

**FACULTATEA: Automatica si Calculatoare**

**SPECIALIZAREA: Calculatoare si tehnologia informatiei**

**DISCIPLINA: Tehnici de Programare**

**AN UNIVERSITAR: 2019/2020**

**PROIECT: Queues Simulator**

**Realizat de :**

Ghiura Darius

**Grupa:** 30227

**CUPRINS**

1. **Obiectivul temei**
2. **Analiza problemei, modelare, scenarii, cazuri de utilizare**
3. **Proiectare**
4. **Implementare**
5. **Rezultate**
6. **Concluzii**
7. **Bibliografie**
8. Obiectivul temei

Obiectivul temei este proiectarea si implementarea unei aplicatii care simuleaza un sistem de cozi de asteptare pentru un anumit numar de clienti. Simulatorul are ca obiectiv minimizarea timpului de asteptare pentru clienti si calcularea timpului mediu de asteptare. Simularea trebuie sa ruleze pana cand timpul curent a ajung la durata maxima de simulare sau se opreste daca nu mai avem clienti. De asemenea, aplicatia trebuie realizata folosind un sistem de multithreading.

1. Analiza problemei, modelare, scenarii, cazuri de utilizare

Conceptul de coada este utilizat in viata de zi cu zi. Practic o coada ofera unei persoane, unui client, un loc in care sa astepte pana cand va fi servit. In mod normal, un client ar alege coada in care timpul de asteptare va fi cel mai mic, astfel acest lucru trebuie simulat si de aplicatie.

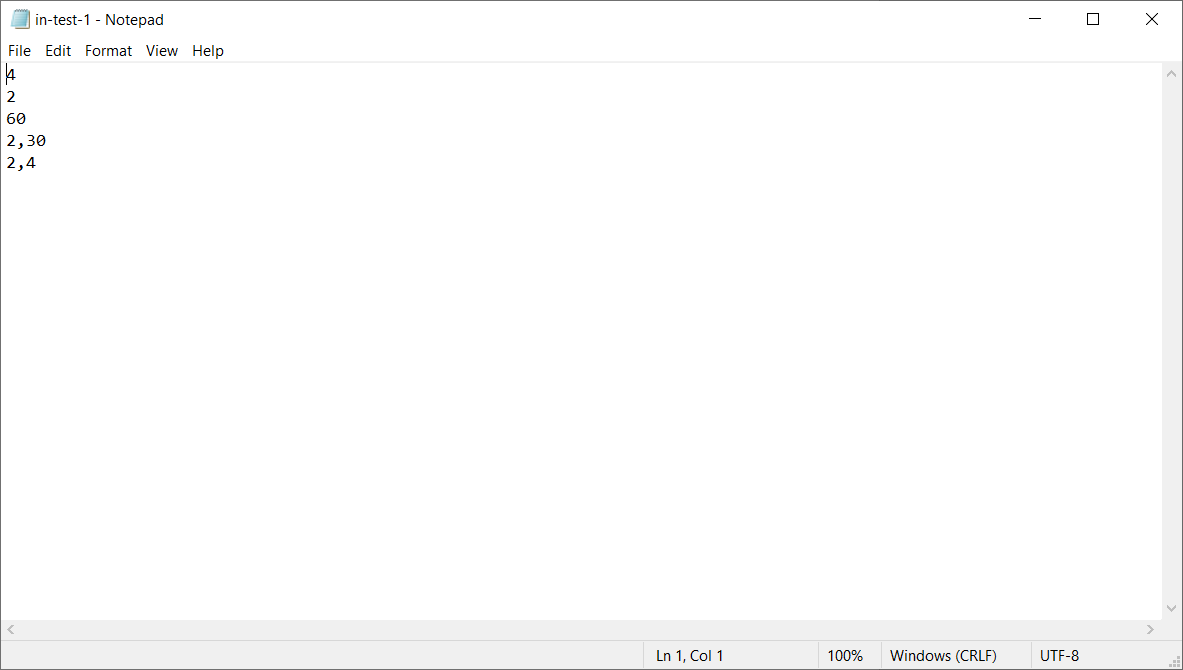
Programul simuleaza urmatorul scenariu : intr-un magazin avem N clienti, dupa ce isi termina cumparaturile, un client intra intr-o coada la un anumit moment de timp numit „Arrival time”. De asemenea fiecare client are un timp de procesare care spune cat dureaza sa fie servit acel client.

