SE2205: Algorithms and Data Structures for Object-Oriented Design
Lab Assignment 2
Assigned: Feb 13, 2019; Due: Mar 8, 2019 @ 10:00 a.m.
If you are working in a group of two, then indicate the associated student IDs and numbers in the Test.java file as a comment in the header.

1 Objectives

The queue data structure can be used for modelling and simulating many interesting systems. In many practical systems, some kind of buffers/queues are used to temporarily place tasks or items prior to processing these. Entities arriving first at the buffer are processed first. Since the FIFO service paradigm is quite natural in a broad range of practical systems, queues can be naturally used to model the behaviour of these. In this lab assignment, you will be modelling a simple data network.

2 Introduction to Data Networks

Transmitting data through the internet (emails, video/voice streaming, etc.) is a complex process. The internet is essentially a large network composed of intermediate devices that forward transmitted data originating from a source device to a destination device through links as illustrated in Figure 1. These intermediate devices include routers and switches that forward data from incoming links to outgoing links based on their destination.

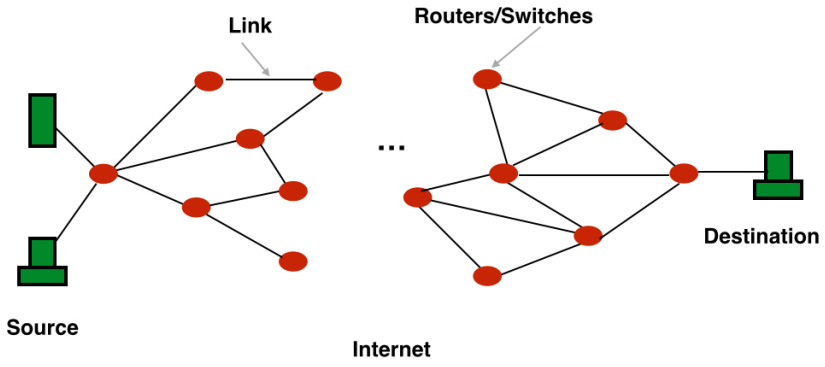

Figure 1: A Simplified Visualization

2.1 A Transportation of People Analogy

In order to understand how data is transmitted across the internet from a source to a destination, consider the following analogy. Suppose that 20 people are to be transported from Point A to Point B via cars only. Everyone is leaving from the same place and intend to arrive at the same place. Cars have fixed space (i.e. can only seat a maximum of 5 people). Assume that all 20 people manage to fit into 4 cars. These cars then will use various roads to travel from Point A to Point B. Since the roads are public, these will be shared by other cars as well. Depending on the time of the day, there will be various degrees of traffic on the roads. Certain roads such as the highways allow cars to travel at a faster speed than others. The cross point of roads will be managed by traffic signals. Based on all these factors, the drivers of the four cars may choose

different paths (composed of various roads) to reach the final destination (i.e. Point B). All four cars will arrive at Point B but possibly at different times depending on the conditions of the roads travelled and speed limits.

2.2 Similarities between Data Network and Transportation

This transportation example has surprisingly many similarities with the process incurred during the transmission of data from a source device to a destination device. Suppose that you would like to send an email from your computer (source) to your friend (destination) who resides in another country. In the transmission process, your email is first broken down into smaller entities called packets (analogous to dividing 20 people into 4 cars). These packets then traverse links (analogous to roads) in order to reach the final destination. Different links have different speeds/bandwidths (analogous to speed limits on roads). Since links in the internet are mostly public, your email packets will share the link with other packets from other sources (roads are shared with other people who are driving from other places). Multiple links intersect at a single point (i.e. junction at a road intersection). Intermediate devices such as switches or routers decide on where to forward packets arriving at these junctions. Packets may arrive at different times at a router. When more than one packet arrives at the router, these are stored in a buffer/queue (i.e. cars waiting at a junction). In this lab assignment, the following assumptions are made: data packets are all of the same size, time taken for a router to forward packets to appropriate links is constant and the queue is infinite in length. Based on the congestion properties and conditions of the network, routers may forward packets to different links (i.e. drivers taking different routes). Even though the packets will arrive at the destination, these might arrive at different times based on the conditions of the paths.

