MINISTRY OF EDUCATION AND RESEARCH

TECHNICAL UNIVERSITY

OF CLUJ-NAPOCA, ROMANIA

FUNDAMENTAL PROGRAMMING TECHNIQUES ASSIGNMENT 2

QUEUES SIMULATOR

By Mihali Vlad

Group: 30424

Content

1. The objective of this assignment

The main goal of this project is to design and implement a queues simulator application aiming to analyse queuing based systems for determining and minimizing clients' waiting time.

The secondary objectives are:

- parsing the input data; (page 8, Chapter 4.3.1.)
- design and implement Client class; (page 5, Chapter 4.1.1.)
- generate and create list of clients; (page 9, Chapter 4.3.2.)
- design and implement a timer for simulation; (page 7, Chapter 4.1.4.)
- design and implement queue for this application; (page 6, Chapter 4.1.2.)
- create and handle the list of queues; (page 7, Chapter 4.1.3)
- design and implement a class that writes in file and connects waiting clients with queues; (page 8, Chapter 4.2.)
- compute the average waiting time; (page 8, Chapter 4.2.)
- system testing; (page 9, Chapter 5.1.)

Parsing input data is done by extracting strings from file which is given as first command-line argument and convert into processable data. This are than by the class ParseInput.
This are inputs data read from a text file for the application:
- Number of clients (numberOfClients);
- Number of queues (numberOfQueues);
- Simulation interval (simulationInterval);
- Minimum and maximum arrival time (minArrivalTime and maxArrivalTime);
- Minimum and maximum service time (minServiceTime and maxServiceTime);

The implementation of the Client class is the base of this assignment. A client is represented by his unique id (ID), arrival time and service time. Example: ID = 1, arrival time = 3 and service time = 4 means that the client is ready to enter the queue after the arrival time (after 3 seconds) and is need to stay int the front of the queue 4 seconds.

In the ClientGenerator class are generated random clients. The number of client is equal with numberOfClients. The bounds of random arrival time are minArrivalTime and maxArrivalTime. The bounds of random service time are minServiceTime and maxServiceTime.

Timer class is a class that implements Runnable interface and is a thread that keeps tracking of simulation time. The Timer stops when the simulation time equals with simulationInterval.

QueueThread class is a class that implements Runnable interface and is a thread that keeps tracking of a queue. Has a unique queue id. The clients are added in the end of the queue. The client which is in front of the queue is proceed and every 1 second his service time is decremented until reaches 0 and is removed from the queue.

QueueList class creates a list of all queues and handle them. This class helps adding client in the queue with the minimum waiting time.

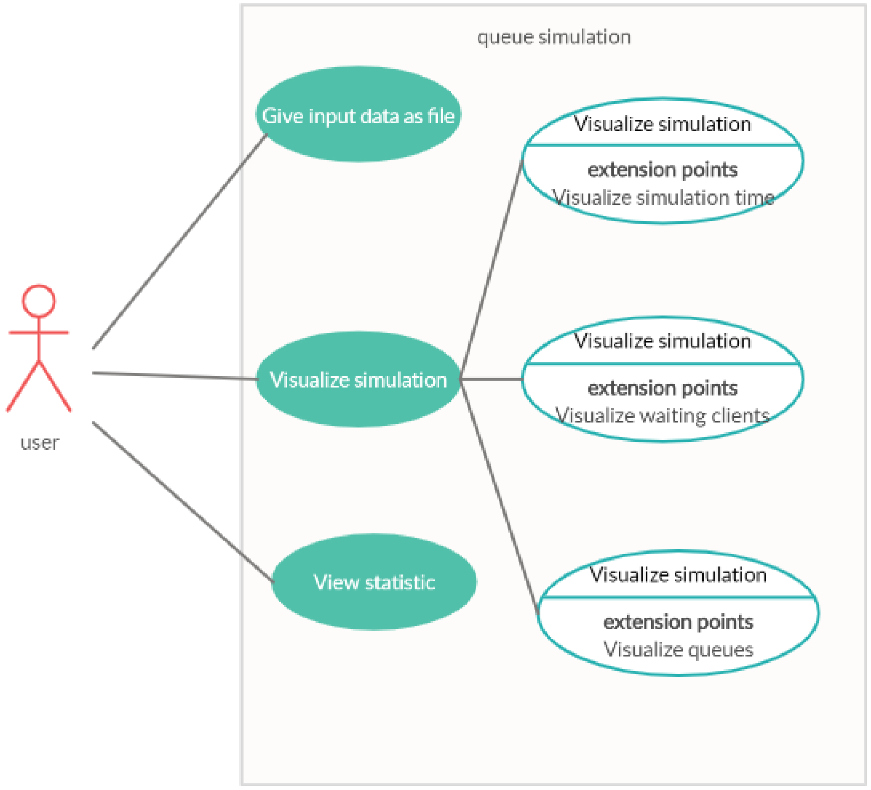
OutputWriter class is a class that implements Runnable interface and is a thread that combines timer, list of queues, list of clients and average waiting time and write there data in output file which is given as second command-line argument.