In momentul in care clientul trebuie sa intre intr-o coada, aplicatia il repartizeaza in coada unde timpul de asteptare este cel mai mic, apoi clientul asteapta pana cand ii vine randul si dupa ce este servit iese din coada. Timpul de astepare al unei cozi este calculat insumand duratele de procesare a clientilor aflati in coada

Timpul mediu de asteptare este calculat astfel : se aduna timpii de asteptare a tuturor clientilor si se imparte la numarul de clienti. Timpul de asteptare pentru un client este reprezentat de durata de procesare adunata cu timpul de asteptare din coada la momentul in care acesta intra in coada.

Aplicatia citeste datele de intrare dintr-un fisier si tot intr-un fisier face afisarea rezultatelor simularii.

Pentru a utiliza aplicatia este nevoie de un fisier de intrare cu formatul :



Unde intrarile reprezinta, in ordine :

* numarul de clienti,
* numarul de cozi,
* durata maxima a simularii,
* momentul de timp minim de intrare in coada,
* momentul de timp maxim de intrare in coada,
* durata minima de procesare,
* durata maxima de procesare.

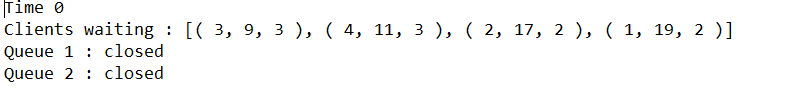
Aplicatia poate fi rulata din consola utilizand fisierul jar, folosind comanda :

***java -jar PT2020\_30227\_Darius\_Ghiura\_Assignment\_2.jar in.txt out.txt***

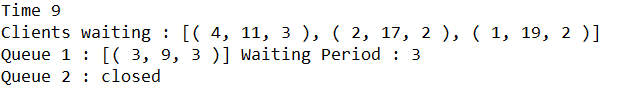
Unde „in.txt” este numele fisierului cu date de intrare care trebuie sa existe,

iar „out.txt” este numele fisierului unde se va face afisarea, care poate sa existe sau va fi creat daca nu de catre program.

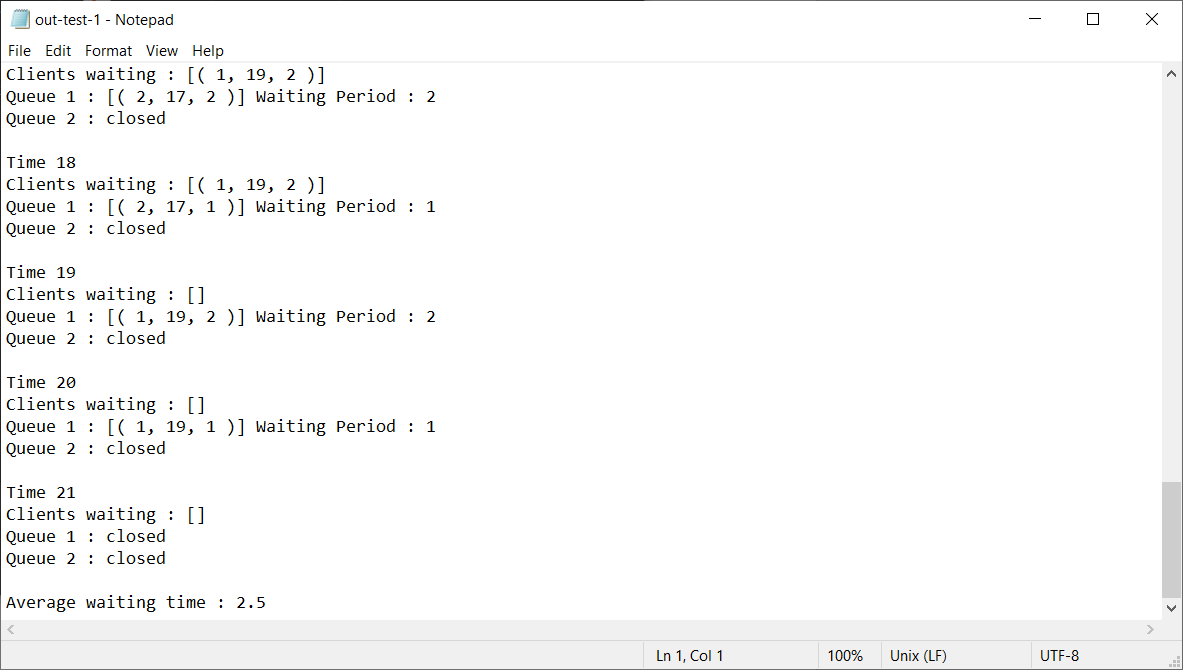
La fiecare secunda din simulare in fisier este afisata starea cozilor si lista de clienti care inca nu au intrat intr-o coada.



