DUE: 6/5/2013

Project #3—Print Queue Simulation

This program will give you practice with a queue data structures. Queues occur naturally in situations where the rate at which customer's demands for services can exceed the rate at which those services can be supplied. For example, cars arriving at a toll booth may have to wait in a queue before proceeding to the booth. The diagram here shows a queue of four cars waiting at a toll plaza with three booths. The same kind of situation can occur on a local area network where many computers share only a few printers: the print jobs may accumulate in a print queue. Supermarket checkout counters, bank teller lines and movie lines are similar. They all implement the first-in-first-out (FIFO) protocol.

You are to write a program that solves this classic computer science problem known as client/server problem for print jobs and printers at Alderwood Hall. Assume for this simulation there are four servers: Printer A, Printer B, Printer C, and Printer D. Each printer has a different print rate, expressed as a percentage. For example, Printer A has a print rate of 89, meaning that it averages 0.89 page per second. Each printer's print job also has a print rate. For example, on Job #1, Printer A has a print rate of 84, meaning that it is printing 0.84 page per second on that job. A single printer's print rates vary among jobs because of other factors such as network traffic. For example, Printer B, which has an average print rate of 0.97 page per second, prints Job #2 at 0.91 pps (pages per second) and Job #6 at 0.95 pps. The print jobs arrive at the print queue random times. In this run of the simulation Job #1 arrived at time 2 (i.e., 2 seconds after the start time), Job #2 arrived at time 10, and Job #3 arrived at time 18. The first part of the output from a single run of the simulation follows.

When jobs arrive faster than the printers can begin printing them, they accumulate in the print queue. In this simulation, that happens when Job #8 arrives at time 127, when all four printers are busy. Prior to that time, there was always at least one idle printer that could begin printing a job as soon as it arrived:

Printer A begins printing Job #1 as soon as it arrives at time 2.
Printer B begins printing Job #2 as soon as it arrives at time 10.
Printer A begins printing Job #3 as soon as it arrives at time 18.
Printer C begins printing Job #4 as soon as it arrives at time 44.
Printer A begins printing Job #5 as soon as it arrives at time 78.
Printer B begins printing Job #6 as soon as it arrives at time 113.
Printer D begins printing Job #7 as soon as it arrives at time 121.

But Job #8 has to wait 28 seconds before Printer D can begin printing it at time 155. It waits in the print queue. And of course, all the other jobs that arrive during that 28-second period must also wait in the print queue. By the time 155, the queue contains four jobs: #8, #9, #10, and #11.

