DOCUMENTATIE TEMA 2

QUEUES SIMULATOR

Nume student: Buzilă-Gârda Andra-Maria
Grupa: 30229
Profesor de laborator: assist. Antal Marcel

Cuprins:

1. Cerinte functionale:

- Implementati o simulare a unei aplicatii care analizeaza un sistem de cozi pentru a determina si minimalize asteptarea timpului la coada de catre client;
- Aplicatia ar trebui sa simuleze o serie de n client care se aseaza la coada si asteapta sa fie serviti. Se introduce un numar de Q cozi. Fiecare client ajunge la cate o coada, asteapta sa fie servit, este servit, iar apoi pleaca. Clientii sunt generate aleatory atunci cand incepe simularea si sunt caracterizati de urmatorii parametrii: ID (un numar intre 1 si N), tArrival (timpul la care clientul merge sa se aseze la coada) si tService (timpul de care are nevoie clientul pentru a fi servit). Sistemul calculeaza timpul fiecarui client aflat in cozi si calculeaza media timpului de asteptare. Fiecare client, atunci cand ii vine randul, este adaugat in coada cu cel mai scurt timp de asteptare;
- Urmatoarele date ar trebui considerate date de intrare pentru aplicatie, utilizatorul le introduce in momentul pornirii aplicatiei in interfata:
    - Number of clients;
    - Number of quques;
    - Simulation interval;
    - Minimum arrival time;
    - Maximum arrival time;
    - Minimum service time;
    - Maximum service time;
- Cerinte pentru notare:
    - Utilizati un limbaj de programare orientat pe obiecte;
    - Generati aleator clientii;
    - Folositi multithreading: un thread per coada;
    - Asigurati siguranta threadurilor folosind date sincronizate potrivite;
    - Salvati intr-un fisier .txt datele obtinute in urma simularii;
    - Implementati clase cu maxim 300 linii (cu exceptia celor ce apartin interfetei cu utilizatorul) si metode cu maxim 30 linii;
    - Folositi nume pertinente conform conventiei din Java;
    - Creati o interfata grafica pentru simulare si afisarea in timp real a evolutiei cozilor;
    - Afisati rezultatele simularii (timpul mediu de asteptare, timpul mediu de servire a clientilor, ora de varf), pentru fiecare interval, fie in interfata grafica fie in fisierul .txt create anterior;
    - Rulati aplicatia cu datele specificate.

2. Obiective

2.1. Obiectiv principal

Proiectarea si implementarea unui simulatori de cozi pentru a simplifica asteptarea clientilor la Coada prin minimizarea timpului de asteptare.

2.2. Obiective secundare

Obiectiv	Descriere	Capitol
Dezvoltarea de use case-uri si scenarii	Dorim sa cunoastem ce se intampla in momentul introducerii datelor si apasarii butonului de procesare	3
Alegerea structurilor de date	Modul de reprezentare in memorie	4
Impartirea problemei pe clase	Cream clasele de care avem nevoie	4
Implementarea solutiilor	Descrierea modului in care am creat fiecare clasa si fiecare metoda in parte	5
Testare (cu datele specificate)	Verificarea functionalitatii	6

3. Analiza problemei

Utilizatori:

- Manageri de restaurante;
- Manageri de supermarket-uri;
- Manageri sau detinatori de orice tip de magazine.

Pre-conditii:

- Utilizatorul a introdus corect numarul de clienti;
- Utilizatorul a introdus corect numarul de cozi;
- Utilizatorul a introdus corect intervalul de simulare;
- Utilizatorul a introdus corect timpul minim de sosire;
- Utilizatorul a introdus corect timpul maxim de sosire;
- Utilizatorul a introdus corect timpul minim de servire;
- Utilizatorul a introdus corect timpul maxim de servire;

Post-conditii:

- Se realizeaza in timp real statul la coada de catre clientii generate aleator;
- Se afiseaza timpul mediu de asteptare;
- Se afiseaza timpul mediu de servire;
- Se afiseaza ora de varf (ora la care au fost cei mai multi client la cozi);

Modul de functionare:

1. Utilizatorul introduce de la tastatura input-urile necesare pentru functionarea aplicatiei. Dupa introducerea corecta a datelor cerute, programul va genera aleator atatia clienti cati sunt specificati de numarul introdus de utilizator. Pe interfata va aparea o lista de client care asteapta, fiecare client este reprezentat de 3 numere: ID, tArrival, tService. Clientul intra in coada cu cel mai scurt timp de asteptare in momentul in care timpul simularii este egal cu timpul sau tArrival. Dupa ce a intrat in coada si daca nu mai are pe nimeni in fata lui, i se va scadea timpul de servire cu cate o unitate odata cu incrementarea timpului curent (utilizatorul vede toata aceasta poveste, deoarece ea este dispusa in timp real, atat in interfata, cat si in consola si in fisierul .txt). Clientul dispare din coada cand are timpul de servire 0. Dupa ce timpul curent este acelasi cu intervalul de simulare introdus de

utilizator, programul se termina, iar in fisierul .txt sunt afisate rezultatele desprinse in urma simularii, pe care utilizatorul le poate vedea, si anume: media timpului de asteptare, media timpului de servire si ora de varf.

