Queue Simulator

Documentation

**Vlad Lujerdeanu**

Group 30226 | Year 2 | Semester 2

# Contents

1. [The Objective](#_The_Objective) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 3
2. [Analysis, Modeling and Use Cases](#_Analysis,_Modeling,_Scenarios) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 3
3. [Designing Phase](#_DesignING_Phase) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 4
4. [Implementation](#_Implementation) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 7
5. [Results](#_Results) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 9
6. [Conclusion](#_Conclusions) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 10
7. [References](#_References) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 10

# The Objective

The Objective of this project is to design and implement a Queue Simulator. Specifically, a simulator aimed for analyzing queuing based systems for determining and minimizing clients’ waiting time. The simulator is implemented using multi-threading.

The design and implementation process of this project was as follows:

* Designing and implementing the Client class with its methods, including a static random client generator.
* Designing and implementing the Queue class which implements Runnable so it can be used as object for the Queue Threads.
* Designing and implementing the SimulationManager class which also implements Runnable, this one being the main thread which gives commands for starting all the other threads.
* Designing the Scheduler class, a class which is called by the Simulation Manager to:
  + Start all the Queue Threads
  + Add clients to Queue based on a predefined strategy, described in Strategy Interface and implemented by ShortestQueueStrategy class and ShortestTimeStrategy class.
  + Compute the Average Waiting Time of a client in queue.
* Describing the Strategy interface and implementing the ShortestQueueStrategy and ShortestTimeStrategy classes.
* Implementing the Scheduler class based on the previously created design.
* Implementing the MainClass.
* Building the project as a jar file.
* Testing the complete project and solving the bugs.

# [Analysis](#_Analysis), [Modeling](#_Modeling) and [Use Cases](#_Use_Cases)

### Analysis

A queue is a list of data items, commands, etc., stored so as to be retrievable in a definite order, usually the order of insertion. Queues are commonly used to model real world applications such as in transportation, retail and banking. In computer science, queues can be used for scheduling, synchronization and as buffers.

The operations of a queue are adding an element to the rear, also known as enqueue, removing the first element, known as dequeue, and peek or front which returns the value of the first element in the queue.

### Modeling

The application will be receiving data from an input text file given as the first argument when running the jar file. The correct file input is as follows:

<Number of clients>

<Number of queues>

<Maximum simulation time>

<Minimum Arrival Time>,<Maximum Arrival Time>

<Minimum Service Time>,<Maximum Service Time>

Each client is modeled after the Client class and the Queues are implemented using the Queue class. The application computes the state of each queue, a list of clients waiting to enter the queue and for each queue a list of clients that are already the queue.

After the computation is finished, the program outputs the information in a file given as the second argument when running the jar file.

### Use Cases

* For Operations between two polynomials (Default):
  1. Running in terminal the command:

java -jar Queue\_Simulation.jar <input File> <Output File> [Time]

Note: Argument “Time” is optional.

* 1. The output will be written at the path mentioned in the “Output File” argument.
* For Obtaining result based on Shortest Queue:
  1. Running in terminal the command:

java -jar Queue\_Simulation.jar <Input File> <Output File> Queue

* 1. The output will be written at the path mentioned in the “Output File” argument.

