**QUEUES MANAGEMENT APPLICATION USING THREADS AND SYNCHRONIZATION MECHANISMS**

Tema 2

DOCUMENTAȚIE

**Gavrilescu Andreea-Lavinia**

**Grupa 30228**

# CUPRINS

[1. **Obiectivul temei**](file:///D:\Facultate\UTCN\An%20II\Semestrul%202\Tehnici%20de%20programare\Laboratoare\PT2021-2022_Documentation_Template.doc#_Toc95297885) 1

[2. **Analiza problemei, modelare, scenarii, cazuri de utilizare**](file:///D:\Facultate\UTCN\An%20II\Semestrul%202\Tehnici%20de%20programare\Laboratoare\PT2021-2022_Documentation_Template.doc#_Toc95297886) 2

[3. **Proiectare**](file:///D:\Facultate\UTCN\An%20II\Semestrul%202\Tehnici%20de%20programare\Laboratoare\PT2021-2022_Documentation_Template.doc#_Toc95297887) 3

[4. **Implementare**](file:///D:\Facultate\UTCN\An%20II\Semestrul%202\Tehnici%20de%20programare\Laboratoare\PT2021-2022_Documentation_Template.doc#_Toc95297888) 9

[5. **Rezultate**](file:///D:\Facultate\UTCN\An%20II\Semestrul%202\Tehnici%20de%20programare\Laboratoare\PT2021-2022_Documentation_Template.doc#_Toc95297889) 22

[6. **Concluzii**](file:///D:\Facultate\UTCN\An%20II\Semestrul%202\Tehnici%20de%20programare\Laboratoare\PT2021-2022_Documentation_Template.doc#_Toc95297890) 23

[7. **Bibliografie**](file:///D:\Facultate\UTCN\An%20II\Semestrul%202\Tehnici%20de%20programare\Laboratoare\PT2021-2022_Documentation_Template.doc#_Toc95297891) 24

Obiectivul temei

**Obiective principale**

* Proiectarea și implementarea unei aplicații care are ca scop analizarea simulării unor clienți la coadă prin intermediul unei interfețe grafice care să permită introducerea datelor corespunzătoare și să afișeze schimbările produse în timp real, precum și timpul de așteptare și serivce mediu și ora de vârf.

**Obiective secundare**

* Analizarea problemei și identificarea cerințelor
* Proiectarea aplicației de simulare a cozilor
* Implementarea aplicației de simulare a cozilor
* Testarea aplicației de simulare a cozilor

Analiza problemei, modelare, scenarii, cazuri de utilizare

**Analiza**

QUEUES MANAGEMENT APPLICATION

**Cerințe funcționale**

* Aplicația de simulare ar trebui să permită utilizatorilor să introducă datele dorite
* Aplicația de simulare ar trebui să permită utilizatorilor să pornească simularea
* Aplicația de simulare ar trebui să afișeze modificările produse în cozi în timp real
* Aplicația de simulare ar trebui să afișeze timpul mediu de aștepatere și service, precum și ora de vârf

**Cerințe non-funcționale**

* Aplicația de simulare ar trebui să fie interactivă
* Aplicația de simulare ar trebui să fie ușor de folosit de către utilizator

**Cazuri de utilizare**

* Caz de utilizare:setarea datelor dorite pentru simulare

Actor principal: utilizatorul

Scenariu principal

1. Utilizatorul introduce valori pentru: numărul de clienți, numărul de cozi, intervalul de simulare, timpul minim și maxim de arrival, timpul minim și maxim de service
2. Utilizatorul apasă pe butonul de start pentru a porni simularea

Scenariu alternativ

1. Utilizatorul introduce date incorecte
2. Scenariul se întoarce la punctul 1

Proiectare

**Design general**

Queue

Simulation

**Date**

Task 6

Task 5

Task 4

Task 2

Task 1

Task 3

Client

Client

**Divizare în pachete**

Conține clasele care implementează interfața

Interfața grafică

Modele

Logica

Conține clasele care implementează operațiile

Conține clasele care modelează datele aplicației

Pentru afișarea rezulatelor potrivite confrom datelor introduse de utilizatori prin interfața grafică voi folosi un Model-View-Controller pattern care împarte aplicația în 3 categorii: procesare, input și output.

