Ex 2: Simulation II

Basic Information

- **Deadline:** 10 September 2024, Tuesday, 23:59 SGT
- **Difficulty:** ★★★★★

Prerequisite

- Completed Programming Exercise 1.
- Caught up to Unit 20 of Lecture Notes.

Goal

This is a continuation of Programming Exercise 1. Programming Exercise 2 changes some of the requirements of Programming Exercise 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. If your design for Programming Exercise 1 follows the OOP principles, then only about 50 lines of changes/additions are required.

For Programming Exercise 2, you should (i) improve upon your design for Programming Exercise 1 if needed, (ii) update CoffeeSimulation and associated classes to simulate the extended scenarios, and (iii) update the input and output components of the classes to conform to the specification below.

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

Tasks

We have two extensions that we want to do on our coffee shop. Before we do that, we need to make sure that your Ex 1 code are sufficiently good.

Task 1: Address the Concern

Hopefully, by now your friendly TA should have written comments about your submission. You should address the concerns pointed by your TA. If those concerns remained unaddressed, the same deductions will be applied to the current submission too.

Task 2: Extend

Extension 1: Simulating a Coffee Shop with a Queue

Recall that, no waiting was allowed inside the coffee shop we are simulating. The coffee shop is losing customers as a customer departs if all the counters are busy.

Programming Exercise 2 adds an entrance queue to the coffee 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[1].

If some counters are available when a customer arrives, the customer will go the first available counter, just like in Programming Exercise 1.

Extension 2: Customers with Orders

Customers now come to the coffee shop with a coffee they intend to order. The order can be either an espresso or a latte for now.

Note

We are not going to actually make the coffee using our code but it is possible if you have an espresso machine[2].

Extension 3: Changes to Input

As we want to add a queue of some length m and customer order, we need to modify the input as well. The changes to the input are summarized below. You will need to implement this changes first before you can do testing as our input test files follow this new format.

1. There is an additional input parameter in the first line of the input file, 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. There is an additional input parameter at the end of each line in the input file that describes a customer's arrival. The new parameter is an `int`, which is either 0 (*for espresso order*) or 1 (*for latte order*).

CHANGES

Ex1.1.in

```
3 1
1.0 1.0
3.0 1.0
5.0 1.0
```

Ex2.1.in

```
3 1 2
1.0 1.0 0
3.0 1.0 1
5.0 1.0 0
```

In `Ex2.1.in`, the first line of the input has an additional integer input `2`. This specifies the length of the queue.

Additionally, for subsequent lines we have additional integer inputs for each line. For line 2 and 4, the customers are ordering espresso. For line 3, the customer is ordering latte.

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 their order*). As per all exercises, we assume that the input is correctly formatted.

Extension 4: Changes to Output

1. A customer will now be printed with a single letter prefix `C`. For instance, instead of printing `Customer 1`, we print `C1`.
2. A counter will be manned by a barista. As such, a counter will now be printed with a single letter prefix `B`. So, instead of printing `Counter 1`, we print `B1`.
3. The entrance queue of the coffee shop will be printed with the arrival event. For example, the following shows that `C4` arrived at time `2.100` and at the time of interval, there were two customers `C2` and `C3` already waiting in the entrance queue.

From Ex2.10.out

```
2.100: C4 arrives [ C2 C3 ]
```

4. If a customer joins the entrance queue, the customer along with the queue before joining should be printed.

From Ex2.4.out

```
4.000: C4 joined queue [ C2 C3 ]
```

5. When a customer ordered a coffee or has been served their order, the customer order should also be printed. For example, Customer `C2` is ordering a latte on counter `B0` would be printed as follows.

From Ex2.4.out

```
5.100: C2 ordered Coffee Latte (by B0)
7.100: C2 served Coffee Latte (by B0)
```

Skeleton for Programming Exercise 2

We only provide two classes for Programming Exercise 2, the main `Ex2.java` (*which is simply* `Ex1.java` *renamed*) and `Queue.java`. The `Ex2.java` is similar to `Ex1.java`. `Queue.java` is new. It models a first-in first-out (FIFO) queue of objects. Its usage will be explained further below.

Do NOT Edit

You should **NOT** edit `Ex2.java` or `Queue.java`. However, you may change the `@author` tag of these two files if you wish.

Additionally, if you have made changes to `Event.java`, `Simulation.java`, or `Simulator.java`, you will need to use the original file given for Programming Exercise 1 as those are not supposed to be edited.

Building on Programming Exercise 1

You are required to build on top of your Programming Exercise 1 submission for this exercise.

Assuming you have `ex1-username` and `ex2-username` under the same directory, and `ex2-username` is your current working directory, you can run

```
username@pe111:~/ex2-username$ cp -i ../ex1-username/*.java .
username@pe111:~/ex2-username$ rm -i Ex1.java
```

to copy all your Java code over and remove the main file for `Ex1`.

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

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

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:

Usage of Queue

```java
// 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` if 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();
```

Following CS2030S Style Guide

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

You can use checkstyle tool and the given configuration `ex2_style.xml` to check your code:

```
username@pe111:~/ex2-username$ java -jar ~cs2030s/bin/checkstyle.jar -c
ex2_style.xml *.java
```

Compiling, Testing, and Debugging

Compiling

To compile your code, you can compile all the Java file.

```
username@pe111:~/ex2-username$ 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

```
username@pe111:~/ex2-username$ java Ex2 < file
```

A set of test inputs is provided as part of the skeleton, named `Ex2.x.in` under the inputs directory. You can run them with, for instance,

```
username@pe111:~/ex2-username$ java Ex2 < inputs/Ex2.4.in
```

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

```
username@pe111:~/ex2-username$ java Ex2 < inputs/Ex2.4.in > OUT
```

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

```
username@pe111:~/ex2-username$ java Ex2 < inputs/Ex2.4.in > OUT
```

Debugging

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

```
username@pe111:~/ex2-username$ vim -d OUT outputs/Ex2.4.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 3 in `OUT`, run

```
username@pe111:~/ex2-username$ grep ": C3" OUT
```

You should see the following output:

```
1.400: C3 arrives [ C1 C2 ]
1.400: C3 joined queue [ C1 C2 ]
9.100: C3 departed
```

Suppose you want to see all the customers served by `B0`, run:

```
username@pe111:~/ex2-username$ grep "B0" OUT
```

You should see the following output:

```
1.100: C0 ordered Coffee Espresso (by B0)
3.100: C0 served Coffee Espresso (by B0)
3.100: C1 ordered Coffee Espresso (by B0)
5.100: C1 served Coffee Espresso (by B0)
5.100: C2 ordered Coffee Latte (by B0)
7.100: C2 served Coffee Latte (by B0)
7.100: C3 ordered Coffee Latte (by B0)
9.100: C3 served Coffee Latte (by B0)
9.100: C4 ordered Coffee Latte (by B0)
11.100: C4 served Coffee Latte (by B0)
```

Documentation (Optional)

Documenting your code with Javadoc is optional for Programming Exercise 2. It is, however, always a good practice to include comments to help readers understand your code.

---

1. This is known as "balking" in queueing theory. ↩
2. If you want to try creating an arduino controlled espresso machine, you can follow the following guide: https://gaggiuino.github.io/#/ ↩