Alex Averbook
COSI 105b
Elevator Simulation README

This is the README for Alex Averbook's Elevator Simulation Final Project. The code can be found in the github public repo:

https://github.com/alaverbook/Elevator_Simulation

The repo, when run through code climate, received a grade of 3.3:

https://codeclimate.com/github/alaverbook/Elevator_Simulation

Upon analyzing the results of the code climate report, the biggest room to improve would be to break apart the lengthy elevator logic within building.rb that acts as the brain of the elevators' movements. It at first appears to be some nasty if-logic containing duplication throughout, however as a result of the importance of context for the elevator's movement and from having decoupled the Elevator class from the Floor class, this chunk of logic is necessary.

My simulation can be run with 6 floors, 2 elevators, and 50 people by copying:

ruby simul.rb

and pasting it into the command line. If you would like to change the number of floors, elevators, or people, go to the bottom of *simul.rb* and change the numbers in the line: *sim = Simulation.new(6, 2, 50)*

It runs for 20 ticks by default but this can be changed by going to the bottom of *simul.rb* and changing the number in the line: *sim.run(20)*

The building, floors, and elevators are generated consistently, and the people are generated and randomly placed on previously generated floors with random desired destinations. The object classes in my program are a Simulation class within *simul.rb*, a Building class within *building.rb*, a Floor class within *floor.rb*, an Elevator class within *elevator.rb*, and a Person class within *person.rb*. The structure of my classes are as follows:

Simulation (creates the simulation and controls the ticking of the clock)
- Building (knows floors and elevators; controls elevator movement and direction at every tick)
  - Floor (knows people on the floor; controls elevator call buttons at every tick)
  - Elevator (knows people on the elevator and what floor it's at; controls who gets on and off the elevator at every tick)
    - Person (knows where they are and where they want to go; controls nothing)

This structure makes the most sense because it decouples as many objects as possible. Inherently the classes rely on each other since if a building knows the floors and elevators, but the elevators and floors do not know each other except for the current location, then the building must control the elevators between floors.

Alex Averbook
COSI 105b
Elevator Simulation README

Apart from the structure and how to run it, I need only explain the logic of the simulation itself. First, elevators are ticked to remove people from them if need be then add people to them if need be. Next, people are ticked. Then, the floors are ticked and elevator calls are made based off of the desires of the people on each floor. Finally the building is ticked and if people are on the elevator, then it will move in the direction of the people's desire; otherwise the elevator will determine if elevator calls have been made and begin to move to the nearest floor with an elevator call. The elevator swapping directions from up to down or down to up is a single tick as well, and if there are no elevator calls, and the elevator is not on the ground floor, and the elevator is empty, then it will travel to the ground floor.

A single building printout with 6 floors, 2 lifts, and 4 people can look like (note: I chose this design because although it is not the most consolidated, it shows the most accurate information):

```
-^-^-^-^-^-^-^-^-^-^-^-^-^-^-^-^-^-^-^-^-^-^-^-
Floor 5 : elevator calls? no
Floor 4 : elevator calls? no
Floor 3 : elevator calls? down
        Person desires Floor 1
                [Lift 1 : 2 passengers : up]
Floor 2 : elevator calls? down
        Person desires Floor 1
                [Lift 2 : 0 passengers : down]
Floor 1 : elevator calls? no
Floor 0 : elevator calls? no
---------------------------------------------
```