Thank you for your interest in Kuali. After reviewing your application and discussing the opportunity with you, we think you may be a good fit for the team and culture we are building.

We kindly ask that you complete the attached assignment as part of your application. Please don't open the .pdf until you can commit two hours to the challenge. We'd prefer you write the exercise in javascript, however, you may use whatever language you'd like. The most important thing is implementing in a language that will clearly demonstrate your technical excellence. No frameworks or other dependencies are required, but during your work you can pull in whatever libraries you think might help. IDE(or not) is completely up to you. You should plan to **spend 2 hours from opening the attachment, committing to a github repo approximately every 15 minutes, please include your commits**. At the end of the time you should send us the url to the github repo for further review.

Once completed, we'll review your code and let you know the next steps.

Programming Challenge

You're designing the software that simulates elevators in a building.

Design a set of objects that will manage elevator movement and interaction between the elevators. The elevator simulation should support these features:

1. Initialize the elevator simulation with the desired number of elevators, and the desired number of floors. Assume ground/min of 1.
2. Each elevator will report as is moves from floor to floor.
3. Each elevator will report when it opens or closes its doors.
4. An elevator cannot proceed above the top floor.
5. An elevator cannot proceed below the ground floor (assume 1 as the min)
6. An elevator request can be made at any floor, to go to any other floor.
7. When an elevator request is made, the unoccupied elevator closest to it will answer the call, unless an occupied elevator is moving and will pass that floor on its way. The exception is that if an unoccupied elevator is already stopped at that floor, then it will always have the highest priority answering that call.
8. The elevator should keep track of how many trips it has made, and how many floors it has passed. The elevator should go into maintenance mode after 100 trips, and stop functioning until serviced, therefore not be available for elevator calls.

Some things to think about in this design:

1. Should you take an approach using an elevator controller, or have the elevators communicate amongst themselves to find an optimal solution?
2. How will your solution scale up and down, from 1 elevator, to a reasonably large number, like 10?

Rules of the Challenge:

1. The goal of this is to design classes and their interactions. It is primarily a design exercise, not strictly a coding exercise. So if you run out of time, stub the methods and comment on how they should work.
2. If you run out of time, focus on the features in the order they are listed as priority order
3. Be prepared to talk about and defend your design.
4. The code does not have to run! We'll just assume that the time will exist to work out the bugs and make it actually run. So, pretend it will run, but don't worry about actually making it run.