! In cazul in care utilizatorul a introdus gresit vreunul dintre input-uri, sau chiar toate, va fi atentionat printr-un mesaj de eroare afisat in consola.

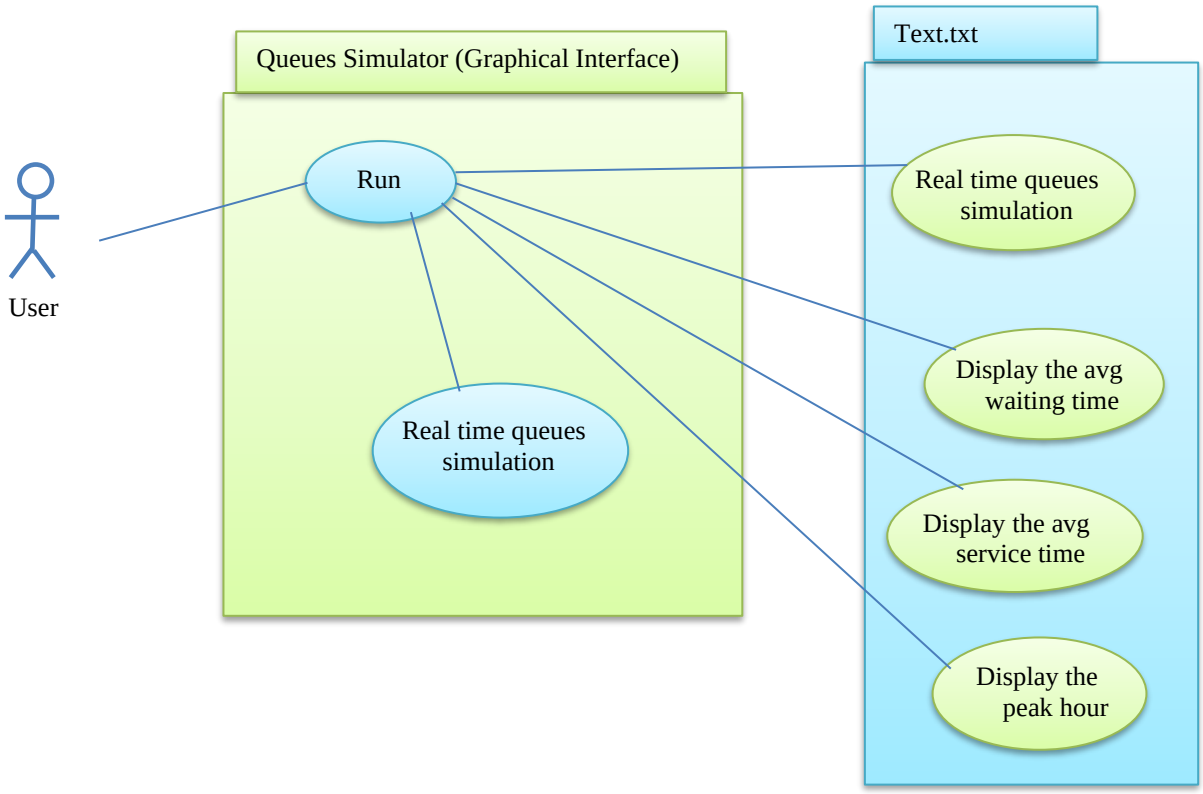

Figura 1: REPREZENTARE DIAGRAMA USE CASE

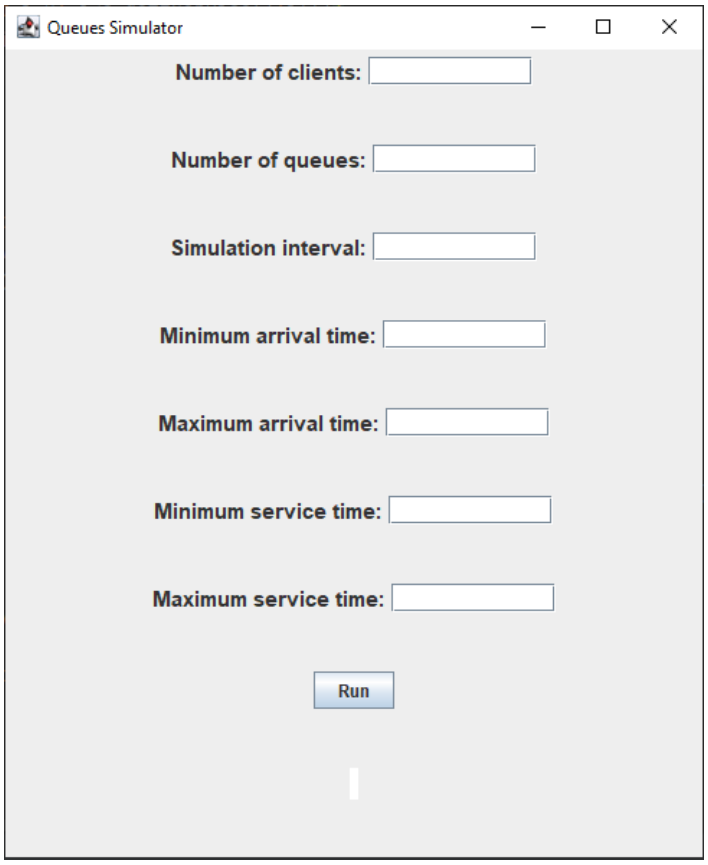

Figura 2: SIMULATORUL VAZUT DE UTILIZATOR

4. Proiectare

4.1. Alegerea structurilor de date

In clasa Store (vederea in ansamblu a magazinului) am create o lista care continue toti clientii, o alta lista pentru cozile existente, o lista in care voi introduce clientii pe care doresc sa ii sterg din lista mare odata ce s-au asezat la cozi si mai exista 2 liste: una cu toti clientii de la coada de la fiecare ora si alta cu orele in care numarul de client de la cozi este maxim.

In clasa Queuee mai am o lista de clienti, care reprezinta practice numarul de clienti aflati la fiecare coada.

4.2. Diagrama de clase

Unified Modeling Language (prescurtat UML) este un limbaj standard pentru descrierea de modele și specificatii pentru software. UML ofera o larga gama de diagrame pentru modelarea diferitelor situatii in cadrul unui proiect de dezvoltare software.

Diagrama de clasa este folosita pentru reprezentarea vizuala a claselor ai a interdependentelor, taxonomiei si a relatiilor de multiplicitate dintre ele. Aceste diagrame sunt folosite si pentru reprezentarea concreta a unor instante de clasa, asadar obiecte, si a legaturilor concrete dintre acestea.

Am create aceasta aplicatie luamndu-ma dupa structura Model-View-Controller (MVC) si anume: am facut clasa ClassView in care am schitat imaginea de ansamblu a simulatorului, unde utilizatorul introduce datele, contine butoane, textField-uri, label-uri si un textPane in care se va afisa rezultatul pe parcursul evolutiei. Am construit apoi clasa ClassController, unde am manipulat datele introduse prin clasa ClassView, iar Modelul este reprezentat de clasele Client, Store, Queuee, FileData si am creat, de asemenea, clasa Main.