La momentul de timp t = arrivalTime, afisarea va arata astfel :



La sfarsitul fisierului se va afisa si timpul mediu de astepare :



1. Proiectare

Procesul de proiectare a simulatorului de cozi a fost concentrat asupra folosirii mai multor threaduri. Fiecare coada ruleaza pe un fir de executie care incepe doar cand un client intra in coada si se opreste daca coada este goala.

Exista si un thread principal care se ocupa de simularea scenariului si are rolul de a contoriza timpul, a comanda introducerea unui client intr-o coada si a afisa in fisier.

Clasele proiectului sunt :

* Client
* Queue
* Scheduler
* SimulationManager

Clasa Scheduler este responsabila pentru stabilirea strategiei de inserare in cozi. Aceasta strategie este determinarea cozii care are timpul de asteptare cel mai mic si inserarea in aceasta.

Clasa Client descrie un client care este caracterizat de : ID, timpul in care intra in coada si timpul de care are nevoie pentru a fi servit.

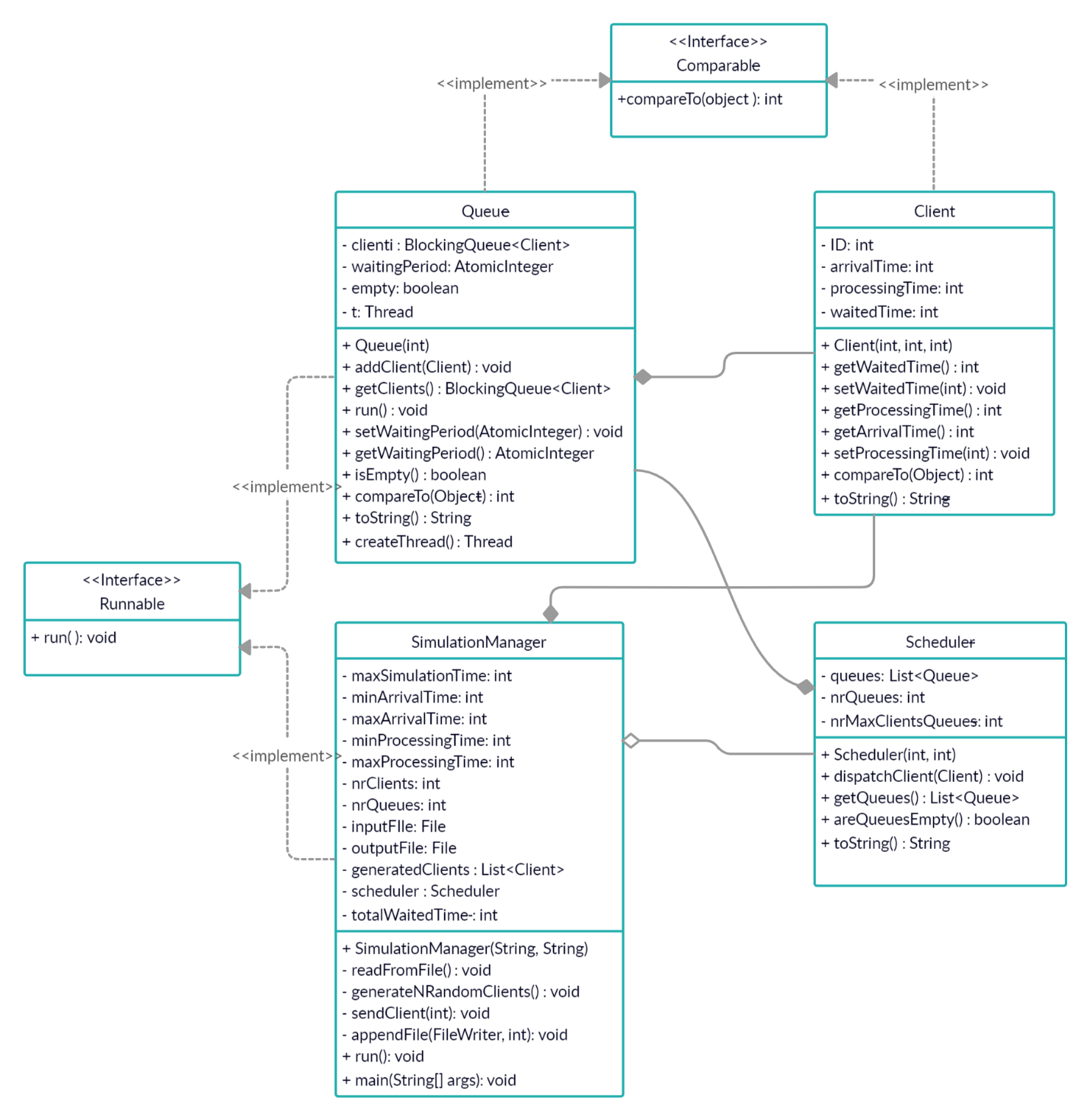
Clasa Queue descrie threadurile care reprezinta cozile simularii. In aceste threaduri clientii sunt procesati si sunt scosi din coada dupa ce timpul de procesare a trecut.

Clasa SimulationManager este responsabila de managementul simularii. In aceasta clasa se afla generatorul random de clienti, care genereaza clienti cu un timp de intrare in coada si un timp de procesare random, in conformitate cu datele de intrare.