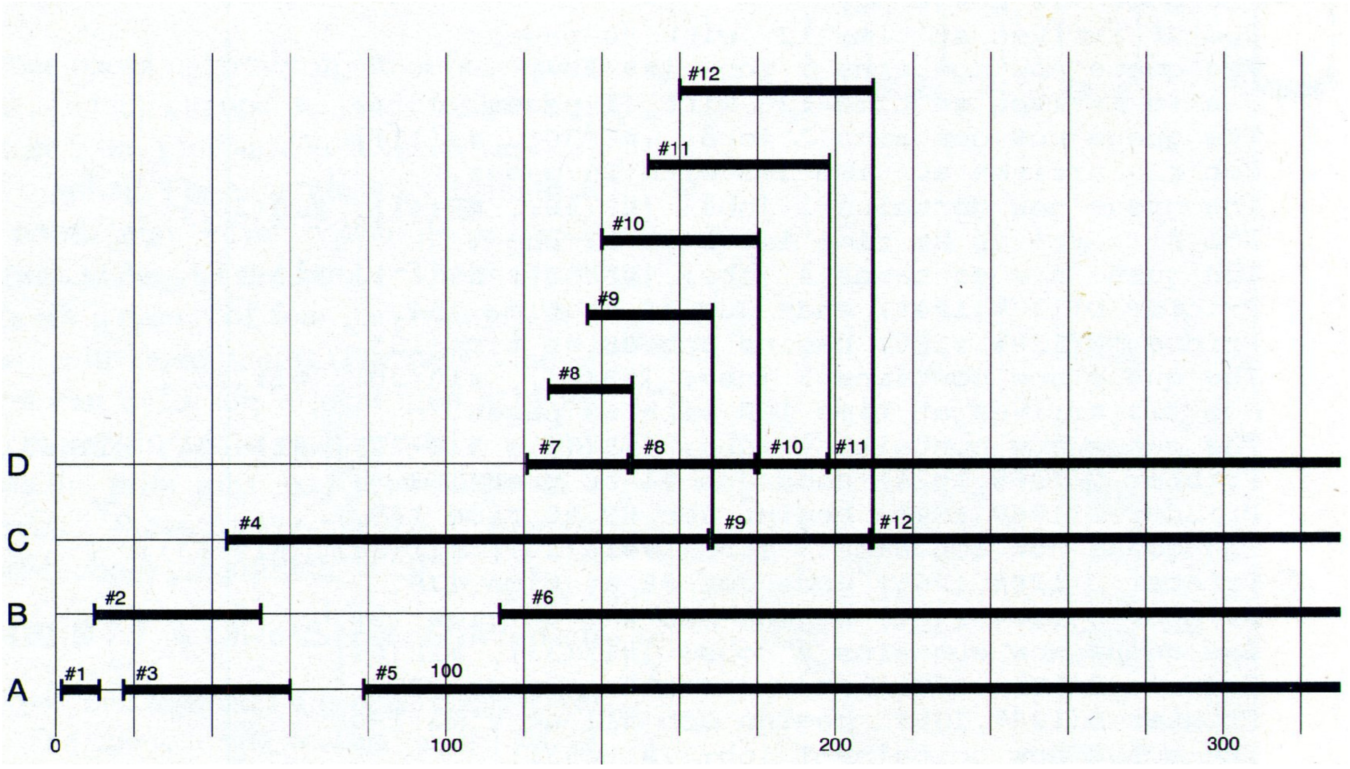
```
Job #1 arrives at time 2 with 7 pages.
The queue now contains 1 job: [#1(7)]
Printer A(89%,84%) begins Job #1 at time 2.
The queue is now empty.
Job #2 arrives at time 10 with 39 pages.
The queue now contains 1 job: [#2(39)]
Printer B(97%,91%) begins Job #2 at time 10.
The queue is now empty.
Printer A(89%,84%) ends Job #1 at time 11.
Job #3 arrives at time 18 with 36 pages.
The queue now contains 1 job: [#3(36)]
Printer A(89%,87%) begins Job #3 at time 18.
The queue is now empty.
Job #4 arrives at time 44 with 126 pages.
The queue now contains 1 job: [#4(126)]
Printer C(106%,102%) begins Job #4 at time 44.
The queue is now empty.
Printer B(97%,91%) ends Job #2 at time 53.
Printer A(89%,87%) ends Job #3 at time 60.
Job #5 arrives at time 78 with 170 pages.
The queue now contains 1 job: [#5(170)]
Printer A(89%,92%) begins Job #5 at time 78.
The queue is now empty.
Job #6 arrives at time 113 with 172 pages.
The queue now contains 1 job: [#6(172)]
Printer B(97%,95%) begins Job #6 at time 113.
The queue is now empty.
Job #7 arrives at time 121 with 40 pages.
The queue now contains 1 job: [#7(40)]
Printer D(128%,124%) begins Job #7 at time 121.
The queue is now empty.
Job #8 arrives at time 127 with 30 pages.
The queue now contains 1 job: [#8(30)]
Job #9 arrives at time 136 with 41 pages.
The queue now contains 2 jobs: [#8(30), #9(41)]
Job #10 arrives at time 140 with 20 pages.
The queue now contains 3 jobs: [#8(30), #9(41), #10(20)]
Job #11 arrives at time 147 with 31 pages.
The queue now contains 4 jobs: [#8(30), #9(41), #10(20), #11(31)]
Printer D(128%,124%) ends Job #7 at time 154.
Printer D(128%,126%) begins Job #8 at time 155.
The queue now contains 3 jobs: [#9(41), #10(20), #11(31)]
Job #12 arrives at time 160 with 63 pages.
The queue now contains 4 jobs: [#9(41), #10(20), #11(31), #12(63)]
Printer C(106%,102%) ends Job #4 at time 168.
Printer C(106%,104%) begins Job #9 at time 169.
The queue now contains 3 jobs: [#10(20), #11(31), #12(63)]
Printer D(128%,126%) ends Job #8 at time 179.
Printer D(128%,118%) begins Job #10 at time 180.
The queue now contains 2 jobs: [#11(31), #12(63)]
Printer D(128%,118%) ends Job #10 at time 197.
Printer D(128%,138%) begins Job #11 at time 198.
The queue now contains 1 job: [#12(63)]
Printer C(106%,104%) ends Job #9 at time 209.
Printer C(106%,96%) begins Job #12 at time 210.
The queue is now empty.
```

