Queue Simulator

Documentatie

Nume:Ungurianu Daniel

Grupa 30228 An 2 semestrul al doilea

Disciplina: Tehnici de programare

Profesor indrumator: Cristina Bianca Pop

Cuprins

[1.Obiective 3](file:///C:\Users\danie\AppData\Local\Temp\Rar$DIa14352.33322\Documentatie_QueueSimulator_VATAFU_30228.docx#_Toc68768151)

[1.1 Obiectiv principal 3](file:///C:\Users\danie\AppData\Local\Temp\Rar$DIa14352.33322\Documentatie_QueueSimulator_VATAFU_30228.docx#_Toc68768152)

[1.2 Obiective secundare 3](file:///C:\Users\danie\AppData\Local\Temp\Rar$DIa14352.33322\Documentatie_QueueSimulator_VATAFU_30228.docx#_Toc68768153)

[2.Analiza problemei,modelare,scenarii,cazuri de utilizare 3](file:///C:\Users\danie\AppData\Local\Temp\Rar$DIa14352.33322\Documentatie_QueueSimulator_VATAFU_30228.docx#_Toc68768154)

[2.1 Analiza 3](file:///C:\Users\danie\AppData\Local\Temp\Rar$DIa14352.33322\Documentatie_QueueSimulator_VATAFU_30228.docx#_Toc68768155)

[2.2 Cerințe funcționale 4](file:///C:\Users\danie\AppData\Local\Temp\Rar$DIa14352.33322\Documentatie_QueueSimulator_VATAFU_30228.docx#_Toc68768156)

[2.3 Cerințe non-funcționale 4](file:///C:\Users\danie\AppData\Local\Temp\Rar$DIa14352.33322\Documentatie_QueueSimulator_VATAFU_30228.docx#_Toc68768157)

[2.4 Diagrama Use-Case 4](file:///C:\Users\danie\AppData\Local\Temp\Rar$DIa14352.33322\Documentatie_QueueSimulator_VATAFU_30228.docx#_Toc68768158)

[2.5 Scenarii de funcționare 5](file:///C:\Users\danie\AppData\Local\Temp\Rar$DIa14352.33322\Documentatie_QueueSimulator_VATAFU_30228.docx#_Toc68768159)

[3.Proiectare 5](file:///C:\Users\danie\AppData\Local\Temp\Rar$DIa14352.33322\Documentatie_QueueSimulator_VATAFU_30228.docx#_Toc68768160)

[3.1 Decizii de proiectare 5](file:///C:\Users\danie\AppData\Local\Temp\Rar$DIa14352.33322\Documentatie_QueueSimulator_VATAFU_30228.docx#_Toc68768161)

[3.2 Diagrame de pachete și clase 6](file:///C:\Users\danie\AppData\Local\Temp\Rar$DIa14352.33322\Documentatie_QueueSimulator_VATAFU_30228.docx#_Toc68768162)

[3.2.1. Diagrama de pachete 6](file:///C:\Users\danie\AppData\Local\Temp\Rar$DIa14352.33322\Documentatie_QueueSimulator_VATAFU_30228.docx#_Toc68768163)

[3.2.2.Diagrama de clase 7](file:///C:\Users\danie\AppData\Local\Temp\Rar$DIa14352.33322\Documentatie_QueueSimulator_VATAFU_30228.docx#_Toc68768164)

[7](file:///C:\Users\danie\AppData\Local\Temp\Rar$DIa14352.33322\Documentatie_QueueSimulator_VATAFU_30228.docx#_Toc68768165)

[3.3 Structuri de date și algoritmi 7](file:///C:\Users\danie\AppData\Local\Temp\Rar$DIa14352.33322\Documentatie_QueueSimulator_VATAFU_30228.docx#_Toc68768166)

[3.4.Interfețe și pachete 7](file:///C:\Users\danie\AppData\Local\Temp\Rar$DIa14352.33322\Documentatie_QueueSimulator_VATAFU_30228.docx#_Toc68768167)