Clasele au fost proiectate respectand conceptul de programare orientata pe obiect de incapsulare. Variabilele instanta sunt declarate cu modificatorul de acces „private” ele fiind accesibile din exteriorul claselor doar prin gettere si settere.

Cand se opereaza cu mai multe threaduri pentru a nu avea probleme, trebuie sa se foloseasca structuri de date speciale astfel incat o variabila sau un obiect sa poata fi accesat doar de un thread la un anumit moment de timp. Pentru asta in dezvoltarea aplicatiei au fost folosite structuri de date precum colectii sincronizate, BlockingQueue, AtomicInteger si variabile volatile.

**Diagrama UML**



1. Implementare

**Clasa : Client**

Clasa implementeaza interfata Comparable.

Variabile instanta :

* **private int ID**;
* **private int arrivalTime** - reprezinta momentul de timp in care clientul intra intr-o coada;
* **private int processingTime** - reprezinta durata de timp necesara pentru procesarea clientului;
* **private int waitedTime** = 0 – reprezinta durata de timp pe care clientul o asteapta intr-o coada. Aceasta variabila este folosita pentru calcularea timpului mediu de asteptare. Este initializata cu 0 si in momentul in care un client intra intr-o coada primeste timpul de asteptare din coada respectiva + timpul de procesare a clientului;

Constructor :   
 **public Client(int ID, int arrivalTime, int processingTime)** – initializeaza variabilele instanta.