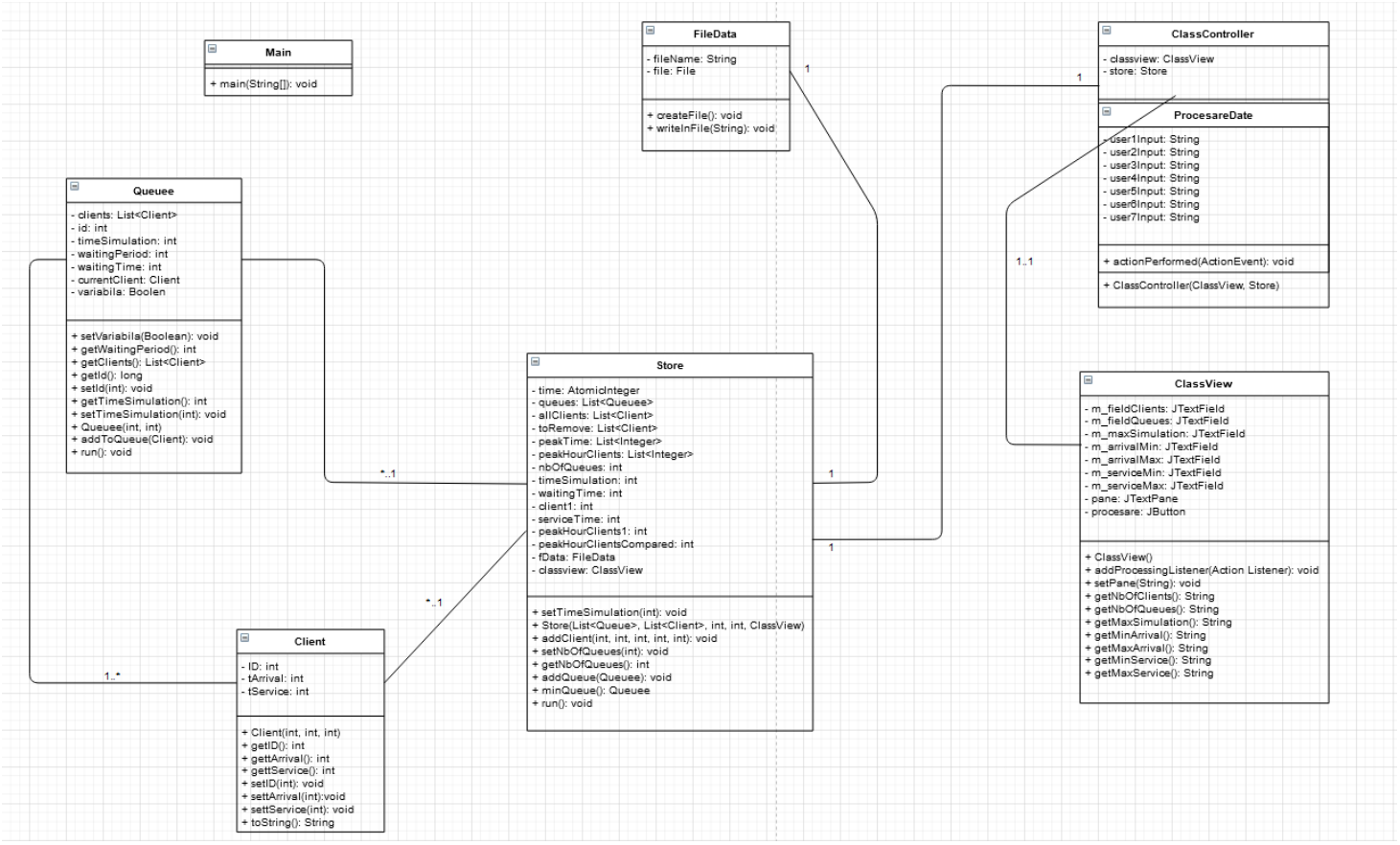

Figura 3: DIAGRAMA UML

5. Implementare

Clasa Client

Clasa Client continue 3 variabile instanta, valori specific fiecarui client, si anume: ID (fiecare client are propriul numar de identificare), tArrival (timpul la care clientul intra in coada) si tService (timpul cat dureaza procesarea clientului odata ce nu mai are pe nimeni in fata lui). Tot in aceasta clasa exista constructorul Client si metode de getter si setter pentru fiecare variabila mentionata anterior, dar mai este si metoda toString care afiseaza cele 3 numere pentru fiecare client.

```java
public class Client {
    public int ID;
    public int tArrival;
    public int tService;

    public Client(int ID, int tArr, int tSer){
        this.ID = ID;
        this.tArrival = tArr;
        this.tService = tSer;
    }

    public int getID() {
        return ID;
    }
    public int gettArrival() {
        return tArrival;
    }
    public int gettService() {
        return tService;
    }
    public void setID(int ID) {
        this.ID = ID;
    }
    public void settArrival(int tArrival) {
        this.tArrival = tArrival;
    }
    public void settService(int tService) {
        this.tService = tService;
    }
    public String toString(){
        return "("+this.ID+","+this.tArrival+","+this.tService+")";
    }
}
```

Clasa Queuee

Variabilele instanta din clasa Queuee sunt: lista de client (care va continue clientii aflati in coada la un anumit moment), un id (pentru a stii despre care coada este vorba), o variabila de timeSimulation (care se introduce de catre utilizator), variabila waitingPeriod (reprezinta timpul de asteptare la coada la un moment dat), currentClient (primul client de la coada) si o variabila de tip Boolean (care marcheaza inchiderea buclei while din metoda run).

Metodele prezente in aceasta clasa sunt cateva metode de getter si setter de care o sa avem nevoie si anum: gettes si setters pentru id si timeSimulation, metoda de setVariabila, metoda care va seta variabila declarata la true atunci cand se incepe executia cozilor si la false pentru a se incheia. Tot in clasa Queuee exista constructorul Queue, dar si metoda addToQueue care adauga client in lista de client a cozii respective si adauga la waitingPeriod-ul curent, timpul de servire al clientului adaugat. Deoarece Clasa Queuee extinde Clasa Thread va exista si o metoda de run. Aceasta metoda run este apelata in metoda run din clasa store prin apelul de sistem start(). Practic metoda run va decrementa tService din secunda in secunda (datorita apelului sleep(1000), unde 1000 reprezinta numarul de milisecunde => 1 secunda) pana ajunge la 0. Daca tService-ul clientului