

**QUEUES SIMULATOR**

**DOCUMENTAȚIE**

**Facultatea:** Automatică și Calculatoare

**Specializarea:** Calculatoare și Tehnologia Informației

**Materia:** Tehnici de programare fundamentale

**Nume student:** Petricele Mihaela **Profesor coordonator**

**Grupa:** 30228 Dr. Pop Cristina Bianca

# **Obiectivul temei**

*Obiectivul principal* al temei este proiectarea și implementarea unei simulări de cozi, care urmărește timpul de așteptare a clienților la cozi și calculeaza timpul mediu de așteptare.

*Obiectivele secundare* ale temei sunt:

* **Analiza problemei** (în capitolul 2) => se prezintă cerințele funcționale, cazurile de utilizare sub formă de

diagrame use-case

* **Proiectare** (în capitolul 3) => proiectarea OOP aplicației, diagramele UML de clase și de pachete,

structurile de date folosite, interfețele definite și algoritmii folosiți.

* **Implementare** (în capitolul 4) => descrierea claselor, metodelor folosite și descrierea implementării

interfeței cu utilizatorul

* **Rezultate** ( în capitolul 5) => prezentarea scenariilor de testare

# **Analiza problemei, modelare, scenarii, cazuri de utilizare**

Aplicația ar trebui să simuleze (prin definirea unui timp de simulare) o serie de N clienți care ajung la cozi, așteaptă, sunt serviți și, în cele din urmă, părăsesc cozile. Toți clienții sunt generați la pornirea simulării și sunt caracterizați de trei parametri: ID (un număr între 1 și N), timpul de sosire (timpul de simulare când clienții sunt gata să meargă la coadă) și timpul de servire(intervalul de timp sau durata necesară pentru a servi clientul). Aplicația urmărește timpul total petrecut de fiecare client la coadă și calculează timpul mediu de așteptare. Fiecare client este adăugat la coadă cu timp minim de așteptare atunci când timpul său de sosire este mai mare sau egal cu timpul de simulare.

Aplicația trebuie să permită utilizatorului:

* Să introducă prin intermediul unei interfețe grafice următoarele date:
* timpul simulării
* numărul clienților
* timpul minim și timpul maxim de servire
* timpul minim și timpul maxim de sosire la coadă
* Să observe rezultatele prin intermediul interfeței și a unui fișier text

## ***Diagrama Use – case a aplicației***

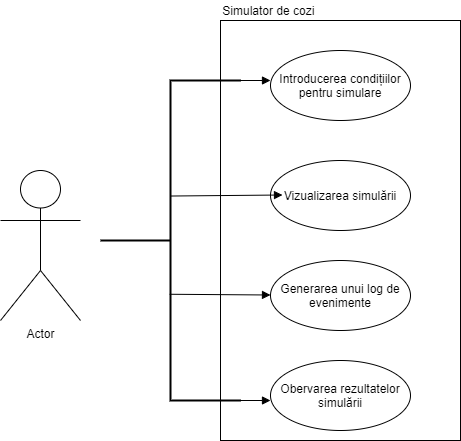


Figura 2.1 Diagrama Use-case

## ***Descrierea cazurilor de utilizare***

***Use case:*** Introducerea condițiilor de simulare

***Scenariul de success:*** Utilizatorul introduce toate valorile corect și în momentul apăsării butonului de submit simularea pornește.

***Scenariu alternativ:*** Utilizatorul cel mult o valoare greșită, caz în care se va afișa un mesaj de eroare.

***Use Case:*** Vizualizarea simulării

***Scenariul de succes:*** Când este pornit butonul de rulare a aplicației, fereastra de simulare apare și se pot introduce datele dorite. La apăsarea butonului de „Submit” se pornește simularea și se afișeaza datele corect la fiecare moment de timp.

***Scenariu alternativ:*** În momentul în care se apasă butonul de rulare, aplicația aruncă o eroare sau aplicația afișează fereastra de simulare dar la apăsarea butonului de ,,Submit” se afișează în mod incorect datele.

***Use case:*** Generarea unui log de evenimente

***Scenariul de succes:*** La apăsarea butonului de „Submit” se crează fisierul .txt corespunzator rezultatului simulării si se afișează corect log-ul de evenimente, conform celui afișat în caseta de text.