* Componentele Model – încapsulează datele și funcționalitatea
* Componentele View – afișează informația utilizatorului – obține datele afișate din model
* Controller – fiecare view are un controller asociat; acesta primește un input, de obicei un eveniment care denotă mișcarea mouse-ului, apăsarea mouse-ului sau a tastaturii; aceste evenimente sunt traduse ca apeluri care sunt trimise către model sau view.

Controller-ul modifică apeluri View-ului în funcție de modificările din Model

Datele din Model sunt folosite în View

Model

View-ul poate cere date din Model

View

Controller

Interacțiunea cu View-ul face apeluri către Controller

**Divizare în clase**

Client

Model

Queue

Simulation

Controller

Controller

View

View

O imagine care conține text

Descriere generată automat

Pachetul Models - conține 3 clase pentru a asigura lizibilitatea codului, dar și intuitivitatea, astfel:

* Clasa Client – conține 3 atribute numite sugestiv, ID și serviceTime, arrivalTime, precum și un constructor, metodele de get(), set() și toString().
* Clasa Queue – are ca atribute o listă de clienți, număul cozii, timpul de așteptare total la coadă, o variabilă open de tip boolean care indică daca coadă e deschisă sau nu, un constructor corespunzător , alături de metodele standard de get(), set(), precum și metode de adăugare a clienților în coadă și run().
* Clasa Simulation – are ca atribute datele preluate de pe interfață, o variabila de tipul AtomicInteger pentru timpul curent, una care va indica ora de vârf și numărul clienților corespunzători, iar pe lângă constructor, avem metode de generareClientiRandom(), de identificare coadaOptima(), stergereClientiGata(), deschidereCozi() și inchidereCozi(), nrClientiInCozi(), scrieInFisier() și run().

Pachetul View – conține clasa View care implementează interfața folosind Java Swing.

Pachetul Controller – conține clasa Controller care receptează acțiunile utilizatorului și returnează rezultatul potrivit prin interfața grafică.

Clasa Main – care are doua atribute, view-ul și controller-ul.

Implementare

**1. Clasa Client** – are ca atribute private, ID, arrivalTime și serviceTime, toate de tipul int.

Avem un constructor care primește ca parametrii atributele declarate.

Cât despre metode, pe lângă metodele de get() și set() necesare, avem metoda toString() care afișează clientul cu câmpurile sale.

public class Client *{* // Declararea campurilor specifice unui client  
 private int ID;  
 private int serviceTime;  
 private int arrivalTime;  
  
 // Constructorul clasei Client  
 public Client*(*int ID, int arrivalTime, int serviceTime*) {* this.ID = ID;  
 this.arrivalTime = arrivalTime;  
 this.serviceTime = serviceTime;  
 *}* // Getters si setters  
  
 public int getArrivalTime*() {* return arrivalTime;  
 *}* public int getServiceTime*() {* return serviceTime;  
 *}* public void setServiceTime*(*int serviceTime*) {* this.serviceTime = serviceTime;  
 *}* // Metoda de afisare a datelor corespunzatoare unui client  
 @Override  
 public String toString*() {* return "(" + ID + "," + arrivalTime + "," + serviceTime + ")";  
 *}  
}*

**2. Clasa Queue –** are ca atribute o listă de clienți, un număr, un timp de așteptare și o variabilă boolean care indică dacă o coadă este deschisă sau nu.

// Declararea campurilor necesare  
private List*<*Client*>* clients;  
  
private int queueNumber;  
private int waitingTime;  
private boolean open; // variabila care stabileste daca coada este deschisa sau nu  
  
// Constructor-ul clasei Queue care primeste ca parametru numarul cozii  
public Queue*(*int queueNumber*) {* this.queueNumber = queueNumber;  
 this.clients = new ArrayList*<>()*; // initializam lista de clienti  
 this.waitingTime = 0; // setam timpul de asteptare la coada cu 0  
*}*

Avem un constructor care primește ca parametru numărul cozii, metodele de get() și set() necesare.

