CS2030 Project: Discrete Event Simulator

Tags & Categories

Tags:

Categories:

Related Tutorials

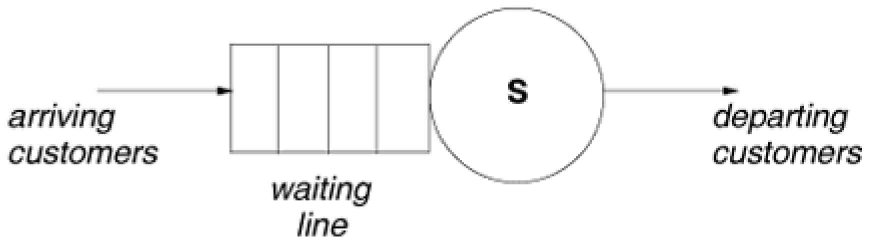
Task Content

CS2030 Project — Discrete Event Simulator

A discrete event simulator is a software that simulates the changes in the state of a system across time, with each transition from one state of the system to another triggered via an event. Such a system can be used to study many complex real-world systems such as queuing to order food at a fast-food restaurant.

An event occurs at a particular time, and each event alters the state of the system and may generate more events. States remain unchanged between two events (hence the term **discrete**), and this allows the simulator to jump from the time of one event to another. Some simulation statistics are also typically tracked to measure the performance of the system.

The following illustrates a queuing system comprising a single service point **S** with one customer queue.



In this exercise, we consider a multi-server system comprising the following:

- There are a number of **servers**; each server can serve one customer at a time.
- Each customer has a **service time** (time taken to serve the customer).
- When a **customer** arrives (`ARRIVE` event):
  - if the server is idle (not serving any customer), then the server starts serving the customer immediately (`SERVE` event).
  - if the server is serving another customer, then the customer that just arrived waits in the queue (`WAIT` event).
  - if the server is serving one customer with a second customer waiting in the queue, and a third customer arrives, then this latter customer leaves (`LEAVE` event). In other words, there is at most one customer waiting in the queue.
  - When the server is done serving a customer (`DONE` event), the server can start serving the customer waiting at the front of the queue (if any).
- If there is no waiting customer, then the server becomes idle again.

Notice from the above description that there are five events in the system, namely: `ARRIVE`, `SERVE`, `WAIT`, `LEAVE` and `DONE`. For each customer, these are the only possible event transitions:

- ARRIVE → SERVE → DONE
- ARRIVE → WAIT → SERVE → DONE
- ARRIVE → LEAVE

In essence, an event is tied to one customer. Depending on the current state of the system, triggering an event will result in the next state of the system, and possibly the next event. Events are also processed via a queue. At the start of the simulation, the queue only contains the customer arrival events. With every simultation time step, an event is retrieved from the queue to be processed, and any resulting event added to the queue. This process is repeated until there are no events left in the queue.

Priority Queuing

The `PriorityQueue` (a mutable class) can be used to keep a collection of elements, where each element is given a certain priority.

- Elements may be added to the queue with the `add(E e)` method. The queue is modified;
- The `poll()` method may be used to retrieve and remove the element with the highest priority from the queue. It returns an object of type `E`, or `null` if the queue is empty. The queue is modified.

Take Note

The requirements of this project will be released in stages. For each level `N`, you will need to create a driver class `MainN` in the file `MainN.java`.

All classes related to the simulation are to be packaged in `cs2030.simulator`, with `Simulator` as the only `public` facing class. The only outside-package clients allowed are the `MainN` driver classes. Define a `main` method in each driver class that reads input and passes the input to the `Simulator::simulate(..)` method. This method should be overloaded due to the different level requirements.

You are encouraged to write `assert` assertion statements to check for bugs.

Level 0

The objective of this level is to provide guidance in setting up some of the classes, as well as the priority queue for the actual simulation. As this level is not tested, feel free to skip the level entirely and proceed to the next level.

Define a driver class `Main` that reads a series of distinct arrival times (not necessarily in chronological order) for each customer, creates events out of them and adds them to a priority queue. The priority queue is then polled so that the earlier arrival event is removed from the queue and output.

A skeleton class is given below:

```
import java.util.Scanner;
import java.util.List;
import java.util.ArrayList;

class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        List<Double> arrivalTimes = new ArrayList<Double>();

        while (sc.hasNextDouble()) {
            arrivalTimes.add(sc.nextDouble());
        }
        Simulator s = new Simulator(arrivalTimes);
        s.simulate();
    }
}
```

The sequence of input arrival times are associated with each `Customer` having an identifier `1`, `2`, `3`, ... respectively. For each of the arrival events, create an `Event` that associates to the customer. Note that since we only consider arrival events here, it suffices to have one `Event` class. However, you would soon realize there are more types of events to be considered.

Create a `PriorityQueue` with an appropriate `Comparator` that implements `Comparator`. In your `Comparator` class, define the `compare` method that takes in two events, and returns an appropriate integer value depending on whichever is the earlier arrival event.