[3.5 Interfața utilizator 7](file:///C:\Users\danie\AppData\Local\Temp\Rar$DIa14352.33322\Documentatie_QueueSimulator_VATAFU_30228.docx#_Toc68768168)

[4.Implementarea 8](file:///C:\Users\danie\AppData\Local\Temp\Rar$DIa14352.33322\Documentatie_QueueSimulator_VATAFU_30228.docx#_Toc68768169)

[4.1 Clasa Task 8](file:///C:\Users\danie\AppData\Local\Temp\Rar$DIa14352.33322\Documentatie_QueueSimulator_VATAFU_30228.docx#_Toc68768170)

[4.2 Clasa Server 8](file:///C:\Users\danie\AppData\Local\Temp\Rar$DIa14352.33322\Documentatie_QueueSimulator_VATAFU_30228.docx#_Toc68768171)

[4.3 Strategy 9](file:///C:\Users\danie\AppData\Local\Temp\Rar$DIa14352.33322\Documentatie_QueueSimulator_VATAFU_30228.docx#_Toc68768172)

[4.3.1 ConcreteStrategyQueue 9](file:///C:\Users\danie\AppData\Local\Temp\Rar$DIa14352.33322\Documentatie_QueueSimulator_VATAFU_30228.docx#_Toc68768173)

[4.3.2 ConcreteStrategyTime 9](file:///C:\Users\danie\AppData\Local\Temp\Rar$DIa14352.33322\Documentatie_QueueSimulator_VATAFU_30228.docx#_Toc68768174)

[4.3.3 SelectionPolicy 10](file:///C:\Users\danie\AppData\Local\Temp\Rar$DIa14352.33322\Documentatie_QueueSimulator_VATAFU_30228.docx#_Toc68768175)

[4.4 Clasa Scheduler 10](file:///C:\Users\danie\AppData\Local\Temp\Rar$DIa14352.33322\Documentatie_QueueSimulator_VATAFU_30228.docx#_Toc68768176)

[4.5 Clasa SimulationManager 10](file:///C:\Users\danie\AppData\Local\Temp\Rar$DIa14352.33322\Documentatie_QueueSimulator_VATAFU_30228.docx#_Toc68768177)

[4.5.1 generateNRandomTasks() 10](file:///C:\Users\danie\AppData\Local\Temp\Rar$DIa14352.33322\Documentatie_QueueSimulator_VATAFU_30228.docx#_Toc68768178)

[4.5.2 waitingTask() 11](file:///C:\Users\danie\AppData\Local\Temp\Rar$DIa14352.33322\Documentatie_QueueSimulator_VATAFU_30228.docx#_Toc68768179)

[4.5.3 run() 11](file:///C:\Users\danie\AppData\Local\Temp\Rar$DIa14352.33322\Documentatie_QueueSimulator_VATAFU_30228.docx#_Toc68768180)

[4.6 Clasa WriteToFile 12](file:///C:\Users\danie\AppData\Local\Temp\Rar$DIa14352.33322\Documentatie_QueueSimulator_VATAFU_30228.docx#_Toc68768181)

[4.7 View 12](file:///C:\Users\danie\AppData\Local\Temp\Rar$DIa14352.33322\Documentatie_QueueSimulator_VATAFU_30228.docx#_Toc68768182)

[4.8 Clasa Controller 13](file:///C:\Users\danie\AppData\Local\Temp\Rar$DIa14352.33322\Documentatie_QueueSimulator_VATAFU_30228.docx#_Toc68768183)

[5.Testarea 14](file:///C:\Users\danie\AppData\Local\Temp\Rar$DIa14352.33322\Documentatie_QueueSimulator_VATAFU_30228.docx#_Toc68768184)

[6.Concluzii 19](file:///C:\Users\danie\AppData\Local\Temp\Rar$DIa14352.33322\Documentatie_QueueSimulator_VATAFU_30228.docx#_Toc68768185)

[7.Surse 19](file:///C:\Users\danie\AppData\Local\Temp\Rar$DIa14352.33322\Documentatie_QueueSimulator_VATAFU_30228.docx#_Toc68768186)

**1.Obiective**

**1.1Obiectiv principal**

Principalul obiectiv al acestui proiect este proiectarea și implementarea unei simulări care să analizeze timpul de așteptarea al clienților la cozi.

