CodeCrunch

CS2030 Discrete Event Simulator (Question)

Tags & Categories

Tags:

Categories:

Related Tutorials

Task Content

Discrete Event Simulator

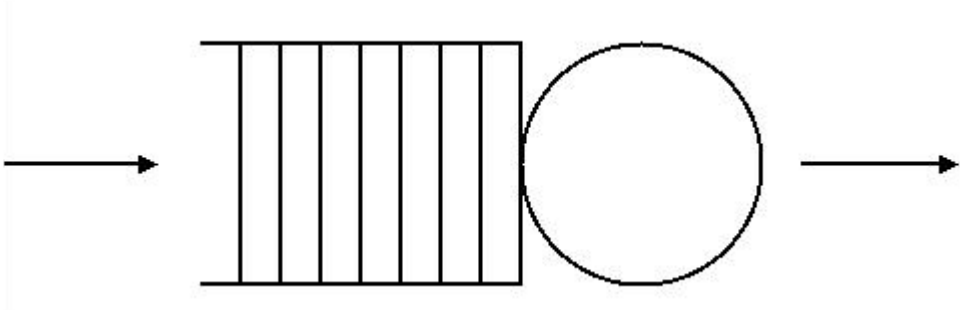
Requirements

- OOP Principles and Design
- Program style: adherance to CS2030 Java Style Guide
- Java documentation: adherence to CS2030 Javadoc Specification

Problem Description

A discrete event simulator is a software that simulates a system with events and states, and can be used to study many complex real-world systems. An event occurs at a particular time, and each event alters the states 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. A simulator typically also keeps track of some statistics to measure the performance of the system.

We shall simulate a queuing system comprising a single customer queue and single service point as shown below.



The system comprises the following:

- There is one **server** (a person providing service to the customer); the server can serve one customer at a time.
- The **service time** (time take to serve a customer) is constant.
- When a **customer** arrives (**`ARRIVES`**):
    - if the server is idle (not serving any customer), then the server starts serving the customer immediately (**`SERVED`**)
    - if the server is serving another customer, then the customer that just arrives waits (**`WAITS`**).
    - if the server is serving one customer, and another customer is waiting, then the customer that just arrives simply leaves (**`LEAVES`**); in other words, there is at most one waiting customer.
    - When the server is done serving a customer (**`DONE`**), the served customer leaves.
- If there is another customer waiting, the server starts serving the waiting customer immediately.
- If there is no waiting customer, then server becomes idle again.

There are five states of the system, namely `ARRIVES`, `SERVED`, `WAITS`, `LEAVES` and `DONE`. Clearly, there is only one of three possible state transitions for each customer:

- `ARRIVES` → `SERVED` → `DONE`
- `ARRIVES` → `WAITS` → `SERVED` → `DONE`
- `ARRIVES` → `LEAVES`

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

As an example, given the arrival times of three customers and assuming a service time of `1.0`

```
0.500
0.600
0.700
```

the following states, each of the form , are produced

```
0.500 1 arrives
0.500 1 served
0.600 2 arrives
0.600 2 waits
0.700 3 arrives
0.700 3 leaves
1.500 1 done
```

```
1.500 2 served
2.500 2 done
```

with the statistics being respectively, `[0.450 2 1]`.

The Task

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

Take note of the following assumptions:

- The maximum number of events is 100 **at any one time**;
- The format of the input is always correct;
- Output of a `double` value, say `d`, is to be formatted with `String.format("%.3f", d)`;

This task is divided into several levels. Read through all the levels to see how the different levels are related. **You may start from any level.**

In each level, you are to

- define a `Main` class with the `main` method to handle input and output;
- check for styling errors by invoking `checkstyle`. For example, to check styling for all java files

```
$ java -jar checkstyle-8.2-all.jar -c cs2030_checks.xml *.java
```

A copy of the files for checkstyle is available in the IVLE Workbin.

Level 1

Customer arrivals

Each arriving customer should be tagged with a customer ID (starting from `1`) as well as an arrival time. We need to also keep track of the number of customers arriving.

Note that input is simply a list of arrival times without any prior indication of the number of arrivals. Check out the Java API on what other `Scanner` methods you can use to achieve this.

The following is a sample run of the program. User input is underlined.

```
0.500
0.600
0.700
1 arrives at 0.500
2 arrives at 0.600
3 arrives at 0.700
Number of customers: 3
```

Click here to submit to CodeCrunch.

Level 2

Serving the Customer

A customer that arrives is served immediately, but only if the server is not currently serving; otherwise, the customer leaves.

The following is a sample run of the program. User input is underlined.

```
0.500
0.600
0.700
1 arrives at 0.500
Customer served; next service @ 1.500
2 arrives at 0.600
Customer leaves
3 arrives at 0.700
Customer leaves
Number of customers: 3
```

```
0.500
0.600
0.700
1.500
1.600
1.700
1 arrives at 0.500
Customer served; next service @ 1.500
2 arrives at 0.600
Customer leaves
3 arrives at 0.700
Customer leaves
4 arrives at 1.500
Customer served; next service @ 2.500
5 arrives at 1.600
Customer leaves
6 arrives at 1.700
Customer leaves
Number of customers: 6
```

Click here to submit to CodeCrunch.

Level 3

Incorporating States

In this level, we take into consideration customer arrivals, customers being served and customers leaving.

