UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

PROGRAMMING TECHNIQUES

QUEUE MANAGEMENT SYSTEM

STUDENT: ELEKES PÉTER
UNIVERSITY: UNIVERSITY OF TECHNICAL ENGINEERING CLUJ-NAPOCA
FACULTY: COMPUTER SCIENCE
YEAR: II.
GROUP: 30422

Table of Contents

Assignment Task

Design and implement a queues management application which assigns clients to queues such that the waiting time is minimized.

The project should use an object-oriented design(with encapsulation, inheritance, decomposition), should use lists instead of arrays, use foreach instead of for, have a GUI using JavaFX or Swing and every class should have less than 300 lines of code.

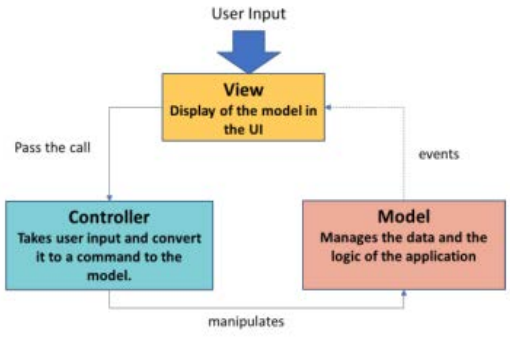
An additional requirement is to use an architectural pattern.

Queues are commonly used to model real world domains. The main objective of a queue is to provide a place for a "client" to wait before receiving a "service". The management of queue-based systems is interested in minimizing the time amount their "clients" are waiting in queues before they are served. One way to minimize the waiting time is to add more servers, i.e., more queues in the system (each queue is considered as having an associated processor) but this approach increases the costs of the service supplier.

As a bonus, i used RegEx to identify the polynomial, and used JUnit for testing each operation.

Requirements	Points
Object-oriented design ✓	minimal requirement
Random Client Generator✓	minimal requirement
Multithreadig: one thread per queue ✓	minimal requirement
Java Swing or JavaFX ✓	minimal requirement
Appropriate synchronized data structures to assure thread safety ✓	minimal requirement
Log of events displayed in a .txt file ✓	Minimal requirement
Graphical user interface for: (1) simulation setup, and (2) displaying the real-time queue evolution.✓	3 points
Display of simulation results (average waiting time, average service time, peak hour for the simulation interval) in the graphical user interface/.txt file corresponding to the log events✓	1 point
Run the application on predefined tests ✓	1 point

Assignment analysis, scenario, approach, and use cases



The first step I took was to create the architectural pattern. I declared the packages neccesary (GUI, BusinessLogic and Model).

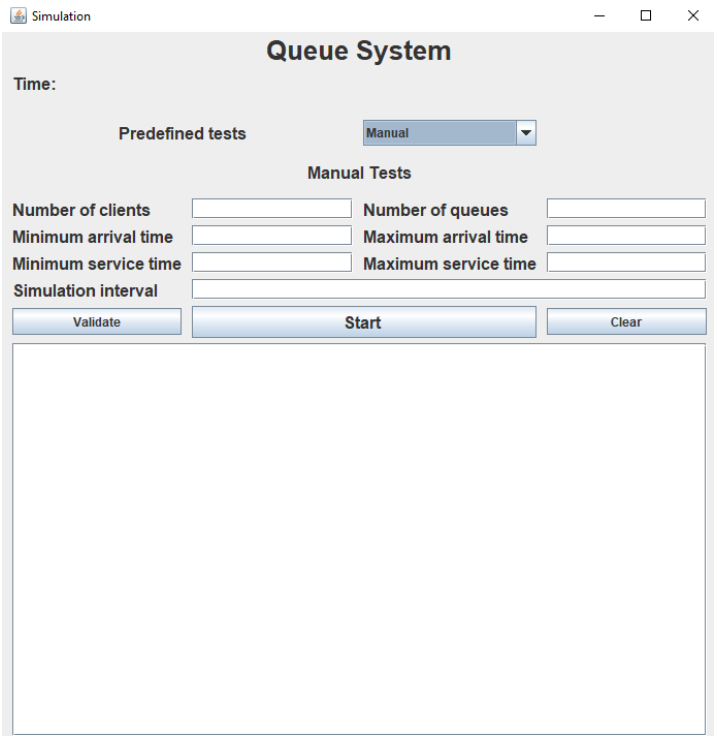
To introduce you to the MVC architecture design I will talk a bit about it. We have to split all the components of the project into 3 main packages: Model, View and Controller. Each of these elements have a seperate role they have to fulfill. The Model has to manage the data and the logic of the application, the View describes the way the Model is displayed in the UI and the Controller takes user input and converts it into a command that is then forwarded to the Model. This creates a circular motion of logic between the packages: the user interfaces with the View, which then passes the call to the Controller, which manipulates the Model, which in turn creates events to pass back to the View. This approach is really efficient for code re-use and parallel development.