The following code fragment demonstrates how the priority queue `pq` is polled. Events are polled and removed from `pq` until it becomes empty.

```
while (!pq.isEmpty()) {
    Event event = pq.poll();
    System.out.println(event);
}
```

Sample runs of the program is given below. User input is underlined. Input is terminated with a `^D` (CTRL-d).

```
0.500
0.600
```

```
0.700
^D
0.500 1 arrives
0.600 2 arrives
0.700 3 arrives
```

```
1.500
0.600
0.700
^D
0.600 2 arrives
0.700 3 arrives
1.500 1 arrives
```

As you proceed through the levels, you would realize that some events would create other events which need to be added to the priority queue.

Level 1

Given the number of servers and a set of customer arrival times in chronological order, output the discrete events. Also output the statistics at the end of the simulation.

Define a driver class `Main1` (stored in `Main1.java`) that reads a series of arrival times, creates the arrival events, and starts the simulation.

Take note of the following assumptions:

- The format of the input is always correct;
- Output of a `double` value, say `d`, is to be formatted with `String.format("%.3f", d);`

As an example, given the arrival times (represented as a floating point value for simplicity) of three customers and one server, and assuming a constant service time of `1.0`,

```
0.500
0.600
0.700
```

The simulation starts with three arrival events

```
<0.500 1 arrives>
<0.600 2 arrives>
<0.700 3 arrives>
```

The next event to pick is `<0.500 1 arrives>`. This schedules a `SERVE` event.

```
<0.500 1 serves by server 1>
<0.600 2 arrives>
<0.700 3 arrives>
```

The next event to pick is `<0.500 1 served>`. This schedules a `DONE` event.

```
<0.600 2 arrives>
<0.700 3 arrives>
<1.500 1 done serving by server 1>
```

The next event to pick is `<0.600 2 arrives>`...

This process is repeated until there are no more events.

Using our example, the entire similation run results in the following output, each of the form `<time_event_occurred, customer_id, event_details>`

```
0.500 1 arrives
0.500 1 serves by server 1
0.600 2 arrives
0.600 2 waits at server 1
0.700 3 arrives
0.700 3 leaves
1.500 1 done serving by server 1
1.500 2 serves by server 1
2.500 2 done serving by server 1
```

Finally, statistics of the system that we need to keep track of are:

1. the average waiting time for customers who have been served;
2. the number of customers served;
3. the number of customers who left without being served.

In our example, the end-of-simulation statistics are respectively, `[0.450 2 1]`.

Sample runs of the program is given below. User input is underlined, and starts with an integer value representing the number of servers, followed by the arrival times of the customers. Input is terminated with a `^D` (CTRL-d).

You will also need to compute the statistics (see problem description above) at the end of the simulation.

```
1
0.500
0.600
0.700
^D
0.500 1 arrives
0.500 1 serves by server 1
0.600 2 arrives
0.600 2 waits at server 1
0.700 3 arrives
0.700 3 leaves
1.500 1 done serving by server 1
1.500 2 serves by server 1
2.500 2 done serving by server 1
[0.450 2 1]
```

```
1
0.500
0.600
0.700
1.500
1.600
1.700
^D
0.500 1 arrives
0.500 1 serves by server 1
0.600 2 arrives
0.600 2 waits at server 1
0.700 3 arrives
0.700 3 leaves
1.500 1 done serving by server 1
1.500 2 serves by server 1
1.500 4 arrives
1.500 4 waits at server 1
1.600 5 arrives
1.600 5 leaves
1.700 6 arrives
1.700 6 leaves
2.500 2 done serving by server 1
2.500 4 serves by server 1
3.500 4 done serving by server 1
[0.633 3 3]
```

```
2
0.500
0.600
0.700
1.500
1.600
1.700
^D
0.500 1 arrives
0.500 1 serves by server 1
0.600 2 arrives
0.600 2 serves by server 2
0.700 3 arrives
0.700 3 waits at server 1
1.500 1 done serving by server 1
1.500 3 serves by server 1
1.500 4 arrives
1.500 4 waits at server 1
1.600 2 done serving by server 2
1.600 5 arrives
1.600 5 serves by server 2
1.700 6 arrives
```

```
1.700 6 waits at server 2
2.500 3 done serving by server 1
2.500 4 serves by server 1
2.600 5 done serving by server 2
2.600 6 serves by server 2
3.500 4 done serving by server 1
3.600 6 done serving by server 2
[0.450 6 0]
```

Level 2

Rather than a constant service time, each customer now has its own service time. In addition, each server now has a queue of customers to allow multiple customers to queue up. A customer that chooses to join a queue joins at the tail. When a server is done serving a customer, it serves the next waiting customer at the head of the queue. Hence, the queue should be a first-in-first-out (FIFO) structure.

Define a driver class `Main2.java` (stored in `Main2.java`)that reads the number of servers, the maximum queue length, as well as arrival and service times of each customer. The simulation is then performed and statistics output at the end.

