**Queues Simulator Documentation**

1. Assignment objective

The main objective is to design and implement a simulation application aiming to analyze queuing-based systems for determining and minimizing clients’ waiting time. The application should simulate (by defining a simulation time 𝑡𝑠𝑖𝑚𝑢𝑙𝑎𝑡𝑖𝑜𝑛) a series of N clients arriving for service, entering Q queues, waiting, being served and finally leaving the queues. All clients are generated when the simulation is started, and are characterized by three parameters: ID (a number between 1 and N), 𝑡𝑎𝑟𝑟𝑖𝑣𝑎𝑙 (simulation time when they are ready to go to the queue; i.e. time when the client finished shopping) and 𝑡𝑠𝑒𝑟𝑣𝑖𝑐𝑒 (time interval or duration needed to serve the client; i.e. waiting time when the client is in front of the queue). The application tracks the total time spent by every client in the queues and computes the average waiting time. Each client is added to the queue with minimum waiting time when its 𝑡𝑎𝑟𝑟𝑖𝑣𝑎𝑙 time is greater than or equal to the simulation time (𝑡𝑎𝑟𝑟𝑖𝑣𝑎𝑙≥𝑡𝑠𝑖𝑚𝑢𝑙𝑎𝑡𝑖𝑜𝑛).

Secondary objectives:

* A random client generator
* A log of events
* A graphical user interface
* Display simulation results (statistics)

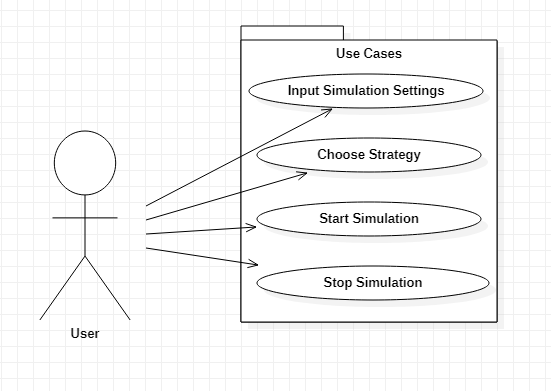
1. Problem analysis
2. Problem analysis

The randomly generated clients will be placed in a waiting queue where they “haven’t arrived yet”. The time will pass each turn and when the arrival time of a client matches the actual time, he will be placed one of the queues, where he will be served over time. The queues must be separate threads and when one has finished serving a client, that client will leave the queue. The clients must be placed in the queues such that their total waiting time is minimum. When all clients have been served or the simulation reaches the end, all threads will be stopped.

1. Modelling of the problem

The user will be able to input a maximum simulation time, a number of queues, a number of clients, a minimum and maximum arrival and service time from a graphical user interface. If there are multiple strategies available, he will also be able to pick one. There will be a start button that starts the simulation and a display area for the log. The clients will be randomly generated with an id, arrival time and serving time chosen between the selected limits and will be displayed entering and leaving the queues, waiting and being processed in real time.

1. Use cases & scenarios



The actor is the user that chooses the simulation settings, strategy and when to start and stop the simulation.

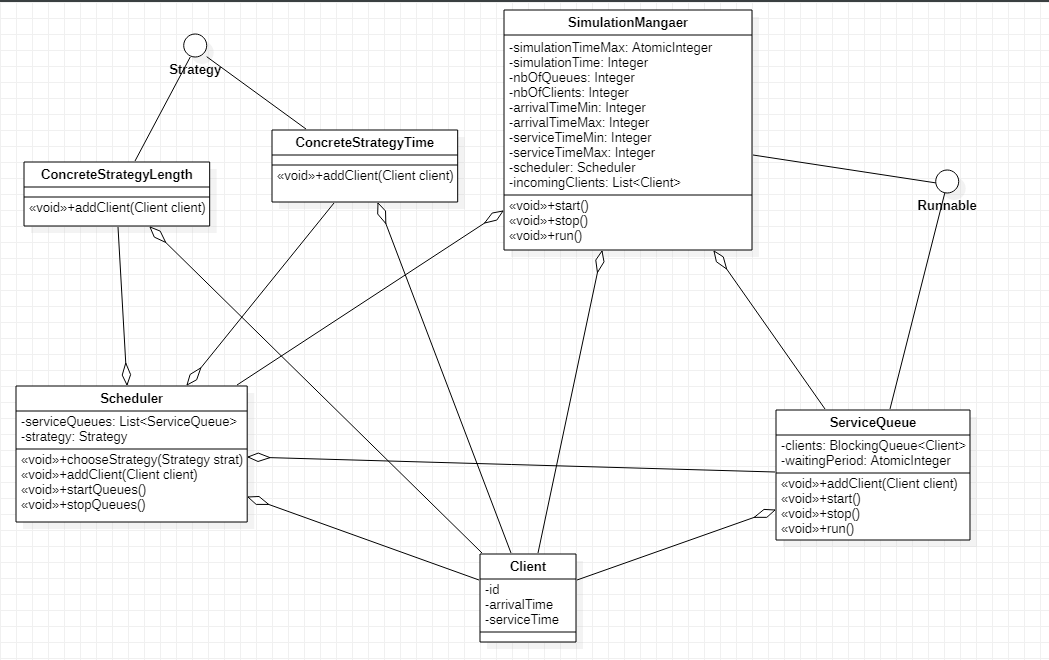
* Input Simulation Settings: All data has to be numeric and positive, otherwise an error message will be shown and the maximum values need to be greater or equal with the minimum values. Also, some values should not be 0, like number of queues and maximum serving time.
* Choose Strategy: One of the provided strategies must be chosen. No way to fail this one with radio buttons (why am I always the only one that uses them, they are so cool)
* Start Simulation: A simple button should do the trick. At this point the input data is checked and the errors caused are displayed, if any where encountered.
* Stop Simulation: If the simulation is running, stop it midway, otherwise fail and show a message.