// Constructor-ul clasei Queue care primeste ca parametru numarul cozii  
public Queue*(*int queueNumber*) {* this.queueNumber = queueNumber;  
 this.clients = new ArrayList*<>()*; // initializam lista de clienti  
 this.waitingTime = 0; // setam timpul de asteptare la coada cu 0  
*}*// Getters si setters  
  
public void setOpen*(*boolean open*) {* this.open = open;  
*}*public int getWaitingTime*() {* return waitingTime;  
*}*public int getQueueNumber*() {* return queueNumber;  
*}*public List*<*Client*>* getClients*() {* return clients;  
*}*

În continuare avem metoda de adăugare a unui client în coadă, precum și metoda de run() care ilustrează modul de preluare și servire al clienților.

// Metoda de adaugare a unui client in coada  
public void adaugareClient*(*Client client*) {* this.clients.add*(*client*)*; // adaugam clientul in lista de clienti  
 this.waitingTime += client.getServiceTime*()*; // adaugam timpul de procesare al noului client la timpul de asteptare a cozii  
*}*// Metoda de functionare a unei cozi  
@Override  
public void run*() {* // Cat timp coada e deschisa  
 while *(*open*) {* // Si daca lista de clienti nu e vida  
 if *(*clients.size*()* != 0*) {* Client primulClient = clients.get*(*0*)*; // primul client care asteapta la coada  
 int timp = primulClient.getServiceTime*()*; // timpul de procesare a clientului care e primul la coada  
  
 if *(*timp > 1*) {* // daca timpul de procesare e mai mare ca 1, adica daca inca mai trebuie sa fie procesat  
  
 primulClient.setServiceTime*(*timp - 1*)*; // ii decrementam primului client timpul de procesare  
  
 this.waitingTime--; // decrementam timpul de asteptare al cozii  
 *}* else *{* // timp == 1, adica terminam procesarea cu clientul curent  
  
 clients.remove*(*clients.get*(*0*))*; // stergem primul client din coada pentru ca timpul lui de procesare s-a incheiat  
  
 this.waitingTime--; // decrementam timpul de asteptare al cozii  
 *}  
 }* try *{* Thread.*sleep(*1000*)*; // asteapta 1 secunda  
 *}* catch *(*InterruptedException ex*) {* ex.printStackTrace*()*;  
 *}  
 }  
}*

**3. Clasa Simulation** – primește ca atribute datele de pe interfață, precum și o varibila de tip AtomicInteger care contorizează timpul curent, doua de tip int pentru ora de vârf și numărul de clienți corespunzători și 2 liste de tip BlockingQueue pentru o bună sincronizare, una pentru cozi, iar cealaltă pentru clienți.

// Declararea campurilor necesare  
private View view;  
  
private AtomicInteger timpCurent;  
  
private int numarCozi;  
private int timpAsteptareTotal;  
private int timpProcesareTotal;  
private int timpMaxSimulare;  
private int nrClientiProcesati;  
private int numarClienti;  
  
private int tMinArrival;  
private int tMaxArrival;  
private int tMinService;  
private int tMaxService;  
  
private int peakHour;  
private int peakHourClients;  
  
private BlockingQueue*<*Queue*>* queues;  
private BlockingQueue*<*Client*>* clients;

Pe lângă câteva dintre câmpurile de mai sus, constructorul va primi ca parametru si view-ul pentru a putea fi folosite datele de pe interfață.

public Simulation*(*View view, int numarCozi, int numarClienti, int timpMaxSimulare, int tMinArrival, int tMaxArrival, int tMinService, int tMaxService*) {* this.view = view;  
 this.timpCurent = new AtomicInteger*()*;  
 this.clients = new LinkedBlockingQueue*<>()*;  
 this.queues = new LinkedBlockingQueue*<>()*;  
 this.numarCozi = numarCozi;  
  
 // Adaugam numar de cozi primit ca parametru  
 for *(*int i = 0; i < this.numarCozi; i++*) {* Queue q = new Queue*(*i + 1*)*; // pentru ca indexarea incepe de la 0  
 this.queues.add*(*q*)*;  
 *}* this.numarClienti = numarClienti;  
 this.timpMaxSimulare = timpMaxSimulare;  
 this.nrClientiProcesati = 0; // initializam cu 0 clientii care au fost in coada  
 this.tMinArrival = tMinArrival;  
 this.tMaxArrival = tMaxArrival;  
 this.tMinService = tMinService;  
 this.tMaxService = tMaxService;  
  
 this.peakHour = 0; // initializam cu 0 ora de varf (cand sunt cei mai multi clienti la coada)  
  
 this.generareClientiRandom*()*; // generam clienti random  
*}*

Apoi, urmează metoda de generare random a câmpurilor de timp specifice unui client, metodă, care la rândul ei, folosește o metodă de generare random a unui număr dintr-un interval.