The program itself is quite straightforward to use.

I created textfields for all the input data (Number of clients, Number of queues, Minimum arrival time, Maximum arrival time, Minimum service time, Maximum service time and Simulation interval)

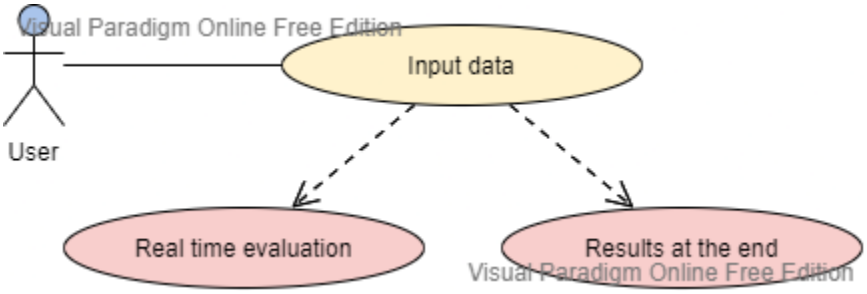
I also added a ComboBox, where the user can choose whether they want to provide their own set of inputs or want to use the predefined tests.

There are 3 predefined tests provided by the Assignment task, each increasing in complexity.

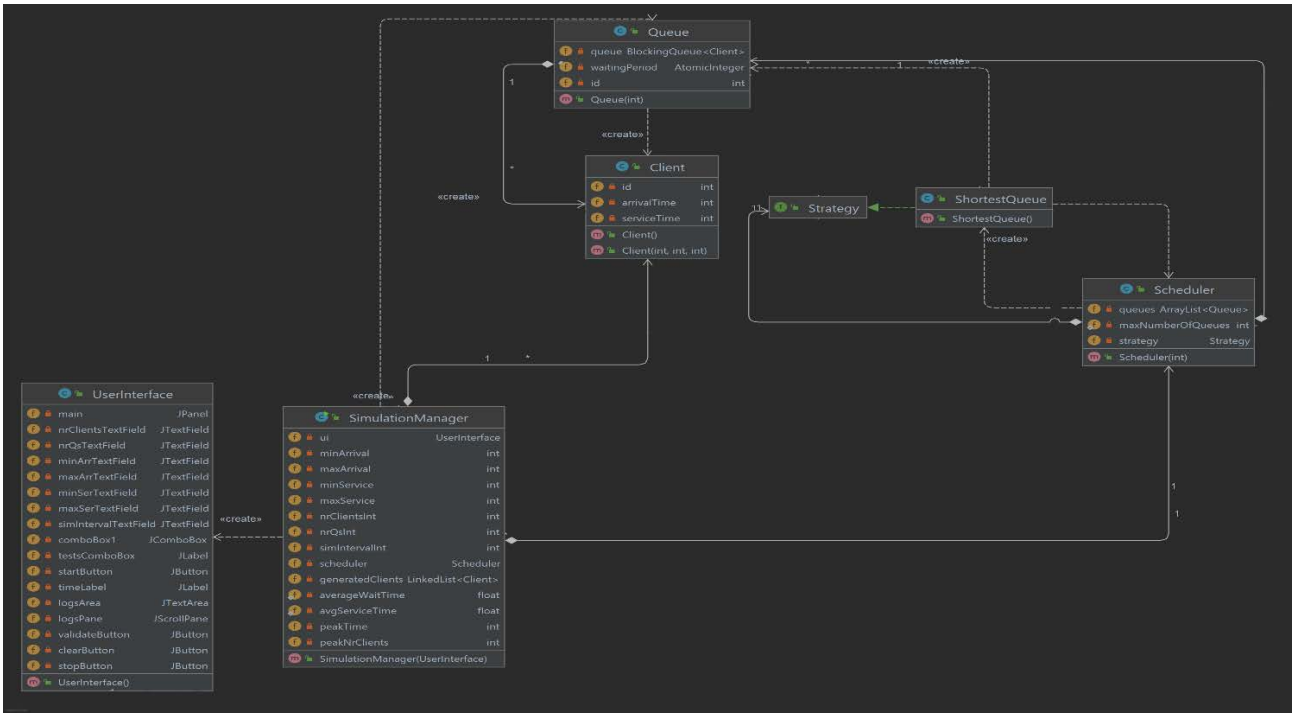


Test 1	Test 2	Test 3
Number of clients = 4	Number of clients = 50	Number of clients = 1000
Number of Queues = 2	Number of Queues = 5	Number of Queues = 20
Simulation interval = 60	Simulation interval = 60	Simulation interval = 200
[MinArrival , MaxArrival] = [2, 30]	[MinArrival , MaxArrival] = [2, 40]	[MinArrival , MaxArrival] = [10, 100]
[MinService, MaxService] = [2, 4]	[MinService, MaxService] = [1, 7]	[MinService, MaxService] = [3, 9]

Use Case Diagram



Projection