1. Design
2. Design decisions

There will be textboxes for the data that needs to be introduced, each with its own label and radio buttons for the choosing the strategies. There will also be 2 buttons, one for starting the simulation, and one for stopping it when it is running. I don’t think the strategies should be changed during simulation so disabling them in some way after the start buttons is pressed may be necessary, because radio button shenanigans. Also, there will be a display area, most likely an un-editable text area, where the results of the log will be displayed in real-time. If any errors will occur, because users are monkeys with the intelligence of a pickle, an error window will be displayed, but since I am not that good at GUIs I might just display the error in the log display.

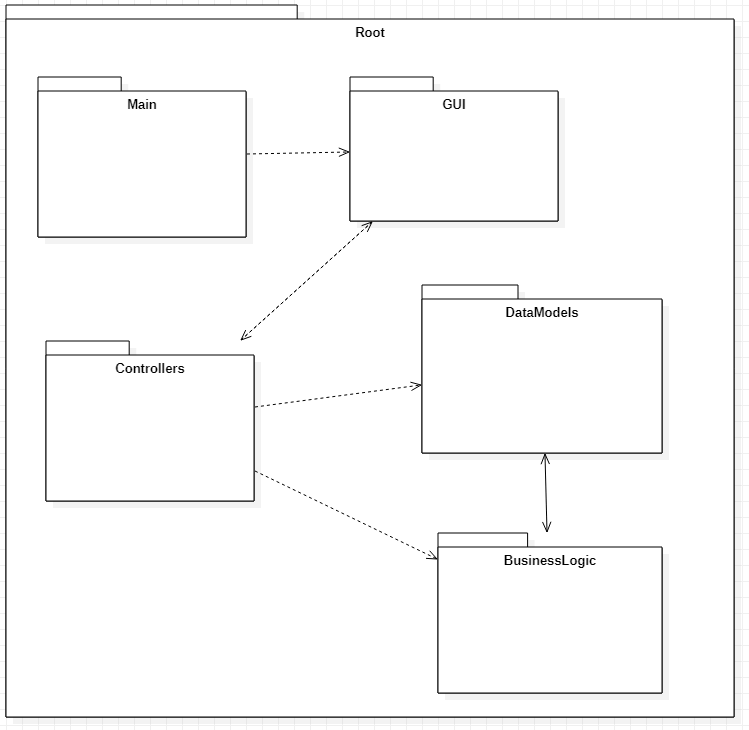
1. Classes

Here is the UML diagram of the classes:



I have already explained the classes a lot in the rest of the documentation so just enjoy the diagram.

1. Packages



As we can see, the project clearly respects the MVC design.

* Main: holds the main function and is used to start up the GUI
* GUI: our View, it holds the interface components and instantiates the controllers
* Controllers: our Controller, which are used to communicate between View and Model
* DataModels: part of our Model, holds the data structure classes, the backbone of the program
* BusinessLogic: the other part of Model, holds classes that are not used to store data and have only methods
* Root: the application package that holds all the other packages, basically the project package

1. Data structures

There should be data structures for the clients, queues, the scheduler and the simulation manager.

* Client: has an id, an arrival time and a service time
* Queue: has a list of clients waiting at that queue and a AtomicInteger for the total waiting time at that queue at a given time
* Scheduler: has a list of queues and a strategy
* SimulationManager: has a variable for every simulation property (max sim time, nb of queues, nb of clients, min arrival time, max arrival time, min service time, max service time), the scheduler and a list for the waiting clients

1. Interfaces

There will be only one interface, the Strategy interface that will be implemented by the ConcreteStrategy\* classes. It will have only one method, addClient() that takes the list of queues and a client and add it to the appropriate queue.

1. Alogorithms

Simple algorithms will be implemented in the ConcreteStrategy classes, ConcreteStrategyTime will search for the queue with the smallest waiting time and add the client there, while ConcreteStrategyLength will search for the queue with the fewest clients in it and add the client there. The time strategy is better, but the length one is more realistic, as this is how a real client would choose the queue.

The queues will have the run() method perform operations on the clients. When time passes, the first client will have his serve time decreased by one, and when it reaches 0 the client will be removed. If there are no clients it will do nothing until the next passing of time.

The simulation manager will contain the main loop in its run() method, where the clients will be sent to the scheduler when their arrivalTime == simulationTime and then the time will pass, until simulationTime == simulationTimeMax or there are no more clients the waiting queue or any other queue. Also, before starting it sorts all clients by arrival time, for convenience.

1. Implementation
2. Classes and methods

Here are the classes and their methods, divided by packages. Of course, all of them have their needed getters, setters and constructors that I did not include here.

* BusinessLogic:

The 2 strategies and their interface:

*/\*\*  
 \* Strategy interface  
 \*/*public interface Strategy {  
 */\*\*  
 \* Adds a client in a queue  
 \*  
 \** ***@param*** *serviceQueues The queues  
 \** ***@param*** *client The client to be added  
 \*/* void addClient(List<ServiceQueue> serviceQueues, Client client);  
  
}

*/\*\*  
 \* Strategy that puts the client in the queue with the fewest clients  
 \*/*public class ConcreteStrategyLength implements Strategy{  
  
 */\*\*  
 \* Adds a client in the shortest queue  
 \*  
 \** ***@param*** *serviceQueues The queues  
 \** ***@param*** *client The client to be added  
 \*/* @Override  
 public void addClient(List<ServiceQueue> serviceQueues, Client client) {  
 ServiceQueue shortestServiceQueue = serviceQueues.get(0);  
 for (ServiceQueue serviceQueue : serviceQueues) {  
 if (serviceQueue.getLength() < shortestServiceQueue.getLength())  
 shortestServiceQueue = serviceQueue;  
 }  
 shortestServiceQueue.addClient(client);  
 }  
}

*/\*\*  
 \* Strategy that puts the client in the queue with the lowest waiting time  
 \*/*public class ConcreteStrategyTime implements Strategy{  
  
 */\*\*  
 \* Adds a client in the queue with the lowest waiting time  
 \*  
 \** ***@param*** *serviceQueues The queues  
 \** ***@param*** *client The client to be added  
 \*/* @Override  
 public void addClient(List<ServiceQueue> serviceQueues, Client client) {  
 ServiceQueue fastestServiceQueue = serviceQueues.get(0);  
 for (ServiceQueue serviceQueue : serviceQueues) {  
 if (serviceQueue.getWaitingPeriod() < fastestServiceQueue.getWaitingPeriod())  
 fastestServiceQueue = serviceQueue;  
 }  
 fastestServiceQueue.addClient(client);  
 }  
}

