Lab 2: Simulation 2

- Deadline: 6 September 2022, Tuesday, 23:59
- Marks: 3%

Prerequisite:

- Completed Lab 1
- Caught up to Unit 19 of Lecture Notes
- Familiar with CS2030S Java style guide

Goal

This is a continuation of Lab 1. Lab 2 changes some of the requirements of Lab 1 and adds some new things to the world that we are simulating. The goal is to demonstrate that, when OO-principles are applied properly, we can adapt our code to changes in the requirement with less effort.

Lab 2 also nudges you towards following good coding practice by adhering to a published coding convention.

Simulating a Shop with a Queue

Recall that, due to COVID-19 restrictions, no waiting is allowed inside the shop we are simulating. The shop is losing customers as a customer departs if all the counters are busy.

Lab 2 adds an entrance queue to the shop. If all counters are busy when a customer arrives, the customer will join the queue and wait. When a counter becomes available, the customer at the front of the queue will proceed to the counter for service.

The entrance queue has a maximum queue length of m. If there are already m customers waiting in the entrance queue, any arriving customer will be turned away.

Building on Lab 1

You are required to build on top of your Lab 1 submission for this lab.

Assuming you have `lab1-<username>` and `lab2-<username>` under the same directory, and `lab2-<username>` is your current working directory, you can run

```
cp -i ../lab1-<username>/*.java .
rm -i Lab1.java
```

to copy all your Java code over.

If you are still unfamiliar with Unix commands to navigate the file system and processing files, please review our Unix guide.

You are encouraged to consider your tutor's feedback and fix any issues with your design for your Lab 1 submission before you embark on your Lab 2.

Skeleton for Lab 2

We only provide two classes for Lab 2, the main `Lab2.java` (which is simply `Lab1.java` renamed) and `Queue.java`.

Both files should not be modified for this lab.

The `Queue` class

`Queue` is a general class for a first-in, first-out queue of objects. Here is an example of how it is used:

```
// Create a queue that holds up to 4 elements
Queue q = new Queue(4);

// Add a string into the queue.  returns true if successful;
// false otherwise.
boolean b = q.enq("a1");

// Remove a string from the queue.  `Queue::deq` returns an
// `Object`, so narrowing type conversion is needed.  Returns
// `null` is queue is empty.
String s = (String) q.deq();

// Returns the string representation of the queue (showing
// each element)
String s = q.toString();

// Returns true if the queue is full, false otherwise.
boolean b = q.isFull();

// Returns true if the queue is empty, false otherwise.
boolean b = q.isEmpty();

```

```
// Returns the number of objects in the queue
int l = q.length();
```

Other Changes Needed

In addition to adding an entrance queue to the shop, we need to make the following changes to the input and output of the program.

1. There is an additional input parameter, an integer m, indicating the maximum allowed length of the entrance queue. This input parameter should be read immediately after reading the number of customers and the number of service counters.
2. A customer will now be printed with a single letter prefix `C`. For instance, instead of `Customer 1`, we print `C1`.
3. A service counter will now be printed with a single letter prefix `S`. For instance, instead of `Counter 1`, we print `S1`.
4. The entrance queue of the shop will be printed with the arrival event. E.g., the following shows that C3 arrived at time 1.400 and at the time of arrival, there are two customers, C1 and C2, waiting in the entrance queue.

   ```
   1.400: C3 arrived [ C1 C2 ]
   ```

5. If a customer joins the entrance queue, the customer along with the queue *before* joining should be printed. E.g.,

   ```
   1.400: C3 joined queue [ C1 C2 ]
   ```

Following CS2030S Style Guide

In addition to the changes above, you should also make sure that your code follows the given Java style guide

Assumptions

We assume that no two events involving two different customers ever occur at the same time (except when a customer departs and another customer begins its service). As per all labs, we assume that the input is correctly formatted.

Your Task

Your task for this lab is to (i) improve upon your design for Lab 1 if needed, (ii) update `ShopSimulation` and associated classes to simulate the entrance queue, (iii) update the input and output components of the classes to conform to the specification above.

If the design for Lab 1 follows the OOP principles, then only about 40 lines of changes/additions are required.

Compiling, Testing, and Debugging

Compilation

To compile your code,

```
1   javac *.java
```

Running and Testing

You should not test your code by manually entering the inputs. Instead, enter the inputs into a file, and run

```
1   java Lab2 < file
```

A set of test inputs is provided as part of the skeleton, named `Lab2.x.in` under the `inputs` directory. You can run them with, for instance,

```
1   java Lab2 < inputs/Lab2.1.in
```

You can save the output by redirecting it into a file.

```
1   java Lab2 < inputs/Lab2.1.in > OUT
```

You can automatically test your code against all the given inputs/outputs as well as against the `checkstyle` by running:

```
1   ./test.sh Lab2
```

Debugging

The expected outputs are given in the `outputs` directory. You can compare `OUT` with the expected output with `diff` or `vim`. Using `vim`,

```
vim -d OUT output/Lab2.1.out
```

will open both files and highlight the differences.

As the output becomes too long, you can focus on tracing a particular counter or customer with the help of `grep`. Suppose you want to focus on what happened to Customer 1 in `OUT`, run

```
$ grep ": C1" OUT
1.200: C1 arrived [ ]
1.200: C1 service begin (by S1)
2.200: C1 service done (by S1)
2.200: C1 departed
```

Suppose you want to see all the customers served by `S1`, run:

```
$ grep "S1" OUT
1.200: C1 service begin (by S1)
2.200: C1 service done (by S1)
2.200: C4 service begin (by S1)
3.200: C4 service done (by S1)
```

Submission

Upload the following files to CodeCrunch.

1. `Lab2.java`
2. *any other* `.java` *files you use*