***Scenariu alternativ:*** În momentul în care se apasă butonul de submit, apare o eroare la crearea fișierului .txt sau se creează fișierul dar generarea evenimentelor nu se realizează în mod corect.

***Uuse case:*** Observarea rezultatelor simulării

***Scenariul de succes:*** Rezultatele simulării sunt:

* average waiting time
* average service time
* peak hour

Aceste rezultate se afișează în mod corect în fereastra de simulare sau în fișierul .txt la finalul simulării.

***Scenariu alternativ:*** Nu se afișează toate rezultatele simulării din pricina unei erori sau dacă se afișează, valoarea acestora nu este una corectă.

# **Proiectare**

## *Proiectarea OOP a aplicației*

Aplicația este împărțită în patru pachete: Business, Logica, Model și GUI. În pachetul Business se află

Interfața Strategy cu metoda *addTask()* și clasa de tip enum Selection, iar în clasele ConctreteStrategyTime și ConcreteStrategyQueue se descrie strategia de distibuire la cozi a clienților. În pachetul Model se implemetează clasa Task reprezentând clasa pentru clienți și clasa Server, reprezentând cozile. În pachetul Logica se descrie logica aplicației, în clasa SimulationManager implementându-se generarea random a clienților distribuiți la cozi. Într-un final în pachetul GUI se implementează cu ajutorul claselor SimulationFrame și ViewController, interfața cu utilizatorul.

1. *Diagramele UML ale aplicației*

* **Diagrama de pachete**

**O imagine care conține text, electronice, captură de ecran

Descriere generată automat**

*Figura 3.1 Diagrama de pachete*

* O imagine care conține text, ecran, negru, captură de ecran

  Descriere generată automat**Diagrama de clase**

*Figura 3.2 Diagrama de clase*

1. *Structurile de date folosite*

Pentru implementarea acestei aplicații am folosit fire de lucru, deoarece dorim ca executia evenimentelor să se întâmple în același moment de timp. Pentru lucrul cu thread-uri am folosit tipurile de date: AtomicInteger și BlockingQueue, deoarece sunt deja sincronizate.

Totodată am folosit fișiere pentru a afișa log-ul de evenimte.

1. *Interfețele definite*

În cadrul acestei teme am lucrat cu următoarele interfețe pentru a mă ajuta sa implementez mai eficient

aplicația:

* + Interfața ActionListener
  + Interfața generică Comparator
  + Interfața Runnable

# **Implementare**

În acest capitol se prezintă clasele și metode corespunzătoare care au ajutat la realizarea aplicației

Simulator de cozi.

1. **Pachetul Model**
2. **Clasa Task**

Această clasă conține metodele pentru descrierea task-urilor, adică a clienților care sunt

distribuiți la cozi. Clasa are trei atribute: ID care reprezintă id-ul fiecărui client, timp\_sosire care reprezintă momentul in care clientul ajunge la coadă și timp\_servire care reprezintă timpul în care clientul este procesat. Clasa mai conține și atributul waitingTime, care reprezintă timpul cât a stat un client la coadă. Totodată, metodele din această clasă sunt reprezentate de gettere și settere.

1. **Clasa Server**

Această clasă conține metodele pentru descrierea cozilor la care vor fi distribuiți clienții. Pentru a

reține task-urile am folosit o coadă de tip BlockingQueue, deoarece aceasta este deja sincronizată și timpul de așteptare al fiecărei cozi este de tip AtomicInteger pentru ca operațiile de incrementare și decrementare să se realizeze în mod coret, acest tip fiind la randul său deja sincronizat. Un alt atribut al fiecărei cozi este reprezentat de id-ul sau numărul acesteia *nrCoada.*

Această clasă extinde Interfața *Runnable* pentru a permite lucrul cu thread-uri. Odată extinsă interfața *Runnable* trebuie implementată metoda aferentă ***run()****.* În metoda *run()* descrisă mai jos cât timp mai există clienți de servit scad pentru fiecare client timpul de servire aferent pe măsură ce este procesat. Totodată decrementez și timpul de așteptare aferent fiecărei cozi. Dacă timpul de servire al unui client ajunge la 0, atunci îl scod din coadă, marcând că a fost servit.