Sample runs of the program is given below. User input is underlined. The first line comprises an integer value representing the number of servers followed by an integer value for the maximum queue length. This is followed by the arrival and service of the customers. Input is terminated with a `^D` (CTRL-d).

```
2 1
0.500 1.000
0.600 1.000
0.700 1.000
1.500 1.000
1.600 1.000
1.700 1.000
^D
0.500 1 arrives
0.500 1 serves by server 1
0.600 2 arrives
0.600 2 serves by server 2
0.700 3 arrives
0.700 3 waits at server 1
1.500 1 done serving by server 1
1.500 3 serves by server 1
1.500 4 arrives
1.500 4 waits at server 1
1.600 2 done serving by server 2
1.600 5 arrives
1.600 5 serves by server 2
1.700 6 arrives
1.700 6 waits at server 2
2.500 3 done serving by server 1
2.500 4 serves by server 1
2.600 5 done serving by server 2
2.600 6 serves by server 2
3.500 4 done serving by server 1
3.600 6 done serving by server 2
[0.450 6 0]
```

```
1 1
0 0.313141
0.313508 0.103753
1.204909 0.699522
2.776498 0.014224
3.876961 0.154173
0.000 1 arrives
0.000 1 serves by server 1
0.313 1 done serving by server 1
0.314 2 arrives
0.314 2 serves by server 1
0.417 2 done serving by server 1
1.205 3 arrives
1.205 3 serves by server 1
1.904 3 done serving by server 1
2.776 4 arrives
2.776 4 serves by server 1
2.791 4 done serving by server 1
3.877 5 arrives
```

```
3.877 5 serves by server 1
4.031 5 done serving by server 1
[0.000 5 0]
```

```
2 1
0 0.313141
0.313508 0.103753
1.204909 0.699522
2.776498 0.014224
3.876961 0.154173
3.909736 0.012659
9.006390 1.477717
9.043360 2.593020
9.105379 0.296847
9.159629 0.051732
0.000 1 arrives
0.000 1 serves by server 1
0.313 1 done serving by server 1
0.314 2 arrives
0.314 2 serves by server 1
0.417 2 done serving by server 1
1.205 3 arrives
1.205 3 serves by server 1
1.904 3 done serving by server 1
2.776 4 arrives
2.776 4 serves by server 1
2.791 4 done serving by server 1
3.877 5 arrives
3.877 5 serves by server 1
3.910 6 arrives
3.910 6 serves by server 2
3.922 6 done serving by server 2
4.031 5 done serving by server 1
9.006 7 arrives
9.006 7 serves by server 1
9.043 8 arrives
9.043 8 serves by server 2
9.105 9 arrives
9.105 9 waits at server 1
9.160 10 arrives
9.160 10 waits at server 2
10.484 7 done serving by server 1
10.484 9 serves by server 1
10.781 9 done serving by server 1
11.636 8 done serving by server 2
11.636 10 serves by server 2
11.688 10 done serving by server 2
[0.386 10 0]
```

```
2 2
0 0.313141
0.313508 0.103753
1.204909 0.699522
2.776498 0.014224
3.876961 0.154173
3.909736 0.012659
9.006390 1.477717
9.043360 2.593020
9.105379 0.296847
9.159629 0.051732
0.000 1 arrives
0.000 1 serves by server 1
0.313 1 done serving by server 1
0.314 2 arrives
0.314 2 serves by server 1
0.417 2 done serving by server 1
1.205 3 arrives
1.205 3 serves by server 1
1.904 3 done serving by server 1
2.776 4 arrives
2.776 4 serves by server 1
2.791 4 done serving by server 1
3.877 5 arrives
3.877 5 serves by server 1
3.910 6 arrives
3.910 6 serves by server 2
```

```
3.922 6 done serving by server 2
4.031 5 done serving by server 1
9.006 7 arrives
9.006 7 serves by server 1
9.043 8 arrives
9.043 8 serves by server 2
9.105 9 arrives
9.105 9 waits at server 1
9.160 10 arrives
9.160 10 waits at server 1
10.484 7 done serving by server 1
10.484 9 serves by server 1
10.781 9 done serving by server 1
10.781 10 serves by server 1
10.833 10 done serving by server 1
11.636 8 done serving by server 2
[0.300 10 0]
```

Level 3

Implementing Server Rest

The servers are now 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.

Define a driver class `Main3.java` (stored in `Main3.java`) that first reads the number of servers, the maximum queue length and the number of customers *N*. This is followed by *N* lines, each representing the arrival time and service times of each customer. The next *N* lines depict a chronological sequence of resting times a server takes upon completion of each service (a value of `0` denotes no rest). Note that we do not pre-determine which rest time is accorded to which server at the beginning; it can only be decided during the actual simulation. That is to say, whenever a server rests, it will then know how much time it has to rest.

As usual, perform the simulation and output the statistics at the end.

Sample runs of the program is given below. User input is underlined. Input is terminated with a `^D` (CTRL-d).

