INTRODUCTION

The Nodal Exchange offices are currently on the third floor of a building with four elevators. For our software engineers on their way to work this provides a few moments each day to consider the interesting problem of elevator control systems - a few simple known inputs and a wide array of algorithmic possibilities. This programming challenge asks you to write an elevator controller and then let it interact with our elevator emulator. This file contains everything you need to know to get started. Our purpose in using this exercise is to give strong candidates a chance to show their best work. Rather than coding on the spot in a first round of interviews, we would like to offer a chance for you to solve a problem the way, the time, and in the place you prefer to work. After we review your solution, we will invite for final round interviews those whose work and potential we feel fits well with our vision and culture. Please note that we do grade on the curve and are looking for people whose work is impressive given where they are in their career. Good luck and have some fun!

OBJECTIVE

Write an elevator controller that gets all passengers where they wish to go.

RULES

\* Please do all of the work yourself without consulting others.

\* Please keep this exercise confidential as we would like it to be fresh for future candidates.

\* Please let us know if you think there is a bug in the emulator. We will want to fix it quickly and get you an improved version.

\* This is a problem with many possibilities and optimal implementations could certainly take many, many hours. We are looking for interested candidates to spend between one and three hours here (about what it would take for a first round of interviews). You are welcome to spend more but our experience has been that if in three hours you are having trouble getting an implementation that works then there might be some strong fit questions that will likely show up during the in-person interview round.

QUICK START

\* In the com.nodalexchange.elevator.controllers package, write a class that implements com.nodalexchange.elevator.ElevatorController as well as any supporting classes you wish

\* Test your class with the SystemLauncher:

java com.nodalexchange.elevator.operation.SystemLauncher 10 4 10 3 com.nodalexchange.elevator.operation.RandomPassengerGenerator com.nodalexchange.elevator.controllers.<your Elevator implementation class name> 2500

\* Observe the output and confirm that all passengers get where they are going (use a wide console so the output formatting is correct).

\* Write a short description of your approach as well as any notes for future possible implementations.

\* Send your com.nodalexchange.elevator.controllers source code only (not a copy of our full code base) back to Nodal Exchange.

\* We will run your implementation under the above conditions and review your code and notes.

\* We may also run your implementation under other conditions just for fun (e.g. other passenger generators, other building sizes, elevator capacities, etc.).

INPUTS TO SYSTEM LAUNCHER

\* number of floors in building

\* number of elevators in building

\* elevator capacity

\* number of cycles required to load elevator

\* class that generates new passengers

\* your class that controls elevators

\* last time cycle with new arriving passengers

EMULATOR NOTES

\* The emulator operates in some unit of time called a cycle.

\* The emulator generates new passengers each cycle.

\* In 1 cycle, an elevator can move 1 floor.

\* It takes the same number of cycles for an elevator to open and close, no matter how many people are getting on or off. The actual number is one of the launcher's inputs.

\* For each time cycle that passes, the controller can issue an instruction for each elevator. An instruction consists of a floor number to which you are sending the elevator for its next stop AND an indicator of which arrow (the up or the down) the elevator is answering. This last is what communicates to the passengers in a waiting area which way the elevator will go next and therefore whether or not they should get on.

\* The program ends when all passengers have reached their destinations. Note that this will take longer than the number of cycles the system is launched with since it will take extra cycles for the elevators to get to all the passengers.

\* The emulator is single-threaded and sequential, meaning that the elevators' behaviors and actions are processed one at a time in the same order during each cycle.

**\* The visibility in the operations package is intentional. Your implementation should only interact with the four interfaces in the com.nodalexchange.elevator package. You are certainly free to examine the com.nodalexchange.elevator.operation package code but it may be more of a distraction than a help.**

\* Elevators WILL NOT STOP unless you instruct them to. Elevator buttons and lobby buttons do not themselves control stops. Your controller must respond to those inputs and decide where to stop.

\* The output below is produced by the emulator after every cycle. The numbers are a count of the passengers on the elevators and in the waiting areas (as well as their intended direction). The hyphens on an elevator mean that it is currently loading and thus may be in the same place for several cycles.

Status after n cycles:

[FLOOR 9] XXXXX XXXXX -010- XXXXX 000 008

[FLOOR 8] XXXXX XXXXX XXXXX XXXXX 000 005

[FLOOR 7] XXXXX XXXXX XXXXX XXXXX 002 005

[FLOOR 6] XXXXX 000 XXXXX XXXXX 003 002

[FLOOR 5] XXXXX XXXXX XXXXX XXXXX 005 002

[FLOOR 4] XXXXX XXXXX XXXXX XXXXX 006 001

[FLOOR 3] XXXXX XXXXX XXXXX XXXXX 009 000

[FLOOR 2] XXXXX XXXXX XXXXX XXXXX 005 000

[FLOOR 1] -001- XXXXX XXXXX XXXXX 000 000

[FLOOR 0] XXXXX XXXXX XXXXX 009 000 000

------------------------------------------------

ELEV0 ELEV1 ELEV2 ELEV3 WAITING WAITING

TO GO UP TO GO DOWN

Interior buttons currently selected:

ELEVATOR 0: 5

ELEVATOR 1:

ELEVATOR 2: 2 3 4 5 6 7

ELEVATOR 3: 2 6 8 9