A comparator for sorting:

*/\*\*  
 \* Comparator for client sorting  
 \*/*

public class ClientsComparator implements Comparator<Client> {

*/\*\*  
 \* Compares 2 clients by arrival time.  
 \* If it is equal it also reverse compares them by service time  
 \*  
 \** ***@param*** *client1 first client  
 \** ***@param*** *client2 second client  
 \** ***@return*** *the result of the comparison  
 \*/*@Override  
public int compare(Client client1, Client client2) { ... }

}

And the textbox validator:

*/\*\*  
 \* Class that validates the text boxes  
 \*/*public class TextBoxValidator{  
  
 */\*\*  
 \* Validates the textboxes, throwing exceptions by case  
 \*  
 \** ***@param*** *fieldSimTime simulation time textbox  
 \** ***@param*** *fieldQueues number of queues textbox  
 \** ***@param*** *fieldClients number of clients textbox  
 \** ***@param*** *fieldArrivalMin minimum arrival time textbox  
 \** ***@param*** *fieldArrivalMax maximum arrival time textbox  
 \** ***@param*** *fieldServiceMin minimum service time textbox  
 \** ***@param*** *fieldServiceMax maximum service time textbox  
 \*  
 \** ***@throws*** *NumberFormatException Textbox is empty or not numerical  
 \** ***@throws*** *NumberNegativeException Textbox contains a negative number  
 \** ***@throws*** *MinMaxException Minimum arrival/service time is higher than their maximum counterpart  
 \*/* public void validate(String fieldSimTime, String fieldQueues, String fieldClients, String fieldArrivalMin,  
 String fieldArrivalMax, String fieldServiceMin, String fieldServiceMax)  
 throws NumberFormatException, NumberNegativeException, MinMaxException, NumberZeroException

* Controllers:

The controllers for the 2 buttons, the radio buttons and the display (text area):

*/\*\*  
 \* Controller for "STOP" button  
 \*/*public class AbortController implements ActionListener {  
  
 SimulationManager simulationManager;  
 JTextArea areaLog;

*/\*\*  
 \* Stops the simulation if running  
 \*  
 \** ***@param*** *e Action event  
 \*/*@Override  
public void actionPerformed(ActionEvent e) { ... }

}

*/\*\*  
 \* Controller for "START" button  
 \*/*public class SimulationController implements ActionListener {  
 private JTextArea areaLog;  
 private JTextField fieldSimTime;  
 private JTextField fieldQueues;  
 private JTextField fieldClients;  
 private JTextField fieldArrivalMin;  
 private JTextField fieldArrivalMax;  
 private JTextField fieldServiceMin;  
 private JTextField fieldServiceMax;  
 private SimulationManager simManager;  
 private ArrayList<JRadioButton> radioButtons;

*/\*\*  
 \* Starts the simulation.  
 \* First it checks if the simulation is running, then it validates the textboxes and initializes  
 \* the SimulationManager with that data. At the end it creates a WriteController thread and starts it.  
 \*  
 \** ***@param*** *e Action event  
 \*/*@Override  
public void actionPerformed(ActionEvent e) { ... }

}

*/\*\*  
 \* Controller for strategy radio buttons  
 \*/*public class StrategyController implements ActionListener {  
 private ArrayList<JRadioButton> radioButtons;  
 private SimulationManager simManager;

*/\*\*  
 \* Changes the strategy  
 \*  
 \** ***@param*** *e Action event  
 \*/*@Override  
public void actionPerformed(ActionEvent e) { ... }

}

*/\*\*  
 \* Controller for writing in the textarea and the log file. It runs as a thread and is the one that starts the  
 \* SimulationManager thread.  
 \*/*public class WriteController implements Runnable{  
 private static final String *FILE\_PATH* = "log.txt";  
  
 private SimulationManager simManager;  
 private ArrayList<JRadioButton> radioButtons;  
 private JTextArea areaLog;

*/\*\*  
 \* Waits for new messages to display, until the simulation stops  
 \*/*@Override  
public void run() {  
 try {  
 FileWriter writer = new FileWriter(*FILE\_PATH*);  
 String msg;  
 simManager.startSimulation();  
 while (simManager.isRunning() || simManager.getMessages().size() != 0) {  
 while (simManager.isRunning() && simManager.getMessages().size() == 0) {  
 Thread.*sleep*(10);  
 }  
 if (!simManager.isRunning() && simManager.getMessages().size() == 0)  
 Thread.*sleep*(1100);  
 msg = simManager.getMessages().take();  
 writer.write(msg);  
 areaLog.append(msg);  
 areaLog.setCaretPosition(areaLog.getDocument().getLength());  
 }  
 writer.close();  
 } catch (InterruptedException | IOException e) {  
 e.printStackTrace();  
 }  
 for (JRadioButton radioButton : radioButtons) {  
 radioButton.setVisible(true);  
 }  
}

}

* DataModels:

The client model:

*/\*\*  
 \* The client data structure  
 \*/*public class Client {  
 private Integer id;  
 private Integer arrivalTime;  
 private Integer serviceTime;  
  
 */\*\*  
 \* Decrements serviceTime by 1  
 \*  
 \** ***@return*** *True if serviceTime is negative, false otherwise  
 \*/* public boolean decrementServiceTime() {  
 serviceTime--;  
 return serviceTime <= 0;  
 }  
  
 @Override  
 public String toString() {  
 return "(" + id + ", " + arrivalTime + ", " + serviceTime + ")";  
 }  
}

A enumeration for the strategies:

public enum QueuePolicy {  
 *SHORTEST\_QUEUE*,  
 *FASTEST\_TIME*}

The scheduler:

*/\*\*  
 \* The scheduler that manages the queues  
 \*/*public class Scheduler {  
 private ArrayList<ServiceQueue> serviceQueues;  
 private Strategy strategy = new ConcreteStrategyTime();  
  
 public void chooseStrategy(QueuePolicy policy) {  
 if (policy == QueuePolicy.*FASTEST\_TIME*)  
 this.strategy = new ConcreteStrategyTime();  
 else if (policy == QueuePolicy.*SHORTEST\_QUEUE*)  
 this.strategy = new ConcreteStrategyLength();  
 }  
  
