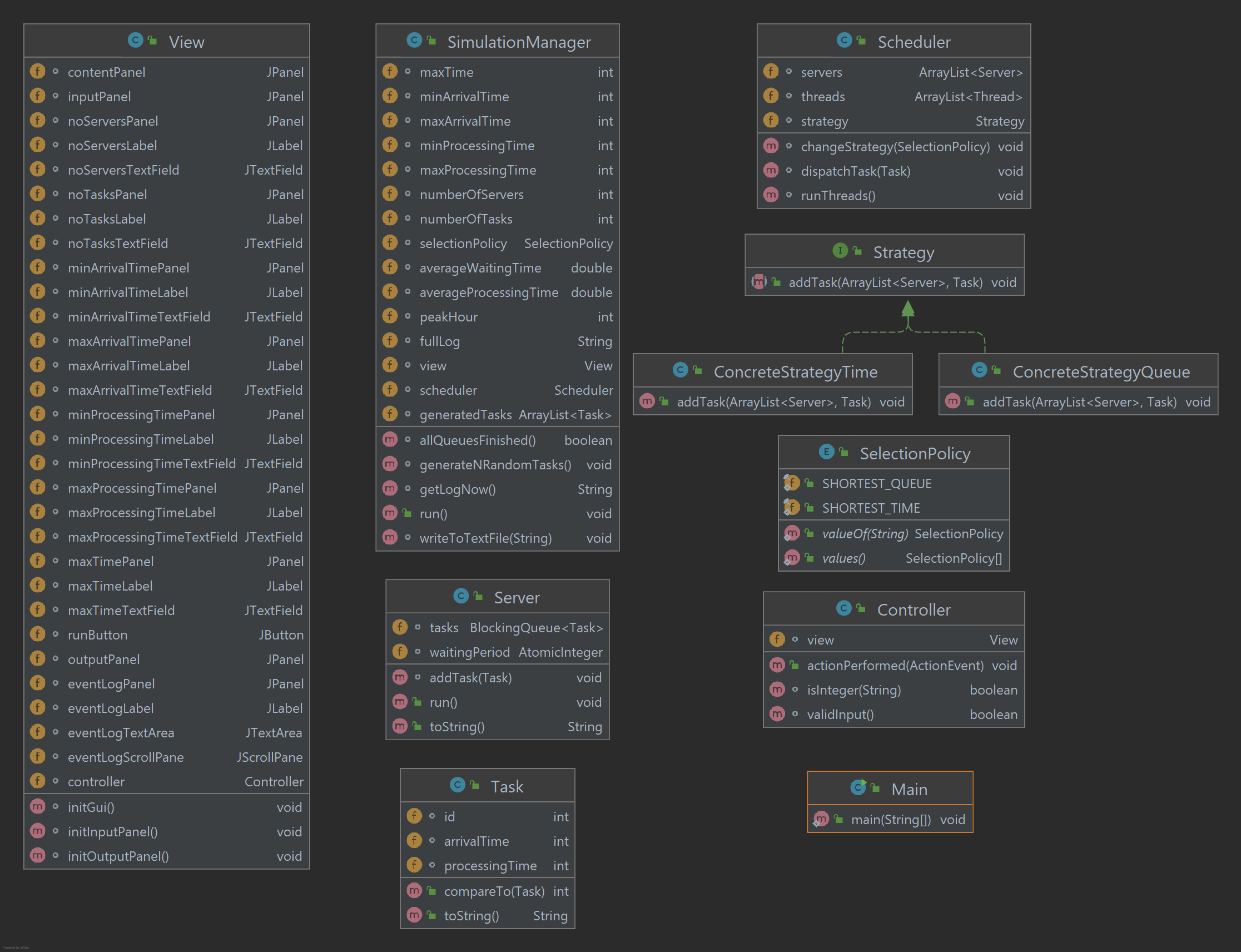


In interfata grafica se pot insera datele pe partea stanga a ferestrei. Dupa inserare, acesta poate apasa butonul „Run”, care va porni simularea cu parametrii inserati. Log-ul va fi afisat in partea dreapta a ferestrei.

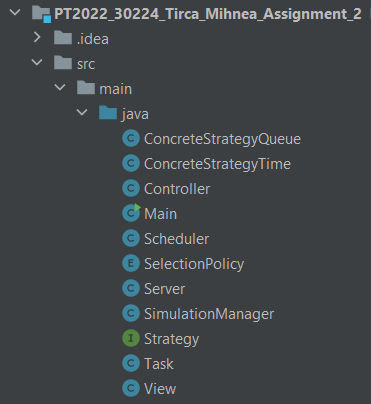
# Proiectare

Atributele fiecarei clasa sunt declarate implicit package-private.