Hint: To implement this behavior, you might want to consider additional event(s).

```
2 2 10
0 0.313141
0.313508 0.103753
1.204909 0.699522
2.776498 0.014224
3.876961 0.154173
3.909736 0.012659
9.006390 1.477717
9.043360 2.593020
9.105379 0.296847
9.159629 0.051732
0
0
0
0
0
0
0
0
0
0
0.000 1 arrives
0.000 1 serves by server 1
0.313 1 done serving by server 1
0.314 2 arrives
0.314 2 serves by server 1
0.417 2 done serving by server 1
1.205 3 arrives
1.205 3 serves by server 1
1.904 3 done serving by server 1
2.776 4 arrives
2.776 4 serves by server 1
2.791 4 done serving by server 1
3.877 5 arrives
3.877 5 serves by server 1
```

```
3.910 6 arrives
3.910 6 serves by server 2
3.922 6 done serving by server 2
4.031 5 done serving by server 1
9.006 7 arrives
9.006 7 serves by server 1
9.043 8 arrives
9.043 8 serves by server 2
9.105 9 arrives
9.105 9 waits at server 1
9.160 10 arrives
9.160 10 waits at server 1
10.484 7 done serving by server 1
10.484 9 serves by server 1
10.781 9 done serving by server 1
10.781 10 serves by server 1
10.833 10 done serving by server 1
11.636 8 done serving by server 2
[0.300 10 0]
```

```
2 2 10
0 0.313141
0.313508 0.103753
1.204909 0.699522
2.776498 0.014224
3.876961 0.154173
3.909736 0.012659
9.006390 1.477717
9.043360 2.593020
9.105379 0.296847
9.159629 0.051732
0
3.138762
0.847804
0
0.848968
0
0
3.863139
0
0
0.000 1 arrives
0.000 1 serves by server 1
0.313 1 done serving by server 1
0.314 2 arrives
0.314 2 serves by server 1
0.417 2 done serving by server 1
1.205 3 arrives
1.205 3 serves by server 2
1.904 3 done serving by server 2
2.776 4 arrives
2.776 4 serves by server 2
2.791 4 done serving by server 2
3.877 5 arrives
3.877 5 serves by server 1
3.910 6 arrives
3.910 6 serves by server 2
3.922 6 done serving by server 2
4.031 5 done serving by server 1
9.006 7 arrives
9.006 7 serves by server 1
9.043 8 arrives
9.043 8 serves by server 2
9.105 9 arrives
9.105 9 waits at server 1
9.160 10 arrives
9.160 10 waits at server 1
10.484 7 done serving by server 1
10.484 9 serves by server 1
10.781 9 done serving by server 1
11.636 8 done serving by server 2
14.644 10 serves by server 1
14.696 10 done serving by server 1
[0.686 10 0]
```

Level 4

Implementing Self-CheckOut Counters

There are now N_{self} self-checkout counters 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 always 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.

Define a driver class `Main4.java` (stored in `Main4.java`) that first reads the number of servers, the number of self-checkout counters, the maximum queue length and the number of customers *N*. This is followed by *N* lines, each representing the arrival time and service times of each customer. The next *N* lines depict a chronological sequence of resting times a server takes upon completion of each service (a value of `0` denotes no rest). Note that we do not pre-determine which rest time is accorded to which server at the beginning; it can only be decided during the actual simulation. That is to say, whenever a server rests, it will then know how much time it has to rest.

As usual, perform the simulation and output the statistics at the end.