 */\*\*  
 \* Starts a thread for each queue  
 \*/* public void startQueues() {  
 for (ServiceQueue serviceQueue : serviceQueues) {  
 Thread thread = new Thread(serviceQueue);  
 thread.start();  
 }  
 }  
  
 */\*\*  
 \* Stops all queue threads  
 \*/* public void stopQueues() {  
 for (ServiceQueue serviceQueue : serviceQueues) {  
 serviceQueue.stop();  
 }  
 }  
  
 */\*\*  
 \* Adds a client to a queue according to the strategy  
 \*  
 \** ***@param*** *client The client to be added  
 \*/* public void addClient(Client client) {  
 strategy.addClient(serviceQueues, client);  
 }  
  
 */\*\*  
 \* Checks if all the queues are empty  
 \*  
 \** ***@return*** *True if they are, false otherwise  
 \*/* public Boolean areQueuesEmpty() {  
 for (ServiceQueue serviceQueue : serviceQueues) {  
 if (serviceQueue.getWaitingPeriod() != 0)  
 return false;  
 }  
 return true;  
 }  
  
 */\*\*  
 \* Gets the number of queues that are serving a client  
 \*  
 \** ***@return*** *the number of clients being served  
 \*/* public Integer getServingClients() {  
 Integer nbServingClients = 0;  
 for (ServiceQueue serviceQueue : serviceQueues) {  
 if (!serviceQueue.isEmpty())  
 nbServingClients++;  
 }  
 return nbServingClients;  
 }  
  
 */\*\*  
 \* Gets the number of clients that are waiting in any queue  
 \*  
 \** ***@return*** *the number of clients in the queues  
 \*/* public Integer getWaitingClients() {  
 Integer nbClients = 0;  
 for (ServiceQueue serviceQueue : serviceQueues) {  
 nbClients += serviceQueue.getLength();  
 }  
 return nbClients;  
 }  
  
 @Override  
 public String toString() {  
 StringBuilder str = new StringBuilder();  
 int i = 1;  
 for (ServiceQueue serviceQueue : serviceQueues) {  
 str.append("Queue ").append(i++).append(": ").append(serviceQueue.toString()).append("\n");  
 }  
 return str.toString();  
 }

The service queues threads:

*/\*\*  
 \* A queue thread  
 \*/*public class ServiceQueue implements Runnable {  
 private LinkedBlockingQueue<Client> clients;  
 private AtomicInteger waitingPeriod;  
 private AtomicBoolean running;  
 private AtomicInteger simulationTime;  
  
 public ServiceQueue(AtomicInteger simulationTime) {  
 clients = new LinkedBlockingQueue<>();  
 waitingPeriod = new AtomicInteger(0);  
 running = new AtomicBoolean(true);  
 this.simulationTime = simulationTime;  
 }  
  
  
 public void addClient(Client client) {  
 waitingPeriod.addAndGet(client.getServiceTime());  
 try {  
 clients.put(client);  
 } catch (InterruptedException e) {  
 e.printStackTrace();  
 }  
 }  
  
 public void stop() {  
 running.set(false);  
 }  
  
 */\*\*  
 \* Processes clients  
 \*/* @Override  
 public void run() {  
 Integer oldSimulationTime = simulationTime.get();  
  
 while (running.get()) {  
 while (oldSimulationTime.equals(simulationTime.get())) {  
 try {  
 Thread.*sleep*(10);  
 } catch (InterruptedException e) {  
 e.printStackTrace();  
 return;  
 }  
 if (!running.get())  
 return;  
 }  
 oldSimulationTime = simulationTime.get();  
 if (clients.size() != 0) {  
 waitingPeriod.decrementAndGet();  
 if (clients.peek().decrementServiceTime()) {  
 clients.remove();  
 }  
 }  
 }  
 }  
  
 @Override  
 public String toString() {  
 if (clients.isEmpty())  
 return "closed";  
 StringBuilder str = new StringBuilder();  
 for (Client client : clients) {  
 if (client != clients.peek())  
 str.append("; ");  
 str.append(client.toString());  
 }  
 return str.toString();  
 }  
}

The simulation manager:

*/\*\*  
 \* A simulation manager  
 \*/*public class SimulationManager implements Runnable {  
 private AtomicBoolean running = new AtomicBoolean(false);  
 private AtomicInteger simulationTime = new AtomicInteger(0);  
 private Integer simulationTimeMax;  
 private Integer serviceTimeMin;  
 private Integer serviceTimeMax;  
 private Integer arrivalTimeMin;  
 private Integer arrivalTimeMax;  
 private Integer nbOfQueues;  
 private Integer nbOfClients;  
 private Scheduler scheduler = new Scheduler();  
 private SimulationStatistics simulationStatistics;  
 private LinkedList<Client> incomingClients = new LinkedList<>();  
 private LinkedBlockingQueue<String> messages = new LinkedBlockingQueue<>();  
  
 public void initialize(Integer simulationTimeMax, Integer nbOfQueues, Integer nbOfClients, Integer arrivalTimeMin,  
 Integer arrivalTimeMax, Integer serviceTimeMin, Integer serviceTimeMax) {  
 this.simulationTimeMax = simulationTimeMax;  
 this.serviceTimeMin = serviceTimeMin;  
 this.serviceTimeMax = serviceTimeMax;  
 this.arrivalTimeMin = arrivalTimeMin;  
 this.arrivalTimeMax = arrivalTimeMax;  
 this.nbOfQueues = nbOfQueues;  
 this.nbOfClients = nbOfClients;  
 simulationTime.set(0);  
 running.set(false);  
 incomingClients.clear();  
 simulationStatistics = new SimulationStatistics();  
 }  
  