Diagrama UML:



Ierarhia de fisiere a proiectului:



Din main se apeleaza constructorul clasei View, care mosteneste clasa JFrame, astfel instantiindu-se interfata grafica.

# Implementare

**Clasa View:**

Clasa view contine toate componentele interfetei grafice. Atributele sunt:

JPanel contentPanel;

Panel-ul principal

JPanel inputPanel;

Panel-ul de unde se citesc datele de intrare.

JPanel noServersPanel;  
JLabel noServersLabel;  
JTextField noServersTextField;

Panel-ul de unde se citeste numarul de cozi. Acesta contine un Label si un TextField unde sunt inserate datele.

JPanel noTasksPanel;  
JLabel noTasksLabel;  
JTextField noTasksTextField;

Panel-ul unde se citeste numarul de clienti. Acesta contine un Label si un TextField unde sunt inserate datele.

JPanel minArrivalTimePanel;  
JLabel minArrivalTimeLabel;  
JTextField minArrivalTimeTextField;

Panel-ul unde se citeste timpul minim de ajungere. Acesta contine un Label si un TextField unde sunt inserate datele.

JPanel maxArrivalTimePanel;  
JLabel maxArrivalTimeLabel;  
JTextField maxArrivalTimeTextField;

Panel-ul unde se citeste timpul maxim de ajungere. Acesta contine un Label si un TextField unde sunt inserate datele.

JPanel minProcessingTimePanel;  
JLabel minProcessingTimeLabel;  
JTextField minProcessingTimeTextField;

Panel-ul unde se citeste timpul minim de procesare. Acesta contine un Label si un TextField unde sunt inserate datele.

JPanel maxProcessingTimePanel;  
JLabel maxProcessingTimeLabel;  
JTextField maxProcessingTimeTextField;

Panel-ul unde se citeste timpul maxim de procesare. Acesta contine un Label si un TextField unde sunt inserate datele.

JPanel maxTimePanel;  
JLabel maxTimeLabel;  
JTextField maxTimeTextField;

Panel-ul unde se citeste timpul maxim de simulare. Acesta contine un Label si un TextField unde sunt inserate datele.

JButton runButton;

Butonul care da start simularii.

JPanel outputPanel;

Panel care contine datele de iesire, adica log-ul.

JPanel eventLogPanel;  
JLabel eventLogLabel;  
JTextArea eventLogTextArea;  
JScrollPane eventLogScrollPane;

Panel-ul unde este afisat log-ul. Acesta contine un Label si un TextArea, unde sunt afisate informatiile. Panel-ul contine de asemenea o bara de scroll (Scroll Pane).

Controller controller = new Controller(this);

Instantierea clasei Controller, responsabila pentru actiunile utilizatorului asupra interfetei grafice.

void initGui()

Metoda responsabila cu initializarea interfetei grafice.

void initInputPanel()

Metoda responsabila cu initializarea Panel-ului cu datele de intrare.

void initOutputPanel()

Metoda responsabila cu initializarea Panel-ului cu informatiile simularii (datele de iesire).

**Clasa Controller**

Clasa Controller este responsabila cu interpretarea actiunilor utilizatorului asupra interfetei grafice.

View view;

Obiect de tip View.

boolean isInteger(String s)

Metoda ajutatoare pentru validarea input-ului.

boolean validInput()

Valideaza input-ul.

@Override  
public void actionPerformed(ActionEvent e)

Metoda din interfata ActionListener. Cand utilizatorul apasa butonul de „Run” al simularii, datele sunt preluate din interfata grafica.

**Clasa Task**

Clasa Task reprezinta clientii.

int id;  
int arrivalTime;  
int processingTime;

Datele specifice fiecarui client.

@Override  
public int compareTo(Task o) {  
 return arrivalTime - o.arrivalTime;  
}

Metoda din interfata Comparable<>. Aceasta este folosita pentru sortarea clientilor (dupa generarea lor) in functie de timpul de ajungere a fiecaruia.

@Override  
public String toString()

Metoda de scriere intr-un string a clientului in format (id, arrivalTime, processingTime).

**Clasa Server**

Clasa Server reprezinta cozile. Implementeaza interfata Runnable.

BlockingQueue<Task> tasks;  
AtomicInteger waitingPeriod;

Coada cu clienti si timpul de asteptare a cozii.

@Override  
public void run()

Aceasta metoda este implementata din interfata Runnable. La fiecare secunda, clientului din fata cozii i se decrementeaza timpul de procesare. Se decrementeaza de asemenea timpul de procesare a cozii in intregime. Daca timpul de procesare este dupa decrementare 0, clientul este scos din coada.

void addTask(Task newTask)

Metoda responsabila strict cu adaugarea unui client in coada (fara alte conditii). Timpul de asteptare total al cozii este incrementat.

@Override  
public String toString()

Metoda de scriere intr-un string a cozii, prin apelarea metodei toString() din clasa Task, pentru fiecare client din coada.