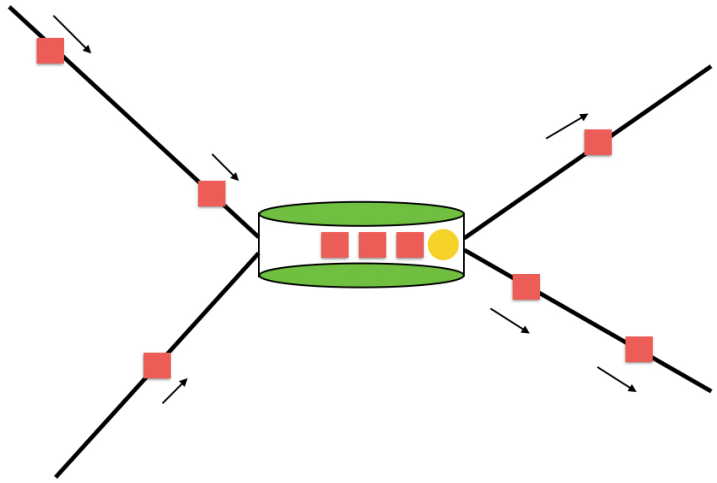

Figure 2: A Simple Representation of a Router Queue

2.3 Your Task for this Lab Assignment

In this lab assignment, you will model various properties of a queue in a single router when packets arrive into the queue at a particular rate and depart from the queue at a particular rate. As depicted in Figure 2, multiple incoming links can converge at the router. Packets arriving through these links are forwarded to appropriate outgoing links by the router. Service time (i.e. the forwarding of packets to appropriate links) is assumed to be constant. Intuitively, it is pretty reasonable to postulate that if the arrival rate of packets into a queue is lower than the service rate, then the queue will be not be congested and the average wait time of a packet in the queue before being processed will not be excessive. On the other hand, when the arrival rate of packets into the queue is greater than the service rate, then the queue will get filled up quickly and each packet in the queue will have to wait for very large periods of time before being served.

In this lab assignment, you will compute the average waiting times of packets departing a router queue by simulating the router queue for various packet arrival rates. You will notice that as the arrival rate gets closer to the service rate, the average waiting times of packets in the queue will increase exponentially and explode. This trend has been modelled by a well known law called the *Little's Law*. The average waiting time $E(S)$ or sojourn time of a packet in a queue can be related to the arrival rate λ of packets into the queue and service rate μ of the queue which denotes the departure rate of packets from the queue via this equation: $E(S) = \frac{1}{\lambda}(\frac{\lambda}{\mu} + \frac{\frac{1}{2}(\frac{\lambda}{\mu})^2}{1-\frac{\lambda}{\mu}})$. You can use your simulations to verify whether the router queue obeys this law.

2.4 Simulation of a Router Queue

A router queue can have no packets at a time or some packets waiting to be serviced. Packets arrive at random times. These arrival times depend on the arrival rates of packets. The function `public double getRandTime(double arrivalRate)` is provided to you in the `QueueSimulator.java` file. This function returns the duration (seconds) within which the next packet will arrive into the queue for a particular arrival rate λ which is passed as an argument to the function. For instance, suppose $\lambda = 0.1$ *packets/sec*, the function call `getRandTime(0.1)` may result in 1.344 *sec*. This means that the next packet will arrive into the queue within 1.344 *sec*. Hence, packets can be queued into the router according to the times returned by `getRandTime`. Every packet queued into the router will be processed by the router according to the first come first serve policy. Time taken to serve each packet is assumed to be *constant*. Suppose the service rate is $\mu = 10$ *packets/sec*, then the service time is $\frac{1}{10} = 0.1$ *sec*. This means that the time required for the router to process every packet at the front of the queue is 0.1 *sec*.