*public void run() {*

*while (true) {*

*if (!tasks.isEmpty()) {*

*try {*

*int timp\_servire;*

*Thread.sleep(1000);*

*timp\_servire = tasks.peek().getTimp\_servire(); //extrag timpul de servire al clientului din*

*//capul cozii*

*timpAsteptare.decrementAndGet(); //decrementez timpul de așteptare*

*//System.out.println(c);*

*timp\_servire—; //scad timpul de servire*

*tasks.peek().setTimp\_servire(timp\_servire); //setez timpul de servire nou decrementat*

*//System.out.println(tasks);*

*if (timp\_servire == 0) { //dacă timpul de servire este 0*

*tasks.take(); //extrag clientul din coadă*

*}*

*} catch (InterruptedException ex) {*

*ex.printStackTrace();*

*}*

*}*

*}*

*}*

1. **Pachetul Business**
2. **Clasele ConcreteStrategyTime și ConcreteStrategyQueue**

Aceste clase implementează Interfața *Strategy* și descriu metoda *addTask()* prin care se adaugă

clienți la coadă. În prima clasă se aduagă clienții la cozi în funcție de timpul de așteptare minim al cozii, iar în a doua clasă se aduagă în funcție de numărul minim de clienți existenți în coadă. În cadrul acestor metode inainte să se adauge clientul în coadă se calculează și timpul în care a fost procesat în cadrul cozii.

1. **Interfața Strategy și clasa Selection**

Interfața Strategy conține metoda addTask() prin care se adaugă clienții la cozile corespunzătoare, iar

clasa Selection este de tip enum și descrie cele două politici de distribuire la cozi.

1. **Pachetul Logica**
2. **Clasa Scheduler**

În cadrul acestei clase se realizează inițializarea cozilor și adăugarea fiecărei cozi la o listă de tip List.

Totodată în cadrul constructorului se pornește un thread. Această clasă conține și o metodă pentru oprirea thread-urilor la finalul execuției. În același timp conține o metodă în care se inițializează tipul de strategie ales, numită *changeStrategy(Selection selectionPolicy),* având ca parametru politica de selecție selectată.

*public void changeStrategy(Selection selectionPolicy)*

*{*

*if(selectionPolicy == Selection.SHORTEST\_TIME)*

*{*

*strategie = new ConcreteStrategyTime();*

*}*

*else*

*{*

*if(selectionPolicy == Selection.SHORTEST\_QUEUE)*

*{*

*strategie = new ConcreteStrategyQueue();*

*}*

*}*

*}*

1. **Clasa Sortare**

Această clasă am utilizat-o pentru a sorta lista de task-uri în ordinea crescătoare a timpului de sosire.

Clasa implementează interfața Comparator cu metoda aferentă acesteia.

1. **Clasa SimulationManager**

Această clasă se ocupă cu menagerierea cozilor, distribuirea clienților la cozi și calculul rezultatelor

simulării. Totodată în cadrul acestei clase se realizează și generatorul de clienți random în funcție de timpul de sosire și timpul de servire. Totodată am inițializat și un fișier text pentru a afișa rezultatele din caseta text, pentru a le vizualiza mai bine. Clasa extinde interfața *Runnable* deci v-a implementa metoda *run().* Implementarea metodei este următoarea:

*public void run() {*

*try {*

*double avgTime = 0, avgServTime = 0;*

*int currentTime = 0, nr = 0, peakSecond = 0, peakTime = 0; // tSimulation*

*List<Task> toRemove = new ArrayList<>();*

*String rezultat, rezAvgTime, rezAvgServTime, rezPeakSecond;*

*while (currentTime < timeLimit) {*

*for (Task task : generatedTasks) {*

*if (currentTime == task.getTimp\_sosire()) {*

*scheduler.dispatchTask(task); //trimit task-ul la coada*

*toRemove.add(task); //adaug task-ul intr-o alta lista ca sa il elimin la final*

*avgTime += task.getWaitingTime();*

*avgServTime += task.getTimp\_servire();*

*nr++;*

*if(peakTime< scheduler.maxTime())*

*{*

*peakTime = scheduler.maxTime();*

*peakSecond = currentTime;*

*}*

*}*

*}*

*for (Task objRemove : toRemove) {*

*generatedTasks.remove(objRemove); //elimin task-urile din coada cu clientii in asteptare*