In the output, each print job is identified by its ID number and also its size. For example, #1(7) means that Job #1 has 7 pages to be printed, and #8 (30) means that Job #8 has 30 pages.

Effective simulation requires the use of randomly generated numeric input. This simulation should use three random number generators: one to generate the average print rates for each printer, one to generate the actual print rates for each printer's print job, and one to generate the time elapsed between job arrivals. They are instances of the following extension of the java.util.Random class, which you need to use in your application (with appropriate Javadocs added).

```
package printqueue;
//  A Random class extension for a Client/Server Simulation

public class Random extends java.util.Random
{
    private double mean;
    private double standardDeviation;

    public Random(double mean)
    {
        this.mean = mean;
        this.standardDeviation = mean;
    }

    public Random(double mean, double standardDeviation)
    {
        this.mean = mean;
        this.standardDeviation = standardDeviation;
    }

    @Override
     /**
    * @return a double random number that is normally distributed with
     * the given mean and standard deviation
    */
    public double nextGaussian()
    {
        double x = super.nextGaussian();  // x = normal(0.0, 1.0)
        return x*standardDeviation + mean;
    }

    /**
    * @return a double random number that is exponentially distributed
     * with the given mean
    */

    public double nextExponential()
    {
        return -mean*Math.log(1.0 - nextDouble());
    }

    public int intNextExponential()
    {
        return (int)Math.ceil(nextExponential());
    }
}
```

The nextGaussianO method returns random numbers that are normally distributed with the given mean and standard deviation. It invokes and overrides the synonymous method in the java.util.Random class, which returns random numbers that are normally distributed with mean 0.0 and standard deviation 1.0. The nextExponential() method returns random numbers that are exponentially distributed with the given mean. This is the correct distribution for unbiased interarrival times. It is also used to generate the job sizes, which determine the service times.



