Documentation

Fundamental Programming Techniques

Queue system

Assignment 2

Dávid Soós

group 3

Table of contents

1. Assignment objective

The application had to model the way servers work, by making use of threads and some data structures that ensure thread-safety. The task was to design and implement a queues management application that assigns clients to queues in a way that minimizes waiting time. The application should simulate a series of N clients arriving for service, entering Q queues, waiting, being served, and finally leaving the queues. Each client has an ID, a ready time when they can enter the queue, and a service time needed to serve them. The application should track the total time spent by every client in the queues and compute the average waiting time. The user should provide input data, including the number of clients, number of queues, simulation interval, minimum and maximum arrival time, and minimum and maximum service time. The application should add each client to the queue with the minimum waiting time when its ready time is greater than or equal to the simulation time.

2. Problem Analysis, Modeling, Scenarios, Use Cases

The first problem one can encounter when searching for edge cases or inconsistencies is the problem of input processing. The user can enter whatever information they want and when the typed in data does not adhere to the rules, it should be signaled towards them. This is crucial because further operations depend on the fact that the data has been correctly given. To combat this issue, I introduced a validator class and a custom-made exception, namely WrongInputException.

There is a Graphical User Interface that allows the user to type in the data: the number of clients, the number of queues, the simulation interval, the minimum arrival time, the maximum arrival time, the minimum service time and the maximum service time.

By pressing the START button, the user may start the simulation, which is displayed in real time and can be followed on the text area. In case it overflows, there is a scrollbar that only appears in such situations and makes the whole process viewable.

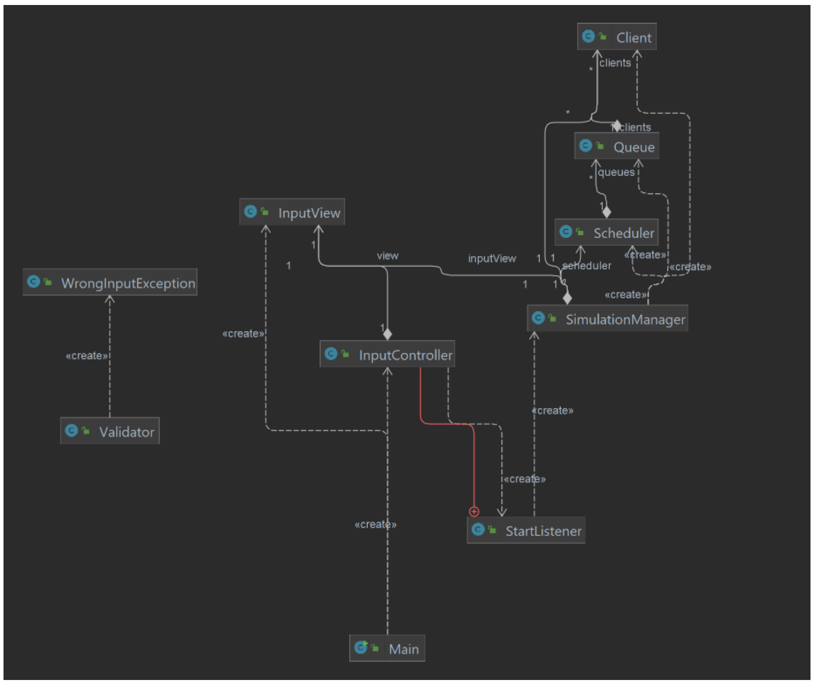
Besides getting the information in the Graphical User Interface, the whole of the process is logged in the .txt file named output.

There are done some mathematical calculations at the end of the program that show the average waiting time, the average processing time and peak time, when the queues had been the busiest. Under waiting time, I have considered the time it took inside a queue, after arrival for a client to get to the front and start the processing phase. By average processing time I mean the average of working time units amongst the servers.

The project has a variety of real-world applications such as helping businesses keep track of orders or assisting a team of co-workers on partitioning the workload, and in the same time increasing productivity.

Design

The project uses MVC architecture which stands for Model, View and Controller. This structure enables the separation of unrelated tasks a program should handle. The views class holds Graphical User Iterface related data, the models package handles the classes the operations are processed with, and controllers class performs the instructions that are given by the user, by making use of the models.



Models

In the Client class, we define a client object that has four attributes: id, arrivalTime, processingTime, and waitingTime. The id attribute is an auto-generated unique identifier for each client. The arrivalTime attribute represents the time at which the client enters the simulation, and the processingTime attribute represents the time it takes for the client to be serviced by the queue.

The Queue class can accept clients and process them. It has six attributes: id, clients, waitingTime, totalWaitingTime, totalServiceTime, and clientsProcessed. The id attribute is an auto-generated unique identifier for each queue.