Sample runs of the program is given below. User input is underlined. Input is terminated with a `^D` (CTRL-d).

```
2 0 2 20
0 0.313141
0.313508 0.103753
1.204909 0.699522
2.776498 0.014224
3.876961 0.154173
3.909736 0.012659
9.00639 1.477717
9.04336 2.59302
9.105379 0.296847
9.159629 0.368667
9.224613 0.051732
10.147993 3.489595
11.204932 0.006333
12.428914 0.160532
13.109177 2.86915
15.263626 0.576351
15.524295 0.89579
15.939973 1.04302
17.793096 3.007573
18.765423 0.74237
0
3.138762
0.847804
0
0.848968
0
0
3.863139
0
0
25.460873
0
0
0
0
36.96365
0
3.577445
2.169913
0
0.000 1 arrives
0.000 1 serves by server 1
0.313 1 done serving by server 1
0.314 2 arrives
0.314 2 serves by server 1
0.417 2 done serving by server 1
1.205 3 arrives
1.205 3 serves by server 2
```

```
1.904 3 done serving by server 2
2.776 4 arrives
2.776 4 serves by server 2
2.791 4 done serving by server 2
3.877 5 arrives
3.877 5 serves by server 1
3.910 6 arrives
3.910 6 serves by server 2
3.922 6 done serving by server 2
4.031 5 done serving by server 1
9.006 7 arrives
9.006 7 serves by server 1
9.043 8 arrives
9.043 8 serves by server 2
9.105 9 arrives
9.105 9 waits at server 1
9.160 10 arrives
9.160 10 waits at server 1
9.225 11 arrives
9.225 11 waits at server 2
10.148 12 arrives
10.148 12 waits at server 2
10.484 7 done serving by server 1
10.484 9 serves by server 1
10.781 9 done serving by server 1
11.205 13 arrives
11.205 13 waits at server 1
11.636 8 done serving by server 2
11.636 11 serves by server 2
11.688 11 done serving by server 2
11.688 12 serves by server 2
12.429 14 arrives
12.429 14 waits at server 2
13.109 15 arrives
13.109 15 waits at server 2
14.644 10 serves by server 1
15.013 10 done serving by server 1
15.178 12 done serving by server 2
15.178 14 serves by server 2
15.264 16 arrives
15.264 16 waits at server 1
15.338 14 done serving by server 2
15.338 15 serves by server 2
15.524 17 arrives
15.524 17 waits at server 2
15.940 18 arrives
15.940 18 waits at server 2
17.793 19 arrives
17.793 19 leaves
18.207 15 done serving by server 2
18.207 17 serves by server 2
18.765 20 arrives
18.765 20 waits at server 2
19.103 17 done serving by server 2
19.103 18 serves by server 2
20.146 18 done serving by server 2
40.474 13 serves by server 1
40.480 13 done serving by server 1
40.480 16 serves by server 1
41.056 16 done serving by server 1
57.110 20 serves by server 2
57.852 20 done serving by server 2
[6.025 19 1]
```

```
2 1 2 20
0 0.313141
0.313508 0.103753
1.204909 0.699522
2.776498 0.014224
3.876961 0.154173
3.909736 0.012659
9.00639 1.477717
9.04336 2.59302
9.105379 0.296847
9.159629 3.489595
```

```
9.224613 0.89579
10.147993 0.051732
11.204932 0.368667
12.428914 0.160532
13.109177 2.86915
15.263626 1.04302
15.524295 0.006333
15.939973 0.576351
17.793096 0.74237
18.765423 3.007573
0
3.138762
0.847804
0
0.848968
0
0
3.863139
0
0
25.460873
0
0
0
0
36.96365
0
3.577445
2.169913
0
0.000 1 arrives
0.000 1 serves by server 1
0.313 1 done serving by server 1
0.314 2 arrives
0.314 2 serves by server 1
0.417 2 done serving by server 1
1.205 3 arrives
1.205 3 serves by server 2
1.904 3 done serving by server 2
2.776 4 arrives
2.776 4 serves by server 2
2.791 4 done serving by server 2
3.877 5 arrives
3.877 5 serves by server 1
3.910 6 arrives
3.910 6 serves by server 2
3.922 6 done serving by server 2
4.031 5 done serving by server 1
9.006 7 arrives
9.006 7 serves by server 1
9.043 8 arrives
9.043 8 serves by server 2
9.105 9 arrives
9.105 9 serves by self-check 3
9.160 10 arrives
9.160 10 waits at server 1
9.225 11 arrives
9.225 11 waits at server 1
9.402 9 done serving by self-check 3
10.148 12 arrives
10.148 12 serves by self-check 3
10.200 12 done serving by self-check 3
10.484 7 done serving by server 1
10.484 10 serves by server 1
11.205 13 arrives
11.205 13 serves by self-check 3
11.574 13 done serving by self-check 3
11.636 8 done serving by server 2
12.429 14 arrives
12.429 14 serves by self-check 3
12.589 14 done serving by self-check 3
13.109 15 arrives
13.109 15 serves by self-check 3
13.974 10 done serving by server 1
```

```
13.974 11 serves by server 1
14.869 11 done serving by server 1
15.264 16 arrives
15.264 16 serves by server 1
15.524 17 arrives
15.524 17 serves by server 2
15.531 17 done serving by server 2
15.940 18 arrives
15.940 18 waits at server 1
15.978 15 done serving by self-check 3
16.307 16 done serving by server 1
16.307 18 serves by server 1
16.883 18 done serving by server 1
17.793 19 arrives
17.793 19 serves by server 1
18.535 19 done serving by server 1
18.765 20 arrives
18.765 20 serves by server 1
21.773 20 done serving by server 1
[0.322 20 0]
```

```
1 2 2 20
0 3.131408
0.313508 1.037533
1.204909 6.995223
2.776498 0.142237
3.876961 1.541726
3.909736 0.126590
9.00639 14.777172
9.04336 25.930199
9.105379 2.968470
9.159629 3.686675
9.224613 1.605316
10.147993 0.517321
11.204932 34.895950
12.428914 28.691500
13.109177 8.957896
15.263626 1.043020
15.524295 0.006333
15.939973 0.576351
17.793096 0.742370
18.765423 3.007573
0
3.138762
0.847804
0
0.848968
0
0
3.863139
0
0
25.460873
0
0
0
0
36.96365
0
3.577445
2.169913
0
0.000 1 arrives
0.000 1 serves by server 1
0.314 2 arrives
0.314 2 serves by self-check 2
1.205 3 arrives
1.205 3 serves by self-check 3
1.351 2 done serving by self-check 2
2.776 4 arrives
2.776 4 serves by self-check 2
2.919 4 done serving by self-check 2
3.131 1 done serving by server 1
3.877 5 arrives
3.877 5 serves by server 1
```

```
3.910 6 arrives
3.910 6 serves by self-check 2
4.036 6 done serving by self-check 2
5.419 5 done serving by server 1
8.200 3 done serving by self-check 3
9.006 7 arrives
9.006 7 serves by server 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 server 1
9.225 11 arrives
9.225 11 waits at server 1
10.148 12 arrives
10.148 12 waits at self-check 2
11.205 13 arrives
11.205 13 waits at self-check 2
12.074 9 done serving by self-check 3
12.074 12 serves by self-check 3
12.429 14 arrives
12.429 14 waits at self-check 2
12.591 12 done serving by self-check 3
12.591 13 serves by self-check 3
13.109 15 arrives
13.109 15 waits at self-check 2
15.264 16 arrives
15.264 16 leaves
15.524 17 arrives
15.524 17 leaves
15.940 18 arrives
15.940 18 leaves
17.793 19 arrives
17.793 19 leaves
18.765 20 arrives
18.765 20 leaves
23.784 7 done serving by server 1
24.631 10 serves by server 1
28.318 10 done serving by server 1
28.318 11 serves by server 1
29.923 11 done serving by server 1
34.974 8 done serving by self-check 2
34.974 14 serves by self-check 2
47.487 13 done serving by self-check 3
47.487 15 serves by self-check 3
56.445 15 done serving by self-check 3
63.665 14 done serving by self-check 2
[6.320 15 5]
```