The average waiting time is result of the total time spend by every customer in the queues divided by total number of clients. The average waiting time is computed and appended to the file.

In the system testing we will test the functionality of the application and the response of the input data and the simulation flow.

2. Analysis of the problem

Use case of the application[1]:



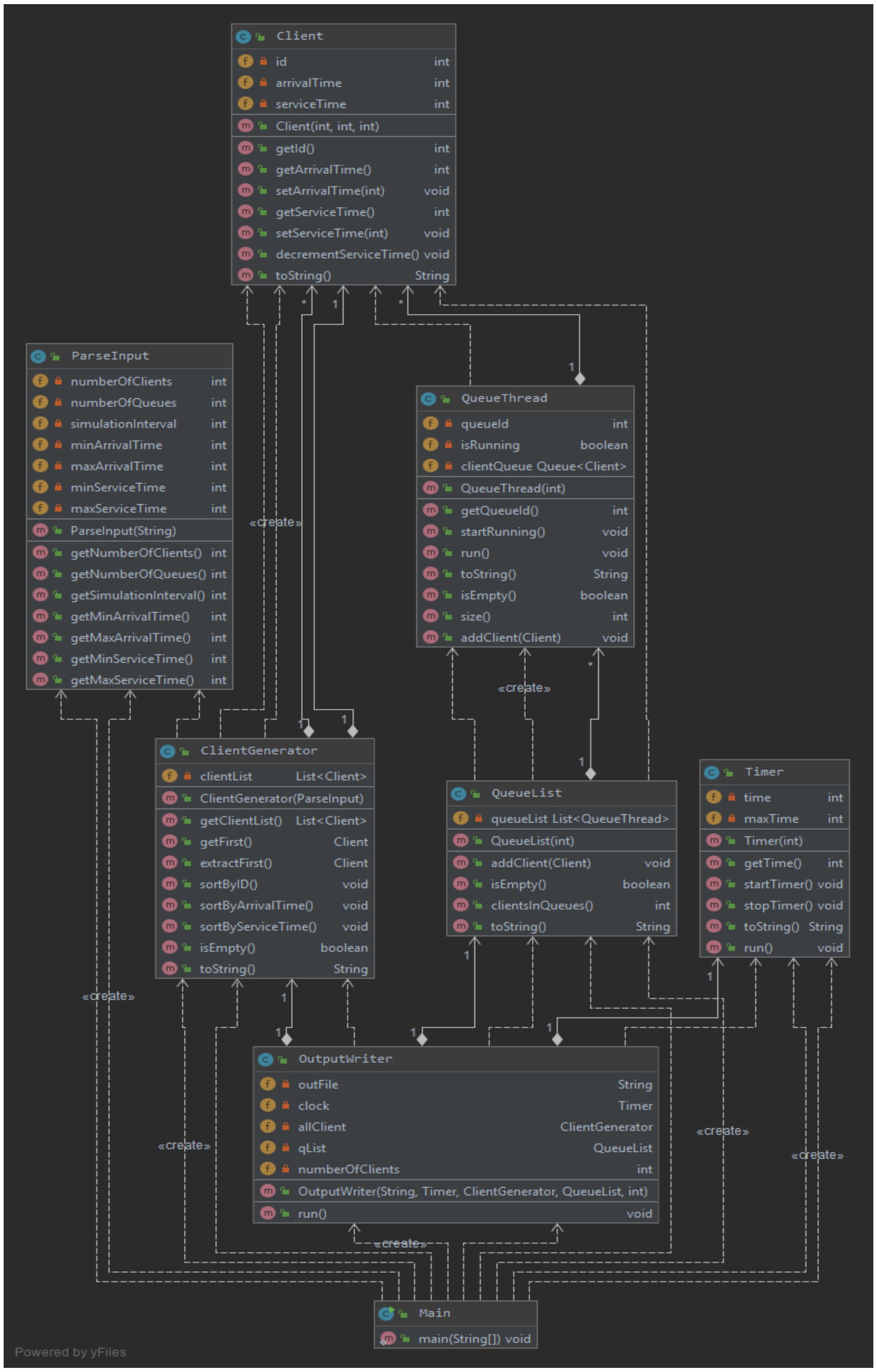
System: queue simulation (application)