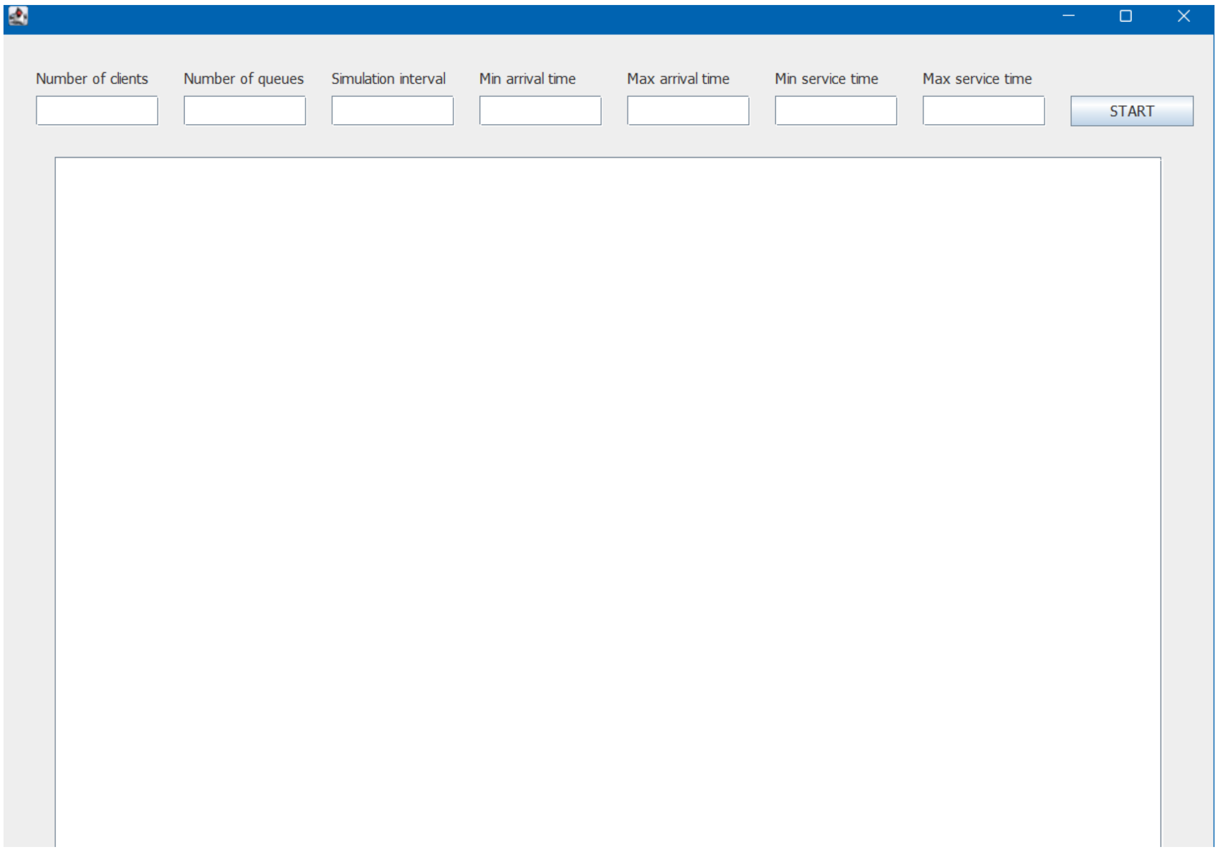
The Scheduler class is responsible for managing multiple queues in the simulation. It has three attributes: queues, maxNumberOfQueues, and time. The queues attribute is a List of Queue objects, representing the queues in the simulation. The maxNumberOfQueues attribute is an Integer that represents the maximum number of queues that can be active in the simulation at any given time. The time attribute is an Integer that represents the current time in the simulation.

The Validator class takes in several input parameters and throws a custom exception named WrongInputException if the input is invalid.

These five classes work together in order to run and maintain the simulation, being able to handle between each other the processing of clients amongst queues according to a pre-defined strategy. Also they are able to communicate with the view part of the program and in the same time log what has happened in text file.

View

There is only one class in the views package, namely InputView. It collects all the Graphical User Interface related objects and some of their informations. Here are embedded the listeners and this class makes sure that the user receives a customized pop-up when they do not adhere to the input related rules.



Controller

The controllers package also has only one, the InputController class. Here is the main thread started at the push of the START button. Of course, only after processing and validating the user defined input data. The acquired data is made sure that is inconsistency-free (minimum < maximum and there is at least one queue / client).

3. Implementation

Once the button is clicked, the StartListener reads the user input from the Graphical User Interface, validates it using the Validator class, creates a SimulationManager object with the validated input parameters, and creates a new thread to run the simulation. The output of the simulation is redirected to a file named "output.txt" or to one of the test files if we want to rewrite the initial tests, using a PrintStream object. If there is an exception while printing to the file or if the input is invalid, an appropriate error message is displayed in the GUI using the InputView object.

The generateRandomClients() method creates a list of Client objects with random arrival and service times within the specified ranges. For this amount of objects (there is no upper bound to how many clients to be generated), I needed to sort the clients' list. The createQueues() method creates a list of Queue objects based on the specified number of queues.