private int generareNumarInterval*(*int a, int b*) {* Random random = new Random*()*;  
  
 return random.nextInt*(*b - a*)* + a; // generam numere random dintr-un interval  
*}*private void generareClientiRandom*() {* // Parcurgem lista de clienti pana la numarul de clienti primit de pe interfata  
 for *(*int i = 0; i < this.numarClienti; i++*) {* // Generam random un timp de arrival si service  
 int randomTArrival = this.generareNumarInterval*(*tMinArrival, tMaxArrival*)*;  
 int randomTService = this.generareNumarInterval*(*tMinService, tMaxService*)*;  
  
 // Clientul o sa aiba un id crescator si cele doua campuri de timp generate random  
 Client client = new Client*(*i + 1, randomTArrival, randomTService*)*;  
  
 this.clients.add*(*client*)*; // adaugam clientul generat in lista  
 *}  
}*

După aceea avem metoda de găsire a cozii cu timpul de așteptare total cel mai scurt pentru a putea adăuga clienți eficient în coadă.

private Queue coadaOptima*() {* // Intr-un BlockingQueue, peek returneaza primul element din coada (in cazul nostru, un obiect de tipul Queue).  
 Queue coadaMinima = this.queues.peek*()*; // initializam minimul cu prima coada  
  
 int timpMinim = coadaMinima.getWaitingTime*()*; // si timpul minim al primului client din coada  
  
 // Parcurgem cu for each lista de cozi  
 for *(*Queue q : this.queues*) {* // Daca timpul de asteptare al cozii este mai mic decat timpul minim  
 if *(*q.getWaitingTime*()* < timpMinim*) {* // Reactualizam timpul minim si coada care are timpul minim  
 timpMinim = q.getWaitingTime*()*;  
 coadaMinima = q;  
 *}  
 }* this.timpAsteptareTotal += timpMinim; // crestem timpul de asteptare total al cozii  
  
 return coadaMinima;  
*}*

Atunci când un client mai are o secundă până termină timpul de service care i s-a alocat, trebuie eliminat din coadă, așadar avem:

private void stergereClientiGata*() {* List*<*Client*>* stersi = new ArrayList*<>()*; // initializam lista de clienti de sters  
  
 // Parcurgem cu for each lista de clienti  
 for *(*Client client : this.clients*) {* if *(*client.getArrivalTime*()* == this.timpCurent.get*())* stersi.add*(*client*)*; // adaugam clientul in lista de sters  
 *}* this.clients.removeAll*(*stersi*)*; // stergem din lista de clienti, clientii care trebuie stersi  
*}*

Urmează metodele de deschidere și închidere a cozilor:

private void deschidereCozi*() {* // Parcurgem cozile cu for each  
 for *(*Queue coada : this.queues*) {* // Pornim un thread pentru fiecare dintre acestea  
 coada.start*()*;  
 coada.setOpen*(*true*)*; // deschidem fiecare coada  
 *}  
}*private void inchidereCozi*() {* // Parcurgem cozile  
 for *(*Queue coada : this.queues*)* coada.setOpen*(*false*)*; // inchidem fiecare coada  
*}*

Mai avem o metodă care detemină numărul total de clienți din toate cozile, deoarece va fi folosit în calcularea timpului mediu și al numărului de clienți din ora de vârf.

public int nrClientiInCozi*() {* int nrClienti = 0; // initializam cu 0 intial  
  
 // Parcurgem cozile cu for each  
 for *(*Queue q : this.queues*) {* nrClienti += q.getClients*()*.size*()*; // crestem numarul de clienti  
 *}* return nrClienti; // numarul total de clienti din cele doua cozi  
*}*

După care urmează o metodă de scriere într-un fișier a rezultatelor obținute, așa cum se specifică în cerință.