Metode :

* **public int getWaitedTime()** – getter pentru waitedTime;
* **public void setWaitedTime(int waitedTime)** – setter pentru waitedTime;
* **public int getProcessingTime()** – getter pentru processingTime;
* **public int getArrivalTime()** – getter pentru arrivalTime;
* **public void setProcessingTime(int processingTime)** – setter pentru processingTime;
* **public int compareTo(Object o)** – Metoda folosita pentru sortarea clientilor in ordine crescatoare a timpului de intrare in coada;
* **public String toString()** – Metoda folosita la afisarea unui client.

**Clasa : Scheduler**

Variabile instanta :

* **private List<Queue> queues** – Lista de cozi;
* **private int nrQueues** – Numarul de cozi care e dat in fisierul cu date de intrare;
* **private int nrMaxClientsQueue –** Numarul maxim de clienti intr-o coada**;**

Constructor :

**public Scheduler(int nrQueues, int nrMaxClientsQueue)** – Creeaza o lista de cozi folosind o colectie sincronizata, si introduce cozile in aceasta lista. Coada este creata ca un BlockingQueue si are marimea maxima nrMaxClientsQueue care primeste numarul maxim de clienti din fisier – 1. De asemenea pentru fiecarea coada se creeaza un thread si se da drumul la acesta folosind metoda start().

Metode :

* **public void dispatchClient(Client c) throws InterruptedException** - Metoda stabileste care este coada cu cel mai mic timp de asteptare, parcurgand lista de cozi si apoi apeleaza metoda de adaugare in coada pentru coada aia;
* **public List<Queue> getQueues()**;
* **public boolean areQueuesEmpty()** – Metoda determina daca exista o coada care nu este goala. Este folosita pentru a stii cand sa se intrerupa executia simularii;
* **public String toString()** – Metoda folosita pentru afisarea starii tututor cozilor. Daca o coada este goala se afiseaza „Closed”.

**Clasa : Queue**

Clasa implementeaza interfata Runnable.

Variabile instanta :

* **private BlockingQueue<Client> clients** – Structura speciala de coada folosita cand se lucreaza cu threaduri;
* **private AtomicInteger waitingPeriod** – Reprezinta timpul de asteptare pe care un client ar trebui sa il astepte daca ar intra in coada;
* **private volatile boolean empty** – Variabila folosita pentru iesirea din bucla while a metodei run a threadului cand coada este goala. În cazul în care o variabilă este declarată cu cuvântul cheie volatile, atunci este garantat că orice fir care citeste un câmp va vedea valoarea cea mai recent scris;
* **private Thread t**;

Constructor :

**public Queue(int nrClients)** – Creeaza un ArrayBlockingQueue pentru si initializeaza waitingPeriod cu 0 si empty cu true.

Metode :

* **public void addClient(Client c) throws InterruptedException** – Metoda adauga un client in coada si seteaza variabile empty = false; de asemenea daca marimea cozii este 1 dupa ce elementul a fost inserat, se porneste un nou thread, deoareace inseamna ca inainte coada era goala si threadul era mort;
* **public BlockingQueue<Client> getClients()**
* **public void run()** – Metoda reprezinta instructiunile executate de thread. Daca coada nu este goala, cand se porneste, threadul doarme o secunda. Apoi se trezeste scade timpul de asteptare din coada cu 1, ia primul element din coada si verifica daca trebuie scos din coada sau doar ii scade timpul de procesare cu 1. Daca elementul a fost scos din coada, se verifica si daca coada e goala, caz in care variabila volatila empty primeste valoarea true si se iese din bucla de while. Threadul doarme o secunda cand incepe, deoarece threadul porneste atunci cand un client este inserat in coada, iar noi vrem sa incepem sa scadem timpul de procesare doar cu o secunda dupa ce a intrat in coada;
* **public void setWaitingPeriod(AtomicInteger waitingPeriod)**
* **public AtomicInteger getWaitingPeriod()**
* **public boolean isEmpty()** – Returneaza true daca coada este goala;
* **public int compareTo(Object o)**
* **public String toString()** – Metoda folosita pentru afisarea cozii
* **public Thread createThread()** – Metoda creeaza un nou thread cu obiectul curent;

**Clasa : SimulationManager**

Clasa implementeaza interfata Runnable si contine metoda main a aplicatiei.