 */\*\*  
 \* Starts the simulation.  
 \* Generates the queues and random clients, then starts the simulation thread.  
 \*/* public void startSimulation() {  
 Random random = new Random();  
 for (int i = 0; i < nbOfClients; i++) {  
 incomingClients.add(new Client(i + 1,  
 random.nextInt(arrivalTimeMax - arrivalTimeMin + 1) + arrivalTimeMin,  
 random.nextInt(serviceTimeMax - serviceTimeMin + 1) + serviceTimeMin));  
 }  
 incomingClients.sort(new ClientsComparator());  
 ArrayList<ServiceQueue> serviceQueues = new ArrayList<>();  
 for (int i = 0; i < nbOfQueues; i++) {  
 serviceQueues.add(new ServiceQueue(simulationTime));  
 }  
 scheduler.setServiceQueues(serviceQueues);  
 running.set(true);  
 scheduler.startQueues();  
 Thread thread = new Thread(this);  
 thread.start();  
 }  
  
 */\*\*  
 \* Stops the simulation and all queue threads  
 \*/* public void stopSimulation() {  
 scheduler.stopQueues();  
 running.set(false);  
 }  
  
 */\*\*  
 \* Sends all clients that have arrived to the queues  
 \*/* private void sendClients() {  
 Iterator<Client> iterator = incomingClients.iterator();  
  
 while (iterator.hasNext()) {  
 Client client = iterator.next();  
 if (client.getArrivalTime().equals(simulationTime.get())) {  
 scheduler.addClient(client);  
 iterator.remove();  
 } else  
 break;  
 }  
 }  
  
 */\*\*  
 \* Updates the statistics  
 \*/* private void updateStatistics()  
  
 */\*\*  
 \* Puts the statistics in the message BlockingQueue  
 \*  
 \** ***@throws*** *InterruptedException the thread has been interrupted  
 \*/* private void sendStatistics() throws InterruptedException  
  
  
 */\*\*  
 \* The main thread.  
 \* Manages the clients, messages and the passing of time.  
 \*/* @Override  
 public void run() {  
 try {  
 while (running.get()) {  
 sendClients();  
 updateStatistics();  
  
 messages.put(this + "\n");  
 simulationTime.incrementAndGet();  
  
 Thread.*sleep*(1000);  
 if (simulationTime.get() >= simulationTimeMax ||  
 (incomingClients.isEmpty() && scheduler.areQueuesEmpty()))  
 stopSimulation();  
 }  
 messages.put(this + "\n");  
 sendStatistics();  
 } catch (InterruptedException e) {  
 e.printStackTrace();  
 stopSimulation();  
 }  
 }  
  
 @Override  
 public String toString() {  
 StringBuilder str = new StringBuilder("Time " + simulationTime + "\n" + "Waiting clients: ");  
 if (incomingClients.isEmpty())  
 str.append("none");  
 for (Client client : incomingClients) {  
 if (client != incomingClients.getFirst())  
 str.append("; ");  
 str.append(client.toString());  
 }  
 str.append("\n").append(scheduler.toString());  
 return str.toString();  
 }  
  
}

And lastly a data structure for the simulation information:

*/\*\*  
 \* Class that stores the statistics of the simulation  
 \*/*public class SimulationStatistics {  
 private Integer totalWaitTime = 0;  
 private Integer totalServeTime = 0;  
 private Integer peakHour = 0;  
 private Integer maxClients = 0;

}

* Exceptions:

A few exceptions to help my implementation:

public class MinMaxException extends Exception

public class NumberNegativeException extends Exception

public class NumberZeroException extends Exception

* GUI:

The main frame and 2 panels implemented in it. There is no reason to include the panel code as it is too big, but the MainFrame is managable:

*/\*\*  
 \* Main frame.  
 \* Completes some initializations for the GUI.  
 \* Creates the panel where all the interactions will take place.  
 \*/*public class MainFrame extends JFrame {  
  
 public MainFrame(String title) {  
 super(title);  
 setLayout(new BorderLayout());  
  
 SimulationController simCtrl = new SimulationController();  
  
 Container container = getContentPane();  
  
 container.add(new LogPanel(simCtrl), BorderLayout.*CENTER*);  
 container.add(new MainPanel(simCtrl), BorderLayout.*WEST*);  
  
 }  
}

*/\*\*  
 \* The panel for the log.  
 \* Contains the text area for the log, blame swing layouts for the existence of this separate class.  
 \*/*public class LogPanel extends JPanel {  
 public LogPanel(SimulationController simCtrl) { ... }

}

*/\*\*  
 \* The main panel that contains all the GUI elements, except the textarea for the log.  
 \* Creates the elements, sets their positions and binds them to their respective controllers.  
 \*/*public class MainPanel extends JPanel {  
 public MainPanel(SimulationController simCtrl) { ... }

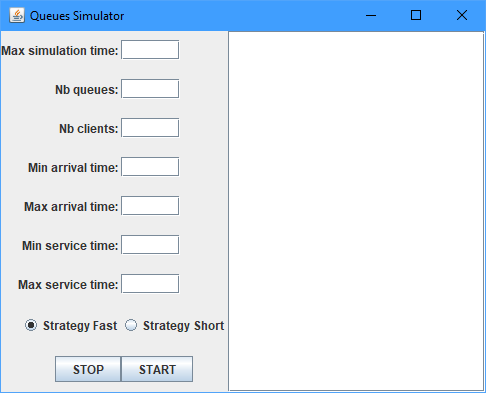
}

* Main:

The main function that starts the application:

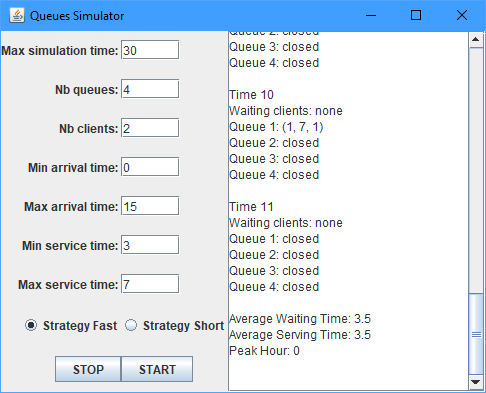
public class Main {  
  
 public static void main(String[] args) {  
  
 SwingUtilities.*invokeLater*(new Runnable() {  
 */\*\*  
 \* Starts a thread for the Swing graphical interface and completes the setup for the frame.  
 \*/* @Override  
 public void run() {  
 JFrame frame = new MainFrame("Queues Simulator");  
 frame.setSize(500, 400);  
 frame.setDefaultCloseOperation(JFrame.*EXIT\_ON\_CLOSE*);  
 frame.setVisible(true);  
 }  
 });  
  
 }  
}

1. User interface



On the left, we have the textboxes that are used to give the simulation its parameters, the radio buttons that are used to choose the strategy and the 2 buttons that start the simulation and stop it when it is running. The input text must be positive integers different from 0 (except for arrival time that can be 0). The start button can’t be pressed while the simulation is running, while the stop button can’t be pressed while the simulation is not running.

