CMSC 221 — Data Structures
Project 5
A Traffic Simulation

Due: 5:00pm Tuesday, Nov. 22 2005

Overview: In this project, you will implement the ADT queue using the Java-supplied LinkedList class. You will then use your queue class to implement a traffic simulation to determine an optimal green light duration for the intersection of two one-way streets.

Project Description: Your first task is to implement the ADT queue by writing a class that implements the textbook’s QueueInterface. Your class should be called SimQueue, and should contain an instance of the Java-supplied LinkedList class as a data field that you will use to store the entries in your queue. This is an instance of encapsulation that will restrict the use of LinkedList methods that would violate the access properties we must maintain for a queue, as we have discussed in class.

After you have implemented and tested your SimQueue class, you must implement a simple traffic simulation model of the problem described below.

![Figure 1: Intersection of two one-way streets controlled by traffic lights.](image)

Simulation Problem Description:

Consider a traffic intersection that consists of two one-way streets controlled by a pair of traffic lights, as shown in Figure 1. Obviously, traffic may only proceed in one direction at a time, so that when the traffic light is green for one direction, it is red for the other direction. Because construction is underway at the intersection, the road is extremely hazardous, forcing drivers to proceed through the intersection at an extremely slow pace. Assume that the duration of the green light for direction one has already been determined to be 45 seconds. Further assume that when the light for direction one turns red, the light for direction two remains red for a period of 30 seconds, to permit
cars passing in direction one to completely clear the intersection before cars in the other
direction are permitted to proceed. The same delay is also used for the other light. To
summarize, the lights work as follows: when the light in direction one turns red, the
light in the other direction remains red for the fixed period of time before turning green.
The same things happens when the second light turns red; the first light remains red for
the fixed period before turning green. In our example with the direction one light green
for 45 seconds and the delay period for both directions fixed at 30 seconds, the period
of time between the light for direction two turning red and turning green again would
be 105 seconds 30 seconds waiting for direction 2 traffic to clear the intersection, 45
seconds for green in direction one, and 30 more seconds waiting for the direction one
traffic to clear the intersection. Note that the purpose of the fixed period of red light is
like the function of the yellow light in a real traffic signal.

The purpose of your simulation study is to determine a green duration for direction two
that will minimize the average wait time for cars arriving in both directions.

Some important notes:

- **A car may only enter the intersection if the light is green for that direction.**
- **A car may not enter the intersection until exactly two seconds after a preceding
car has entered the intersection (the safe following distance).**
- **If a car arrives at the intersection to find the light green in that direction, and if
there are no cars ahead in the queue and it has been at least two seconds since a
car has entered the intersection, the car may proceed immediately.**

**Simulation Particulars:** The simulation code must reside in a file named *TrafficSim.java.*
For this simulation project, you should use *next-event simulation.* Remember that rather than
having a variable representing a clock that increments, in a next-event simulation we record the
times that events will occur, and always choose the future event with the lowest time to process.
This allows us to skip a lot of checking at clock values where nothing happens. To construct a
next-event simulation, three things must be done:

- **Construct a set of variables that collectively provide a state description for the system;**
- **Identify those events that can change the state of the system; and**
- **Construct a set of algorithms that define the changes in state when each type of event occurs.**

**Event types:** For this project, you are only concerned with modeling the traffic flow for one
direction (the red times and the green time for the other direction are already determined, and we
will be calculating the average waiting time for that direction directly, so traffic in that direction
need not be simulated). As a result, there are three types of events that can change the state of
the system: a car arriving at the intersection in direction two; a light change; and a car entering
the intersection from direction two.

**State variables:** At a minimum, I suggest that you maintain an array of *double* as the event
list that contains the next time that an event of each type will occur. It will make understanding
your code much easier if you define a named constant for each event type that you can use as the
index into the array and to determine which type of event you are handling. Initializing your event list should look something like the following.

```java
double[] eventList = new double[3];
eventList[CAR_ARRIVAL] = /* arrival time of first car */
eventList[LIGHT_CHANGE] = /* time of first light change */
eventList[CAR_ENTERES] = /* time of first car to enter the intersection */
```

(These are just suggested names; use ones that make sense to you.) You will be updating the event times stored in the event list as you process events. The next event to process in simulated time will be the entry in the event list with the smallest time. Note that the index of the time in the event list (either 0, 1, or 2) determines the type of event that is occurring.

Also note that each entry may not always have a valid time, for example, how do you know when the first car will enter the intersection until you have processed the first arrival? You may assume that the light starts green for direction two (the direction you are simulating).

