Assignment 2: Sushi Bar Simulation

The overall purpose:

In this assignment you are asked to simulate a sushi bar. During the implementation, you need to manage several threads that are using shared resources. Shared resources should be protected using the monitor concept. Java monitors (Synchronized methods) are described in chapter 2.3 of the text book.

In order to see what is happening when running the simulation, you must print out important happenings of the bar.

The problem:

The sushi bar consists of a clock, a door, a number of seats and customers. The bar is open for a specific time period and the customers will enter the bar and try to enter the serving area. Since there are a limited number of seats in the serving area, customers will stay in a queue, waiting for a free seat. When a customer finds a seat, he/she will order some sushi and then starts eating. Customers have the choice to eat their sushi in the bar or have takeaways.

When it is closing time, the bar cannot accept more customers and only those who have already entered will be served.

The Door:

The door creates customers on random time intervals. It also defines the ID of each customer. This number is a counter, which is increased whenever a customer enters the bar.

The Customer:

When a customer enters the shop, he/she tries to go to the serving area. After entering the serving area, each customer spends some time for eating and then leaves the serving area.

- Each customer has
    1. ID number, which is assigned by the Door while creating customers.
    2. Number of orders, which is a random number less than 10.
- Each customer mainly performs these three tasks:
    1. Entering the serving area
    2. Eating sushi
    3. Leaving the serving area

Note that each customer should be simulated as a separate thread.

The Serving Area:

The Serving area is a common recourse, shared between several customers. A customer can only enter the area, when there is a free seat.
Note that the customers enter the area in a **first come- first serve** manner, which means the earlier a customer arrives, the sooner he can enter the serving area. (This manner should be seen in the print outs.)

The serving area has:

1. Capacity
2. A list of customers that are in the serving area

The Clock:

The clock in the bar serves two purposes:

1. It will alert the door the closing time; so the door will generate no more customers.
2. The clock gets the current time of simulation, which is used for printing actions.

You do not need to write the clock yourself; instead you must use the provided class: "clock.java". This clock sets the "`isOpen`" variable to false when the simulation time ends. The only thing you have to do is to initiate the clock (`new Clock(`*`duration`*`)`) and to use the "`isOpen`" variable properly.

Statistics:

At the end of the simulation, you should print out some statistics on the run. Our suggestion is to write your program without considering statistics, to make sure everything works fine. Then change the simulation to calculate these statistics and print them out when the last thread terminates.

The statistics should contain:

1. Total number of orders.
2. Total number of takeaway orders
3. Total number of orders that customers eat at serving area.

Note about orders:
When a customer enters the serving area, he/she will order some sushi (This number should be less than 10 for each customer). Each customer has a choice to eat some orders in the bar and have the remaining takeaway. Remember "takeaway orders" + "eaten orders" is equal to "number of orders".

Here the eating time of the customer will not be affected by number of takeaways. The only thing you need to do is to randomly assign numbers to each type of order.

Requirements:

- You need to submit documentation, which is as important as your implementation. In your documentation you should give an explanation of your implementation. In addition, you have to address the following points:
    - What are the functionality of wait(), notify() and notifyAll() and what is the difference between notify() and notifyAll()?
    - Which variables are shared variable and what is your solution to manage them?
    - Which method or thread will report the statistics and how it will recognize the proper time for writing statistics?

- You should not kill a thread. Every thread should terminate normally, when it does all its tasks. So do not use any "Stop" or "Exit" methods.

- Manipulate the settings of the sushi bar to make sure that your program works in all situations. These settings can be number of seats, simulation time, frequency of generating customers and length of eating time.

- Correct statistics does not necessary mean that your implementation is correct. You have to check all outputs of you program to make sure that everything works in your simulation.

Instruction for output messages:

To see what is happing in the Sushi Bar you should print out these messages at the time they happen.

- `Customer #Id is now created.`
- `Customer #Id has a seat now.`
- `Customer #Id is eating sushi.`
- `Customer #Id has left the shop.`
- `Customer #Id is waiting for free seat.`
- `Now there is a free seat in the shop.`
- `***** NO MORE CUSTOMERS - THE SHOP IS CLOSED NOW. *****`

In order to print these messages, you have to use this code, which adds time and thread number to the message.

```
SushiBar.write(Thread.currentThread().getName()
                    +": Customer " + customer.customerId
                    + " has left the bar.");
```

So at the end, the messages should look like this:

```
15:15:14, Thread-7: Customer 6 has left the bar.
```

Please stick to this format and only print out statistics and the messages listed above.