*}*

*rezultat = rezultat(currentTime);*

*textArea.setText(rezultat); myWriter.write(rezultat);*

*currentTime++;*

*if(generatedTasks.isEmpty() && scheduler.simulare())*

*{*

*break;*

*}*

*try {*

*Thread.sleep(1000);*

*} catch (InterruptedException e) {*

*e.printStackTrace();*

*}*

*}*

*scheduler.stopServer();*

*rezAvgTime = "Average waiting time: " + avgTime / nr + "\n";*

*rezAvgServTime= "Average service time: " + avgServTime / nr + "\n";*

*rezPeakSecond = "Peak second: " + peakSecond + "\n";*

*textAreaSim.append(rezAvgTime); textAreaSim.append(rezAvgServTime); textAreaSim.append(rezPeakSecond);*

*myWriter.write(rezAvgTime); myWriter.write(rezAvgServTime);myWriter.write(rezPeakSecond);myWriter.close();*

*} catch (IOException e) { System.out.println("A aparut o eroare!"); }*

*}*

1. **Pachetul GUI**
2. **Clasa SimulationFrame**

În cadrul acestei clase am implementat interfața cu utilizatorul a aplicației. Aceasta este formata din

trei zone:

1. *Introducerea datelor*

În această zona ultilizatorul poate să introducă datele aferente: numărul de clienți, numărul de cozi,

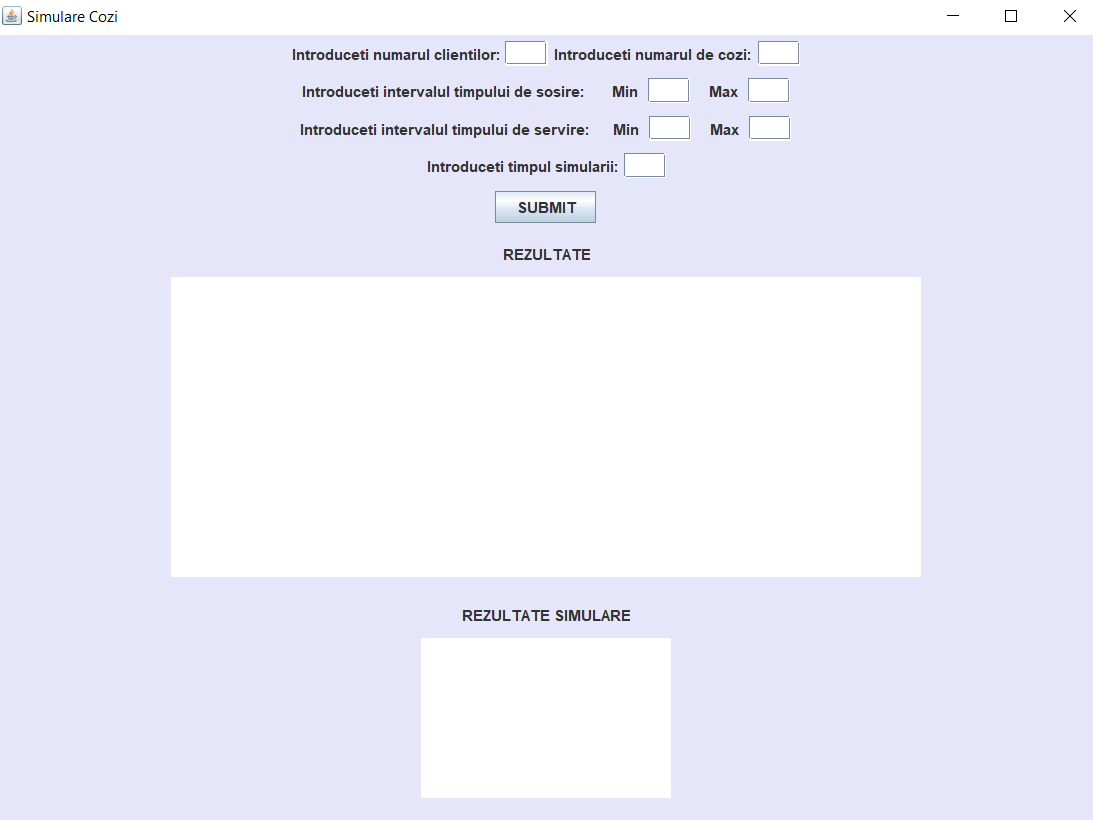
Intervalul timpului de sosire și intervalul timpului de servire.

1. *Rezultate*

Această zonă este destinată afișării în timp real a evoluției cozii. Log-ul de evenimente se afișează prin intermediul unei casete de text, fiecare eveniment afișându-se pe rând la un moment de timp.