// Metoda de scriere a rezultatelor in fisier  
private void scrieInFisier*(*String deScris*) {* try *{* BufferedWriter bw = new BufferedWriter*(*new FileWriter*(*"D:\\Facultate\\UTCN\\An II\\Semestrul 2\\Tehnici de programare\\Laboratoare\\PT2022\_30228\_Gavrilescu\_AndreeaLavinia\_2\\src\\main\\java\\mvc\\models\\log.txt"*))*;  
 bw.append*(*deScris*)*;  
 bw.close*()*;  
 *}* catch *(*IOException ex*) {* ex.printStackTrace*()*;  
 *}  
}*

În cele din urmă, avem metoda de run() care simulează modul de desfășurarea al cozilor și afișează modificările produse în timp real, în consolă, fișier, dar și interfață.

@Override  
public synchronized void run*() {* String deScris = ""; // string care va fi folosit pentru a scrie rezultatul in fisier  
  
 this.deschidereCozi*()*; // deschidem cozile  
  
 // Metoda get() obtine valoarea unui AtomicInteger  
 while *(*this.timpCurent.get*()* < this.timpMaxSimulare*) {* String str = "";  
  
 System.*out*.println*(*"Time " + timpCurent.get*())*; // afisare consola  
 deScris += "Time " + timpCurent.get*()* + "\n"; // afisare in fisier  
 str += "Time " + timpCurent.get*()* + "\n"; // afisare interfata  
  
 // Parcurgem lista de clienti  
 for *(*Client client : this.clients*) {* if *(*this.timpCurent.get*()* == client.getArrivalTime*()) {* if *(*client.getArrivalTime*()* + client.getServiceTime*()* <= timpMaxSimulare*) {* Queue coadaAleasa = this.coadaOptima*()*;  
 coadaAleasa.adaugareClient*(*client*)*; // adaugam clientul in coada cu timpul de asteptare cel mai mic  
  
 this.timpProcesareTotal += client.getServiceTime*()*; // timpul de service al tuturor clientilor  
  
 this.nrClientiProcesati++; // numarul de clienti deserviti  
 *}  
  
 }  
 }* this.stergereClientiGata*()*; // stergerea clientilor deserviti  
  
 System.*out*.print*(*"Waiting clients: "*)*;  
 deScris += "Waiting clients: ";  
 str += "Waiting clients: ";  
  
 // Daca lista de clienti e goala  
 if *(*this.clients.size*()* == 0*) {* System.*out*.println*(*"none"*)*;  
 deScris += "none\n";  
 str += "none\n";  
 *}* else *{* // Parcurgem lista de clienti  
 for *(*Client client : this.clients*) {* // Afisam ca in cerinta  
 System.*out*.print*(*client.toString*()* + "; "*)*;  
 deScris += client.toString*()* + "; ";  
 str += client.toString*()* + "; ";  
 *}* System.*out*.println*()*;  
 deScris += "\n";  
 str += "\n";  
 *}* // Parcurgem cozile  
 for *(*Queue coada : this.queues*) {* System.*out*.print*(*"Queue " + coada.getQueueNumber*()* + ": "*)*;  
 deScris += "Queue " + coada.getQueueNumber*()* + ": ";  
 str += "Queue " + coada.getQueueNumber*()* + ": ";  
  
 // Daca coada e goala  
 if *(*coada.getClients*()*.size*()* == 0*) {* System.*out*.println*(*"closed"*)*;  
 deScris += "closed\n";  
 str += "closed\n";  
 *}* else *{* // Parcurgem lista de clienti  
 for *(*Client client : coada.getClients*()) {* System.*out*.print*(*client.toString*()* + "; "*)*;  
 deScris += client.toString*()* + "; ";  
 str += client.toString*()* + "; ";  
 *}* System.*out*.println*()*;  
 deScris += "\n";  
 str += "\n";  
 *}  
 }* try *{* wait*(*1000*)*; // asteapta o secunda  
 *}* catch *(*InterruptedException ex*) {* ex.printStackTrace*()*;  
 *}* // Metoda nrClientiInCozi() returneaza numarul total de clienti din toate cozile  
 int currentClients = this.nrClientiInCozi*()*;  
  
 if *(*currentClients > this.peakHourClients*) {* this.peakHourClients = currentClients;  
 this.peakHour = this.timpCurent.get*()*;  
 *}* // Metoda addAndGet(x) pentru un AtomicInteger, incrementeaza acel AtomicInteger cu x  
 this.timpCurent.addAndGet*(*1*)*;  
  