Actors: The user

Use case is a tool to analyze and divide the problem in small pieces easy to resolve. The main success scenario is the user provide a valid input file and the simulation is done properly and the user can see the evolution of the simulation. The alternative (failure) scenario is when the user provide an invalid data and an error message will be displayed in the console.

3. Design

Class diagram[1]:



In my design I used three packages (Model View Controller) :
•model –contains all the data-related logic that the user works with, represents the data that is being transferred between the view and controller components. Here are timer, client and queues;(Timer, Client, QueueThread, QueueList)

•view –contains all the output logic of the application (OutputWriter);

•controller –acts as an interface between the model and the view to process all the business logic and incoming requests, manipulate data using the model and interact with the views to render the final output. Here are done reading data and generate clients (ParseInput, ClientGenerator). [2]

Encapsulation is a mechanism of wrapping the variables and code acting on the methods together as a single unit. In encapsulation, the variables of a class will be hidden from other classes and can be accessed only through the methods of their current class. Therefore, it is also known as data hiding. [3]

To obtain an object-oriented programming design, I used following principles: encapsulation, inheritance, polymorphism and threads.

Data structures used:

Queue (LinkedList) is used in QueueThread class as a field and to manage the queue simulation. The Queue is used to insert elements at the end of the queue and removes from the beginning of the queue. It follows FIFO concept.

ArrayList is used in ClientGenerator (clientList) and QueueList to store dynamically sized collection of Client objects and QueueThread objects. Contrary to Arrays that are fixed in size, an ArrayList grows its size automatically when new elements are added to it and are easy to use.

4. Implementation

4.1.Package model

4.1.1. Client

```
public class Client
```

A client is represented by three private integer variable: his unique id, arrival time (arrivalTime) and service time (serviceTime).

```
private int id;
private int arrivalTime;
private int serviceTime;
```

This constructor set variable of the object Client.

```
public Client(int id, int arrivalTime, int serviceTime)
```

Setters are methods that updates value of a variable. Getters are methods that reads value of a variable.

```
public int getId()
public int getArrivalTime()
public void setArrivalTime(int arrivalTime)
public int getServiceTime()
public void setServiceTime(int serviceTime)
public void decrementServiceTime()
@Override
public String toString()
```

Method decrementServiceTime decrement the variable serviceTime by 1.

The toString() method returns the string representation of the Client object.

Ex: id = 2; arrivalTime = 3; serviceTime = 4; result => “(2,3,4)”

4.1.2. QueueThread

QueueThread class implements the Runnable interface because each queue need to be a different thread.

```
public class QueueThread implements Runnable
```

Private variable clientQueue is a queue of clients which makes the object idea.

I used LinkedList object for simplicity and to simulate a queue because I needed in my implementation its properties.

Boolean variable isRunning tell the object if the thread is running or not.

Variable waiting tell us the waiting time until the queue procced the future client.

```
private int queueId;
private boolean isRunning;
private int waiting;
private Queue<Client> clientQueue = new LinkedList<>();
```

The constructor take as parameter the id of the queue and the other parameters are initialize there (waiting = 0; isRunning = false)

```
public QueueThread(int queueId)
```

queueId and waiting have getters.

Method startRunning start a thread and variable isRunning takes value true.

The toString() method returns the string representation of the QueueThread object.

In override method run it will decrement service time of the top of the queue, if this will have service time 0 it will be removed. It stops when has no clients and starts one is at least one.