In a simulation, certain metrics about the system are measured throughout the simulation period. We are particularly interested in the average waiting time of departing packets for various combinations of λ and μ. The simulation will run for ρ *sec*. Time progression `currTime` in a simulation is based on events occurring in the simulation. Simulation will start at `0` sec. In a router queue, an event will occur if a packet is scheduled to arrive into or depart from the queue. Suppose that a packet is scheduled to arrive into the queue at `timeForNextArrival`. Suppose that the number of packets in the queue is at least one. Then the first packet in the queue is scheduled to depart from the queue at `timeForNextDeparture`. The next event that will occur in the simulation is an `ARRIVAL` if `timeForNextArrival < timeForNextDeparture`. Otherwise, the next event is a packet `DEPARTURE`. If there are no packets in the queue, then the only event that can occur next is an `ARRIVAL`. When an event occurs, the simulator updates the current time variable `currTime` to the time at which the event occurs. The simulator operates a loop conditioned upon the current simulation time. The loop will terminate when `currTime` is greater than or equal to the total simulation time `simTime` allocated to the simulation. At each iteration of the while loop, the simulator will perform operations on two queue data structures. If the current event is an `ARRIVAL`, the simulator will enqueue a node into the `buffer` queue (which mimics the router queue) and compute `timeForNextArrival`. On the other hand, if the current event is a `DEPARTURE`, then the first node in the `buffer` queue is dequeued and this node is enqueued into `eventQueue`. The simulator then computes `timeForNextDeparture`. All nodes in `eventQueue` contain information about the arrival time and departure time of a packet into and from the `buffer` queue. At the end of the simulation, the simulator uses the `calcAverageWaitingTime` function to dequeue all nodes from `eventQueue` and compute the average waiting time or sojourn time of packets that have departed the `buffer` queue. Packets still remaining in the `buffer` queue are not accounted for in this calculation. The average packet waiting time computed is returned by the function `runSimulation` which is running the simulation.

3 Materials Provided

You will download content in the folder `SE2205B-LabAssignment2` which contains two folders (`code` and `expOutput`) onto your workspace. Folder `code` contains a skeleton of function implementations and declarations. Your task is to expand these functions appropriately in the `SinglyLinkedList.java`, `LinkedListQueue.java`, `Data.java`, and `QueueSimulator.java` files. `Test.java` evokes all the functions you will have implemented and is similar to the file that will be used to test your implementations.

Use `Test.java` file to test all your implementations. Folder `expOutput` contains outputs expected for function calls in `Test.java` (for part 2 the answers should be fairly close to the expected output (to one decimal place) with the exception of $\lambda = 10$packets/sec). Note that we will **NOT** use function calls with the same parameters for grading your lab assignment. Do **NOT** change the name of these files or functions.

4 Grading: Final Mark Composition

It is **IMPORTANT** that you follow all instructions provided in this assignment very closely. Otherwise, you will lose a *significant* amount of marks as this assignment is auto-marked and relies heavily on you accurately following the provided instructions. Following is the mark composition for this assignment (total of 40 points):

- Successful compilation of all program files i.e. the following command results in no errors (3 points):
  `javac Queue.java SinglyLinkedList.java LinkedListQueue.java Data.java QueueSimulato`
  `Test.java`
- Successful execution of the following commands: (2 points)
  `java Test 1`
  `java Test 2`
- Output from Part 1 matches expected output (7 points)
- Output from Part 2 matches expected output (to one decimal place for the first four outputs of the `runSimulation` function) (18 points)
- Code content (10 points)

Sample expected outputs are provided in folder `expOutput`. We will test your program with a set of completely different data files. Late submissions will not be accepted. All late submissions will be assigned a grade of 0/40.

5 Implementation of the Simulation Program

You will be using `queue` ADT as a basic building block for this lab assignment. Your simulation program will make use of the `queue` ADT to model a queue in a router. This queue is assumed to be infinite in length (i.e. no restriction on the size of the queue). Packets will arrive into the queue at rate λ. Time taken to serve these packets (i.e. forward to the appropriate link) is fixed as dictated by the service rate μ.