este 0 acesta va disparea si din coada in care era (clientul paraseste magazinul). Am luat pe cazuri in felul urmator: primul caz, cand primul client are tService > 1 atunci se decrementeaza si timpul de asteptare sit Service-ul clientului.In al doilea caz, data tService este 1, clientul de va elimina din coada ca la urmatoarea afisare coada sa fie goala si de asemenea se decrementeaza perioada de asteptare. Daca tService este mai mic decat 1 se va arunca o exceptie in consola.

```java
import java.util.ArrayList;
import java.util.List;

class Queuee extends Thread{

    private List<Client> clients = new ArrayList<>();
    private int id;
    private int timeSimulation;
    private int waitingPeriod;
    private Client currentClient;
    private Boolean variabila;

    public void setVariabila(Boolean variabila) { this.variabila = variabila; }


    public int getWaitingPeriod() { return waitingPeriod; }

    public List<Client> getClients() { return clients; }


    @Override
    public long getId() { return id; }

    public void setId(int id) { this.id = id; }

    public int getTimeSimulation() { return timeSimulation; }

    public void setTimeSimulation(int timeSimulation) { this.timeSimulation = timeSimulation; }


    public Queuee(int id, int timeSimulation){
        this.id = id;
        this.timeSimulation = timeSimulation;
```

```
48      }
49
50      public void addToQueue(Client c){
51          this.clients.add(c);
52          for(int i=0; i<c.gettService();i++){
53              waitingPeriod++;
54          }
55      }
56
57
58      @Override
59      public void run(){
60          while(variabila){
61              if(!clients.isEmpty()){
62                  currentClient=clients.get(0);
63                  if(currentClient.gettService()>1){
64                      currentClient.settService(currentClient.gettService()-1);
65                      waitingPeriod--;
66                  }
67                  else{
68                      if(currentClient.gettService()==1){
69                      clients.remove(currentClient);
70                      waitingPeriod--;
71                      }else{
72                          try {
73                              throw new Exception("Error");
74                          } catch (Exception e) {
75                              e.printStackTrace();
76                          }
77                      }
78                  }
79              }
80              try{
```

```
79              }
80              try{
81                  sleep( millis: 1000);
82              } catch (InterruptedException e) {
83                  e.printStackTrace();
84              }
85          }
86      }
87  }
```