Furthermore, you will need a queue to hold cars that arrive in direction two but cannot proceed immediately, and you will need a variable to keep track of the light status (green or red).

**Set of Algorithms:** You must then have some mechanism that processes events in the event list and then updates the event list appropriately until some maximum time has been reached.

Let \( t \) be the time of the current event (the one with the minimum time in the event list) in the simulation. I suggest you use a structure similar to the following to process events:

```java
// do what is necessary to set up the first event to occur (an arrival)
// ...
// Get next event to process
eventType = /* index of event list entry with lowest time */
t = eventList[eventType]; /* time of next event to process */

while (t < maxTime)
{
    switch (eventType)
    {
        case CAR_ARRIVAL: // an arriving car
            // do what is necessary to process an arriving car
            // update eventList[CAR_ARRIVAL] to be the time for the
            // next car to arrive
            ...
            break;

        case LIGHT_CHANGE: // a light change
            // do what is necessary to process a light change
            // update eventList[LIGHT_CHANGE] to be the time for the
            // next light change
            ...
            break;
    }
}
```
case CAR.ENTERS:  // a car entering the intersection
    // do what is necessary to process a car entering the intersection
    // compute wait time for the car
    // update eventList[CAR.ENTERS] to be the time for the next
    // car to enter (if there is anything waiting to enter)
    ...
    break;
}

    // Get the next event to process
    eventType = /* index of event list entry with lowest time */
    t = eventList[eventType]; /* time of next event to process */

You must define what happens for each of the cases. For example, when the first car arrives and you are processing that event, you can then set up the first “enter the intersection” event in the event list.

**Accompanying Software:** On the course assignments web page, you will find a file named Car.java. Provided in this class are, among other things, the methods

```java
    public static Car getNextArrival(double meanArrivalTime)

    public double getArrivalTime()
```

You can use the static method `getNextArrival()` to automatically generate a `Car` object with an *appropriate arrival time* for you. Note that you must supply a *mean arrival time* (and this should be the same for all cars during a simulation run) The *mean arrival time* for direction two will be provided to your program as a command line argument as described below.

The non-static method `getArrivalTime()` can be used to get the arrival time of a particular `Car` object when so desired. The queue for direction two described above will hold `Car` objects.

**Experiments and Analysis:** Your TrafficSim executable must accept three command line *arguments*: the maximum time of the simulation in seconds, the length of the green light for direction two (in seconds), and the mean arrival time of cars arriving to the intersection.

Once your simulator is working, perform experiments to answer the following questions:

1. Based on experimentation, what should be the length of the green light to minimize the average wait time of cars (from both directions) if the mean interarrival time of cars for direction two is 12.0? What did you choose for the maximum time of the simulation, and why? What can you provide as evidence to support your choice for the “optimal” green time?

2. Same as the above question, except the mean interarrival time of cars for direction two is 9.0.

To facilitate answering these questions, consider the following methods available in *Car.java*.

```java
    public static void setDirectionTwoGreenLength(double length)

    public static double getDirectionOneAvgWait()
```
The first of these methods should be used to set the length of the green light for direction two. Set this before you start your simulation loop. This value is needed in the Car class to help calculate the wait for cars in direction one.

The second of these methods should be used when computing the average wait for cars in both directions. For this project (although you should not do this in general), compute the average wait for both directions as follows. First, compute the average wait for cars in direction two by summing the wait times of those cars processed by your simulation and then dividing by the number of cars you processed. Then compute the average wait for cars in both directions as the average of the wait time for direction two (what you just computed) and the average wait time for direction one (obtained using the second of the static methods above). We resort to this process for computing the average wait because we aren’t actually simulating the traffic for direction one.

Write-up and Submission:

- Your program must be properly documented so that I can execute javadoc and obtain a complete, meaningful Java documentation HTML page.

- You must supply a separate well-written, logical discussion of your program with sufficient detail so that by reading this discussion but not looking at your program another student in the class could construct a correct program and understand the theory on which it is based. At a minimum, the write-up must include discussion of the algorithm(s) you used in your implementation, evidence of your experiments that supports your choice for the optimal green times, and answers to the questions outlined in the previous section. Note that since you are simply encapsulating an existing data structure for your queue, you do not need to discuss the queue class in your writeup.

- Submit your source code files and your write-up to larnett@richmond.edu as email attachments by 5:00pm Tuesday, Nov. 22 2005.

Your work on this and all programming projects in this course is subject to the conditions of the Honor Code as described in the course syllabus.