Part 1: Defining Functions Associated with a Queue

In this part, function implementations of `enqueue`, and `dequeue` will be tested. The `Queue` interface is provided in `Queue.java` file. These functions are to be defined in the `LinkedListQueue.java` file. Prior to implementing the `LinkedListQueue` class, you will need to define the `SinglyLinkedList` class in `SingleLinkedList.java` file. This class is composed of a local private `Node` class that consists of a generic element `E` and a reference to the `next` Node. It should also support the following interface functions:

- `public Node(E e, Node<E> n);`
- `public E getElement();`
- `public Node<E> getNext();`
- `public void setNext(Node<E> n);`

where the constructor initializes the local fields with the arguments passed, `getElement` returns the generic element stored in the node, `getNext` returns the reference to the next node locally stored and `setNext` changes the reference stored in the local variable `next`. After this private class is defined, you can proceed to defining the `SinglyLinkedList`. Here you will need to keep track references to the first and last nodes along with the size of the linked list as private field variables. The following functions will be defined to support operations on the linked list:

- `public SinglyLinkedList();`
- `public int size();`
- `public boolean isEmpty();`
- `public E first();`
- `public E last();`
- `public void addFirst(E element);`
- `public void addLast(E element);`
- `public E removeFirst();`

where you will define the constructor to initialize the private field variables, `size` returns the number of nodes in the linked list, `isEmpty` returns `TRUE` if the linked list has no nodes and `FALSE` otherwise, `first` returns the content stored in the node at the front of the linked list, `last` returns the content stored in the node at the end of the linked list, `addFirst` adds a node containing `element` to the front of the linked list, `addLast` adds a node containing `element` to the end of the linked list and `removeFirst` removes the node at the front of the linked list. The `SinglyLinkedList` class will then be utilized to implement the Queue interface in `LinkedListQueue.java`. Here you will implement the interface functions defined in the `Queue.java` file.

- `public LinkedListQueue();`
- `public int size();`
- `public boolean isEmpty();`
- `public E first();`
- `public void enqueue(E node);`
- `public E dequeue();`

where constructor will initialize the private field variables (e.g. linked list), `size` returns the number of nodes in the queue, `isEmpty` checks whether the queue has any nodes, `first` returns the element stored at the front of the queue, `enqueue` inserts a node to the back of the queue, and `dequeue` removes a node from the front of the queue. The element stored in each node is defined to be of generic type. For this lab assignment, we will utilize a specific element of type `Data` which is defined in the `Data.java` file. It is composed of the two private field variables of type `double` that are called `arrivalTime` and `departureTime`. Operations on these variables will be conducted via the following functions:

- `public Data();`
- `public void setArrivalTime(double a);`
- `public void setDepartureTime(double d);`
- `public double getDepartureTime();`
- `public double getArrivalTime();`

where the constructor initializes the private field variables to 0, `setArrivalTime` sets the `arrivalTime` variable to be the argument passed into the function, `setDepartureTime` sets the `departureTime` variable to be the argument passed into the function, `getDepartureTime` returns the value stored in the `departureTime` variable and `getArrivalTime` returns the value stored in the`arrivalTime` variable. This part of the lab assignment will be tested via the following commands:

- `java Test 1`

Outputs from these tests should match the content of `Part1.txt` which is the result of the parameters passed from function calls in `Test.java`.

Part 2: Implementing the Router Queue Simulator

You will build a queue simulator that computes the average waiting time of data packets departing from a router queue. You will analyze the impact of various packet arrival rates on average waiting times. In this implementation, you will define the class called `QueueSimulator` and three functions. The `QueueSimulator` class stores all parameters and ADTs associated with a simulation via field variables. These consist of the following field variables: `double currTime`, `double arrivalRate`, `double serviceTime`, `double timeForNextArrival`, `double timeForNextDeparture`, `double totalSimTime`, `LinkedListQueu buffer`, `LinkedListQueue<Data> eventQueue` and `Event e;`. `currTime` keeps tracks of the time progression of a simulation. `arrivalRate` and `serviceTime` store λ and $\frac{1}{\mu}$ respectively. `timeForNextArrival` stores the time at which the next packet will arrive into the `buffer` queue. `timeForNextDeparture` will keep track of the time at which a packet will depart from the `buffer` queue. `totalSimTime` denotes the total time duration for which the simulation is run for. It is assumed that the `buffer` is initially empty and packets start entering the queue only when the simulation begins. `buffer` is a `LinkedListQueue` data structure that will mimic an actual queue in a router. `eventQueue` is another queue that is used to store information about packets entering and exiting the `buffer` queue. `Event` is an enum with two constants: `ARRIVAL` and `DEPARTURE`. `e` is used to store information about the next event that will occur in the simulation. Three functions which are to be implemented for this part are:

- `public QueueSimulator(double arrivalRate, double serviceTime, double simTime);`
- `public double runSimulation(double arrivalRate, double serviceTime, double simTime);`
- `public double calcAverageWaitingTime(struct Simulation * S);`

The service rate is denoted by μ is fixed and denotes the rate at which packets depart from the queue. The arrival rate of packets into the queue can vary depending on the congestion in the network. Suppose that the service rate is $\mu = 10$ *packets/sec* (i.e. the service time for a packet is fixed and therefore if $\mu = 10$ *packets/sec* then it takes the router $\frac{1}{10} = 0.1$ *sec* to process a packet (also known as service time)). Packets will arrive into a queue at random times. However, on average, these packets arrive at a rate λ. A simulation is evoked via the creation of a new `QueueSimulator` instance. The simulation is initialized by the constructor `QueueSimulator`. Three arguments `arrivalRate, serviceTime` and `simTime` are passed into this function. `arrivalRate` is λ, `serviceTime` is $\frac{1}{\mu}$ and `simTime` is ρ. The arrival time of the first packet into the queue is computed via the `getRandTime` function and this value is stored in the `timeForNextArrival` member. The departure time of the first packet from the queue is computed by adding the fixed service time to the previously computed packet arrival time and this value is stored into the `timeForNextDeparture` member. The next function `runSimulation` will launch a while loop and progress through the simulation by performing appropriate enqueueing or dequeueing operations based on the current event. Packets are enqueued into `buffer` in the form `Node`. The arrival time of the packet into `buffer` is recorded in the `arrivalTime` member of the `Data` variable which is the `E` member of the `Node` struct. When a packet is dequeued from `buffer`, its departure time is recorded in the `departureTime` member of the `Data` variable and this node is then enqueued into the `eventQueue` queue which records all packets departing the `buffer` queue. When the simulation runs for ρ *seconds*, it exits the while loop and evokes `calcAverageWaitingTime` to compute the average waiting time of packets that have already departed from the `buffer` queue. This value is printed to the console. This part of the assignment will be tested via the following commands:

- `java Test 2`

Outputs from these tests should match the content ((to one decimal place for the first four outputs of the `runSimulation` function)) in `Part2.txt` which is the result of the parameters passed from function calls

in `main.c`. You can test if your implementation is correct by printing out the arrival time and departure time of packets when these are removed from `buffer` queue.

6 Code Submission

7 Code Submission

You will use `git` to submit this assignment. Create a new repository for `LabAssignment2`:

- Download the lab assignment file `SE2205B-LabAssignment2.zip` from OWL into your course folder say `SE2205B`.
- Unzip this downloaded file which will create a folder in the same directory called `SE2205B-LabAssignment2`.
- Open GitHub Desktop.
- Run `File` → `Add local repository`.
- Click `Choose` and browse for the folder `SE2205B-LabAssignment2`.
- You will get the following warning: `This directory does not appear to be a Git repository. Would you like to create a repository here instead?`
- Click on `Create a Repository` and then click on `Create Repository`.
- In the GitHub Desktop tool bar, click `Publish` repository, check the option `Keep this code private`, choose your GitHub account, and then click `Publish` repository.
- Now a new repository called "SE2205B-LabAssignment2" should appear in your GitHub account.
- You will need to add the instructor and the TAs to this repository. For this,
  - In the GitHub account click the repository `SE2205B-LabAssignment2` and then click `Setting`
  - Click `Collaborators & teams` option.
  - At the bottom of the page and in the Collaborators section, enter the account id of the GitHub user you would like to add and then click `Add collaborator`.
  - You need to add the following accounts to your repo:
    - **psrikan**
    - **gdmanandamohon**
    - **navidfekri**
    - **EvanTian233**

ENSURE that your work satisfies the following checklist:

- You submit before the deadline
- All files and functions retain the same original names
- Your code compiles without error (if it does not compile then your maximum grade will be 3/40)