CS 4345: Operating Systems

Assignment 4 (Spring 2019)
Due date: Wednesday, 24 April 2019, 11:00 p.m.

This is a group project. Each group will have two (2) members from the current class. However, only one submission is required from the group.

Write a Java program to solve the following synchronization & deadlock problem.

Two mountain resorts are connected by a road that becomes **single lane inside a tunnel** that goes through the mountain. One resort is at the left end of the tunnel and the other at the right end of the tunnel. Cars moving from left to right are called "Right-bound" cars and are indicated with odd numbers starting from 1 (1, 3, 5, …). "Left-bound" cars move from right to left and are indicated with even numbers starting from 0 (0, 2, 4, ..). The tunnel can become deadlocked if a left-bound and a right-bound car enter into the tunnel at the same time.

Task: Write a program that prevents the deadlock. You may use any synchronization technique. For example, semaphores would be good choice. You may ignore starvation, that is, the situation in which left-bound cars prevent right-bound cars to enter into the tunnel, or vice versa. You can assume that there will be steady stream of cars from each side, however, there can be more than one car from one side before we see a car from the opposite side. The tunnel may allow more than one car passing through it, but only in one direction.

User is not expected to supply any input. The user will simply compile and run your program. A segment from a sample output could be like the following:

…
Right-bound Car 1 wants to enter the tunnel.
Right-bound Car 1 is in the tunnel.
Right-bound Car 1 is exiting the tunnel.
Left-bound Car 4 wants to enter the tunnel.
Left-bound Car 4 is in the tunnel.
Left-bound Car 4 is exiting the tunnel.
Right-bound Car 5 wants to enter the tunnel.
Right-bound Car 5 is in the tunnel.
Left-bound Car 8 wants to enter the tunnel. // note: Car 8 cannot enter as Car 5 is already in tunnel
Right-bound Car 5 is exiting the tunnel.
Left-bound Car 8 is in the tunnel. // note: now car 8 can go as car 5 has left the tunnel
Right-bound Car 7 wants to enter the tunnel. // note: Car 7 cannot enter as Car 8 is already in tunnel
Left-bound Car 10 wants to enter the tunnel. // note: the tunnel may have allowed 10 to enter
Left-bound Car 8 is exiting the tunnel.
Left-bound Car 10 is in the tunnel. // note: this could have been before the previous statement
Left-bound Car 12 wants to enter the tunnel.
…

CAUTION: two cars from opposite sides cannot be in the tunnel at the same time. So if "Right-bound car x in the tunnel" appears in the output with 'x' being odd, there must be a "Right-bound car x exiting the tunnel" in the output before a "Left-bound car y in the tunnel" appears in the output with 'y' being even. Another caution: if 'car m' and 'car n' are waiting to enter tunnel from same direction, the car which is first will enter the tunnel first. Also, once a particular car enters and then leaves the tunnel, it does not come back again to enter the tunnel.

[*Hint: you may create two threads for controlling left-bound and right-bound cars. You may also use Thread.sleep() with different sleep times for left-bound and right-bound cars when they enter tunnel. This would allow simulating different time taken to pass the tunnel. You may also use synchronized controller method and use wait() and notify()/notifyAll() appropriately.*]

Submission instruction: Submit a .zip file containing your program file(s) or package through BlazeVIEW submission box.. **[Important] Naming of the .zip file:** If your group has members *Alice Dean* and *Bob Bryan* (hypothetical names) then name the .zip file as ***AD_BB.zip.***