**1.2 Obiective secundare**

\*Analiza problemei si modelarea:Cerinte, cazuri de utilizare, diagrame Use-Case

\*Proiectare: Decizii de proiectare, diagrame UML, Structuri de date, clase, interfete, pachete, algoritmi,GUI

\*Implementare:Descrierea claselor si metodelor

\*Rezultate:Seturi de valori

**2.Analiza problemei, modelare, scenarii, cazuri de utilizare**

**2.1 Analiza**

Aplicatia aceasta ar trebui sa simuleze o multime de cozi cu diversi clienti ce isi asteapta randul, fiecare coadă servind clienți simultan, la fel ca în munca reală, cum ar fi un supermarket, o bancă sau un server. Se analizează câți clienți pot fi deserviți într-un anumit interval de simulare, prin introducerea parametrilor într-o interfață grafică

**2.2Cerinte functionale**

Utilizatorul trebuie sa aiba posibilitatea de a:

\*Introduce detaliile simularii

-Nr de clienti

-Nr de cozi

-Timpul maxim de simulare

-Intervalul de sosire al clientilor

-Intervalul de servire al clientilor

-strategia de asezare la coada

\*Observa rezultatele in cadrul interfetei grafice

\*Evalua procesul total in cadrul unui fisier text

\*Analiza statistici

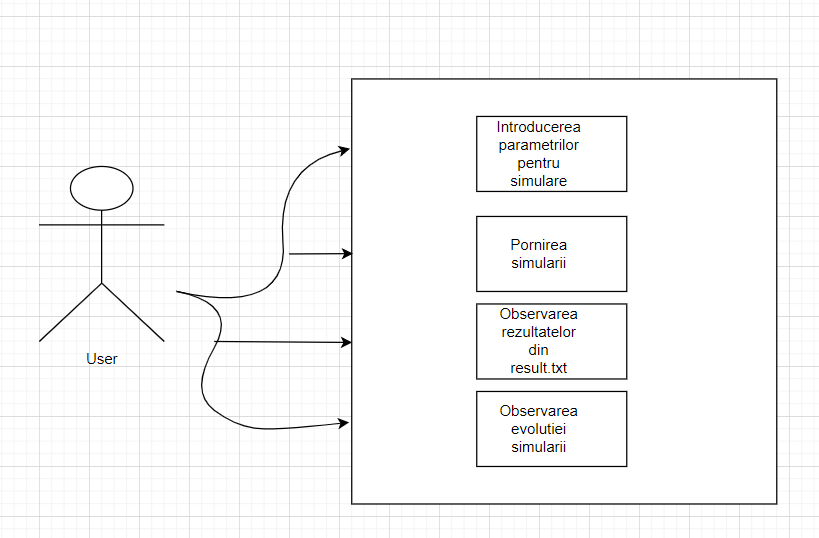
**2.3 Cerinte non-functionale**

Se doreste proiectarea unei interfete intuitive si usor de folosit pentru utilizator

-Nu se poate reveni la primul panel decat dupa terminarea simularii curente

-Datele invalide de simulare nu sunt tolerate

**2.4 Diagrama Use-Case**



Aceasta diagrama use-case evidentiaza principalele facilitati puse la dispozitie user-ului.

**2.5 Scenarii de functionare**

Mai întai utilizatorul va introduce condițiile pe care dorește ca simularea să le îndeplinească.

Există 2 scenarii posibile:

1. Scenariul de succes

Userul introduce date valide de simulare, iar acesta va putea observa evoluția acesteia în interfața grafică. Apoi poate analiza procesul și statisticile din fișierul text. După terminarea simulării, se poate reveni la meniul inițial și să se efectueze altă simulare.