For example, the following arrivals

```
0.500
0.600
0.700
```

will generate the following output:

```
0.500 1 arrives
0.500 1 served
0.600 2 arrives
0.600 2 leaves
0.700 3 arrives
0.700 3 leaves
```

A suggestion is to define the following constants (or use an Enumeration):

```
public static final int ARRIVES = 1;
public static final int SERVED = 2;
public static final int LEAVES = 3;
```

The following is a sample run of the program. User input is underlined.

```
0.500
0.600
0.700
0.500 1 arrives
0.500 1 served
0.600 2 arrives
0.600 2 leaves
0.700 3 arrives
0.700 3 leaves
Number of customers: 3
```

```
0.500
0.600
0.700
1.500
1.600
1.700
0.500 1 arrives
0.500 1 served
0.600 2 arrives
0.600 2 leaves
0.700 3 arrives
0.700 3 leaves
1.500 4 arrives
1.500 4 served
1.600 5 arrives
1.600 5 leaves
1.700 6 arrives
1.700 6 leaves
Number of customers: 6
```

Click here to submit to CodeCrunch.

Level 4

Scheduling Events

As an example, suppose three arrival events are initially scheduled.

```
# Adding arrivals
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.

```
# Get next event: 0.500 1 arrives
0.500 1 served
0.600 2 arrives
0.700 3 arrives
```

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

```
# Get next event: 0.500 1 served
0.600 2 arrives
0.700 3 arrives
1.500 1 done
```

The process is repeated until there are no more events.

The following is a sample run of the program. User input is underlined. Note that during each iteration of the simulation, there is **no need** to maintain any order on the list of scheduled events, as long as they appear (the final level does not have this output requirement anyway). CodeCrunch will order the events as part of program testing.

What is important is that the correct event is picked during each iteration. Specifically, pick the earliest occurring one, and if there is a tie, pick the one with the smallest customer ID.

```
0.500
0.600
0.700
# Adding arrivals
0.500 1 arrives
0.600 2 arrives
0.700 3 arrives

# Get next event: 0.500 1 arrives
0.500 1 served
0.600 2 arrives
0.700 3 arrives
```

1

1.
I:a
>a
>s1
2.
I:s
>a
>d
3.
I:a
>a
>l
>d
4.
I:L
>a
>d

```
# Get next event: 0.500 1 served
0.600 2 arrives
0.700 3 arrives
1.500 1 done

# Get next event: 0.600 2 arrives
0.600 2 leaves
0.700 3 arrives
1.500 1 done

# Get next event: 0.600 2 leaves
0.700 3 arrives
1.500 1 done

# Get next event: 0.700 3 arrives
0.700 3 leaves
1.500 1 done

# Get next event: 0.700 3 leaves
1.500 1 done

# Get next event: 1.500 1 done

Number of customers: 3
```

```
0.500
0.600
0.700
1.500
1.600
1.700
# Adding arrivals
0.500 1 arrives
0.600 2 arrives
0.700 3 arrives
1.500 4 arrives
1.600 5 arrives
1.700 6 arrives

# Get next event: 0.500 1 arrives
0.500 1 served
0.600 2 arrives
0.700 3 arrives
1.500 4 arrives
1.600 5 arrives
1.700 6 arrives

# Get next event: 0.500 1 served
0.600 2 arrives
0.700 3 arrives
1.500 1 done
1.500 4 arrives
```

a, add serve/leave
d/l, display
s, add done

```
1.600 5 arrives
1.700 6 arrives

# Get next event: 0.600 2 arrives
0.600 2 leaves
0.700 3 arrives
1.500 1 done
1.500 4 arrives
1.600 5 arrives
1.700 6 arrives

# Get next event: 0.600 2 leaves
0.700 3 arrives
1.500 1 done
1.500 4 arrives
1.600 5 arrives
1.700 6 arrives

# Get next event: 0.700 3 arrives
0.700 3 leaves
1.500 1 done
1.500 4 arrives
1.600 5 arrives
1.700 6 arrives

# Get next event: 0.700 3 leaves
1.500 1 done
1.500 4 arrives
1.600 5 arrives
1.700 6 arrives

# Get next event: 1.500 1 done
1.500 4 arrives
1.600 5 arrives
1.700 6 arrives

# Get next event: 1.500 4 arrives
1.500 4 served
1.600 5 arrives
1.700 6 arrives

# Get next event: 1.500 4 served
1.600 5 arrives
1.700 6 arrives
2.500 4 done

# Get next event: 1.600 5 arrives
1.600 5 leaves
1.700 6 arrives
2.500 4 done

# Get next event: 1.600 5 leaves
```

```
1.700 6 arrives
2.500 4 done

# Get next event: 1.700 6 arrives
1.700 6 leaves
2.500 4 done

# Get next event: 1.700 6 leaves
2.500 4 done

# Get next event: 2.500 4 done

Number of customers: 6
```

Click here to submit to CodeCrunch.

Level 5

Discrete event simulation

To complete the discrete event simulator, you will now need to consider that an arriving customer can either be served, wait to be served or leaves.

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

The following is a sample run of the program. User input is underlined.

```
0.500
0.600
0.700
0.500 1 arrives
0.500 1 served
0.600 2 arrives
0.600 2 waits
0.700 3 arrives
0.700 3 leaves
1.500 1 done
1.500 2 served
2.500 2 done
[0.450 2 1]
```

```
0.500
0.600
0.700
1.500
1.600
1.700
0.500 1 arrives
0.500 1 served
0.600 2 arrives
```

waiting time / no of served

```
0.600 2 waits
0.700 3 arrives
0.700 3 leaves
1.500 1 done
1.500 2 served
1.500 4 arrives
1.500 4 waits
1.600 5 arrives
1.600 5 leaves
1.700 6 arrives
1.700 6 leaves
2.500 2 done
2.500 4 served
3.500 4 done
[0.633 3 3]
```

Click here to submit to CodeCrunch.