 // Daca timpul curent din simulare este mai mare decat timpul maxim de simulare (tMAX Simulare),  
 // inseamna ca s-a terminat simularea  
 if *(*this.timpCurent.get*()* >= this.timpMaxSimulare*)* this.inchidereCozi*()*;  
  
 view.getAfisareTextField*()*.setText*(*str*)*; // afisam pe interfata modul de simulare  
 *}* String str = "";  
 System.*out*.println*(*"Average waiting time:\n" + *(*float*)* this.timpAsteptareTotal / this.nrClientiProcesati*)*;  
 deScris += "Average waiting time: " + *(*float*)* this.timpAsteptareTotal / this.nrClientiProcesati + "\n";  
 str += "Average waiting time: " + *(*float*)* this.timpAsteptareTotal / this.nrClientiProcesati + "\n";  
  
  
 System.*out*.println*(*"Average service time:\n" + *(*float*)* this.timpProcesareTotal / this.nrClientiProcesati*)*;  
 deScris += "Average service time: " + *(*float*)* this.timpProcesareTotal / this.nrClientiProcesati + "\n";  
 str += "Average service time: " + *(*float*)* this.timpProcesareTotal / this.nrClientiProcesati + "\n";  
  
 System.*out*.println*(*"Peak Hour: " + this.peakHour + "\nNumber of clients in Peak Hour: " + this.peakHourClients*)*;  
 deScris += "Peak Hour: " + this.peakHour + "\nNumber of clients in Peak Hour: " + this.peakHourClients + "\n";  
 str += "Peak Hour: " + this.peakHour + "\nNumber of clients in Peak Hour: " + this.peakHourClients + "\n";  
  
 view.getAfisareTextField*()*.setText*(*str*)*; // afisam in interfata  
  
 this.scrieInFisier*(*deScris*)*; // afisam in fisier  
  
*}*

4. **Clasa View** – impementează elementele interfeței grafice

– conține atributele necesare în casete text, a etichetelor corepunzătoare acestora, precum și a butoanului de start care va porni simularea.

În constructor vom poziționa toate elementele interfeței.

// Constructor-ul clasei View  
public View*() {* // Marginile ferestrei interfetei  
 this.setBounds*(*100, 100, 844, 648*)*;  
 this.setDefaultCloseOperation*(*JFrame.*EXIT\_ON\_CLOSE)*;  
 this.getContentPane*()*.setLayout*(*null*)*;  
  
 // Etichetele din interfata  
 titleLabel = new JLabel*(*"QUEUES MANAGEMENT"*)*;  
 titleLabel.setFont*(*new Font*(*"Times New Roman", Font.*PLAIN*, 22*))*;  
 titleLabel.setHorizontalAlignment*(*SwingConstants.*CENTER)*;  
 titleLabel.setBounds*(*233, 10, 405, 49*)*;  
 this.getContentPane*()*.add*(*titleLabel*)*;  
  
 clientiLabel = new JLabel*(*"Numar clienti"*)*;  
 clientiLabel.setHorizontalAlignment*(*SwingConstants.*CENTER)*;  
 clientiLabel.setFont*(*new Font*(*"Times New Roman", Font.*PLAIN*, 20*))*;  
 clientiLabel.setBounds*(*34, 110, 161, 51*)*;  
 this.getContentPane*()*.add*(*clientiLabel*)*;  
  
 coziLabel = new JLabel*(*"Numar cozi"*)*;  
 coziLabel.setHorizontalAlignment*(*SwingConstants.*CENTER)*;  
 coziLabel.setFont*(*new Font*(*"Times New Roman", Font.*PLAIN*, 20*))*;  
 coziLabel.setBounds*(*23, 159, 172, 51*)*;  
 this.getContentPane*()*.add*(*coziLabel*)*;  
  
 simulareLabel = new JLabel*(*"Timp simulare"*)*;  
 simulareLabel.setHorizontalAlignment*(*SwingConstants.*CENTER)*;  
 simulareLabel.setFont*(*new Font*(*"Times New Roman", Font.*PLAIN*, 20*))*;  
 simulareLabel.setBounds*(*23, 206, 172, 51*)*;  
 this.getContentPane*()*.add*(*simulareLabel*)*;  
  