The SimulationManager class implements the Runnable interface, which means that it can be executed in a separate thread. In the run() method of the simulation is executed in a loop. Inside this loop, there is a call to Thread.sleep(1000) which causes the thread to pause for one second before continuing. This is done to simulate the passage of time in the simulation.The loop continues to execute until the simulation interval has elapsed or until there are no more clients left to process and all the queues are empty. During each iteration of the loop, the scheduler.dispatchClients(clients) method is called to try to assign clients to the available queues, based on their arrival time and the availability of the queues. After each iteration of the loop, the state of the simulation is updated and printed to the console using inputView.setSituation() and System.out.println(). The inputView.setSituation() method is used to update the Graphical User Interface with the latest state of the simulation. Finally, once the simulation has ended, the results are printed to the console and the queues are stopped, using scheduler.stopQueues().

showWaitingClients() method returns a string representation of the waiting clients in the simulation and with the help of scheduler.showQueues() and the time getter from the scheduler class displays the current situation of the simulation.

The Scheduler class also has several methods for managing the queues. The showQueues() method returns a String representation of each queue and its clients. The dispatchClients() method takes a List of Client objects and adds them to the queue with the shortest processing time. The findMinQueueIndex() method returns the index of the queue with the shortest processing time. The queuesEmpty() method checks if all queues are empty. The stopQueues() method stops all running queues. The clientsInQueues() method returns the total number of clients in all queues. Finally, there are two methods to calculate the average waiting time and average service time across all queues. This component is basically the brain behind the simulation, here are made the most of the decisions.

When the Queue thread is started, it enters an infinite loop and begins processing clients from its queue. It does this by dequeuing the first client from the waiting queue and sleeping for one second, simulating the time it takes to service the client. It then decrements the processing time for the client and increments the totalServiceTime counter. If the client's processing time is zero, it increments the clientsProcessed counter and removes the client from the queue. If there are still clients in the queue after servicing the first client, it updates the totalWaitingTime counter to reflect the total waiting time for the remaining clients. We need the amount of clients processed and totality the time waited by them to calculate the average waiting time.

The Queue class implements the Runnable interface, which means it can be used as a Runnable object in a new thread. When the start method of the Queue class is called, it sets the running flag to true and creates a new Thread object with the current Queue object as the Runnable. The Thread object then starts executing the run method of the Queue object in a new thread. The class contains several instance variables, including an ID, a BlockingQueue of Client objects, and several AtomicInteger and AtomicBoolean objects that are used to track various metrics related to the queue. Overall, the data structures were chosen for their thread-safety and ability to handle concurrent access in a multi-threaded environment.

The addNewClient method is responsible for adding a new client to the queue of clients to be processed by the server. The method first checks if the server is not currently running, and if not, it starts the server by calling the start method. Then, it adds the new client to the clients queue, which is a thread-safe data structure used to store the clients. The synchronized keyword before the method declaration ensures that only one thread can execute this method at a time. This is important in a multithreaded environment, where multiple threads may attempt to add clients to the queue simultaneously. By synchronizing the method, the server can guarantee that only one thread will be modifying the queue at any given time, preventing any potential race conditions and ensuring thread safety. The clientsProcessed and waitingTime variables are also updated atomically using the getAndIncrement and addAndGet methods, respectively, provided by the AtomicInteger class. This guarantees that these operations are thread-safe and prevent any concurrency issues.

In order to stop the execution of the Queue the running flag is set to false, which causes the run method to exit its infinite loop and stop processing clients.

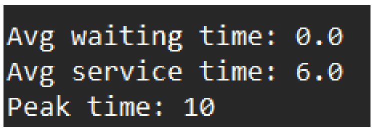
The smallest building block of the simulation is the task, here named the client. The id variable of every Client is a unique identifier and is generated by the Validator.generateClientId() method.

The Client class implements the Comparable interface and defines a compareTo method to allow for comparison between two Client objects. The comparison is first based on the arrivalTime of the clients, and if they have the same arrivalTime, then the comparison is based on their processingTime.

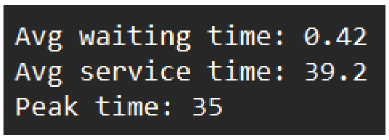
4. Results

All of the the requirements have been fulfilled.

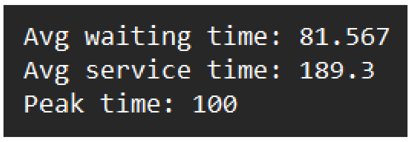
The three files containing the test scenarios can be found in the githhub repository. The first two of them have finished before the given time,



```
Avg waiting time: 0.0
Avg service time: 6.0
Peak time: 10
```



```
Avg waiting time: 0.42
Avg service time: 39.2
Peak time: 35
```



```
Avg waiting time: 81.567
Avg service time: 189.3
Peak time: 100
```

5. Conclusions

There is plenty of room for improvement. For example, the simulation can only be run once at the moment.

I have learned a great deal about threads. When I beginned the project I had limited amount of knowledge about them, so a really big part of the required work was brand new to me.

6. Bibliography

https://www.visual-paradigm.com/guide/uml-unified-modeling-language/uml-class-diagram-tutorial/

https://www.w3schools.com/java/java_threads.asp

https://www.programiz.com/java-programming/printstream

https://www.geeksforgeeks.org/java-threads/