1. Scenariul eșuat – date introduse greșit

Nu se poate deschide fereastra de simulare. Se așteaptă date corecte.

**3.Proiectare**

**3.1 Decizii de proiectare**

În urma analizei am decis să urmez structura de proiect propusă.

Cozile au devenit servere, fiecare cu thread-ul propriu, iar clienții task-uri. Pentru utilizarea thread.urilor am optat pentru folosirea interfeței Runnable.

Pentru o mai bună logică și organizare s-a utilizat Scheduler-Strategy Pattern. Scheduler trimite clienți către cozi în conformitate cu cele stabilite de strategie.

Aceste clase, alături de tasks și server au fost introduse în pachetul MODEL.

Clasa SimulationManager conține thread-ul principal . Aici sunt generați clienții și sunt calculate statisticile.

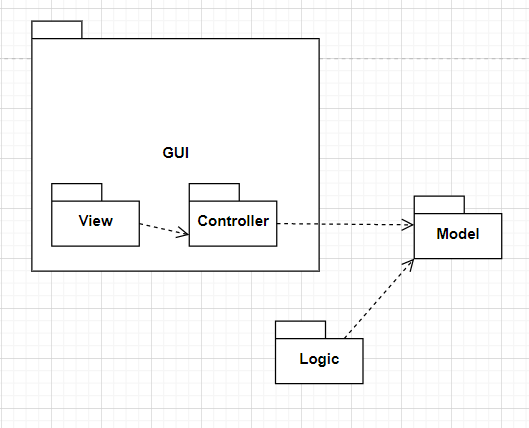
Interfața grafică este dezvoltată în pachetul GUI.

Pachetul View conține clasa cu același nume. Aceasta este responsabilă pentru logica de vizualizare a aplicației. Ea conține titlul proiectului, câmpurile de introducere ale polinoamelor,butonul de efectuare a calculului și câmpul pentru afișarea rezultatului.

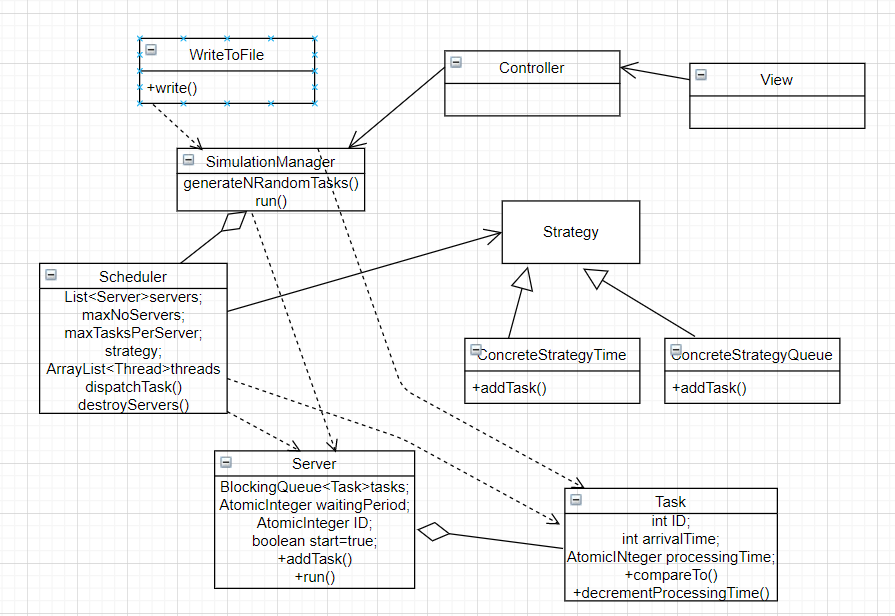
Prin clasa Controller ( pachetul Controller) se face legătura dintre View și Model. Acesta extinde ActionListener și oferă funcționalitate butoanelor din cadrul interfeței.