 minArrivalLabel = new JLabel*(*"Min Arrival time"*)*;  
 minArrivalLabel.setHorizontalAlignment*(*SwingConstants.*CENTER)*;  
 minArrivalLabel.setFont*(*new Font*(*"Times New Roman", Font.*PLAIN*, 20*))*;  
 minArrivalLabel.setBounds*(*23, 253, 172, 51*)*;  
 this.getContentPane*()*.add*(*minArrivalLabel*)*;  
  
 maxArrivalLabel = new JLabel*(*"Max Arrival time"*)*;  
 maxArrivalLabel.setHorizontalAlignment*(*SwingConstants.*CENTER)*;  
 maxArrivalLabel.setFont*(*new Font*(*"Times New Roman", Font.*PLAIN*, 20*))*;  
 maxArrivalLabel.setBounds*(*23, 301, 172, 51*)*;  
 this.getContentPane*()*.add*(*maxArrivalLabel*)*;  
  
 minServiceLabel = new JLabel*(*"Min Service time"*)*;  
 minServiceLabel.setHorizontalAlignment*(*SwingConstants.*CENTER)*;  
 minServiceLabel.setFont*(*new Font*(*"Times New Roman", Font.*PLAIN*, 20*))*;  
 minServiceLabel.setBounds*(*23, 347, 172, 51*)*;  
 this.getContentPane*()*.add*(*minServiceLabel*)*;  
  
 maxServiceLabel = new JLabel*(*"Max Service time"*)*;  
 maxServiceLabel.setHorizontalAlignment*(*SwingConstants.*CENTER)*;  
 maxServiceLabel.setFont*(*new Font*(*"Times New Roman", Font.*PLAIN*, 20*))*;  
 maxServiceLabel.setBounds*(*23, 397, 172, 51*)*;  
 this.getContentPane*()*.add*(*maxServiceLabel*)*;  
  
 afisareLabel = new JLabel*(*"Afisare cozi"*)*;  
 afisareLabel.setHorizontalAlignment*(*SwingConstants.*CENTER)*;  
 afisareLabel.setFont*(*new Font*(*"Times New Roman", Font.*PLAIN*, 25*))*;  
 afisareLabel.setBounds*(*424, 101, 190, 49*)*;  
 this.getContentPane*()*.add*(*afisareLabel*)*;  
  
 // Casetele text in care se vor introduce datele si se va afisa simularea cozilor  
 clientiTextField = new JTextField*()*;  
 clientiTextField.setBounds*(*205, 126, 56, 24*)*;  
 this.getContentPane*()*.add*(*clientiTextField*)*;  
 clientiTextField.setColumns*(*10*)*;  
  
 coziTextField = new JTextField*()*;  
 coziTextField.setColumns*(*10*)*;  
 coziTextField.setBounds*(*205, 175, 56, 24*)*;  
 this.getContentPane*()*.add*(*coziTextField*)*;  
  
 simulareTextField = new JTextField*()*;  
 simulareTextField.setColumns*(*10*)*;  
 simulareTextField.setBounds*(*205, 222, 56, 24*)*;  
 this.getContentPane*()*.add*(*simulareTextField*)*;  
  
 minArrivalTextField = new JTextField*()*;  
 minArrivalTextField.setColumns*(*10*)*;  
 minArrivalTextField.setBounds*(*205, 269, 56, 24*)*;  
 this.getContentPane*()*.add*(*minArrivalTextField*)*;  
  
 maxArrivalTextField = new JTextField*()*;  
 maxArrivalTextField.setColumns*(*10*)*;  
 maxArrivalTextField.setBounds*(*205, 317, 56, 24*)*;  
 this.getContentPane*()*.add*(*maxArrivalTextField*)*;  
  
 minServiceTextField = new JTextField*()*;  
 minServiceTextField.setColumns*(*10*)*;  
 minServiceTextField.setBounds*(*205, 363, 56, 24*)*;  
 this.getContentPane*()*.add*(*minServiceTextField*)*;  
  
 maxServiceTextField = new JTextField*()*;  
 maxServiceTextField.setColumns*(*10*)*;  
 maxServiceTextField.setBounds*(*205, 413, 56, 24*)*;  
 this.getContentPane*()*.add*(*maxServiceTextField*)*;  
  
 afisareTextField = new JTextArea*()*;  
 afisareTextField.setColumns*(*10*)*;  
 afisareTextField.setBounds*(*271, 152, 531, 459*)*;  
 this.getContentPane*()*.add*(*afisareTextField*)*;  
  