Method isEmpty verify if it is any clients in the queue.

Ex: “Queue 1: (1,2,3);(2,2,2);”

```
public int getQueueId()
public void startRunning()
@Override
public void run()
@Override
public String toString()
public int getWaiting()
public boolean isEmpty()
public int size()
public void addClient(Client c)
```

4.1.3. QueueList

```
public class QueueList
```

queueList is an array list of all queues. I decided to put queues in a list because it is much easier to handle.

```
private List<QueueThread> queueList = new ArrayList<>();
```

This constructor takes as parameter size and adds in queueList size unique queues.

```
public QueueList (int size)
```

Method addClient add client in queueList such as the client is add in the queue with the smallest waiting time.

The toString() method returns the string representation of the QueueList object that means all queues represantation.

Method isEmpty verify if it is any clients in any queues.

Ex “Queue 1: (1,2,3);(2,2,2);

” Queue 2: (3,2,3);

```
public void addClient(Client c)
public boolean isEmpty()
public int clientsInQueues()
@Override
public String toString()
```

4.1.4.Timer

Timer class implements the Runnable interface because helps keeping time of the simulation.

```
public class Timer implements Runnable
```

time is actual time.

maxTime is the finishing time of the simulation.

```
private int time;
private int maxTime;
```

Constructor with one parameter and initialize time with 0;

```
public Timer(int maxTime)
```

Method startTimer start a new thread.

Method stopTimer stop the thread by exiting the run method with maxTime = -1.

run() method increments the time by one each second until reaches maxTime. All the other components of the application work according to this timer.

The toString() method returns the string representation of the Timer object.

```
public void startTimer()
public void stopTimer()
```

```
@Override
public String toString()
@Override
public void run()
```

4.2. view (OutputWriter)

OutputWriter class implements the Runnable interface because it represents a thread that is a global timer of the application

```
public class OutputWriter implements Runnable
```

```
private String outFile;
private Timer clock;
private ClientGenerator allClient;
private QueueList qList;
private int numberOfClients;
```

Constructor initialize all fields of the class.

```
public OutputWriter(String outFile, Timer clock, ClientGenerator allClient, QueueList qList, int numberOfClients)
```

run method is running until the timer ends. Here is display the output in outFile file and also here is connected allClient and qList. In this method is computed and displayed average waiting time by adding every step return of the qList.clientsInQueues and in the end divide by numberOfClients.

```
@Override
public void run()
```

4.3. controller

4.3.1. ParseInput

This class is made for parsing a input file into data and validate the parsing and throwing exceptions when is not valid.

```
public class ParseInput
```

```
private int numberOfClients;
private int numberOfQueues;
private int simulationInterval;
private int minArrivalTime;
private int maxArrivalTime;
private int minServiceTime;
private int maxServiceTime;
```

In constructor is done reading of the file and transfer them to fields of the class.

```
public ParseInput(String fileName)
```

Here we have all geters for each field.

```
public int getNumberOfClients()
public int getNumberOfQueues()
public int getSimulationInterval()
public int getMinArrivalTime()
public int getMaxArrivalTime()
public int getMinServiceTime()
public int getMaxServiceTime()
```

4.3.2. ClientGenerator

ClientGenerator class combines generating random clients with storing clients and handle a client list.

```
public class ClientGenerator
```

```
private List<Client> clientList = new ArrayList<Client>();
```

Constructor takes procced data from input and generate random numberOfClients clients adding them to clientList

```
public ClientGenerator(ParseInput data)
```

clientList has getter.

Method getFirst return first element in the clientList.

Method extractFirst same as getFirst but also removes first element.

Methods sortByID, sortByArrivalTime, sortByServiceTime sort clientList by a specific field of the class Client

by using : clientList.sort(Comparator.comparing(Client::getArrivalTime));

The toString() method returns the string representation of the ClientGenerator object.

