CS2030 (2310) Lab #4

Tags & Categories

Tags:

Categories:

Related Tutorials

Task Content

Lab #4

This is the final milestone for the Discrete Event Simulator.

There are two types of servers: human servers and self-checkout counters.

Human Server

Human servers (or simply called servers) are allowed to take occasional breaks. When a server finishes serving a customer, there is a chance that the server takes a rest for a certain amount of time. During the break, the server does not serve the next waiting customer. Upon returning from the break, the server serves the next customer in the queue (if any) immediately.

Just like customer service time, we cannot pre-determine what rest time is accorded to which server at the beginning; it can only be decided during the simulation. That is to say, whenever a server rests, it will then know how much time it has to rest. The rest time is supplied by a random number generator as shown in the program `Main.java` provided.

We use an instance of `java.util.Random` to generate a stream of pseudorandom numbers. A `Random` object is created using a single seed value of type `long`. The seed is used to initialize the internal state of the pseudorandom number generator. Random numbers can then be generated via methods `next()`, `nextInt()`, `nextDouble()`, etc. The example below shows how random floating point values of type `double` can be generated.

```
jshell> new Random(1L)
$.. ==> java.util.Random@6e8cf4c6

jshell> new Random(1L).nextDouble()
$.. ==> 0.7308781907032909

jshell> new Random(1L).nextDouble()
$.. ==> 0.7308781907032909

jshell> Random rand = new Random(1L)
rand ==> java.util.Random@5b6f7412

jshell> rand.nextDouble()
$.. ==> 0.7308781907032909

jshell> rand.nextDouble()
$.. ==> 0.41008081149220166
```

Notice that the first random value generated right after seeding with the same seed (here we use the `long` value representation of one, or `1L`) is always the same. Hence, every unique seed provides a unique sequence of random numbers via subsequent calls to `nextDouble()`.

Self-Checkout Counters

As with all modern supermarkets, there are now a number of self-checkout counters being set up. In particular, if there are `k` human servers, then the self-checkout counters are identified from `k + 1` onwards.

Take note of the following:

- All self-checkout counters share the same queue.
- Unlike human servers, self-checkout counters do not rest.
- When we print out the wait event, we say that the customer is waiting for the self-checkout counter `k + 1`, even though this customer may eventually be served by another self-checkout counter.

Use the program `Main.java` provided to test your program with a default service time of `1.0`. As usual, the utility classes `Pair`, `PQ` and `ImList` have been provided to you. Note that your program will be tested against test cases where the service times could be different when serving different customers.

User input starts with values representing the number of servers, the number of self-check counters, the maximum queue length and the probability of a server resting. This is followed by the arrival times of the customers. Lastly, a number of service times (could be more than necessary) are provided.

Here is the sample run from the last lab, with two human servers, no self-checkouts, maximum queue length of `2` and probablity of rest set to `0.0`.

```
$ cat 1.in
2 0 2 0
0.500
0.600
0.700
1.500
1.600
1.700

$ cat 1.in | java Main
0.500 1 arrives
0.500 1 serves by 1
0.600 2 arrives
0.600 2 serves by 2
0.700 3 arrives
0.700 3 waits at 1
1.500 1 done serving by 1
1.500 3 serves by 1
1.500 4 arrives
1.500 4 waits at 1
1.600 2 done serving by 2
1.600 5 arrives
1.600 5 serves by 2
1.700 6 arrives
1.700 6 waits at 1
2.500 3 done serving by 1
2.500 4 serves by 1
2.600 5 done serving by 2
3.500 4 done serving by 1
3.500 6 serves by 1
4.500 6 done serving by 1
[0.600 6 0]
```