1. *Rezultate Simulare*

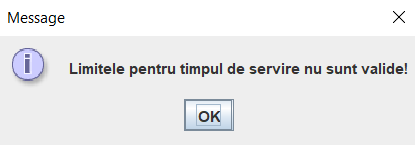
În această zonă se afișează rezultate corespunzătoare simulării: average waiting time, average service time și peak second.

****

*Figura 4.1 Interfața cu utilizatorul*

În cazul în care utilizatorul introduce în cadrul intervalelor de sosire sau plecare un număr mai

mare în JTextField-ul corespunzător minimului decât celui corespunzător maximului va apărea un mesaj de eroarea cu un text corespunzator.

O imagine care conține text

Descriere generată automat

*Figura 4.2 Mesajele de eroare afișate*

1. **Clasa ViewController**

Această clasă implementează interfața *ActionListener*, deci trebuie să implementeze metoda

*actionPerformed().* În această metodă am descris logica care se întâmplă la apăsarea butonului submit de către utilizator. În momentul apăsării butonului de Submit se preiau datele intregi din interfața grafică și se crează un nou obiect de tipul *SimulationManager,* pentru apelarea metodelor de afișare în zona de text. Totodată în această metodă se pornește și firul de execuție asociat aplicației.

*public void actionPerformed(ActionEvent e) {*

*Object src = e.getSource();*

*if (src == frame.getSubmit()) {*

*try {*

*int numberOfClients = parseInt(frame.getTextNrClienti().getText());*

*int numberOfServers = parseInt(frame.getTextNrCozi().getText());*

*int maxTimpServire = parseInt(frame.getTextMaxServire().getText());*

*int minTimpServire = parseInt(frame.getTextMinServire().getText());*

*int maxTimpSosire = parseInt(frame.getTextMaxSosire().getText());*

*int minTimpSosire = parseInt(frame.getTextMinSosire().getText());*

*int timeLimit = parseInt(frame.getTextTimpSimulare().getText());*

*validate(minTimpServire, maxTimpServire, minTimpSosire, maxTimpSosire);*

*SimulationManager sim = null;*

*try {*

*sim = new SimulationManager(numberOfClients, numberOfServers, minTimpSosire, maxTimpSosire, minTimpServire, maxTimpServire, timeLimit);*

*} catch (IOException ioException) {*

*ioException.printStackTrace();*

*}*

*sim.setTextArea(frame.getTextafisare());*

*Thread thread = new Thread(sim);*

*frame.getTextafisare().setText("");*

*sim.setTextArea(frame.getTextafisare());*

*frame.getTextafisareresult().setText("");*

*sim.setTextAreaSim(frame.getTextafisareresult());*

*thread.start();*

*} catch (ExceptionValidate exceptionValidate) {*

*frame.showError(exceptionValidate.getMessage());*

*}*

*}*

*}*

1. **Rezultate**

Pentru testarea aplicației am folosit următoarele date de intrare.

|  |  |  |
| --- | --- | --- |
| Test1 | Test2 | Test3 |
| N = 4  Q = 2  𝑡𝑠𝑖𝑚𝑢𝑙𝑎𝑡𝑖𝑜𝑛𝑀𝐴𝑋= 60 seconds  [𝑡𝑎𝑟𝑟𝑖𝑣𝑎𝑙𝑀𝐼𝑁, 𝑡𝑎𝑟𝑟𝑖𝑣𝑎𝑙𝑀𝐴𝑋] = [2, 30]  [𝑡𝑠𝑒𝑟𝑣𝑖𝑐𝑒𝑀𝐼𝑁,𝑡𝑠𝑒𝑟𝑣𝑖𝑐𝑒𝑀𝐴𝑋]= [2, 4] | N = 50  Q = 5  𝑡𝑠𝑖𝑚𝑢𝑙𝑎𝑡𝑖𝑜𝑛𝑀𝐴𝑋= 60 seconds  [𝑡𝑎𝑟𝑟𝑖𝑣𝑎𝑙𝑀𝐼𝑁, 𝑡𝑎𝑟𝑟𝑖𝑣𝑎𝑙𝑀𝐴𝑋] = [2, 40]  [𝑡𝑠𝑒𝑟𝑣𝑖𝑐𝑒𝑀𝐼𝑁,𝑡𝑠𝑒𝑟𝑣𝑖𝑐𝑒𝑀𝐴𝑋]=[1, 7] | N = 1000  Q = 20  𝑡𝑠𝑖𝑚𝑢𝑙𝑎𝑡𝑖𝑜𝑛𝑀𝐴𝑋= 200 seconds  [𝑡𝑎𝑟𝑟𝑖𝑣𝑎𝑙𝑀𝐼𝑁, 𝑡𝑎𝑟𝑟𝑖𝑣𝑎𝑙𝑀𝐴𝑋] = [10, 100]  [𝑡𝑠𝑒𝑟𝑣𝑖𝑐𝑒𝑀𝐼𝑁,𝑡𝑠𝑒𝑟𝑣𝑖𝑐𝑒𝑀𝐴𝑋]=[3, 9] |