The picture you can see is called a Unified Modelling Language (UML) diagram. It is used to visualize, specify, and document the software system. This is an Implementation level view. As you can see, I implemented quite a few classes to ease the workflow. This is a simplified version, to grasp the idea of the interactions between the packages.

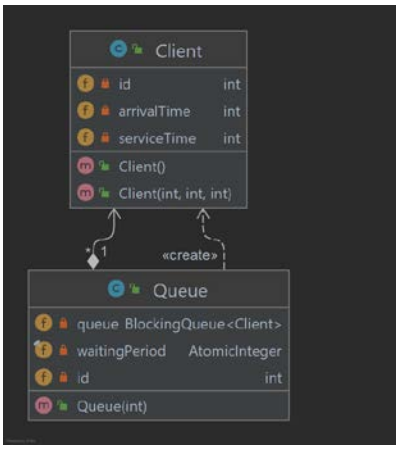
I used some new data structures (to me), like BlockingQueue, AtomicInteger.

The BlockingQueue interface extends the normal Queue. This is a type of Queue that additionally supports operations that wait for the queue to become non-empty when retrieving an element, and wait for space to become available in the queue when storing an element. A BlockngQueue can be capacity bounded, however it was not the case in this assignment.

The AtomicInteger data structure is an Int value that can be updated automatically. The variable that I used replaced a normal int counter.

Implementation

The Model

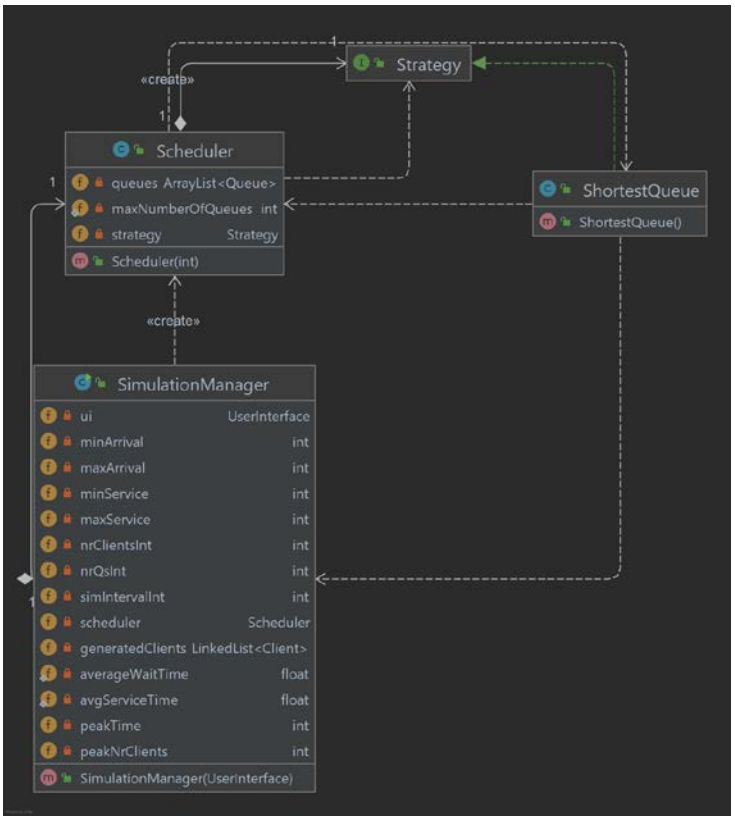


In this package I implemented two Classes, the Client class, and the Queue class. There is aggregation between them because a Queue consists of many Clients.