Level 5

Randomizing Arrival, Service and Resting Times

Rather than reading arrival, service and resting times from input, we will generate them as random times instead. A random number generator is an entity that generates one random number after another. Since it is not possible to generate a truly random number algorithmically, pseudo random number generation is adopted instead. A pseudo-random number generator can be initialized with a seed, such that the same seed always produces the same sequence of (seemingly random) numbers.

Although, Java provides a class `java.util.Random`, an alternative `RandomGenerator` class that is more suitable for discrete event simulation is provided for you that encapsulates different random number generators for use in our simulator. Each random number generator generates a different stream of random numbers. The constructor for `RandomGenerator` takes in the following parameters:

- `int seed` is the base seed. Each random number generator uses its own seed that is derived from this base seed;
- `double lambda` is the arrival rate, λ;
- `double mu` is the service rate, μ;
- `double rho` is the server resting rate, ρ.

The inter-arrival time is usually modeled as an exponential random variable, characterized by a single parameter λ denoting the arrival rate. The `genInterArrivalTime()` method of the class `RandomGenerator` is used for this purpose. Specifically,

- start the simulation by generating the first customer arrival event with timestamp `0`
- if there are still more customers to simulate, generate the next arrival event with a timestamp of `T + now`, where `T` is generated with the method `genInterArrivalTime()`;

The service time is modeled as an exponential random variable, characterized by a single parameter, service rate μ. The method `genServiceTime()` from the class `RandomGenerator` can be used to generate the service time. Specifically,

- each time a customer is being served, a `DONE` event is generated and scheduled;
- the `DONE` event generated will have a timestamp of `T + now`, where `T` is generated with the method `genServiceTime()`.

To decide if the server should rest, a random number uniformly drawn from `[0, 1]` is generated using the `RandomGenerator` method `genRandomRest()`. If the value returned is less than the probability of resting (P_r) then the server rests. Otherwise, the server does not rest but continues serving the next customer.

As soon as the server rests, a random rest period T_r is generated using the `RandomGenerator` method `genRestPeriod()`. This variable is an exponential random variable, governed by the resting rate, ρ.

In addition, an arriving customer is a greedy customer with probability P_g. A *greedy* customer always joins the queue with the fewest customers; in the case of a tie, he/she breaks the tie by choosing the first one while scanning from servers `1` to `k`.

To decide whether a typical or greedy customer is created, a random number uniformly drawn from `[0, 1]` is generated with the `RandomGenerator` method `genCustomerType()`. If the value returned is less than P_g, a greedy customer is generated, otherwise, a typical customer is generated.

You may refer to the API of the `RandomGenerator` class here. The class file can be downloaded from here.

Note that `RandomGenerator` class resides in the `cs2030.simulator` package. So, `RandomGenerator.class` should be saved in the `cs2030/simulator` directory.

Define a driver class `Main5.java` (stored in `Main5.java`) that reads the following as input (in order of presentation):

- an `int` value denoting the base seed for the `RandomGenerator` object;
- an `int` value representing the number of servers
- am `int` value representing the number of self-checkout counters, N_{self}
- an `int` value for the maximum queue length, Q_{max}
- an `int` representing the number of customers (or the number of arrival events) to simulate
- a positive `double` parameter for the arrival rate, λ
- a positive `double` parameter for the service rate, μ
- a positive `double` parameter for the resting rate, ρ
- a `double` parameter for the probability of resting, P_r
- a `double` parameter for the probability of a greedy customer occurring, P_g

Remember to start the simulation by generating the first customer arrival event with timestamp `0`, and then generate the next arrival event of the next customer by adding the time generated using the method `genInterArrivalTime()` from the class `RandomGenerator`.