# DesignING Phase

### Design Decisions

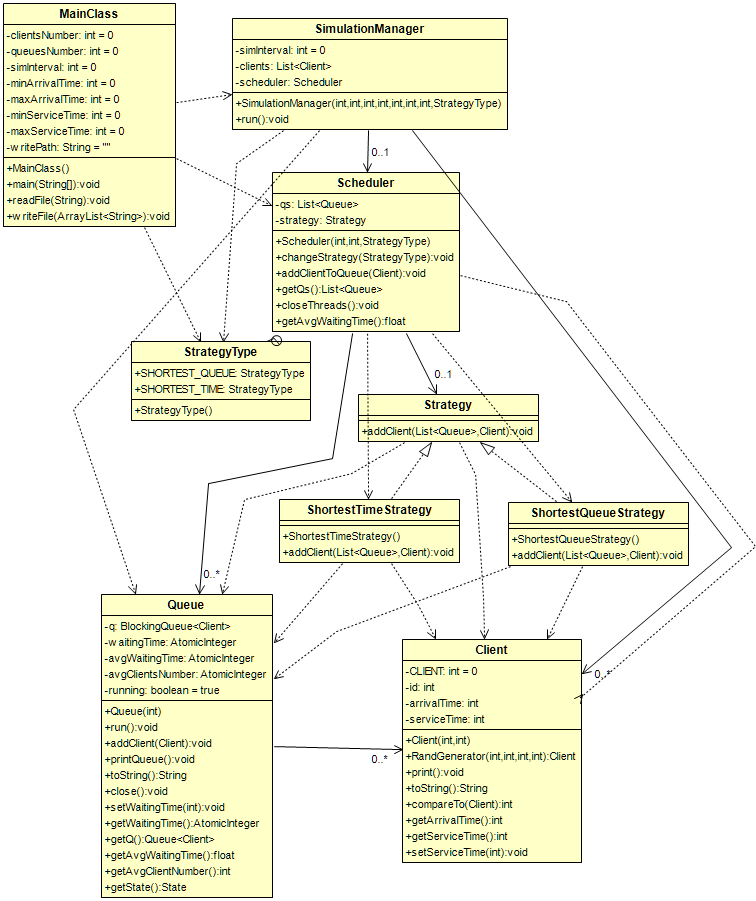
Designing process for this project was a top-down one, and the implementation was bottom-up, firstly implementing the Client and Queue classes, then creating the Strategy interface and implementing it in two ways then adding the Scheduler and implementing all the functionalities in the main thread, namely the Simulation Manager.

Firstly, the application expects the input to be a text file given as the first argument when running it and the results are given as a text file with the name given by the user as the second argument.

Another design decision is following an architecture based on a main thread called Simulation Manager which gives signals to a scheduler which distributes the clients to its corresponding queue based on a predefined strategy.

A very important design decision is implementing the Queue class with Atomic Integers and Array Blocking Queues for synchronizing information between threads.

### UML Diagram



### Class Design

#### Client:

It has a static int variable called client which stores the number of clients, and three other variables which store for each client its ID, Arrival Time and Service Time. Arrival Time and Service Time have getters and setters.

The Random Client Generator is also implemented in Client class.

The class has an implementation for toString method to be given as argument to the printing method.

CompareTo is Overridden so that the Arraylist of clients can be sorted.

It also has a print function for easy debugging.

#### QUEUE:

The class is implemented using an Array Blocking Queue, three Atomic Integers which store information for waiting time, average waiting time, average client number and a Boolean variable called running which is true by default and becomes false when the simulation interval is finished. There are getters and setters implemented for waiting time.

This class implement Runnable and it is used as object when initializing new threads.

The implementation of Runnable, makes the thread active only if there are items in queue, in which case it processes the clients in queue and removes them one by one. If there are no items in queue the queue is put on hold.

The method addClient, adds a client to the queue and if it’s on hold, it starts again the thread.

The class has an implementation for toString method to be given as argument to the printing method.

It also has a print function for easy debugging in the console.

#### Simulation Manager:

The variables used in this class are an integer for storing the upper cap of the simulation interval, a list of clients which are generated randomly and an object of type Scheduler, the object whose job is to distribute the clients to the appropriate queues based on the preferred strategy.

Simulation Manager implements the Runnable interface. This object is used for initializing the main thread in which all the clients are sent to the scheduler for distributing.

This object also creates an Array List of Strings that at the end is sent to the file writing method in the Main Class.

#### Scheduler:

The implementation for the scheduler contains two variables, one of which is a list of queues and the other one is an object of type Strategy.

The scheduler’s constructor creates a synchronized list of Queue objects and starts threads for each of them.

This objects job is to distribute the clients in the optimal queue based on different criteria defined in the Strategy object below.

Other than adding a client to the appropriate queue, it also implements three other methods:

* + - Change Strategy – Changing the criteria by which the clients are distributed.
    - Close Threads – At the end of the simulation, the scheduler closes the queue threads.
    - Get Average Waiting Time – Computes the average waiting time of the clients.