The Client class has the following properties: an id, which is automatically generated, an arrivalTime generated between the constraints specified by the user, and a serviceTime generated between the constraints specified by the user. In addition to the setters and getters this class has one more method that is useful to the program: It has a decreaseServiceTime method to update the remaining time needed to serve the Client.

The Queue class has a BlockingQueue of Clients, a waitingPeriod specific to the queue and an id to be able to differentiate it. This class also extends the Thread class because we have to synchronize many queues. Additional methods I implemented in this class are: a synchronized addClient method which adds a Client to the queue and decrements the waitingPeriod; a synchronized queueLength method to see the real-time size of the queue; a getClients method to return an array of clients that are in the queue; and the run method. Since this class extends the Thread class it is compulsory to add a run method. All this method does is take the first client in the queue and decrease its service time. If the taken client has a service time of 0 then it will be taken out of the queue.

The Controller (BusinessLogic)



This package consists of 4 components: SimulationManager, Scheduler, Strategy and ShortestQueue.

Let us start with the Strategy interface. I created this interface so that it could be implemented by two different classes, one that inserts the new client into the Queue that has the least number of clients, and one that inserts the new client into the Queue that would be empty in the smallest amount of time. Since the requirements did not state that we must implement both approaches I only implemented the ShortestQueue approach, but this interface allows the program to be further developed.

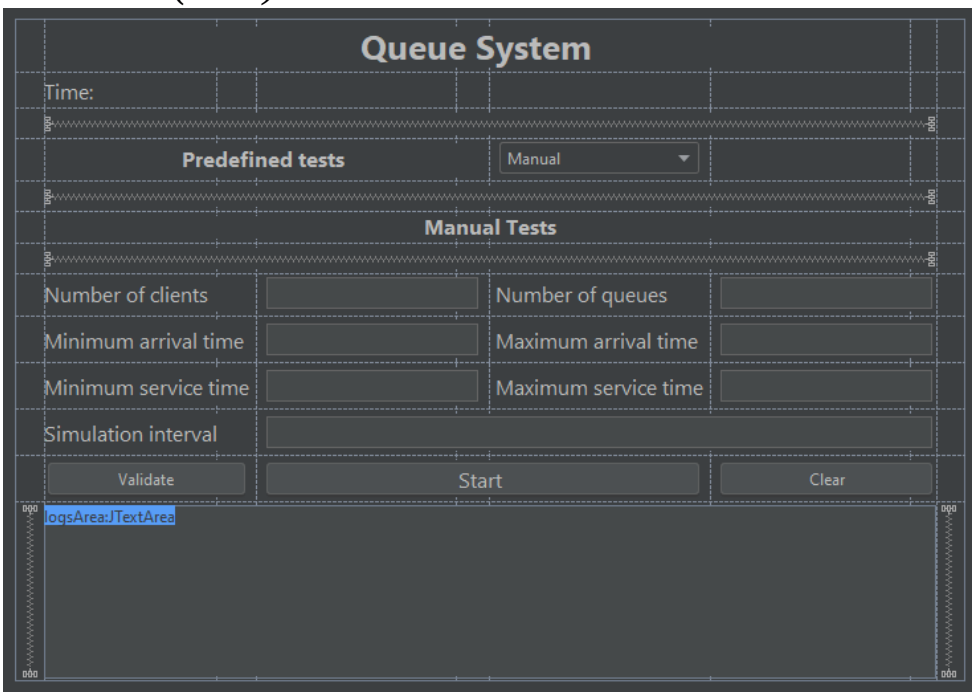
The ShortestQueue class as I stated before implements the Strategy interface and overrides its addClient method. This method gets an ArrayList of Queues and a Client. After that, it goes through all the Queues and checks which Queue has the least number of Clients in it. When it found the Queue with the least number of clients it calls the addClient method from the Queue class, thus completing the task it had to do.