Ex: “waiting clients:(2,3,4); (3,3,4); (4,3,4)”

```
public List<Client> getClientList()
public Client getFirst()
public Client extractFirst()
public void sortByID()
public void sortByArrivalTime()
public void sortByServiceTime()
public boolean isEmpty()
@Override
public String toString()
```

5. Results

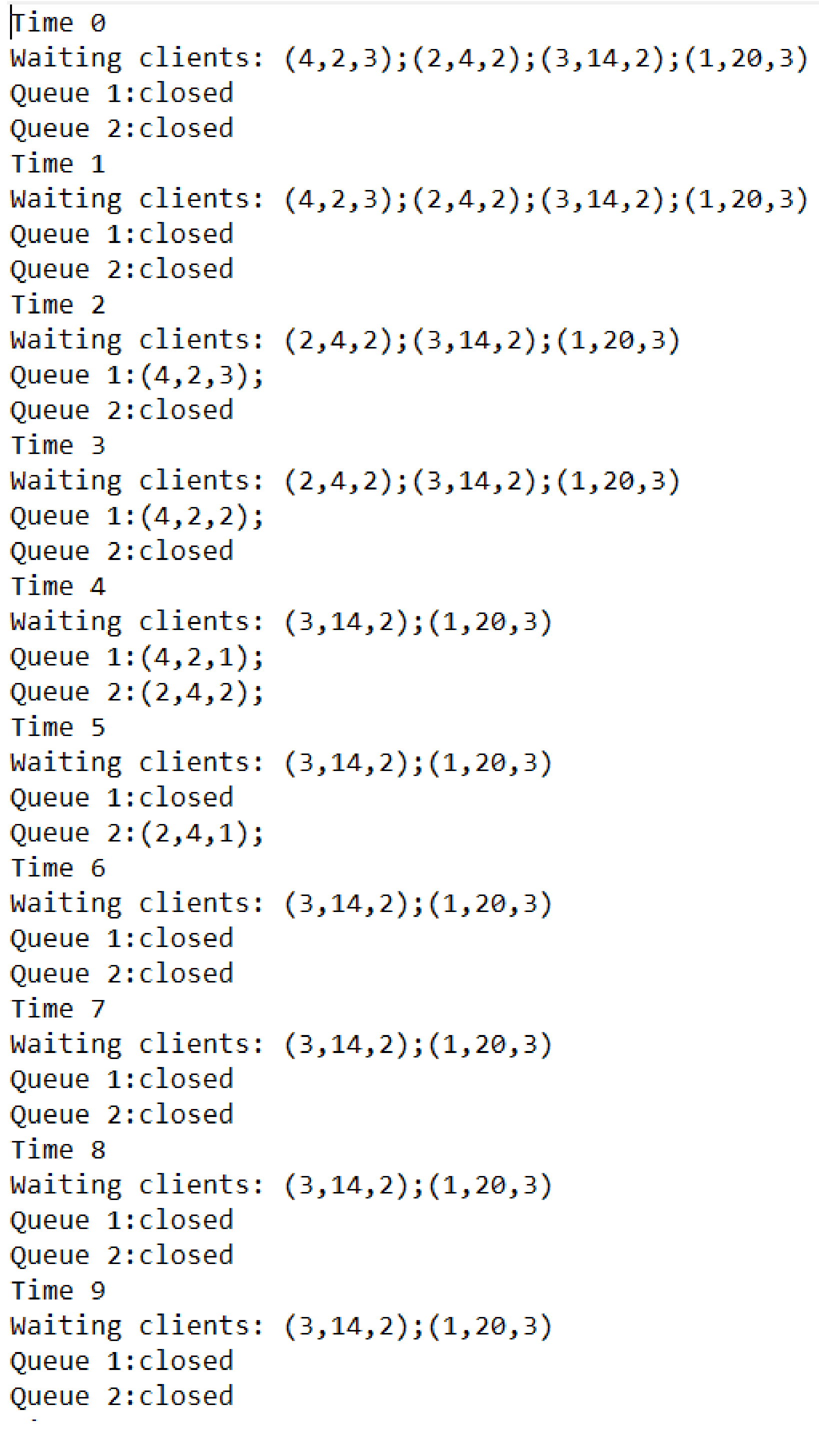
5.1. System testing

System testing is a level of testing that validates the complete and fully integrated software product.

I tested on my own and analyses the response of main scenario and secondary scenarios.