On the right, we have a noneditable text area that displays the log, updated in real-time, or error messages if need be. It auto-scrolls and has a scroll bar for ease of use.



1. Results

Following multiple tests and a lot of time spent debugging, I managed to get it to run correctly in all tested cases. There are various error messages for when the user tries to use the interface in a way that it is not meant to be used.

I have added the results of the tests from the assignment pdf as separate .txt logs file in the project folder, because the second and the third are too big for the documentation. This is how the first test looked like:

Time 0  
Waiting clients: (2, 12, 3); (3, 12, 2); (4, 22, 2); (1, 23, 2)  
Queue 1: closed  
Queue 2: closed  
  
Time 1  
Waiting clients: (2, 12, 3); (3, 12, 2); (4, 22, 2); (1, 23, 2)  
Queue 1: closed  
Queue 2: closed  
  
Time 2  
Waiting clients: (2, 12, 3); (3, 12, 2); (4, 22, 2); (1, 23, 2)  
Queue 1: closed  
Queue 2: closed  
  
Time 3  
Waiting clients: (2, 12, 3); (3, 12, 2); (4, 22, 2); (1, 23, 2)  
Queue 1: closed  
Queue 2: closed  
  
Time 4  
Waiting clients: (2, 12, 3); (3, 12, 2); (4, 22, 2); (1, 23, 2)  
Queue 1: closed  
Queue 2: closed  
  
Time 5  
Waiting clients: (2, 12, 3); (3, 12, 2); (4, 22, 2); (1, 23, 2)  
Queue 1: closed  
Queue 2: closed  
  
Time 6  
Waiting clients: (2, 12, 3); (3, 12, 2); (4, 22, 2); (1, 23, 2)  
Queue 1: closed  
Queue 2: closed  
  
Time 7  
Waiting clients: (2, 12, 3); (3, 12, 2); (4, 22, 2); (1, 23, 2)  
Queue 1: closed  
Queue 2: closed  
  
Time 8  
Waiting clients: (2, 12, 3); (3, 12, 2); (4, 22, 2); (1, 23, 2)  
Queue 1: closed  
Queue 2: closed  
  
Time 9  
Waiting clients: (2, 12, 3); (3, 12, 2); (4, 22, 2); (1, 23, 2)  
Queue 1: closed  
Queue 2: closed  
  
Time 10  
Waiting clients: (2, 12, 3); (3, 12, 2); (4, 22, 2); (1, 23, 2)  
Queue 1: closed  
Queue 2: closed  
  
Time 11  
Waiting clients: (2, 12, 3); (3, 12, 2); (4, 22, 2); (1, 23, 2)  
Queue 1: closed  
Queue 2: closed  
  
Time 12  
Waiting clients: (4, 22, 2); (1, 23, 2)  
Queue 1: (2, 12, 3)  
Queue 2: (3, 12, 2)  
  
Time 13  
Waiting clients: (4, 22, 2); (1, 23, 2)  
Queue 1: (2, 12, 2)  
Queue 2: (3, 12, 1)  
  
Time 14  
Waiting clients: (4, 22, 2); (1, 23, 2)  
Queue 1: (2, 12, 1)  
Queue 2: closed  
  
Time 15  
Waiting clients: (4, 22, 2); (1, 23, 2)  
Queue 1: closed  
Queue 2: closed  
  
Time 16  
Waiting clients: (4, 22, 2); (1, 23, 2)  
Queue 1: closed  
Queue 2: closed  
  
Time 17  
Waiting clients: (4, 22, 2); (1, 23, 2)  
Queue 1: closed  
Queue 2: closed  
  
Time 18  
Waiting clients: (4, 22, 2); (1, 23, 2)  
Queue 1: closed  
Queue 2: closed  
  
Time 19  
Waiting clients: (4, 22, 2); (1, 23, 2)  
Queue 1: closed  
Queue 2: closed  
  
Time 20  
Waiting clients: (4, 22, 2); (1, 23, 2)  
Queue 1: closed  
Queue 2: closed  
  
Time 21  
Waiting clients: (4, 22, 2); (1, 23, 2)  
Queue 1: closed  
Queue 2: closed  
  
Time 22  
Waiting clients: (1, 23, 2)  
Queue 1: (4, 22, 2)  
Queue 2: closed  
  
Time 23  
Waiting clients: none  
Queue 1: (4, 22, 1)  
Queue 2: (1, 23, 2)  
  
Time 24  
Waiting clients: none  
Queue 1: closed  
Queue 2: (1, 23, 1)  
  
Time 25  
Waiting clients: none  
Queue 1: closed  
Queue 2: closed  
  
Average Waiting Time: 2.25  
Average Serving Time: 2.25  
Peak Hour: 12

And some parts of the second test:

Time 0  
Waiting clients: (5, 2, 1); (29, 4, 6); (13, 4, 5); (46, 4, 5); (7, 4, 1); (19, 5, 6); (20, 5, 3); (27, 5, 2); (12, 6, 4); (21, 8, 6); (11, 9, 1); (35, 10, 6); (44, 11, 1); (17, 12, 6); (6, 12, 2); (48, 13, 6); (8, 15, 6); (38, 15, 5); (10, 15, 1); (2, 16, 5); (39, 16, 4); (41, 17, 7); (50, 17, 5); (3, 18, 1); (36, 19, 7); (47, 19, 1); (28, 21, 4); (32, 21, 3); (37, 23, 3); (40, 24, 6); (26, 24, 1); (42, 28, 6); (15, 29, 1); (34, 31, 6); (23, 31, 2); (24, 32, 4); (18, 33, 6); (22, 33, 4); (33, 33, 1); (4, 34, 6); (25, 36, 7); (43, 36, 5); (49, 37, 5); (31, 37, 3); (1, 37, 2); (16, 38, 4); (45, 39, 6); (9, 39, 1); (30, 40, 4); (14, 40, 2)  
Queue 1: closed  
Queue 2: closed  
Queue 3: closed  
Queue 4: closed  
Queue 5: closed  
  