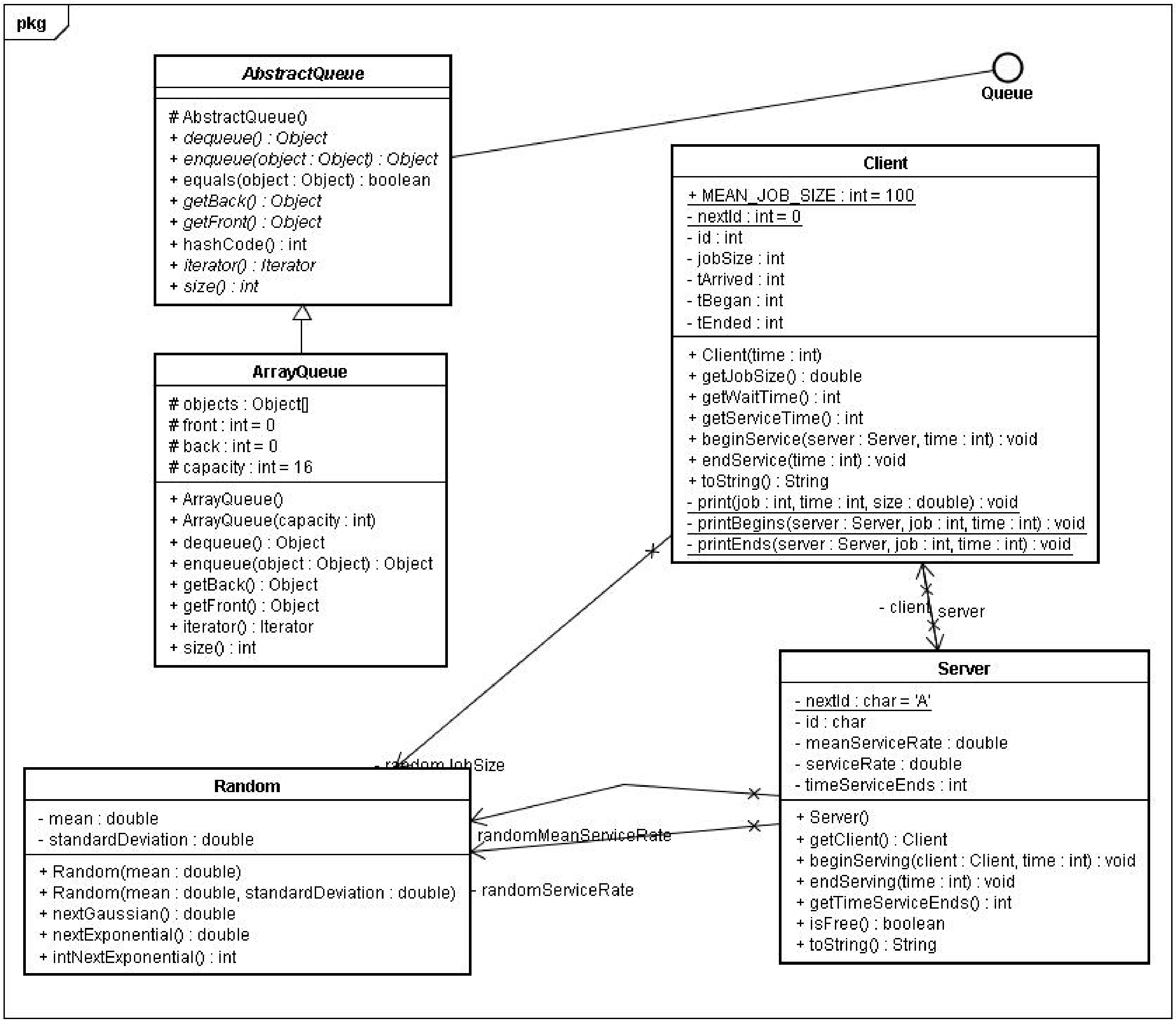
Project Requirements. Your program must contain at least the following classes:

1. PrintQueueSimulationGUI--the simulation driver class. A GUI-based JFrame that allows the user to set/select number of servers (printers), set/select mean interarrival time, and set/select mean duration of each job. This class should contain as instance variables and array of servers (printers), a queue and a random instance variable. It should manage arrival of clients, serve them, enqueue and dequeue them in the queue and display the results.

2. Client--Each print job has an instance of this class. It contains a random number generator randomJobSize that generates the exponentially distributed job sizes with a mean of 100 pages. It is declared static because only one instance is needed to produce all the job sizes. Similarly, the static int nextId is used to generate identification numbers for all the jobs. The constructor uses the next Id counter to set the job id, and it uses the randomJobSize generator to set the jobSize. Then it prints one line of output, announcing that that job has arrived. The beginService () method assigns the server reference to the printer that invoked it and then prints one line of output, announcing that the printing has begun. Similarly, the endService() method nullifies the server reference after printing one line of output that announces that the printing has ended.
3. Server--It has a random number generator randomMeanServiceRate that generates the normally distributed rates with mean 100.0 and standard deviation 20.0. It produces the meanServiceRate for each printer. In the run just shown, it produced the rates 89 for Printer A, 97 for Printer B, 106 for Printer C, and 128 for Printer D. Similarly, the random number generator randomServiceRate generates the normally distributed rates for each print job. In the run just shown, it produced the rates 84 for Job #1, 87 for Job #3, and 92 for Job #5. Those came from a normal distribution with mean 89 (for Printer A). The standard deviation is set at 10 for each printer's distribution. The beginServing () method assigns the client reference to the client job that it is printing and obtains the normally distributed serviceRate from the randomServiceRate generator. Then it sends the beginService message to its client print job. Next, the assignment int serviceTime = (int)Math.ceil(client.getJobSize()/serviceRate); computes the time (number of seconds) that it will take to do the print job by dividing the job size (the number of pages) by the printing rate (pages per second). The integer ceiling of this ratio is used as a count of the number of seconds to elapse. This count is then added to the current time to initialize the timeServiceEnds field of the Server object.
4. Random--extends java.utilRandom. Given above.
5. AbstractQueue--an abstract class that extends AbstractCollection and Implements Queue interface.
6. Queue--an interface that extends Collection. Public method include enqueue, dequeue, getBack, and getFront. You may decide to implement the Queue class differently--it need not be an interface.
7. ArrayQueue--extends AbstractQueue. Overrides all methods from the AbstractQueue.
8. Splash screen with which the program begins and an About form which describes the project among other info (copyright, warning, logo, etc.).
9. A sound data structure of the Person class to hold the preference data. This could be an array, ArrayList, LinkedList (preferable), HashMap, or any other structure you desire.
10. Javadocs, description of the program, and comments, comments everywhere.
11. Menus that synchronize with corresponding buttons and with at least the following menu choices:
    - File with Open, Clear, Print, and Exit menu items.