It is also worth noting that the service time **cannot** be generated together with the arrival time as each customer is created before the simulation starts, since the n^th customer to arrive might not be the n^th customer to be served! Although this might entail passing the `RandomGenerator` along with the Customer. This need not be so! *Hint: Think Lazy...*

Sample runs of the program is given below. User input is underlined.

```
1 2 0 1 10 1.0 1.0 0.0 0.0 0.0
0.000 1 arrives
0.000 1 serves by server 1
0.313 1 done serving by server 1
0.314 2 arrives
0.314 2 serves by server 1
0.417 2 done serving by server 1
1.205 3 arrives
1.205 3 serves by server 1
1.904 3 done serving by server 1
2.776 4 arrives
2.776 4 serves by server 1
2.791 4 done serving by server 1
3.877 5 arrives
3.877 5 serves by server 1
3.910 6 arrives
3.910 6 serves by server 2
3.922 6 done serving by server 2
4.031 5 done serving by server 1
```

```
9.006 7 arrives
9.006 7 serves by server 1
9.043 8 arrives
9.043 8 serves by server 2
9.105 9 arrives
9.105 9 waits at server 1
9.160 10 arrives
9.160 10 waits at server 2
10.484 7 done serving by server 1
10.484 9 serves by server 1
10.781 9 done serving by server 1
11.636 8 done serving by server 2
11.636 10 serves by server 2
11.688 10 done serving by server 2
[0.386 10 0]
```

```
1 2 0 2 10 1.0 1.0 0.0 0.0 0.0
0.000 1 arrives
0.000 1 serves by server 1
0.313 1 done serving by server 1
0.314 2 arrives
0.314 2 serves by server 1
0.417 2 done serving by server 1
1.205 3 arrives
1.205 3 serves by server 1
1.904 3 done serving by server 1
2.776 4 arrives
2.776 4 serves by server 1
2.791 4 done serving by server 1
3.877 5 arrives
3.877 5 serves by server 1
3.910 6 arrives
3.910 6 serves by server 2
3.922 6 done serving by server 2
4.031 5 done serving by server 1
9.006 7 arrives
9.006 7 serves by server 1
9.043 8 arrives
9.043 8 serves by server 2
9.105 9 arrives
9.105 9 waits at server 1
9.160 10 arrives
9.160 10 waits at server 1
10.484 7 done serving by server 1
10.484 9 serves by server 1
10.781 9 done serving by server 1
10.781 10 serves by server 1
10.833 10 done serving by server 1
11.636 8 done serving by server 2
[0.300 10 0]
```

```
1 2 0 3 20 1.0 1.0 0.1 0.5 0.0
0.000 1 arrives
0.000 1 serves by server 1
0.313 1 done serving by server 1
0.314 2 arrives
0.314 2 serves by server 1
0.417 2 done serving by server 1
1.205 3 arrives
1.205 3 serves by server 2
1.904 3 done serving by server 2
2.776 4 arrives
2.776 4 serves by server 2
2.791 4 done serving by server 2
3.877 5 arrives
3.877 5 serves by server 1
3.910 6 arrives
3.910 6 serves by server 2
3.922 6 done serving by server 2
4.031 5 done serving by server 1
9.006 7 arrives
9.006 7 serves by server 1
9.043 8 arrives
9.043 8 serves by server 2
9.105 9 arrives
9.105 9 waits at server 1
```

```
9.160 10 arrives
9.160 10 waits at server 1
9.225 11 arrives
9.225 11 waits at server 1
10.148 12 arrives
10.148 12 waits at server 2
10.484 7 done serving by server 1
10.484 9 serves by server 1
10.781 9 done serving by server 1
11.205 13 arrives
11.205 13 waits at server 1
11.636 8 done serving by server 2
11.636 12 serves by server 2
11.688 12 done serving by server 2
12.429 14 arrives
12.429 14 serves by server 2
13.109 15 arrives
13.109 15 waits at server 2
14.644 10 serves by server 1
15.013 10 done serving by server 1
15.264 16 arrives
15.264 16 waits at server 1
15.524 17 arrives
15.524 17 waits at server 2
15.919 14 done serving by server 2
15.919 15 serves by server 2
15.940 18 arrives
15.940 18 waits at server 2
16.079 15 done serving by server 2
16.079 17 serves by server 2
17.793 19 arrives
17.793 19 waits at server 2
18.765 20 arrives
18.765 20 waits at server 2
18.948 17 done serving by server 2
18.948 18 serves by server 2
19.844 18 done serving by server 2
19.844 19 serves by server 2
20.887 19 done serving by server 2
40.474 11 serves by server 1
40.480 11 done serving by server 1
40.480 13 serves by server 1
41.056 13 done serving by server 1
44.634 16 serves by server 1
45.376 16 done serving by server 1
57.851 20 serves by server 2
60.858 20 done serving by server 2
[7.288 20 0]
```

```
1 2 0 3 20 1.0 1.0 0.1 0.5 0.9
0.000 1(greedy) arrives
0.000 1(greedy) serves by server 1
0.313 1(greedy) done serving by server 1
0.314 2(greedy) arrives
0.314 2(greedy) serves by server 1
0.417 2(greedy) done serving by server 1
1.205 3(greedy) arrives
1.205 3(greedy) serves by server 2