Main scenario (success scenario):

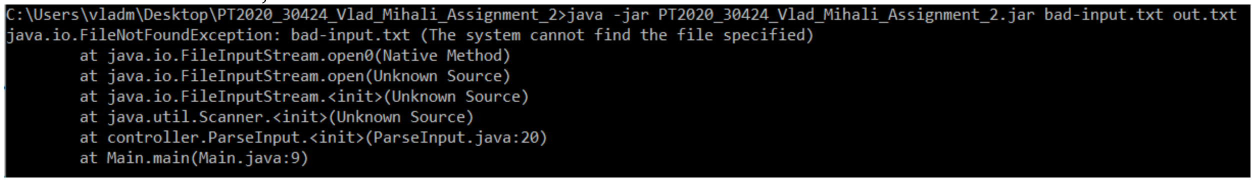
- Introducing valid input file;
- Wait to end the simulation;
- Verify the result by simulate your own next step and then checking the real simulation data;



```
Time 0
Waiting clients: (4,2,3);(2,4,2);(3,14,2);(1,20,3)
Queue 1:closed
Queue 2:closed
Time 1
Waiting clients: (4,2,3);(2,4,2);(3,14,2);(1,20,3)
Queue 1:closed
Queue 2:closed
Time 2
Waiting clients: (2,4,2);(3,14,2);(1,20,3)
Queue 1:(4,2,3);
Queue 2:closed
Time 3
Waiting clients: (2,4,2);(3,14,2);(1,20,3)
Queue 1:(4,2,2);
Queue 2:closed
Time 4
Waiting clients: (3,14,2);(1,20,3)
Queue 1:(4,2,1);
Queue 2:(2,4,2);
Time 5
Waiting clients: (3,14,2);(1,20,3)
Queue 1:closed
Queue 2:(2,4,1);
Time 6
Waiting clients: (3,14,2);(1,20,3)
Queue 1:closed
Queue 2:closed
Time 7
Waiting clients: (3,14,2);(1,20,3)
Queue 1:closed
Queue 2:closed
Time 8
Waiting clients: (3,14,2);(1,20,3)
Queue 1:closed
Queue 2:closed
Time 9
Waiting clients: (3,14,2);(1,20,3)
Queue 1:closed
Queue 2:closed
```

Failure scenario:

- Not a valid data;
- Not a valid file;



```
C:\Users\vladm\Desktop\PT2020_30424_Vlad_Mihali_Assignment_2>java -jar PT2020_30424_Vlad_Mihali_Assignment_2.jar bad-input.txt out.txt
java.io.FileNotFoundException: bad-input.txt (The system cannot find the file specified)
        at java.io.FileInputStream.open0(Native Method)
        at java.io.FileInputStream.open(Unknown Source)
        at java.io.FileInputStream.<init>(Unknown Source)
        at java.util.Scanner.<init>(Unknown Source)
        at controller.ParseInput.<init>(ParseInput.java:20)
        at Main.main(Main.java:9)
```

I tested all three input files given and verify multiple times the correctness of the simulation.

6. Conclusions

In conclusion, results of this paper work show fulfilling of the main objective and secondary goals. This application does simulate service queues for clients. This assignment approached the following items: java naming conventions; object-oriented programming design; Random Client Generator; Multithreading (one thread per queue); Three test run and saved: in-test-1.txt, in-test-2.txt, in-test-3.txt; Appropriate synchronized data structures to assure thread safety; Queues open/close dynamically. Initially all queues are closed. When clients are distributed to the queues, they become open as needed. When a queue becomes empty, it is closed, and the corresponding thread is paused; Compute average waiting time.[4]

For future development of this application can be improved in following points: an graphical user interface can be done to see simulation flow in real time; more complex data can be add in clients and service can be also more complex.

7. References

[1] https://www.visual-paradigm.com

[2] http://users.utcluj.ro/~cviorica/PT2020/

[3] https://www.quora.com/What-is-encapsulation-in-Java

[4] http://coned.utcluj.ro/~salomie/PT_Lic/4_Lab/Assignment_2/Assignment_2.pdf

[5] http://docs.oracle.com/javase/tutorial/essential/concurrency/index.html

[6] http://www.tutorialspoint.com/java/util/timer_schedule_period.htm

[7] http://www.javacodegeeks.com/2013/01/java-thread-pool-example-using-executors-andthreadpoolexecutor.html