The Scheduler class supervises all the Queues and selects which Strategy will be used when adding new Clients. As its constructor it gets an integer value that specifies how many queues should be generated. In a for loop it goes through all of the queues, gives them an id starting from 1 and starts a new thread for each queue. (as the problem stated, I had to start a new Thread for every Queue)

The SimulationManager class is the main controller of the application, and it also includes the main function. It implements the Runnable interface for synchronization and management of the threads. The main function calls the constructor of the User Interface and implements the ActionListeners of the Validate and Start buttons. When we press the Validate button it extracts all the input we provided in the text fields of the GUI and checks their validity. Possible errors could be: any fields left empty, any Not a Number input, or incorrect minimum and maximum values. If an error is found we get a Pop-Up window stating that the input we provided is not correct, otherwise another Pop-Up window tells us that the data is valid. When we press the Start button it calls the class's constructor with the User Interface we provided and starts the main

thread. The constructor sets all the variables that are used to simulate the system, initializes the Scheduler with the number of queues it must handle, and generates random clients according to the restrictions we provided. Two other simple methods implemented here are addAverageWaitingTime and addAverageServiceTime with both getting an integer value that after can be added to the variable that counts the average values. The peakHour method gets as parameters a list of queues and the time. It goes through all the queues, counting the clients that are in them and check if the number of clients is a maximum. If it is greater than the previous value, it stores the time this occurred with the number of clients. The print function is used to display the real-time evaluation of the queues. It displays the current time, the clients who are not yet in the queues and all of the queues with the clients in them. The generateRandomClients method gets as argument the number of clients it has to generate in addition with the minimum arrival time, maximum arrival time, minimum service time and maximum service time. It goes through a for loop starting from 0 to the number of clients, generates a random arrival time between the minimum and maximum, a random service time between the minimum and maximum and adds the client to the list. After all of the clients were generated, I used Collections.sort to compare each client's arrival time and sorted the list in increasing order. The isRunning method returns a Boolean value that is decided by looking at the clients list. If it still has clients in it the program should still run, returning a true value, and if there are no clients in the list, but there still are some in the queues the program should still run too. If none of these comply then a false value is returned, thus stopping the system. Since we want the simulation to be synchronized this class implements the Runnable interface, as I stated before, and a *run* method should be implemented. This method is the bread and butter of the program. It initializes a time counter to 0 and creates a list of Queues. A loop is started and runs if the isRunning method returns true and the time is smaller than the simulation interval. It takes a client from the client list and with the help of the scheduler the client is added to one of the queues. Each iteration it checks if the current time is the peakHour and prints the real-time status of the queues. After the loop is over all the threads are stopped, and shows the average wait time, average service time and the peak hour with the number of clients at the time.

The View (GUI)



A single class is implemented in this package, and it is the UserInterface. I used IntelliJ's built-in Java Swing UI Designer to create the Graphical User Interface. The layout I used was the GridLayoutManager for ease of managing all the elements. I added lables to all the fields and included a DropBox that can be used to fill in the data for the user. The DropBox has the values Manual, Test1, Test2 and Test 3. I also added a Scroll Pane at the bottom with a Text Area that can not be edited by the user. This Text Area displays the real-time status of all the queues.

In the class itself I implemented three methods, one for the drop box, which fills in all the fields according to the predefined values; one for the Clear button, which clears all the text in the Text Area; and one for updating the logs. The updateLogs method gets a String that is appended to the Text Area and updates the scroll bar.

Possible further development

As I stated before, I deliberately designed the program in a way that any new criteria could be implemented. At this time the program only looks at each Queue's length and adds the Client to the smallest Queue. A new approach could be to compute all the Queues' service time needed and add the client to the Queue that has the lowest value.

Conclusion

This management system can simulate any number of queues and display their real-time evolution. The GUI is easy to understand and the way we have to provide the constraints is self explanatory. The project uses many object-oriented paradigms and is easy to develop and work on in the future.

This assignment proved to be useful in helping me to grasp the idea of Object-Oriented programming, encapsulation, and architectural design.

I loved working with threads and I do believe that I will be able to put the skills I have learned while developing this project to use.

Bibliography

Course slides

Assignment documentation

Support presentation

https://dsrl.eu/courses/pt/

https://www.geeksforgeeks.org/

https://www.youtube.com/

https://stackoverflow.com/