 // Butonul de start care va porni simularea cozilor  
 startButton = new JButton*(*"START"*)*;  
 startButton.setFont*(*new Font*(*"Times New Roman", Font.*PLAIN*, 20*))*;  
 startButton.setBounds*(*47, 479, 195, 49*)*;  
 this.getContentPane*()*.add*(*startButton*)*;  
  
 // Afisarea ferestrei actuale  
 this.setVisible*(*true*)*;  
*}*

Pe lângă metodele de get() pentru castele text, avem metode pentru a verifica dacă unul dintre butoane este apăsat, caz în care se trimite un apel către controller pentru a afișa rezulatul corepunzător.

public void addStartListener*(*ActionListener actionListener*) {* this.startButton.addActionListener*(*actionListener*)*;  
*}*

5. **Clasa Controller** – primește un apel când este apăsat un buton de pe interfață, iar după efectuarea operației corespunzătoare butonului, returnează rezultatul în caseta specifică.

Constructorul primește ca parametru view-ul, iar acesta are rolul de a lega butoanul de start de metoda care apelează începerea simulării.

// Constructorul care primeste ca parametru un view  
public Controller*(*View view*) {* this.view = view;  
  
 // Punem listener pe buton pentru a porni simularea la apasare  
 this.view.addStartListener*(*new StartListener*())*;  
*}*

Apoi avem metodele care implementeaza interfața ActionListener prin care se verifică inputul primit după apăsarea unui buton.

class StartListener implements ActionListener *{* @Override  
 public void actionPerformed*(*ActionEvent e*) {* try *{* // Preluare date de pe interfata  
 int nrCozi, nrClienti, timpMaxSimulare, tMinArrival, tMaxArrival, tMinService, tMaxService;  
 nrCozi = Integer.*parseInt(*view.getCoziTextField*())*;  
 nrClienti = Integer.*parseInt(*view.getClientiTextField*())*;  
 timpMaxSimulare = Integer.*parseInt(*view.getSimulareTextField*())*;  
 tMinArrival = Integer.*parseInt(*view.getMinArrivalTextField*())*;  
 tMaxArrival = Integer.*parseInt(*view.getMaxArrivalTextField*())*;  
 tMinService = Integer.*parseInt(*view.getMinServiceTextField*())*;  
 tMaxService = Integer.*parseInt(*view.getMaxServiceTextField*())*;  
  
 // Pornim simularea executiei cozilor cu parametrii preluati de pe interfata  
 Simulation simulation = new Simulation*(*view, nrCozi, nrClienti, timpMaxSimulare, tMinArrival, tMaxArrival, tMinService, tMaxService*)*;  
 simulation.start*()*;  
  
 *}* catch *(*NumberFormatException exception*) {* exception.printStackTrace*()*;  
 *}  
 }  
}*

6. **Clasa Main** – are atributele view și controller-ul care primește ca parametru view-ul respectiv, precum si metoda main.

public static void main*(*String*[]* args*) {* View view = new View*()*; // interfata  
  
 Controller controller = new Controller*(*view*)*; // controller-ul care primeste ca parametru intefata  
  
*}*

Rezultate

Pentru testarea proiectului am folosit exemplele furnizate, iar rezultatul acestora va fi afișat în fișierele corespunzătoare.

O imagine care conține text

Descriere generată automat

Concluzii

Acest proiect mi-a consolidat cunoștințele teoretice și practice de Java pe care le-am învățat în semestrul 1 prin volumul de lucru mult mai intens și avansat pe care l-am depus în scopul finalizării acestei teme.

Ca dezvoltare ulterioară, se mai poate îmbunătăți aspectul interfeței ca să arate mai interesant, se mai poate adăuga tot felul de butoane pentru pauză sau stop, putând fi utlizată eventual pentru magazine online.

Bibliografie

* <https://dsrl.eu/courses/pt/materials/A2_Support_Presentation.pdf>
* <https://dsrl.eu/courses/pt/materials/PT2021-2022_Assignment_2.pdf>
* <https://docs.oracle.com/javase/tutorial/essential/concurrency/index.html>
* <https://www.tutorialspoint.com/java/util/timer_schedule_period.htm>
* https://www.javacodegeeks.com/2013/01/java-thread-pool-example-using-executors-and-threadpoolexecutor.html