#### Strategy:

This class is an interface which describes a single method called addClient.

The interface is implemented by two classes: Shortest Queue Strategy and Shortest Time Strategy.

* Shortest QueueS trategy: adds a client to the queue with the least clients.
* Shortest Time Strategy: adds a client to the queue with the shortest waiting time.

# Implementation

#### Client:

public class Client implements Comparable<Client> {  
 private static int *CLIENT* = 0; // global client id  
 private int id;  
 private int arrivalTime;  
 private int serviceTime; // time needed for the client to be served

/// Class Constructor  
 public Client(int newArrivalTime, int newServiceTime) { ... }

/// Generates and returns a new random client  
 public static Client RandGenerator(int minArrivalTime, int maxArrivalTime,

int minServiceTime, int maxServiceTime) { ... }

/// Print function for debugging  
 public void print() { ... }  
  
 @Override  
 public String toString() { ... }  
  
 public int compareTo(Client c) { ... }  
  
 /// Getters and Setters  
 public int getArrivalTime() { ... }

public int getServiceTime() { ... }

public void setServiceTime(int serviceTime) { ... }  
}

#### Queue:

public class Queue implements Runnable {  
 private BlockingQueue<Client> q;  
 private AtomicInteger waitingTime;  
 private AtomicInteger avgWaitingTime;  
 private AtomicInteger avgClientsNumber;  
 private boolean running = true;  
  
 /// Class Constructor  
 public Queue(int clientsNumber) { ... }  
  
 /// Implementation of Runnable  
 public void run() { ... }  
  
 /// Add a client to the queue  
 public void addClient(Client c) { ... }  
  
 /// Print function for debugging  
 public void printQueue() { ... }  
  
 public String toString() { ... }  
  
 /// Method to stop the thread  
 public void close() { ... }  
  
 /// Getters and Setters  
 public void setWaitingTime(int waitingTime) { ... }

public AtomicInteger getWaitingTime() { ... }  
  
 public java.util.Queue<Client> getQ() { ... }  
  
 public float getAvgWaitingTime() { ... }  
  
 public int getAvgClientNumber() { ... }  
  
 public State getState() { ... }  
}

#### Simulation Manager:

public class Scheduler {  
 private List<Queue> qs;  
 private Strategy strategy;  
  
 public enum StrategyType {*SHORTEST\_QUEUE*, *SHORTEST\_TIME*}  
  
 /// Class Constructor  
 public Scheduler(int maxQueues, int maxClientsPerQueue,

StrategyType strategyType) { ... }  
  
 /// Method to change between strategies  
 public void changeStrategy(StrategyType newStrategy) { ... }  
  
 /// Method that sends the client to be added to the appropriate queue  
 public void addClientToQueue(Client c) { ... }  
  
 public List<Queue> getQs() { ... }  
  
 /// Method that closes all the opened Queue Threads.  
 public void closeThreads() { ... }

/// Method that computes the average waiting time for all the queues  
 public float getAvgWaitingTime() { ... }

}

#### Strategy:

public interface Strategy {  
   
 /// Adds a client based on the preferred implementation  
 void addClient(List<Queue> qs, Client c);  
}

# Results

The project has been tested using “manually” by testing first each classes’ functionalities. The completed project has been tested using different input test files and checking the output files for logical errors.

# Conclusions

To conclude, the project presented above taught me a lot about analyzing a problem and designing a solution to fit a set of strict requirements and also how work with threads and build an application with multi-threading, but most importantly, how to search the web for tutorials and help when I’m stuck on a problem.

# References

1. [Laboratory Guide – Ionel Giosan](http://users.utcluj.ro/~igiosan/teaching_poo.html) ( UML Generating in eclipse )
2. [Java Concurrency](http://coned.utcluj.ro/~salomie/PT_Lic/4_Lab/Assignment_2/Java_Concurrency.pdf) ( Simulation Manager Architecture )
3. [Stackoverflow.com](https://Stackoverflow.com) ( Minor problems with the code )
4. [Wikipedia – Queue (abstract data type)](https://en.wikipedia.org/wiki/Queue_(abstract_data_type)) ( Description of the data type for the documentation )