Lab 3

Submission deadline: 2359, Sunday, September 17, 2017.

Prerequisites

This lab assumes that students:

- have already attempted Lab 2
- have an understanding of the customer/server system being simulated

Learning Objectives

After completing this lab, students should:

- be familiar with the CS2030 Java Coding Style [../style/index.html] and comfortable in following them
- be familiar with `javadoc` syntax [../javadoc/index.html] and comfortable with documenting the code with `javadoc`
- be familiar with generating `javadoc` documentation
- be comfortable reading a Java Collection Framework documentation and use one of the classes provided
- appreciate the usefulness of Java Collection Framework by seeing how much shorter and cleaner the resulting code is
- be familiar with random number generation in `java.lang.Math`
- be exposed to the concept of pseudo random number generator and seeds

Setup

There is no new skeleton code provided. You are to build from your Lab 3 solution based on your Lab 2. To setup Lab 3, do the following.

- Login to `cs2030-i`
- Copy `~/lab02` to `~/lab03`
- Rename `LabTwo.java` to `LabThree.java`
- Rename the class `LabTwo` to `LabThree`
- Copy the test data (`TESTDATA1.txt` .. `TESTDATA5.txt`) from `~cs2030/lab03` to `~/lab03`

If you are still not familiar with how to do the above, please revisit the UNIX [../unix/index.html] guide.

Task

For Lab 3, you will be asked to make two small changes to the code:

- Make the arrival time and service time random
- Use `PriorityQueue<Event>` instead of `Event[]` to schedule the event.

As a result of this, you might realize that there is a better way to encapsulate the data and the behavior of the various entities in the program. In which case, you may want to reorganize your classes, create new classes, etc. Depending on how "changeable" your Lab 2 solution is, you may have ended up with more than two small changes.

In addition, you should edit your code so that:

- it follows the CS2030 Coding Style [../style/index.html]
- it is clearly documented with `javadoc` comments [../javadoc/index.html]

Grading

This lab contributes another 4 marks to your final grade (100 marks).

- 1 marks for coding style
- 1 marks for javadoc
- 1 mark for implementation *and encapsulation* of arrival time/service time generation and the priority queue
- 1 mark for correctness

Priority Queuing

The first change you need to do in this assignment is to use one of the Java Collection classes to manage the events. In `LabTwo.java` , we kept all the events in an array, and scanned through it to find the event with the smallest (i.e., earliest) timestamp. This is not efficient, since scanning through all the events takes time that increases linearly with the number of events [1] [#fn:2].

Java Collection provides a class that is perfect for our use: `PriorityQueue<E>` [https://docs.oracle.com/javase/8/docs/api/java/util/PriorityQueue.html]. A `PriorityQueue` keeps a collection of elements, the elements are given certain priority. Elements can be added with `add(E e)` [https://docs.oracle.com/javase/8/docs/api/java/util/PriorityQueue.html#add-E-] method. To retrieve and remove the elements with highest priority, we use the `poll()` [https://docs.oracle.com/javase/8/docs/api/java/util/PriorityQueue.html#poll--] method, which returns an object of type `E` , or `null` is the queue is empty.

In our case, the event with the smallest timestamp has the highest priority. To tell the `PriorityQueue<E>` class how to order the events so that smaller timestamp has higher priority, we use the `PriorityQueue<E>` constructor [https://docs.oracle.com/javase/8/docs/api/java/util/PriorityQueue.html#PriorityQueue-java.util.Comparator-] that takes in a `Comparator` object, just like we see in Lecture 5 [../lec5/index.html].

If you design is right, you should only change the code in four places: (i) initialize list of events, (ii) schedule an event, (iii) get the next event, (iv) checking if there is still an event.

(Hint: You should be able to implement a `Comparator` without getter `getTime()`)

You should implement this change first, since you can do a sanity check of your correctness against the result of Lab 2 using the test data *from* Lab 2.

Randomized Arrival and Service Time

Next, we are going to change how the arrival time and service time is specified, so that we can easily simulate different settings (e.g., a more efficient server with faster service time, more arrivals during weekends, etc).

Random

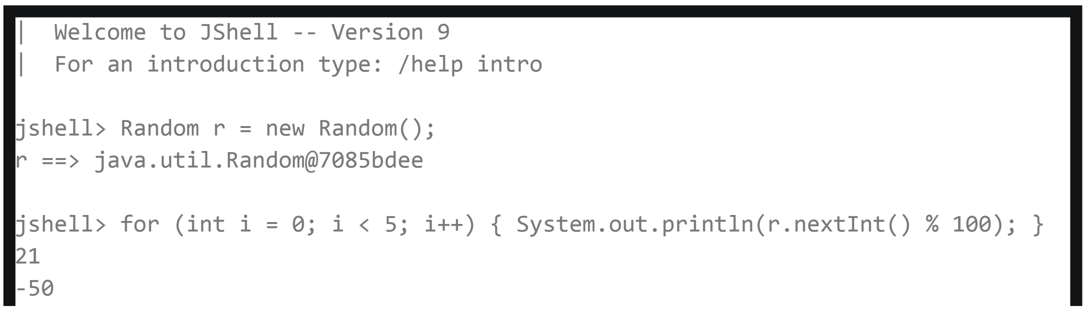
First, an introduction to random number generation. A random number generator is an entity that spews up one random number after another. We, however, cannot generate a truly random number algorithmically. We can only generate a *pseudo* random number. A pseudo random number generator can be initialized with a *seed.* A pseudo random number generator, when initialized with the same *seed*, always produces the same sequence of (seemingly random) numbers.