**3.2 Diagrame de pachete si clase**

**3.2.1 Diagrama de pachete**



**3.2.2 Diagrama de clase**



**3.3 Structuri de date si algoritmi**

Întrucât am implementat proiectul folosind mai multe fire de execuție, am ales structuri de date potrivite pentru această abordare.

Am lucrat cu tipuri de date atomice : AtomicInteger, precum și cu colecția BlockingQueue.

**3.4 Interfete si pachete**

Principalele interfețe folosite sunt Comparable,ActionListener și Runnable.

Am avut nevoie de java.io.FileWriter pentru lucrul cu fișiere.

**3.5 Interfata utilizator**

Pentru interfata grafica am decis sa incerc ceva nou, anume sa lucrez cu doua panel-uri, design-ul este insa unul simplu.

Utilizatorului i se pun la dispoziție o serie de câmpuri prin care se vor seta condițiile simulării.

Se apasă butonul de finalizare , iar apoi se poate vizualiza evoluția în timp real a cozilor . Informația este afișată în cadrul unei JTextArea.

**4.Implementarea**

**4.1 Clasa Task**

Această clasă simbolizează clientul.

Se memorează id-ul, momentul sosirii și timpul de procesare.

Clasa Task implementeaza interfata Comparable, pentru a se realiza sortarea în funcție de timpul sosirii. În acest fel se pot adăuga clienții eficient în funcție de timpul curent.

**4.2 Clasa Server**

Este imaginea cozii. Se asigură mai multe fire de execuție prin implementarea interfeței Runnable.

Clienții ce așteaptă sunt salvați într-o coadă, structura de date sincronizata – ArrayBlockingQueue.

Esențială este metoda run() ce descriere comportamentul fiecărei cozi în timpul execuției.

public void run(){  
 while(start){  
 if(!tasks.isEmpty()){  
 AtomicInteger processingTime=tasks.peek().getProcessingTime();  
 try{  
 Thread.*sleep*(1000);  
 waitingPeriod.addAndGet(-1);  
 tasks.peek().decrementProcessingTime();  
 if(tasks.peek().getProcessingTime().get()==0){  
 tasks.take();  
 }  
 }catch (Throwable e){  
 e.printStackTrace();  
 }  
 }  
 }  
}

Cât timp coada nu este goală, se servește primul client. Se așteaptă o secundă, se decrementează timpul de servire al clientului. Dacă acest timp ajunge 0, clientul a terminat . Deci, se va trece la următorul client.

Tot aici este implementată funcția toString, prin care se facilitează afișarea intuitivă a procesului simulării.

public String toString(){  
 if(!tasks.isEmpty()){  
 return "Server"+ ID+": "+tasks;  
 }  
 return "Server"+ID+": closed";  
}

**4.3 Strategy**

Am implementat doua strategii de pozitionare a clientilor in cozi.

**4.3.1 ConcreteStrategyTime**

@Override  
public void addTask(List<Server> servers, Task t){  
 //TO DO auto-generated method stub  
 int min=Integer.*MAX\_VALUE*;  
 int id=0;  
 for(Server s:servers){  
 if(s.getWaitingPeriod().get()<min){  
 min=s.getWaitingPeriod().get();  
 id=s.getID().get();  
 }  
 }  
 for(Server s:servers){  
 if(s.getID().get()==id){  
 s.addTask(t);  
 }  
 }  
}

Se alege coada cu cel mai mic timp de asteptare.

**4.3.2 ConcreteStrategyQueue**

@Override  
public void addTask(List<Server> servers, Task t){  
 //TO DO auto-generated method stub  
 int min=Integer.*MAX\_VALUE*;  
 int id=0;  
 for(Server s:servers){  
 if(s.getNumberTasks()<min){  
 id=s.getID().get();  
 }  
 }  
 for(Server s:servers){  
 if(s.getID().get()==id){  
 s.addTask(t);  
 }  
 }  
}

Se alege coada cu cel mai mic numar de clienti aflati in asteptare.

