Sleeping Barber Shop Problem

The Problem Definition:

The problem is based on a hypothetical barbershop with a certain number of barbers. When there are no customers, the barber sleeps on his chair. If any customer arrives, he will wake up the barber to get his hair cut. If there are no chairs empty for another customers, they have to wait in the waiting room/chairs. And if there is no empty chair in the waiting room, then the customer leaves.

Pseudocode of the solution:

- **If there is no customer to be served, the barbers are normally sleeping:**

  IF noOfCustomers == 0
  Then
  Barber state == sleeping

- **If the customer enters the barbershop, and the barber is sleeping, the customer wakes up one of the barbers:**

  IF noOfCustomers == 1 AND Barber state == sleeping
  Then
  One of the Barber states == awake
  AND
  availableCBarbers == 1

- **If a customer enters the barbershop, and the one of the two barbers is sleeping and the other is unavailable, the customer wakes up the sleeping barber to cut his hair:**

  IF availableBarbers == 1
  Then
  The other Barber state == awake
  AND
  availableCBarbers == 0

- **If the customer enters the barbershop, and the two barbers are busy, the customer had to wait in the waiting chairs:**

  IF availableBarbers == 0 AND no_of_waiting_chairs != 0
  Then
  Customer state == waiting
  AND
  WaitingChairsCounter –

- **If a customer enters the shop and there are no available chairs, the customer leaves:**

  IF availableBarbers == 0 AND no_of_waiting_chairs == 0
  Then
  Customer state == leaving
  backlatercounter ++

- **If the two barbers are done and there are no more waiting customers, they will return to sleep:**

  IF noOfCustomers == 0 AND UnWorkingBarbers == no_of_barbers
  Then
  Barbers state == sleeping

Example on Deadlock in this problem:

There might be a scenario in which the customer ends up waiting on the barber and a barber waiting on the customer, which would result a deadlock.

Scenario: a customer may arrive to find the barber cutting hair so they return to the waiting room to take a seat but while walking back to the waiting room the barber finishes the haircut and goes to the waiting room, which he finds empty (because the customer walks slowly or went to the restroom) and thus goes to sleep in the barber chair.

Solution: There are several possible solutions, and all solutions require a mutex, which ensures that only one of the participants can change state at once. The barber must acquire the room status mutex before checking for customers and release it when they begin either to sleep or cut hair; a customer must acquire it before entering the shop and release it once they are sitting in a waiting room or barber chair, and also when they leave the shop because no seats were available. This would take care of both of the problems mentioned above. Some semaphores are also required to indicate

the state of the system. For example, one might store the number of people in the waiting room, the second one check status of the barber either idle or not, and the third semaphore to keep count of the available seats, so customers either wait if there are free seats or leave if there are none or went to the restroom) and thus goes to sleep in the barber chair.

Example on Starvation in this problem:

Starvation might happen to customer who is waiting for a long time when the customer don't follow order or when barber calls out customer randomly

Solution: To handle this problem in the code, insert the customers in a linked list which follows the first in first out property. So, every time a customer sits in a waiting room, they will be selected by the barber in first come first serve basis.

```
synchronized(CustomerList){
            while (CustomerList.size() == 0) {
                form.SleepTA(BA_ID);
                System.out.println("\nBA "+BA_ID+" is waiting "
                        + "for the customer and sleeps in his
desk");
                try {
                    CustomerList.wait();
                } catch (InterruptedException ex) {

Logger.getLogger(Shop.class.getName()).log(Level.SEVERE, null,
ex);
                }
            }

            Customer =
(Customer)((LinkedList<?>)CustomerList).poll();
            System.out.println("Customer
"+customer.getCustomerId()+
                    " finds TA available and "
                    + "the TA "+BA_ID);
        }
```

Explanation for real world application and how did apply the problem:

This design is the best analogy for a customer care call center. Initially when there is no customer on-call all call executives just relax and wait for the call. The moment the first customer dials the number he/she is connected to any call executive and in a scenario when all call executives are busy the customer will have to wait in a queue till they are assigned to a call executive. If all executives are busy and the waiting line is full, the customers are disconnected with a message that executives are busy and customers will be contacted later by the company. This best relates to this design as the customers are picked from the queue on a first come first serve basis and call executives are utilized in such a way that everyone executive gets at least one call.

In this scenario, we can have the following design similarities:

- The critical section will be the call between executive and customer.
- The waiting room will be the waiting queue over a call, where customers will be held in a FIFO manner.
- Locks can be acquired on the waiting queue so that no two executives pick the same customer.