CSC 383, Sections 401 and 410
Fall, 2011
Assignment 3
Help for the simulation algorithm

Simulation Parameters (set as Java constant values)

Remember that there are five constants you need to declare and initialize:

- The number of seconds being simulated: This is an integer that we decided will be 3600.
- The probability that a customer arrives each second: This is a floating point value between 0 and 1 and we decided on 0.1 (that is, there is a 10% chance that person arrives any given second).
- The number of servers: This is an integer and we decided on 4.
- The minimum amount of time in seconds to serve a customer: This is an integer and we decided on 60.
- The maximum amount of time in seconds to serve a customer: This is an integer and we decided on 300.

Generating Random Values

There are two points at which you will need to generate a random value and you should use the Java API **Random** class for this.

The first point is when you decide whether a customer has just arrived. Use the method **Random** method **nextDouble()**, which returns a floating point value uniformly chosen between 0 and 1. If the value is less than or equal to the arrival probability, a customer has arrived.

The second point is when you determine how much service time a customer will need. Use the **Random** method **nextInt(int n)**. Note that this uniformly selects a value between 0 and $n - 1$. You will have to use that value to generate a value between the minimum service time (60) and the maximum service time (300).

The Simulation Algorithm

Here is a pseudocode version of the simulation algorithm used for the McDonald's arrangement.

```
create one queue for each server
create one busy timer for each server and set each to 0
set customer served count to 0
set total time waiting to 0
for (seconds = 1; seconds <= simulation time; seconds++) {
  if a customer has arrived {
    enqueue that customer's arrival time on one of the
      server queues
  }
  for each server {
    if that server's busy timer is 0 {
      if that server's queue is not empty {
        dequeue a customer's arrival time
        compute that customer's wait time and add to total
        increment customer served count
        generate customer's service time
        set the server's busy time to that service time
      }
    }
    else {
      decrement that server's busy timer
    }
  }
}
print simulation parameter values (total simulation time,
   number of servers, arrival probability, min and max
   service times)
print number of customers served
calculate and print average wait time
```

Decide for yourself how to select which server's queue to put a new customer onto. One suggestion is to choose the shortest queue, which reflects actual behavior.

For the single line simulation, you will create exactly one queue (and not one for each server) and replace every reference to a particular server's queue with a reference to this single queue.

Design Notes

Use either the **ArrayList** or the **LinkedList** Java API generic class for the queues. As we discussed in class, the enqueue operation is performed with the **add()** method, the dequeue operation with the **remove()** method, and the size operation with the **size()** method. Because you are putting integer values on them, every reference to the class should be **ArrayList<Integer>** or **LinkedList<Integer>**.

You might model a customer as an object but the only data we keep track of for a customer is its arrival time. Of course, we might extend this simulation later and require more information be stored for a customer.

You could model a server as an object but, again, the only data we store here is the server's busy time. If you do create a server class, I strongly recommend not including its queue as a member variable. Declare those directly in the program.