Here is another sample run with the same shop set-up as above, but serving ten other customers.

```
$ cat 2.in
2 0 2 0
0.000000
0.313508
1.204910
2.776499
3.876961
3.909737
9.006391
9.043361
9.105379
9.159630

$ cat 2.in | java Main
0.000 1 arrives
0.000 1 serves by 1
```

```
0.314 2 arrives
0.314 2 serves by 2
1.000 1 done serving by 1
1.205 3 arrives
1.205 3 serves by 1
1.314 2 done serving by 2
2.205 3 done serving by 1
2.776 4 arrives
2.776 4 serves by 1
3.776 4 done serving by 1
3.877 5 arrives
3.877 5 serves by 1
3.910 6 arrives
3.910 6 serves by 2
4.877 5 done serving by 1
4.910 6 done serving by 2
9.006 7 arrives
9.006 7 serves by 1
9.043 8 arrives
9.043 8 serves by 2
9.105 9 arrives
9.105 9 waits at 1
9.160 10 arrives
9.160 10 waits at 1
10.006 7 done serving by 1
10.006 9 serves by 1
10.043 8 done serving by 2
11.006 9 done serving by 1
11.006 10 serves by 1
12.006 10 done serving by 1
[0.275 10 0]
```

Here is the sample run for the same ten customers above, with two servers, no self-checkouts and probability of rest set to `0.5`.

```
$ cat 3.in
2 0 2 0.5
0.000000
0.313508
1.204910
2.776499
3.876961
3.909737
9.006391
9.043361
9.105379
9.159630

$ cat 3.in | java Main
0.000 1 arrives
0.000 1 serves by 1
0.314 2 arrives
0.314 2 serves by 2
1.000 1 done serving by 1
1.205 3 arrives
1.205 3 serves by 1
1.314 2 done serving by 2
2.205 3 done serving by 1
2.776 4 arrives
2.776 4 waits at 1
3.053 4 serves by 1
3.877 5 arrives
3.877 5 waits at 1
3.910 6 arrives
3.910 6 waits at 1
4.053 4 done serving by 1
4.053 5 serves by 1
5.053 5 done serving by 1
5.902 6 serves by 1
6.902 6 done serving by 1
9.006 7 arrives
9.006 7 serves by 1
9.043 8 arrives
9.043 8 serves by 2
```

```
9.105 9 arrives
9.105 9 waits at 1
9.160 10 arrives
9.160 10 waits at 1
10.006 7 done serving by 1
10.006 9 serves by 1
10.043 8 done serving by 2
11.006 9 done serving by 1
11.006 10 serves by 1
12.006 10 done serving by 1
[0.519 10 0]
```

Lastly, the same ten customers with one human server, two self-checkout counters and probability of rest set to `0.5`.

```
$ cat 4.in
1 2 2 0.5
0.000000
0.313508
1.204910
2.776499
3.876961
3.909737
9.006391
9.043361
9.105379
9.159630

$ cat 4.in | java Main
0.000 1 arrives
0.000 1 serves by 1
0.314 2 arrives
0.314 2 serves by self-check 2
1.000 1 done serving by 1
1.205 3 arrives
1.205 3 serves by 1
1.314 2 done serving by self-check 2
2.205 3 done serving by 1
2.776 4 arrives
2.776 4 serves by self-check 2
3.776 4 done serving by self-check 2
3.877 5 arrives
3.877 5 serves by self-check 2
3.910 6 arrives
3.910 6 serves by self-check 3
4.877 5 done serving by self-check 2
4.910 6 done serving by self-check 3
9.006 7 arrives
9.006 7 serves by 1
9.043 8 arrives
9.043 8 serves by self-check 2
9.105 9 arrives
9.105 9 serves by self-check 3
9.160 10 arrives
9.160 10 waits at 1
10.006 7 done serving by 1
10.043 8 done serving by self-check 2
10.105 9 done serving by self-check 3
10.854 10 serves by 1
11.854 10 done serving by 1
[0.169 10 0]
```

Submission (Course)

Select course: CS2030 (2023/2024 Sem 1) - Programming Methodology II

Your Files:

BROWSE

SUBMIT (only .java, .c, .cpp, .h, .jsh, and .py extensions allowed)

```
To submit multiple files, click on the Browse button, then select one or more files.
The selected file(s) will be added to the upload queue. You can repeat this step to add
more files. Check that you have all the files needed for your submission. Then click on
the Submit button to upload your submission.
```