Java provides a class `java.util.Random` that encapsulates a pseudo random number generator. We can create a random number generator with a seed:

```
Random rng = new Random(1);
```

We can then call `rng.nextDouble()` repeatedly to generate random numbers between 0 and 1.

In the demo below, we see that creating a `Random` object with the same seed of 2 towards the end leads to the same sequence of random doubles being generated.



```
|  Welcome to JShell -- Version 9
|  For an introduction type: /help intro

jshell> Random r = new Random();
r ==> java.util.Random@7085bdee

jshell> for (int i = 0; i < 5; i++) { System.out.println(r.nextInt() % 100); }
21
-50
```

Using a fixed seed is important for testing, since the execution of the program will be deterministic, even when random numbers are involved.

Arrival Time

In Lab 2, the arrival time is given in the input text file. This approach is less flexible and requires another program to generate the input file. Further, the original code creates *all* the arrival events before the simulation starts, and therefore limits the total number of arrivals to the size of the initial array `events`.

We are going to improve this part of the program, by generating the arrival one after another. To do this, we need to generate a *random inter-arrival time.* The inter-arrival time is usually modeled as a exponential random variable, characterized by a single parameter λ (`lambda`), known as *arrival rate.*

Mathematically, the inter-arrival time can be generated with $-\ln(U)/\lambda$, where U is a random variable between 0 and 1 [2] [#fn:1].

- Every time an arrival event is processed, it generates another arrival event and schedule it.
- If there are still more customer to simulator, we generate the next arrival event with a timestamp of T + now, where T is generated with the Java expression:

```java
-Math.log(rng.nextDouble()) / lambda
```

- When we first start the simulator, we need to generate the first arrival event with timestamp T, generated with the same expression as above.

You can adapt the expression above to suit your program.

Service Time

In Lab 2, the service time is constant, which is not always realistic. We are going to model the service time as a exponential random variable, characterized with a single parameter, *service rate* μ (`mu`). We can generate the service time with the expression $-\ln(U)/\mu$, where U is a random variable between 0 and 1.

- Every time a customer is being served, we generate a "done" event and schedule it (just like we did it in Lab 2).
- The "done" event generated will have a timestamp of T + now, where T is *no longer constant* `SERVICE_TIME`, but instead is generated with the Java expression:

```
    -Math.log(rng.nextDouble()) / mu
```

You can adapt the expression above to suit your program.

Note that *we should only have a single random number generator* in the simulation. (hint: what access modifier should we use?)

Input and Output

With this change, the input file should now contain only the *simulator parameters.* It has four lines:

- The first line is a `int` value, indicating the seed to the simulator
- The second line is a `double` value, indicating the service rate μ
- The third line is a `double` value, indicating the arrival rate λ
- The fourth line is a `int` value, indicating the total number of customers to simulate

We give you five test inputs: `TESTDATA1.txt` to `TESTDATA5.txt`. The output we get with these inputs are:

```
ooiwt@cs2030-i:~/lab03> java LabThree < TESTDATA1.txt | tail -1
0.000 1 0
```

```
ooiwt@cs2030-i:~/lab03> java LabThree < TESTDATA2.txt | tail -1
0.924 3 2
ooiwt@cs2030-i:~/lab03> java LabThree < TESTDATA3.txt | tail -1
0.019 10 0
ooiwt@cs2030-i:~/lab03> java LabThree < TESTDATA4.txt | tail -1
0.832 3 7
ooiwt@cs2030-i:~/lab03> java LabThree < TESTDATA5.txt | tail -1
0.530 634 366
```

Your Task

Your mission, in Lab 3, is update your solution in Lab 2 to:

- follow the style guideline
- add `javadoc` comments to code (`javadoc` should not produce any warning)
- use `PriorityQueue<E>` to manage the events
- use randomly generated service time and arrival time, specified by a new input file format
- improve your existing encapsulations, create new encapsultions, etc. if necessary

The `main` method should remain in a class named `LabThree` . We must be able to run your code with:

```
javac *.java
java LabThree < TESTDATA1.txt
```

You also must not change the formatting of the *last line* of output:

```
System.out.printf("%.3f %d %d", ..")
```

We rely on it to check for correctness of your logic.

Submission

When you are ready to submit your lab, on `cs2030-i` , run the script

```
~cs2030/submit03
```

which will copy all files matching `*.java` (and nothing else) from your `~/lab03` directory on `cs2030-i` to an internal grading directory. We will test compile and test run with a tiny sample input to make sure that your submission is OK.

You can submit multiple times, but only the most recent submission will be graded.

---

1. For those who are taking CS2040, we say this is $O(n)$ time. A heap-based priority queue, on the other hand, takes $O(logn)$ time.
2. If you are not familiar with exponential distribution and random variable, do not worry, the code is being given to you. These concepts are covered in ST2334.