Time 1  
Waiting clients: (5, 2, 1); (29, 4, 6); (13, 4, 5); (46, 4, 5); (7, 4, 1); (19, 5, 6); (20, 5, 3); (27, 5, 2); (12, 6, 4); (21, 8, 6); (11, 9, 1); (35, 10, 6); (44, 11, 1); (17, 12, 6); (6, 12, 2); (48, 13, 6); (8, 15, 6); (38, 15, 5); (10, 15, 1); (2, 16, 5); (39, 16, 4); (41, 17, 7); (50, 17, 5); (3, 18, 1); (36, 19, 7); (47, 19, 1); (28, 21, 4); (32, 21, 3); (37, 23, 3); (40, 24, 6); (26, 24, 1); (42, 28, 6); (15, 29, 1); (34, 31, 6); (23, 31, 2); (24, 32, 4); (18, 33, 6); (22, 33, 4); (33, 33, 1); (4, 34, 6); (25, 36, 7); (43, 36, 5); (49, 37, 5); (31, 37, 3); (1, 37, 2); (16, 38, 4); (45, 39, 6); (9, 39, 1); (30, 40, 4); (14, 40, 2)  
Queue 1: closed  
Queue 2: closed  
Queue 3: closed  
Queue 4: closed  
Queue 5: closed

...

Time 47  
Waiting clients: none  
Queue 1: closed  
Queue 2: closed  
Queue 3: closed  
Queue 4: (45, 39, 1)  
Queue 5: closed  
  
Time 48  
Waiting clients: none  
Queue 1: closed  
Queue 2: closed  
Queue 3: closed  
Queue 4: closed  
Queue 5: closed  
  
Average Waiting Time: 5.1  
Average Serving Time: 3.9  
Peak Hour: 40

And lastly the time and the results of the third test (putting the first takes up way too much space):

...