Clasa Store

Aceasta clasa contine urmatoarele variabile instanta: time, care este de tipul AtomicInteger si se va incrementa in metoda run pan ava ajunge la timeSimulation, timeSimulation este tot o variabila de tip int, de data aceasta, si reprezinta valoarea introdusa de utilizator in campul pentru “Simulation interval”, o lista cu cozile prezente in magazine, o lista cu toti clientii care au intrat in magazin dar nu stau la vreo coada, o lista cu clientii care se vor sterge din lista mare de clienti (clientii care intra in cozi), o lista peakTime unde se vor inregistra orele de varf, o alta lista: peakHourClients care contine numarul de client de la fiecare ora, dintre aceste numere se va alege cel mai mare pentru ora de varf. Mai exista variabila nbOfQueues (cu numarul de cozi introdus de utilizator), variabila timeSimulation adica intervalul de simulare, valoare introdusa tot de utilizator, variabila client1 in care se stocheaza numarul de client introdusi cu success in cozi, serviceTime, o variabila in care se stocheaza suma tService-urilor tuturor clientilor introdusi in cozi, cu ajutorul careia vom calcula average service time, mai exista variabila peakHourClients1 in care se va salva numarul maxim de client care stau la cozi la un anumit moment, iar variabila peakHourClientsCompared are rol de contor si se compara mereu cu peakHourClients1 pentru a se extrage valoarea maxima. Am create si o instanta de FileData si una de ClassView pentru a afisa resultatul in interfata. Variabila waitingTime salveaza suma tuturor waitingTime-urilor de la fiecare client care intra in coada pentru a se putea realiza average waiting time la final.

Am create un constructor Store unde am initializat variabilele, exista si metode de getter si setter si o metoda de addClient care adauga clienti in lista mare allClients. Metoda minQueue returneaza o coada care are

