Synchronization 2: A Free City-Bike Rental System.

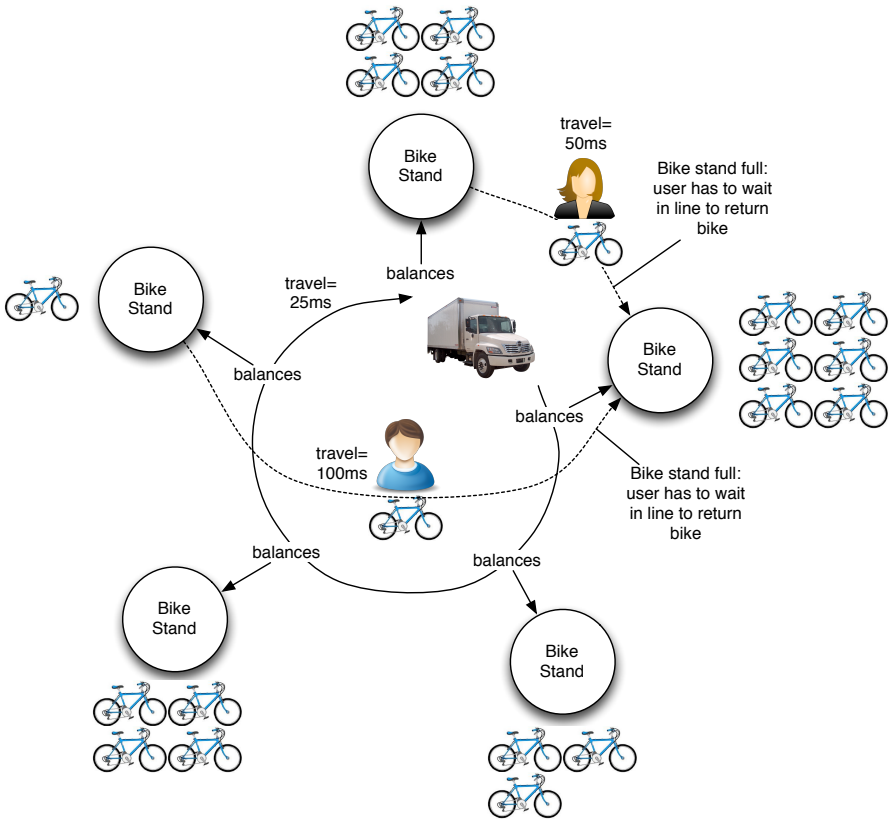

Figure 1: Illustration of the free bike rental system, with the balancing truck and two users.

1 Introduction

The purpose of this practical is to make use of synchronization in a simulation.

Several cities in the world have a system for free bike renting, e.g., Zürich, Paris, Geneva and even Neuchâtel. In a rental system, illustrated in Figure 1, a set of *bike stands* are disseminated throughout the city. Users can borrow a bike at one stand and put it back at any other stand in the city.

For the sake of simplification, we consider that when a stand is empty, a user in need of a bike waits for someone to return one. When the bike stand is full, somebody wishing to return a bike waits for a free slot. We also consider that we are in Zürich and not in Paris: people waiting for either a free slot or a bike, do it in the order they arrived at the stand and must not get any priority over the ones who where waiting before them. In other words, the ordering of accesses is FIFO (first-in, first-out).

Users pick randomly a start stand and a destination stand among the N existing. In order to emulate the fact that some stands are more popular for picking bikes, and some others for returning them (e.g., top and bottom of a hill), a user pick its start and destination stand using a biased choice. We illustrate this with the Java code that follows.

```
1  private static int[] startStands = {1,1,1,1,1,2,2,2,3,3,3,4,4,5,5,6};
2  private static int[] arrivalStands = {1,2,2,3,3,3,4,4,4,5,5,5,6,6,6,6};
3
4  private int getRandStand (int[] source) {
5    return source[(int)(Math.random()*source.length)]
6  }
```

In the above code, stand 1 could be the top of a hill, and stand 6 some popular destination such as the university. Obviously, the popularity of some stands over others result in an unbalance of the number of available slots or bikes. With the biased choice presented above, it is clear that stand 6 will often be full, and stand 1 will often be empty.

To solve this problem, cities use a balancing-truck. This truck circles around stands (in order) forever, and only stops when there are no longer any active clients.

The behavior of the truck is as follows. When it arrives at a stand S_i, it checks the amount of bikes $b(S_i)$. The maximal capacity of a stand is $b_{max}(S_i)$, which we set to 10 for all sites. The truck has the same capacity as a stand. The goal of the truck is to *balance* the number of bikes on each stand to half its maximum capacity, that is 5 bikes in our case. When the truck arrives at a stand with 3 bikes, it unloads (at most) two bikes, provided the truck is carrying enough bikes. When the truck arrives at a stand with 8 bikes it picks (at most) 3 bikes, provided it has enough spare capacity. When the truck arrives at one site, no user can pick or deposit a bike. That is, the complete operation of balancing the number of bikes must appear as atomic to users. The truck always have priority over users to access the stand, and does not wait for more bikes to come or slots to be available; it checks and only perform some operation if there is need, and then continue to the next stand.

2 Instructions

You are welcome to rename your classes and files as you wish, but do not forget to properly comment the code and make way for *concise prints*. Your comments must describe the *logic* of the application, in particular synchronization, and not what can be directly understood from the code itself.

Implement the rental system as a Java program where:

- Users are threads and are independent from each other. A user does *3 to 8* trips and only travels on the ring. It waits for 100 ms to 200 ms between each trip, and a trip takes at least 50 ms, i.e., the minimal difference between two stands along the ring. For instance, a trip between stand 5 and stand 1 takes 100 ms because the traveller has to go through stand 6.
- The truck is a thread that forever balances bikes between stands. The truck takes 25 ms between two stands, and always goes clockwise.
- The stands are shared objects that are accessed by users and the truck.

Of course, you should provide a correct solution to concurrent accesses to shared resources (stands and bikes). In particular, make sure that the priority criteria expressed above are respected: FIFO for waiting users, and the truck has priority over users to access a stand.

You shall consider that there is no resource modeling the bikes themselves. One simply keeps the number of bikes in a stand, in the truck, and used by traveling users.

Due date

The completed skeleton must be submitted before Mon, 23 May 2022 at 16:15.