Time 200  
Waiting clients: none  
Queue 1: (915, 68, 8); (495, 70, 6); (906, 71, 7); (157, 73, 7); (502, 74, 3); (223, 75, 4); (984, 78, 7); (608, 81, 6); (617, 83, 7); (417, 85, 8); (285, 87, 5); (683, 88, 3); (358, 90, 9); (217, 92, 3); (486, 94, 7); (978, 96, 8); (965, 99, 9); (230, 100, 4)  
Queue 2: (250, 68, 7); (949, 70, 8); (857, 71, 4); (776, 72, 4); (846, 74, 8); (981, 75, 3); (807, 78, 8); (69, 81, 5); (439, 82, 4); (318, 83, 4); (209, 85, 7); (104, 86, 4); (310, 87, 3); (526, 89, 6); (672, 91, 3); (611, 92, 4); (902, 94, 9); (306, 96, 5); (645, 98, 9); (170, 100, 8)  
Queue 3: (321, 67, 4); (696, 69, 9); (563, 71, 7); (304, 73, 8); (616, 74, 3); (574, 76, 9); (132, 80, 7); (286, 82, 3); (903, 83, 6); (357, 85, 6); (518, 86, 4); (391, 87, 3); (202, 89, 5); (62, 91, 6); (251, 92, 3); (551, 94, 6); (705, 95, 3); (665, 97, 8); (505, 99, 8)  
Queue 4: (919, 67, 5); (864, 69, 4); (712, 70, 5); (951, 71, 7); (443, 73, 7); (775, 74, 3); (119, 76, 8); (541, 79, 4); (556, 81, 3); (368, 82, 6); (632, 84, 9); (449, 86, 5); (141, 88, 8); (165, 91, 6); (339, 92, 3); (916, 94, 5); (154, 95, 5); (533, 97, 4); (343, 98, 5); (432, 99, 6)  
Queue 5: (577, 67, 4); (44, 69, 8); (55, 70, 3); (459, 71, 3); (483, 72, 4); (389, 73, 5); (646, 74, 5); (27, 78, 8); (175, 80, 7); (426, 82, 3); (158, 83, 5); (441, 85, 7); (714, 86, 4); (606, 88, 8); (390, 91, 6); (98, 93, 9); (136, 96, 8); (372, 98, 5); (565, 99, 5); (507, 100, 3)  
Queue 6: (162, 68, 6); (643, 69, 3); (186, 70, 4); (586, 71, 7); (323, 73, 8); (88, 75, 9); (325, 78, 3); (781, 80, 6); (445, 82, 6); (921, 84, 8); (745, 86, 7); (914, 88, 8); (299, 92, 9); (349, 95, 8); (841, 97, 8); (531, 99, 8)  
Queue 7: (878, 67, 4); (607, 69, 8); (881, 71, 9); (708, 73, 7); (748, 75, 8); (148, 78, 5); (766, 80, 3); (901, 81, 3); (850, 83, 8); (726, 85, 7); (444, 86, 3); (307, 87, 4); (660, 89, 5); (721, 91, 6); (302, 93, 9); (440, 96, 8); (713, 98, 3); (742, 99, 8)  
Queue 8: (110, 67, 3); (270, 68, 4); (968, 70, 8); (383, 72, 9); (455, 74, 7); (688, 76, 8); (580, 79, 3); (454, 81, 5); (243, 83, 7); (529, 84, 5); (598, 85, 3); (524, 87, 8); (397, 89, 4); (10, 91, 3); (255, 92, 6); (767, 94, 3); (602, 95, 6); (204, 97, 6); (45, 99, 8); (604, 100, 8)  
Queue 9: (392, 67, 4); (923, 69, 8); (256, 71, 7); (674, 72, 3); (534, 73, 5); (317, 74, 4); (326, 76, 5); (700, 78, 5); (779, 80, 3); (478, 82, 9); (848, 84, 7); (977, 86, 7); (117, 88, 6); (796, 90, 7); (913, 93, 6); (982, 95, 6); (481, 97, 5); (743, 98, 3); (194, 99, 7)  
Queue 10: (234, 68, 6); (87, 70, 9); (572, 72, 9); (240, 74, 5); (166, 75, 4); (143, 78, 8); (259, 81, 8); (21, 83, 6); (836, 85, 7); (715, 87, 8); (886, 89, 3); (887, 90, 7); (212, 93, 5); (360, 95, 7); (599, 97, 5); (824, 98, 3); (874, 99, 7)  
Queue 11: (868, 67, 3); (334, 68, 4); (102, 70, 7); (81, 71, 5); (760, 72, 3); (53, 74, 9); (634, 76, 5); (510, 78, 4); (907, 80, 6); (584, 82, 5); (859, 83, 4); (933, 85, 7); (780, 87, 8); (980, 89, 3); (52, 90, 5); (947, 92, 4); (171, 94, 8); (315, 96, 7); (861, 98, 3); (975, 99, 7)  
Queue 12: (628, 67, 4); (926, 69, 7); (817, 70, 4); (474, 72, 7); (185, 74, 9); (629, 76, 4); (431, 78, 6); (71, 81, 6); (356, 83, 7); (476, 84, 4); (49, 85, 5); (560, 87, 5); (736, 89, 9); (753, 92, 6); (804, 94, 3); (54, 95, 5); (242, 96, 3); (728, 97, 4); (66, 99, 8); (379, 100, 7)  
Queue 13: (553, 67, 4); (759, 69, 6); (671, 70, 4); (150, 71, 5); (905, 72, 3); (190, 74, 9); (691, 77, 9); (362, 80, 5); (193, 82, 8); (491, 84, 6); (434, 86, 9); (182, 89, 8); (113, 92, 8); (746, 95, 8); (176, 97, 4); (50, 98, 8); (147, 100, 9)  
Queue 14: (578, 68, 6); (90, 70, 9); (487, 72, 7); (393, 74, 9); (937, 77, 8); (842, 79, 3); (754, 81, 4); (970, 82, 5); (490, 83, 3); (819, 84, 3); (896, 85, 6); (925, 87, 5); (276, 89, 8); (839, 92, 7); (7, 95, 8); (466, 96, 3); (101, 98, 9); (430, 99, 3); (676, 100, 7)  
Queue 15: (181, 67, 3); (543, 68, 4); (690, 70, 7); (189, 71, 5); (128, 73, 9); (78, 75, 6); (849, 78, 8); (303, 81, 3); (364, 82, 8); (655, 84, 6); (82, 86, 8); (123, 88, 6); (509, 90, 4); (149, 92, 4); (585, 94, 9); (14, 96, 6); (178, 98, 6); (569, 99, 5)  
Queue 16: (208, 67, 1); (411, 68, 5); (308, 70, 9); (924, 72, 7); (622, 74, 9); (125, 77, 6); (554, 78, 3); (677, 80, 5); (813, 82, 8); (837, 84, 6); (499, 86, 8); (640, 88, 6); (711, 91, 7); (537, 93, 5); (725, 95, 7); (442, 97, 4); (612, 98, 6); (840, 99, 4); (751, 100, 6)  
Queue 17: (37, 67, 3); (236, 68, 3); (335, 70, 9); (763, 72, 6); (133, 73, 6); (433, 74, 4); (458, 77, 3); (891, 78, 8); (469, 81, 3); (331, 82, 7); (935, 84, 8); (183, 86, 5); (365, 87, 4); (869, 89, 5); (482, 91, 3); (284, 92, 4); (667, 94, 9); (340, 96, 6); (710, 98, 6); (84, 99, 3); (488, 100, 8)  
Queue 18: (808, 67, 4); (682, 69, 4); (787, 70, 6); (421, 71, 5); (301, 73, 9); (106, 75, 5); (744, 78, 8); (96, 81, 6); (463, 83, 7); (130, 85, 9); (122, 87, 4); (156, 88, 5); (835, 90, 8); (555, 93, 5); (225, 95, 6); (702, 96, 3); (107, 98, 9); (832, 99, 3); (17, 100, 5)  
Queue 19: (290, 67, 3); (557, 69, 9); (404, 71, 7); (3, 73, 8); (686, 74, 4); (798, 78, 9); (865, 80, 5); (536, 82, 7); (644, 84, 7); (519, 86, 8); (662, 88, 5); (750, 90, 7); (23, 92, 3); (172, 94, 7); (456, 96, 8); (401, 99, 9); (351, 100, 5)  
Queue 20: (693, 68, 7); (957, 70, 7); (971, 71, 5); (83, 73, 8); (852, 74, 4); (799, 78, 9); (311, 80, 4); (618, 82, 9); (48, 84, 5); (539, 85, 5); (288, 87, 4); (283, 88, 4); (698, 90, 9); (649, 93, 4); (908, 95, 8); (462, 97, 4); (18, 98, 5); (252, 99, 6)  
  
Average Waiting Time: 86.276  
Average Serving Time: 3.794  
Peak Hour: 100

1. Conclusion

This was an interesting project that helped me learn how to use threads and how to enforce thread safety. Making the simulation was a more entertaining assignment than the last one, because this basically simulates a real-life shop, or anything with a queue and could be used as reference when designing a similar usable application, that can be sold for lots of money after you made the app at work, just because you were bored and, as a programmer, you don’t really work that much if you chose the right company, so freelancing for the win.

This could be improved by making it more realistic or adding different constraints, like a maximum amount of clients at a queue, queues closing and opening between random hour intervals, faster or slower service advancement at different queues, random events like clients randomly deciding that they don’t want to wait anymore at a queue if not processed, or some stopping from being processed and returning to the “waiting queue” with a different arrival time. There are many ways of making it more useful, complex or just fun to watch.

1. Bibliography

<https://docs.oracle.com/en/>

<https://stackoverflow.com/>