timpul de asteptare minim. Am initializat minWaitingTime cu timpul de asteptare al primei cozi si l-am masurat pe rand cu fiecare timp de asteptare, adica de la fiecare coada, si l-am adaugat pe cel mai mic, iar in final am extras coada cu acel minWaitingTime si am returnat-o. Deoarece si aceasta clasa extinde clasa Thread, am create si aici o metoda run(). In aceasta metoda, prima data am pornit toate cozile cu apelul start si le-am setat variabila la true pentru a functiona while-ul din metodele run ale cozilor. Si in clasa Store in metoda run avem un while, doar ca aici avem conditie si anume se verifica daca timpul curent a ajuns sau nu la timeSimulation, in cazul in care timpul curent este mai mic se intra in bucla. Ceea ce se intampla in bucla respectiva se va repeta pentru fiecare timp incepand cu 0 si pana la timpul simularii introdus de utilizator. In bucla se initializeaza un String care la finalul fiecarei repetari de bucla va fi afisat atat in interfata, cat si in consola si in fisierul .txt, aceasta afisare se face progresiv datorita faptului ca si aici avem un wait(1000) (obligatoriu timpul trebuie sa fie egal cu cel din metoda run al clasei Queuee, pentru ca altfel ar aparea decalari, nu am mai avea aceleasi solutii). In bucla while se verifica daca timpul tArrival al fiecarui client este acelasi cu timpul curent, daca sunt egale se verifica daca timpul curent +tService nu depaseste timpul simularii, daca si aceasta conditie este indeplinita atunci clientul poate intra in coada cu timpul minim de asteptare, numarul de client procesati se incrementeaza, iar la waitingTime si serviceTime li se aduna tService-ul clientului procesat. Se sterg din lista mare de client, clientii care au intrat in cozi, iar pe urma se creeaza string-ul care trebuie afisat. Acesta va fi afisat in consola, in interfata prin apelul unei metode din ClassView si in fisierul .txt prin apelul unei metode din FileData, dupa care se face asteptarea. Se adauga in lista de peakHourClients, numarul de client de la timpul respective, iar apoi acesta se incrementeaza si se reia procedeul din bucla. Dupa terminare se seteaza variabila din fiecare coada la false pentru a se inchide procesele. Tot dupa terminare se salveaza in peakHourClients1 numarul maxim de clienti si apoi se compara cu fiecare numar de client de la fiecare coada si cand se gaseste se salveaza in lista peakTime iteratorul corespunsator pozitiei gasite. Pe urma se scriu average waiting time= waitingTime/client1, average service time= ServiceTime/client1 si peakHour= peakTime in fisier tot prin apelul aceleiasi metode din FileData.

```java
import java.io.FileWriter;
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;
import java.util.Random;
import java.util.concurrent.atomic.AtomicInteger;

public class Store extends Thread {
    private AtomicInteger time = new AtomicInteger();
    private List<Queuee> queues = new ArrayList<>();
    private List<Client> allClients = new ArrayList<>();
    private List<Client> toRemove = new ArrayList<>();
    private List<Integer> peakTime = new ArrayList<>();
    private List<Integer> peakHourClients = new ArrayList<>();
    private int nbOfQueues;
    private int timeSimulation;
    private int waitingTime;
    private int client1;
    private int serviceTime;
    private int peakHourClients1;
    private int peakHourClientsCompared;
    private FileData fData;
    private ClassView classview;

    public void setTimeSimulation(int timeSimulation) { this.timeSimulation = timeSimulation; }

    public Store(List<Queuee> l1, List<Client> l2, int nbOfQueues, int timeS, ClassView v) throws IOException {
        this.queues=l1;
        this.allClients=l2;
        this.nbOfQueues=nbOfQueues;
        this.timeSimulation = timeS;
        this.fData = new FileData();
        this.classview = v;
```

```
        fData.createFile();
    }

    public void addClient(int id,int minArr, int maxArr, int minSer, int maxSer){
        Random r = new Random();
        int arrival=r.nextInt( bound: maxArr-minArr)+minArr;
        int service=r.nextInt( bound: maxSer-minSer)+minSer;
        Client c = new Client(id,arrival,service);
        allClients.add(c);
    }

    public void setNbOfQueues(int nbOfQueues) { this.nbOfQueues = nbOfQueues; }

    public int getNbOfQueues() { return nbOfQueues; }
    public void addQueue(Queuee q) { queues.add(q); }

    public Queuee minQueue(){
        int minWaitingTime=queues.get(0).getWaitingPeriod();
        Queuee q1=null;
        for(Queuee q:queues){
            if(minWaitingTime>q.getWaitingPeriod())
                minWaitingTime=q.getWaitingPeriod();
        }
        for(Queuee q:queues) {
            if (minWaitingTime == q.getWaitingPeriod()){
                q1 = q;
                break;
            }
        }
        waitingTime+=minWaitingTime;
        return q1;
    }
```

```
    @Override
    public synchronized void run(){
        for(Queuee q:queues){
            q.start();
            q.setVariabila(true); }
        while(time.get()<=timeSimulation){
            String s="Time "+time.get()+"\n";
            for(Client c:allClients)
                if(c.gettArrival()==time.get())
                    if(time.get()+c.gettService()<=timeSimulation){
                        Queuee q1=minQueue();
                        q1.addToQueue(c);
                        waitingTime+=c.gettService();
                        client1++;
                        serviceTime+=c.gettService(); }
            for(Client c: allClients)
                if(c.gettArrival()==time.get())
                    toRemove.add(c);
            allClients.removeAll(toRemove);
            s+="Waiting clients: ";
            if(allClients.size()!=0){
                for(Client c:allClients)
                        s+=c.toString()+", ";
                s+="\n"; }
            else{
                s+="closed\n"; }
            for(Queuee q:queues){
                s+="Queue "+q.getId()+":";
                if(!q.getClients().isEmpty())
                    for (Client c : q.getClients()){
                        peakHourClients1++;
                        s += c.toString() + ", ";}
                else
```

```
                    s+="closed";
                s+="\n"; }
            peakHourClients.add(peakHourClients1);
            peakHourClients1=0;
            System.out.println(s);
            classview.setPane(s);
            try {
                fData.writeInFile(s);
            } catch (IOException e) {
                e.printStackTrace();
            }
            try{
                wait( timeout: 1000);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
            time.incrementAndGet();
            if(time.get()>timeSimulation)
                for(Queuee q: queues)
                    q.setVariabila(false); }
        peakHourClients1=peakHourClients.get(0);
        for(int i= 0;i<peakHourClients.size();i++){
            peakHourClientsCompared=peakHourClients.get(i);
            if(peakHourClients1<peakHourClientsCompared)
                peakHourClients1=peakHourClientsCompared;}
        for(int i=0 ;i<peakHourClients.size();i++){
            if(peakHourClients.get(i).compareTo(peakHourClients1)==0)
                peakTime.add(i);
        }
        try {
            fData.writeInFile( s: "Average waiting time is: "+(float) waitingTime/client1+"\n"+"Average service time is: "+(float) serviceTime/client1+"\n"+"Peak hour is: "+peakT
        } catch (IOException e) {
            e.printStackTrace();
```