- Statistics displaying all averages (see extra credit below).
- Help with About menu item for an About form.

12. The project should start with a Splash Screen that closes itself after so many seconds and it should contain an About form activated from the Help menu.
13. Well designed and efficient GUI with images and ease to use.



Project Grading Guidelines: Your project will be graded on the following:

- Correct solution to the proposed problem.
- GUI design.
- Elegance, simplicity, style, and readability of the algorithms and code.
- Comments:
    - ✓ Use of Javadocs.
    - ✓ Description of project.
    - ✓ Description of procedures.
- 30 points maximum. Extra credit points are optional and not required.

EXTRA CREDIT (Optional). There are many other improvements one could make for this project: consider, for example,

- Provide a animation for the project. Show at least the content of the queue in a graphically pleasing way.
- You can obtain lots of extra credit by displaying the following statistics:
  1. The average number of jobs in the system.
  2. The average number of jobs in the queue.
  3. The average time that a job spends in the system.
  4. The average time that a job spends in the queue.
  5. The average service time for each server.
  6. The average service time for all jobs.
  7. The percent of time that each server is idle.

Sample run for the extra credit:

```
run:
Mean interarrival time: 20.0
Mean service time: 100
Number of servers: 4
Average number of jobs in the system: 2.38
Average number of jobs in the queue: 0.0
Average time a job spent in the system: 238.0
Average time a job spent in the queue: 0.0
Total time spent by all jobs in system: 238
Total time spent by all jobs in queue:  0
Duration:                  100
Total number of jobs completed: 1
Average service time for server:
        A(112%,111%): NaN
        B(109%,99%): 66.0
        C(114%,107%): NaN
        D(120%,118%): NaN
Average service time among all jobs: 66.0
Percent idle time for server:
        A(112%,111%): 100.0%
        B(109%,99%): 34.0%
        C(114%,107%): 100.0%
        D(120%,118%): 100.0%
BUILD SUCCESSFUL (total time: 0 seconds)
```

Pay attention to the following criteria for grading:

```
/*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
'Comments by the prof:
'Great effort.  Here are suggestions for improvement: in the works.
'1. Use Javadoc comments throughout the program, not just at the top heading.
'2. Need at least the specified classes.
'3. Reconsider the design of your classes.
'4. It is good style to tab code inside a method.
'5. Declare all constants if there are any.
'6. Implement printing--it is required.
'7. Display jobs as they arrive in the queue with ID (#), arrival time and pages to
be printed.
'8. Display status of the queue: which job # is in it with how many pages to print
and in what order. This should be done with a print method which will print the
queue.
'9. Display status of the printers as they begin each job. Include the printer rate,
job rate, job ID printed, and at what time it started the job.
'10. Add the ability to change the number of printers.
'11. Add the ability to change the mean job size.
'12. Add the ability to change the mean interarrival time.
'13. Add the ability to change the duration.
'14. Add the ability to change the mean job size.
'15. more...
'21.
'22. Disable maximization of the form.
'23. Add an About form which describes the project.
'24. Add a Splash screen that starts the project, displayed in the middle of the
screen.
'25. Provide ToolTip to help user navigate through the form.
'26. Provide an interesting but functional GUI for the main form.
'27. More...
'30. Add icons to all forms.
'
'The ones that apply to your project are:
'
'4, 5, 12, 25
'26/30
'~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~*/
```