**4.3.3 SelectionPolicy**

Acest enum permite diferentierea mai usoara a celor doua strategii.

public enum SelectionPolicy {  
 *SHORTEST\_QUEUE*,*SHORTEST\_TIME*;  
  
}

**4.4 Clasa Scheduler**

Scheduler Trimite clienți către cozi în conformitate cu cele strategia stabilită.

* Metoda dispatchTask() apeleaza metoda addTask() din clasa strategiei alese.
* Metoda destroyServers() intrerupe firele de execuție . După servirea clienților , thread-urile aferente serverelor ar fi continuat să existe, thread-ul principal neputând să încheie execuția.

Am ales să realizez această întrerupere prin intermediul flag-ului start din clasa Server. Acesta marchează finalizarea thread-ului.

public void destroyServers(){  
 for(Server s:servers){  
 s.Start(false);  
 }  
}

}

* Un rol important îl are și metoda isEmpty(). Se verifică dacă toate cozile sunt goale sau nu. Această metodă permite finalizarea simulării dacă toți clienții au fost serviți.
* showServers() permite afișarea conținutului cozii .

**4.5Clasa SimulationManager**

Această clasă tratează simularea propriu-zisă.

Aici sunt reținute condițiile de desfășurare ale simulării și este implementată ca un thread propriu zis. Și aici am optat pentru varianta cu interfața Runnable.

Condițiile esențiale sunt:

* timpul maxim de simulare: timeLimit
* intervalul de servire : minProcessingTime - maxProcessingTime
* intervalul de sosire: minArrivalTime - maxArrivalTime
* numar de clienti – numberOfClients
* numar de cozi – numberOfServers

**4.5.1 generateNRandomTasks()**

public void generateNRandomTasks() {  
 Random r = new Random();  
 generatedTasks=new ArrayList<Task>(numberOfTasks);  
 int i;  
 for (i = 1; i <= numberOfTasks; i++) {  
 Task task=new Task(i,r.nextInt(maxArrivalTime-minArrivalTime)+minArrivalTime,new AtomicInteger(r.nextInt(maxProcessingTime-minProcessingTime)+minProcessingTime));  
 generatedTasks.add(task);  
 }  
 Collections.*sort*(generatedTasks);  
}

Se generează numberOfTasks clienți cu id-uri între 1 și numberOfTasks. Timpul de sosire și de servire sunt generate random în funcție de intervalele primite în interfață.

Apoi, clienții sunt sortați în funcție de timpul de sosire. Coincidența dintre timpul curent și timpul de sosire va genera asocierea clientului cu o coadă.

**4.5.2 waitingTasks()**

public String waitingTasks(Task task,int currentTime){  
 String result="";  
 int index=generatedTasks.indexOf(task);  
 int j;  
 for(j=index;j<generatedTasks.size();j++){  
 if(!(currentTime>= task.getArrivalTime() && index==generatedTasks.size()-1))  
 result+=generatedTasks.get(j)+" ";  
 }  
 return result;  
}

Prin această metodă se identifică clienții al căror timp de sosire nu a fost atins încă, altfel spus, nu așteaptă la o coadă încă.

**4.5.3 run()**

@Override  
public void run(){  
 StringBuilder result;  
 int currentTime=0;  
 int waitMax=0;  
 int processingTime=0;  
 int waitingTime=0;  
 int peak=0;  
  
 Iterator<Task> i=generatedTasks.iterator();  
 Task task=i.next();  
 String toWrite="";  
 while(timeLimit>currentTime) {  
 //daca se depaseste limita de timp, simularea se opreste  
 while (currentTime == task.getArrivalTime()) {  
 //daca se termina task urile , se opreste simularea  
 processingTime = processingTime + task.getProcessingTime().get();  
 scheduler.dispatchTask(task);  
 if (i.hasNext()) {  
 task = i.next();  
 } else  
 break;  
 }  
  