Clasa FileData

In clasa FileData am creat o instanta de File: file si o instanta de String care este numele fisierului. Mai exista si metoda de createFile care pur si simplu creeaza fisierul file prin apelul functiei createNewFile(). Metoda writeInFile scrie in fisierul creat cu ajutorul unei variabile writer de tipul FileWriter, al unui buffer si cu functia append.

```java
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;

public class FileData {
    private String fileName = "text.txt";
    File file = new File( pathname: "text.txt");
    public void createFile() throws IOException {
        file.createNewFile();
    }

    public void writeInFile(String s) throws IOException {
        try {
            FileWriter write = new FileWriter( fileName: "D:\\PT_30229_Buzila_Andra_Assignmenttt\\text.txt", append: true);
            BufferedWriter buffWr = new BufferedWriter(write);
            buffWr.append(s);
            buffWr.newLine();
            buffWr.close();
            write.close();
        } catch (IOException e) {
            e.printStackTrace();
        }

    }
}
```

Clasa ClassView

In aceasta clasa practic am modelat imaginea de ansamblu a interfetei. Exista mai multe variabile instanta de tipul JTextField, o variabila de tipul JTextPane si o variabila de tipul JButton. In metoda ClassView am create instante de Panel-uri, Label-uri, am setat fontul scrisului, titlul interfetei si dimensiunea ei.

Tot in aceeasi clasa am mai multe metode, una se numeste addProcessingListener care implementeaza ActionListener adica interconecteaza apasarea butonului cu functionarea aplicatiei. Tot aici mai exista o metoda prin care se afiseaza rezultatul in in acel TextPane cu ajutorul functiei setText. Mai exista cate o metoda pentru fiecare imput care returneaza un string, adica acele input-uri introduse de utilizator in fiecare TextField.

```java
import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionListener;

public class ClassView extends JFrame {
    private JTextField m_fieldClients = new JTextField( columns: 10);
    private JTextField m_fieldQueues = new JTextField( columns: 10);
    private JTextField m_maxSimulation = new JTextField( columns: 10);
    private JTextField m_arrivalMin = new JTextField( columns: 10);
    private JTextField m_arrivalMax = new JTextField( columns: 10);
    private JTextField m_serviceMin = new JTextField( columns: 10);
    private JTextField m_serviceMax = new JTextField( columns: 10);
    JTextPane pane = new JTextPane();

    private JButton procesare= new JButton( text: "Run");

    ClassView(){

        Font f = new Font( name: "TimesRoman",Font.BOLD, size: 15);
        JPanel panel1 = new JPanel();
        JPanel panel2 = new JPanel();
        JPanel panel3 = new JPanel();
        JPanel panel4 = new JPanel();
        JPanel panel5 = new JPanel();
        JPanel panel6 = new JPanel();
        JPanel panel7 = new JPanel();
        JPanel panel8 = new JPanel();
        JPanel panel9 = new JPanel();

        JLabel label1 = new JLabel( text: "Number of clients:");
        label1.setFont(f);
        panel1.add(label1);
        panel1.add(m_fieldClients);
```

```
        panel1.setLayout(new FlowLayout());

        JLabel label2 = new JLabel( text: "Number of queues:");
        label2.setFont(f);
        panel2.add(label2);
        panel2.add(m_fieldQueues);
        panel2.setLayout(new FlowLayout());

        JLabel label3 = new JLabel( text: "Simulation interval:");
        label3.setFont(f);
        panel3.add(label3);
        panel3.add(m_maxSimulation);
        panel3.setLayout(new FlowLayout());

        JLabel label4 = new JLabel( text: "Minimum arrival time:");
        label4.setFont(f);
        panel4.add(label4);
        panel4.add(m_arrivalMin);
        panel4.setLayout(new FlowLayout());

        JLabel label5 = new JLabel( text: "Maximum arrival time:");
        label5.setFont(f);
        panel5.add(label5);
        panel5.add(m_arrivalMax);
        panel5.setLayout(new FlowLayout());

        JLabel label6 = new JLabel( text: "Minimum service time:");
        label6.setFont(f);
        panel6.add(label6);
        panel6.add(m_serviceMin);
        panel6.setLayout(new FlowLayout());

        JLabel label7 = new JLabel( text: "Maximum service time:");
        label7.setFont(f);
        panel7.add(label7);
        panel7.add(m_serviceMax);
        panel7.setLayout(new FlowLayout());
        panel8.add(procesare);
        panel8.setLayout(new FlowLayout());
        panel9.add(pane);
        JPanel p = new JPanel();
        p.add(panel1);
        p.add(panel2);
        p.add(panel3);
        p.add(panel4);
        p.add(panel5);
        p.add(panel6);
        p.add(panel7);
        p.add(panel8);
        p.add(panel9);
        p.setLayout(new BoxLayout(p, BoxLayout.Y_AXIS));
        setTitle("Queues Simulator");
        setPreferredSize(new Dimension( width: 500, height: 600));
        this.setContentPane(p);
        this.pack();
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
    void addProcessingListener(ActionListener aal) { procesare.addActionListener(aal); }
    void setPane(String s) { pane.setText(s); }
    String getNbOfClients() { return m_fieldClients.getText(); }
    String getNbOfQueues() { return m_fieldQueues.getText(); }
    String getMaxSimulation() { return m_maxSimulation.getText(); }
    String getMinArrival() { return m_arrivalMin.getText(); }
    String getMaxArrival() { return m_arrivalMax.getText(); }
    String getMinService() { return m_serviceMin.getText(); }
    String getMaxService() { return m_serviceMax.getText(); }
```