1.904 3(greedy) done serving by server 2
2.776 4(greedy) arrives
2.776 4(greedy) serves by server 2
2.791 4(greedy) done serving by server 2
3.877 5(greedy) arrives
3.877 5(greedy) serves by server 1
3.910 6(greedy) arrives
3.910 6(greedy) serves by server 2
3.922 6(greedy) done serving by server 2
4.031 5(greedy) done serving by server 1
9.006 7(greedy) arrives
9.006 7(greedy) serves by server 1
9.043 8(greedy) arrives
9.043 8(greedy) serves by server 2
9.105 9(greedy) arrives
9.105 9(greedy) waits at server 1
9.160 10 arrives
```

```
9.160 10 waits at server 1
9.225 11(greedy) arrives
9.225 11(greedy) waits at server 2
10.148 12(greedy) arrives
10.148 12(greedy) waits at server 2
10.484 7(greedy) done serving by server 1
10.484 9(greedy) serves by server 1
10.781 9(greedy) done serving by server 1
11.205 13(greedy) arrives
11.205 13(greedy) waits at server 1
11.636 8(greedy) done serving by server 2
11.636 11(greedy) serves by server 2
11.688 11(greedy) done serving by server 2
11.688 12(greedy) serves by server 2
12.429 14(greedy) arrives
12.429 14(greedy) waits at server 2
13.109 15(greedy) arrives
13.109 15(greedy) waits at server 2
14.644 10 serves by server 1
15.013 10 done serving by server 1
15.178 12(greedy) done serving by server 2
15.178 14(greedy) serves by server 2
15.264 16 arrives
15.264 16 waits at server 1
15.338 14(greedy) done serving by server 2
15.338 15(greedy) serves by server 2
15.524 17(greedy) arrives
15.524 17(greedy) waits at server 2
15.940 18(greedy) arrives
15.940 18(greedy) waits at server 2
17.793 19(greedy) arrives
17.793 19(greedy) waits at server 1
18.207 15(greedy) done serving by server 2
18.207 17(greedy) serves by server 2
18.765 20(greedy) arrives
18.765 20(greedy) waits at server 2
19.103 17(greedy) done serving by server 2
19.103 18(greedy) serves by server 2
20.146 18(greedy) done serving by server 2
40.474 13(greedy) serves by server 1
40.480 13(greedy) done serving by server 1
40.480 16 serves by server 1
41.056 16 done serving by server 1
44.634 19(greedy) serves by server 1
45.376 19(greedy) done serving by server 1
57.110 20(greedy) serves by server 2
60.117 20(greedy) done serving by server 2
[7.065 20 0]
```

```
1 2 1 2 20 1.0 1.0 0.1 0.5 0.9
0.000 1(greedy) arrives
0.000 1(greedy) serves by server 1
0.313 1(greedy) done serving by server 1
0.314 2(greedy) arrives
0.314 2(greedy) serves by server 1
0.417 2(greedy) done serving by server 1
1.205 3(greedy) arrives
1.205 3(greedy) serves by server 2
1.904 3(greedy) done serving by server 2
2.776 4(greedy) arrives
2.776 4(greedy) serves by server 2
2.791 4(greedy) done serving by server 2
3.877 5(greedy) arrives
3.877 5(greedy) serves by server 1
3.910 6(greedy) arrives
3.910 6(greedy) serves by server 2
3.922 6(greedy) done serving by server 2
4.031 5(greedy) done serving by server 1
9.006 7(greedy) arrives
9.006 7(greedy) serves by server 1
9.043 8(greedy) arrives
9.043 8(greedy) serves by server 2
9.105 9(greedy) arrives
9.105 9(greedy) serves by self-check 3
9.160 10 arrives
```

```
9.160 10 waits at server 1
9.225 11(greedy) arrives
9.225 11(greedy) waits at server 2
9.402 9(greedy) done serving by self-check 3
10.148 12(greedy) arrives
10.148 12(greedy) serves by self-check 3
10.200 12(greedy) done serving by self-check 3
10.484 7(greedy) done serving by server 1
10.484 10 serves by server 1
11.205 13(greedy) arrives
11.205 13(greedy) serves by self-check 3
11.574 13(greedy) done serving by self-check 3
11.636 8(greedy) done serving by server 2
12.429 14(greedy) arrives
12.429 14(greedy) serves by self-check 3
12.589 14(greedy) done serving by self-check 3
13.109 15(greedy) arrives
13.109 15(greedy) serves by self-check 3
13.974 10 done serving by server 1
15.264 16 arrives
15.264 16 serves by server 1
15.500 11(greedy) serves by server 2
15.524 17(greedy) arrives
15.524 17(greedy) waits at server 1
15.940 18(greedy) arrives
15.940 18(greedy) waits at server 2
15.978 15(greedy) done serving by self-check 3
16.159 16 done serving by server 1
16.159 17(greedy) serves by server 1
16.166 17(greedy) done serving by server 1
16.543 11(greedy) done serving by server 2
16.543 18(greedy) serves by server 2
17.119 18(greedy) done serving by server 2
17.793 19(greedy) arrives
17.793 19(greedy) serves by server 2
18.535 19(greedy) done serving by server 2
18.765 20(greedy) arrives
18.765 20(greedy) serves by server 2
21.773 20(greedy) done serving by server 2
[0.442 20 0]
```