 waitingTime = waitingTime + scheduler.totalWaitingTime();  
 if (scheduler.totalWaitingTime() > waitMax) {  
 peak = currentTime;  
 waitMax = scheduler.totalWaitingTime();  
 }  
 String stringCurrentTime = "Time" + currentTime;  
 String stringWaitingTasks = "Waiting tasks: ";  
 stringWaitingTasks += waitingTasks(task, currentTime);  
 String stringServer = scheduler.showServers();  
 view.setOutput(stringCurrentTime + "\n" + stringWaitingTasks + "\n" + stringServer);  
 toWrite += stringCurrentTime + "\n" + stringWaitingTasks + "\n" + stringServer + "\n";  
 currentTime++;  
 try {  
 Thread.*sleep*(1000);  
 } catch (InterruptedException e) {  
 e.printStackTrace();  
 }  
 if (scheduler.isEmpty() && task.getProcessingTime().get() == 0) {  
 break;  
 }  
 }  
 scheduler.destroyServers();  
 toWrite+="Avg Processing Time: "+((float)processingTime)/numberOfServers/generatedTasks.indexOf(task)+"\n";  
 toWrite+="Avg Wait Time: "+((float)waitingTime)/numberOfServers/timeLimit+"\n";  
 toWrite+="Peak Time: "+peak+"\n";  
 WriteToFile.*write*("result.txt",toWrite);  
 view.EnableBack(true);  
}

Cat timp nu s-au scurs secundele maxime acceptate sau cat timp mai sunt clienti de servit, simularea va rula.

Clientii deja au fost sortati in functie de timpul de sosire. In momentul coincidentei cu timpul current, se vor adauga clienti in cozi.

Se formeaza String-uri cu informațiile tranzițiilor clienti-cozi, ce urmează să fie afișate atât în fișierul text cât și în interfața grafică.

**4.6 Clasa WriteToFile**

Această clasă se ocupă de scrierea într-un fișier text.

public static void write(String filePath,String toWrite){  
 try{  
 FileWriter writer=new FileWriter(filePath);  
 writer.write(toWrite);  
 writer.close();  
 System.*out*.println("OK");  
 }catch (IOException e){  
 System.*out*.println("Error");  
 e.printStackTrace();  
 }  
}

**4.7 Clasa View**

Această clasă este responsabilă pentru logica de vizualizare a aplicației. Ea conține titlul proiectului,câmpurile de introducere ale condițiilor simulării,butonul de finalizare și câmpuri pentru afișarea rezultatului.

JLabel[] InputLabels;  
JTextField[] InputTexts;  
JPanel[] InputPanels;  
JLabel title;  
JPanel first\_panel;  
JPanel second\_panel;  
JButton result,goBack;  
Controller controller = new Controller(this);  
JTextArea resultText;

După ce se completează aceste informații, se comută la JPanelul de simulare. Revenirea la meniul inițial se face prin butonul “<=”, ce se menține inactiv pana la finalizarea simularii.

Metoda ce permite interschimbarea este următoarea:

public void changePanels(){  
 if(first\_panel.isVisible()){  
 first\_panel.setVisible(false);  
 second\_panel.setVisible(true);  
 goBack.setVisible(true);  
 }else{  
 first\_panel.setVisible(true);  
 second\_panel.setVisible(false);  
 goBack.setVisible(false);  
 for (JTextField t: InputTexts  
 ) {  
 t.setText("");  
 }  
 }  
}

**4.8 Clasa Controller**

Prin clasa Controller ( pachetul Controller) se face legătura dintre View și Model. Acesta extinde ActionListener și oferă funcționalitate butoanelor din cadrul interfeței.

Butonul goBack permite revenirea la meniu.

Butonul result (Result) porneste simularea.

Tot aici este instantiata clasa SimulationManager.