**Interfata Strategy**

Aceasta interfata contine doar o metoda de adaugare a unui client. Ea este apelata din clasa Scheduler cu o anumita strategie: in functie de timpul de asteptare a cozilor sau de dimensiunea cozilor. Antetul metodei este:

void addTask(ArrayList<Server> servers, Task t)

**Clasa ConcreteStrategyQueue**

Aceasta clasa implementeaza interfata Strategy definita de programator. Ea contine doar o metoda care insereaza un client in coada cu cei mai putini clienti.

@Override  
public void addTask(ArrayList<Server> servers, Task t)

**Clasa ConcreteStrategyTime**

Aceasta clasa implementeaza interfata Strategy definita de programator. Ea contine doar o metoda care isnereaza un client in coada cu timpul de asteptare minim.

@Override  
public void addTask(ArrayList<Server> servers, Task t)

**Clasa Scheduler**

Clasa Scheduler este responsabila cu asocierea fiecarei cozi cu un thread, cu pornirea threadurilor si cu adaugarea clientilor in cozi in functie de strategie. Asocierea cozilor se face in constructorul clasei. In constructor se instantiaza de asemenea strategia cu un obiect de tip ConcreteStrategyTime.

ArrayList<Server> servers;

Vector cu cele Q cozi.

ArrayList<Thread> threads;

Vector cu cele Q thread-uri.

Strategy strategy;

Strategia de inserare (dupa lungimea cozii sau timpul de asteptare). In constructor este instantiat cu un obiect de tip ConcreteStrategyTime (dupa timpul de asteptare).

void changeStrategy(SelectionPolicy policy)

Metoda pentru schimbarea strategiei. Metoda nu este apelata niciodata, deoarece cerinta specifica strategia dupa timpul de asteptare.

void dispatchTask(Task t)

Metoda pentru adaugarea unui client t in vectorul servers, dupa strategia strategy.

void runThreads()

Metoda pentru inceperea fiecarui thread (in numar de Q).

**Clasa SimulationManager**

Clasa SimulationManager este responsabila cu stocarea datelor inserate de utilizator, generarea celor N clienti folosing aceste date, si trimiterea clientilor in cozi cand este timpul. Aceasta clasa stocheaza de asemenea log-ul, care este afisat in interfata grafica sau in fisiere.

int maxTime;  
int minArrivalTime;  
int maxArrivalTime;  
int minProcessingTime;  
int maxProcessingTime;  
int numberOfServers;  
int numberOfTasks;

Aceste atribute stocheaza datele introduse de utilizator.

SelectionPolicy selectionPolicy;

Acest atribut reprezinta modul de inserare (cel mai scurt timp de asteptare sau cea mai scurta coada). Atributul nu este folosit, din cauza specificatilor problemei.

double averageWaitingTime;  
double averageProcessingTime;  
int peakHour;

In aceste atribute vor fi calculate timpul mediu de asteptare, timpul mediu de procesare si ora de varf.

String fullLog;

In acest atribut de tip String este retinut la fiecare timp din simulare log-ul total de pana in acel moment.

View view;

Scheduler scheduler;

Obiecte de tip view si scheduler, initializate in constructor.

ArrayList<Task> generatedTasks;

Vector cu obiecte de tip Task(Client) care contine toti clientii generati. Cand un client intra intr-o coada, acesta este scos din vector. Adica, vectorul reprezinta toti clientii in lista de asteptare.

void generateNRandomTasks()

Metoda care genereaza N task-uri, respectand parametrii de intrare (datele inserate in interfata grafica).

boolean allQueuesFinished()

Metoda care returneaza true daca toate cozile sunt terminate (inclusiv coada cu lista de asteptare), sau fals in caz contrar.

@Override  
public void run()

Metoda implementata din interfata Runnable. Cat timp nu am ajuns la sfarsitul simularii (fie ajungand la timpul maxim de simulare, fie toate cozile sunt goale), atunci adaug toti clientii care au ajuns, intr-o coada. In acelasi timp, updatez log-urile si calculez ora de varf.

String getLogNow()

Metoda care returneaza un String cu log-ul la timpul curent.

void writeToTextFile(String fullLog)

Metoda care scrie in fisierul „Simulation Log.txt” String-ul dat ca parametru functiei.

**Clasa Main**

Clasa main instantiaza un nou obiect de tip View.

# Rezultate

Rezultatele testarilor sunt prezente in fisierele “Simulation Log 1-3.txt”, sau in aplicatie prin rularea simularii din interfata grafica.

# Concluzii *.*

Aceasta tema mi-a aprofundat cunostintele despre thread-uri. Mai mult, am invatat cum sa fac o aplicatie de management de tip client-server.

# Bibliografie

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

<https://geeksforgeeks.org/>