Generarea log-ului de evenimente pentru primele doua teste este prezentat mai jos. Pentru cel de-al treilea test nu am mai introdus log-ul de evenimente având în vedere dimensiunea mare a acestuia.

**TEST 1:**

Time: 0

Waiting clients: (1, 7, 2) (2, 10, 2) (3, 25, 2) (4, 25, 2)

Queue 1: closed

Queue 2: closed

Time: 1

Waiting clients: (1, 7, 2) (2, 10, 2) (3, 25, 2) (4, 25, 2)

Queue 1: closed

Queue 2: closed

………………………………………..

Time: 7

Waiting clients: (2, 10, 2) (3, 25, 2) (4, 25, 2)

Queue 1: [(1, 7, 2) ]

Queue 2: closed

Time: 8

Waiting clients: (2, 10, 2) (3, 25, 2) (4, 25, 2)

Queue 1: [(1, 7, 1) ]

Queue 2: closed

Time: 9

Waiting clients: (2, 10, 2) (3, 25, 2) (4, 25, 2)

Queue 1: closed

Queue 2: closed

Time: 10

Waiting clients: (3, 25, 2) (4, 25, 2)

Queue 1: [(2, 10, 2) ]

Queue 2: closed

Time: 11

Waiting clients: (3, 25, 2) (4, 25, 2)

Queue 1: [(2, 10, 1) ]

Queue 2: closed

………………………………………….

Time: 25

Waiting clients:

Queue 1: [(3, 25, 2) ]

Queue 2: [(4, 25, 2) ]

Time: 26

Waiting clients:

Queue 1: [(3, 25, 1) ]

Queue 2: [(4, 25, 1) ]

Time: 27

Waiting clients:

Queue 1: closed

Queue 2: closed

Average waiting time: 0.0

Average service time: 2.0

Peak second: 25

**TEST 2:**

Time: 0

Waiting clients: (18, 2, 2) (29, 2, 4) (30, 3, 5) (35, 3, 1) (20, 4, 3) (31, 4, 4) (50, 4, 5) (6, 5, 5) (23, 5, 1) (42, 5, 5) (8, 6, 6) (26, 8, 1) (44, 8, 5) (49, 10, 1) (3, 12, 3) (24, 13, 6) (46, 13, 2) (13, 14, 3) (4, 15, 5) (27, 15, 5) (39, 18, 4) (36, 19, 2) (45, 19, 1) (37, 20, 4) (47, 20, 6) (40, 21, 2) (48, 21, 3) (7, 24, 4) (1, 25, 5) (17, 25, 6) (19, 25, 1) (14, 26, 2) (34, 26, 4) (5, 27, 1) (25, 27, 5) (32, 27, 4) (15, 28, 2) (16, 29, 1) (22, 30, 1) (10, 31, 1) (33, 31, 3) (2, 32, 4) (38, 33, 5) (43, 33, 5) (12, 34, 2) (21, 34, 6) (41, 37, 2) (9, 38, 5) (28, 38, 6) (11, 39, 2)