@Override  
public void actionPerformed(ActionEvent e) {  
 JButton buttonPressed = (JButton) e.getSource();  
 if(buttonPressed == view.getResult()){  
 try{  
 view.changePanels();  
 numberOfClients = Integer.*parseInt*(view.getInputTexts(0).getText());  
 numberOfServers = Integer.*parseInt*(view.getInputTexts(1).getText());  
 timeLimit = Integer.*parseInt*(view.getInputTexts(2).getText());  
 minArrivalTime = Integer.*parseInt*(view.getInputTexts(3).getText());  
 maxArrivalTime = Integer.*parseInt*(view.getInputTexts(4).getText());  
 minProcessingTime = Integer.*parseInt*(view.getInputTexts(5).getText());  
 maxProcessingTime = Integer.*parseInt*(view.getInputTexts(6).getText());  
 view.EnableBack(false);  
 SimulationManager sim = new SimulationManager(timeLimit,maxProcessingTime,minProcessingTime,maxArrivalTime,minArrivalTime,numberOfServers,numberOfClients,view);  
 Thread t = new Thread(sim);  
 t.start();  
 }catch (Throwable throwable){  
 view.changePanels();  
 }  
  
 }else {  
 view.changePanels();  
 }  
}

**5. Testarea**

Am rulat aplicația pe cazurile din prezentare pe strategia ce vizeaza timpul.

Mai jos voi include primul set de teste, fiind mai scurt. Celelalte teste le voi încărca pe GitLab.

Time0

Waiting tasks: (2 ,3 ,2) (4 ,7 ,3) (3 ,11 ,3) (1 ,18 ,3)

Server0: closed

Server1: closed

Time1

Waiting tasks: (2 ,3 ,2) (4 ,7 ,3) (3 ,11 ,3) (1 ,18 ,3)

Server0: closed

Server1: closed

Time2

Waiting tasks: (2 ,3 ,2) (4 ,7 ,3) (3 ,11 ,3) (1 ,18 ,3)

Server0: closed

Server1: closed

Time3

Waiting tasks: (4 ,7 ,3) (3 ,11 ,3) (1 ,18 ,3)

Server0: [(2 ,3 ,2)]

Server1: closed

Time4

Waiting tasks: (4 ,7 ,3) (3 ,11 ,3) (1 ,18 ,3)

Server0: [(2 ,3 ,1)]

Server1: closed

Time5

Waiting tasks: (4 ,7 ,3) (3 ,11 ,3) (1 ,18 ,3)

Server0: closed

Server1: closed

Time6

Waiting tasks: (4 ,7 ,3) (3 ,11 ,3) (1 ,18 ,3)

Server0: closed

Server1: closed

Time7

Waiting tasks: (3 ,11 ,3) (1 ,18 ,3)

Server0: [(4 ,7 ,3)]

Server1: closed

Time8

Waiting tasks: (3 ,11 ,3) (1 ,18 ,3)

Server0: [(4 ,7 ,3)]

Server1: closed

Time9

Waiting tasks: (3 ,11 ,3) (1 ,18 ,3)

Server0: [(4 ,7 ,1)]

Server1: closed

Time10

Waiting tasks: (3 ,11 ,3) (1 ,18 ,3)

Server0: [(4 ,7 ,0)]

Server1: closed

Time11

Waiting tasks: (1 ,18 ,3)

Server0: [(3 ,11 ,3)]

Server1: closed

Time12

Waiting tasks: (1 ,18 ,3)

Server0: [(3 ,11 ,2)]

Server1: closed

Time13

Waiting tasks: (1 ,18 ,3)

Server0: [(3 ,11 ,2)]

Server1: closed

Time14

Waiting tasks: (1 ,18 ,3)

Server0: closed

Server1: closed

Time15

Waiting tasks: (1 ,18 ,3)

Server0: closed

Server1: closed

Time16

Waiting tasks: (1 ,18 ,3)

Server0: closed

Server1: closed

Time17

Waiting tasks: (1 ,18 ,3)

Server0: closed

Server1: closed

Time18

Waiting tasks:

Server0: [(1 ,18 ,3)]

Server1: closed

Time19

Waiting tasks:

Server0: [(1 ,18 ,3)]

Server1: closed

Time20

Waiting tasks:

Server0: [(1 ,18 ,1)]

Server1: closed

Avg Processing Time: 1.8333334

Avg Wait Time: 0.225

Peak Time: 7