Variabile instanta :

* **private int maxSimulationTime** – Durata maxima a simularii;
* **private int minArrivalTime**;
* **private int maxArrivalTime**;
* **private int minProcessingTime**;
* **private int maxProcessingTime**;
* **private int nrClients**;
* **private int nrQueues**;
* **private File inputFile**;
* **private File outputFile**;
* **private List<Client> generatedClients** – Lista de clienti generati random
* **private Scheduler scheduler;**
* **private int totalWaitedTime** = 0 – Variabila folosita pentru calcularea timpului mediu de asteptare;

Constructor :

**public SimulationManager(String input, String output)** – Constructorul primeste 2 stringuri care sunt argumentele din metoda maine si reprezinta numele fisierelor de intrare si de iesire. Constructorul creeaza obiecte de tipul File din numele fisierelor, apeleaza metoda care citeste datele de intrare din fisier, apeleaza metoda care genereaza clientii si creeaza un nou obiect instanta clasei Scheduler.

Metode :

* **private void readFromFile()** – Metoda citeste folosind un Scanner cate o linie din fisier si folosind metoda parseInt() din clasa Integer, da valoare variabilelor instanta;
* **private void generateNRandomClients()** –Metoda creeaza o lista sincronizata si cu un for de la 0 pana la numarul de clienti – 1 creeaza clienti astfel : ID-ul clientului este valoarea iteratorului + 1 ; Folosind metoda random() din clasa Math, metoda genereaza clienti care au timpul de intrare in coada intre timpul minim si timpul maxim de intrarea si timpul de procesare intre timpul minim si maxim de procesare. La final clientii din aceasta lista vor fi sortati in ordine crescatoare a timpului de intrare in coada;
* **private void sendClient(int currentTime)** – Metoda este apelata din metoda run() si are rolul de a trimite un client intr-o coada la momentul potrivit, in acelasi timp stergandu-l din lista de clienti care sunt in asteptare si adauga la variabila totalWaitedTime timpul asteptat de client pentru a putea calcula timpul mediu asteptat la final;
* **private void appendFile(FileWriter writer, int currentTime)** – Metoda are rolul de a scrie in fisier. Scrisul se pune la finalul fisierului fara a suprascrie ce se afla in el.
* **public void run()** – Metoda threadului principal care se ocupa de simulare. Aceasta are o variabila numita currentTime care reprezinta timpul. Cat timp currentTime este mai mic sau egal decat timpul maxim al simularii se apeleaza metoda sendClient si appendFile pentru afisare. Se verifica daca lista de asteptare si cozile sunt goale, caz in care se intrerupe simularea. La finalul buclei while timpul curent se incrementeaza si threadul doarme o secunda. La finalul metodei se calculeaza timpul mediu de asteptare si se afiseaza in fisier.
* **public static void main(String[] args)** – Metoda creeaza o instanta a clasei SimulationManager si trimite args[0] si args[1] care reprezinta numele fisierelor de intrare si de iesire. Metoda creeaza un thread cu obiectul instantiat si apeleaza start().

1. Rezultate

Aplicatia a fost testata manual, verificand outputul din fisier pentru 3 fisiere de intrare.

1. Concluzii

Aplicatia implementeaza cu succes scenariul de simulare a unui magazin in care clientii intra intr-o coada la un moment de timp arrivalTime si sunt repartizati in coada in care timpul de asteptare este minim, asteapta pana sunt serviti, iar apoi parasesc coada.

Acesta a fost primul proiect in care am lucrat cu threaduri si a fost foarte util pentru ca am invatat despre importanta sincronizarii intre threaduri si folosirea de structuri de date speciale pentru lucrul cu threaduri.

1. Bibliografie

<https://docs.oracle.com/javase/tutorial/essential/concurrency/index.html>

<https://docs.oracle.com/javase/7/docs/api/java/lang/Thread.html>