Clasa ClassController

In clasa ClassController exista o instanta de ClassView si una de Store. Constructorul ClassController initealizeaza acele variabile. Avem nevoie de o noua clasa pentru butonul de procesare in care se va intra atunci cand acesta este apasat. In aceasta noua clasa avem cate o variabila instanta pentru fiecare input, iar in metoda actionPerformed li se atribuie variabilelor acele valori si se face parsarea lor. Se creeaza clienti random si se introduce in lista de client din store. De asemenea se creeaza si atatea cozi cat numarul de cozi introdus si de adauga in lista de cozi din store. Dupa aceea se incepe simulatorul prin apelul functiei start() pentru clasa instanta store.

```java
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

public class ClassController {
    private final ClassView classview;
    private final Store store;

    ClassController(ClassView view, Store s){
        classview=view;
        store=s;

        classview.addProcessingListener(new procesareDate());
    }

    class procesareDate implements ActionListener{
        String user1Input="";
        String user2Input="";
        String user3Input="";
        String user4Input="";
        String user5Input="";
        String user6Input="";
        String user7Input="";

        @Override
        public void actionPerformed(ActionEvent e) {
            user1Input=classview.getNbOfClients();
            user2Input=classview.getNbOfQueues();
            user3Input= classview.getMaxSimulation();
            user4Input=classview.getMinArrival();
            user5Input=classview.getMaxArrival();
            user6Input= classview.getMinService();
            user7Input= classview.getMaxService();
            try {
                user3Input= classview.getMaxSimulation();
                user4Input=classview.getMinArrival();
                user5Input=classview.getMaxArrival();
                user6Input= classview.getMinService();
                user7Input= classview.getMaxService();
                try {
                    int nbQueues = Integer.parseInt(user2Input);
                    for (int i = 0; i < nbQueues; i++) {
                        Queuee qu = new Queuee(i, Integer.parseInt(user3Input));
                        store.addQueue(qu);
                    }
                    int nbClients = Integer.parseInt(user1Input);
                    int minArr = Integer.parseInt(user4Input);
                    int maxArr = Integer.parseInt(user5Input);
                    int minSer = Integer.parseInt(user6Input);
                    int maxSer = Integer.parseInt(user7Input);
                    store.setNbOfQueues(nbQueues);
                    store.setTimeSimulation(Integer.parseInt(user3Input));
                    for (int i = 0; i < nbClients; i++) {
                        store.addClient(i, minArr, maxArr, minSer, maxSer);
                    }
                    store.start();
                } catch (NumberFormatException numberFormatException) {
                    numberFormatException.printStackTrace();
                }
            }
        }
    }
}
```

Clasa Main

In clasa Maine exista metoda main in care am creat o instanta de ClassView pe care am trimis-o ca parametru in momentul in care am creat instanta de clasa Store, iar apoi ambele le-am trimis ca parametrii la crearea instantei de ClassController.

```java
import java.io.IOException;
import java.util.ArrayList;


public class Main {
    public static void main(String[] args) throws IOException {
        ClassView vedere = new ClassView();
        Store s = new Store(new ArrayList<>(), new ArrayList<>(),  nbOfQueues: 0,  timeS: 0,vedere);
        ClassController controller = new ClassController(vedere,s);
        vedere.setVisible(true);
    }
}
```

6. Testare

Testarea a fost facuta pentru datele sugerate in enuntul problemei. Pentru fiecare dintre cele 3 sugestii de input-uri am afisat rezultatele in fisierele: text.txt, text1.txt si text2.txt.

7. Concluzii si dezvoltari ulterioare

Se poate implementa in acelasi mod o aplicatie de simulare a cumparaturilor online. Se pot introduce diferite butoane, de exemplu daca utilizatorul doreste sa puna pauza simularii sau sa reseteze TextField-urile astfel incat sa introduca alte valori.

In concluzie, aceste limbaje OOP sunt foarte usor de folosit si la indemana oricui stie paradigmele OOP de baza. Mi-am perfectionat modul de programare, atat aplicand tehnici noi cat si repetandu-le pe cele deja invatate.

8. Bibliografie

- Documentele puse la dispozitie semestrul trecut de catre facultate: cursuri, documente de la laborator;
- YouTube
- http://www.tutorialspoint.com/java/util/timer_schedule_period.htm
- https://www.javacodegeeks.com/2013/01/java-thread-pool-example-using-executors-and-threadpoolexecutor.html
- https://www.w3schools.com/java/java_threads.asp