Queue 1: closed

Queue 2: closed

Queue 3: closed

Queue 4: closed

Queue 5: closed

Time: 1

Waiting clients: (18, 2, 2) (29, 2, 4) (30, 3, 5) (35, 3, 1) (20, 4, 3) (31, 4, 4) (50, 4, 5) (6, 5, 5) (23, 5, 1) (42, 5, 5) (8, 6, 6) (26, 8, 1) (44, 8, 5) (49, 10, 1) (3, 12, 3) (24, 13, 6) (46, 13, 2) (13, 14, 3) (4, 15, 5) (27, 15, 5) (39, 18, 4) (36, 19, 2) (45, 19, 1) (37, 20, 4) (47, 20, 6) (40, 21, 2) (48, 21, 3) (7, 24, 4) (1, 25, 5) (17, 25, 6) (19, 25, 1) (14, 26, 2) (34, 26, 4) (5, 27, 1) (25, 27, 5) (32, 27, 4) (15, 28, 2) (16, 29, 1) (22, 30, 1) (10, 31, 1) (33, 31, 3) (2, 32, 4) (38, 33, 5) (43, 33, 5) (12, 34, 2) (21, 34, 6) (41, 37, 2) (9, 38, 5) (28, 38, 6) (11, 39, 2)

Queue 1: closed

Queue 2: closed

Queue 3: closed

Queue 4: closed

Queue 5: closed

Time: 2

Waiting clients: (30, 3, 5) (35, 3, 1) (20, 4, 3) (31, 4, 4) (50, 4, 5) (6, 5, 5) (23, 5, 1) (42, 5, 5) (8, 6, 6) (26, 8, 1) (44, 8, 5) (49, 10, 1) (3, 12, 3) (24, 13, 6) (46, 13, 2) (13, 14, 3) (4, 15, 5) (27, 15, 5) (39, 18, 4) (36, 19, 2) (45, 19, 1) (37, 20, 4) (47, 20, 6) (40, 21, 2) (48, 21, 3) (7, 24, 4) (1, 25, 5) (17, 25, 6) (19, 25, 1) (14, 26, 2) (34, 26, 4) (5, 27, 1) (25, 27, 5) (32, 27, 4) (15, 28, 2) (16, 29, 1) (22, 30, 1) (10, 31, 1) (33, 31, 3) (2, 32, 4) (38, 33, 5) (43, 33, 5) (12, 34, 2) (21, 34, 6) (41, 37, 2) (9, 38, 5) (28, 38, 6) (11, 39, 2)

Queue 1: [(18, 2, 2) ]

Queue 2: [(29, 2, 4) ]

Queue 3: closed

Queue 4: closed

Queue 5: closed

Time: 3

Waiting clients: (20, 4, 3) (31, 4, 4) (50, 4, 5) (6, 5, 5) (23, 5, 1) (42, 5, 5) (8, 6, 6) (26, 8, 1) (44, 8, 5) (49, 10, 1) (3, 12, 3) (24, 13, 6) (46, 13, 2) (13, 14, 3) (4, 15, 5) (27, 15, 5) (39, 18, 4) (36, 19, 2) (45, 19, 1) (37, 20, 4) (47, 20, 6) (40, 21, 2) (48, 21, 3) (7, 24, 4) (1, 25, 5) (17, 25, 6) (19, 25, 1) (14, 26, 2) (34, 26, 4) (5, 27, 1) (25, 27, 5) (32, 27, 4) (15, 28, 2) (16, 29, 1) (22, 30, 1) (10, 31, 1) (33, 31, 3) (2, 32, 4) (38, 33, 5) (43, 33, 5) (12, 34, 2) (21, 34, 6) (41, 37, 2) (9, 38, 5) (28, 38, 6) (11, 39, 2)

Queue 1: [(18, 2, 1) ]

Queue 2: [(29, 2, 3) ]

Queue 3: [(30, 3, 5) ]

Queue 4: [(35, 3, 1) ]

………………………………………..

Time: 44

Waiting clients:

Queue 1: closed

Queue 2: closed

Queue 3: closed

Queue 4: closed

Queue 5: closed

Average waiting time: 0.52

Average service time: 3.42

Peak second: 5

1. **Concluzii**

În cocluzie acest proiect mi-a aprofundat cunoștințele despre limbajul Java. Am învățat cum să lucrez

cu fire de lucru, deoarece este prima dată când le folosesc. Totodată am învățat cum să creez fișiere și cum să afișez într-un fișier.

Ca dezvoltări ulterioare aș propune crearea unei interfețe mult mai ,,prietenoase’’ și perfecționarea codului, deoarece în lucrul cu fire de lucru pot să apară unele greșeli din cauza sincronizării.

1. **Bibliografie**
2. <https://app.diagrams.net/>
3. <https://www.w3schools.com/java/java_files_create.asp>
4. Stack Overflow
5. Youtube
6. Assignment\_2\_Suport\_Presentation