CISC 310, Fall 2014

Project #2: Concurrency and Deadlock

Details

- We will do this project with Java threads
- You can work in a group of four

Overview

Your assignment is to design and program a multithreaded simulation of the operation of the elevators in the NerdsRun, Inc. headquarters.

There should be one thread for each elevator. Each elevator thread should be responsible for all interaction with the hardware of its respective elevator: it should control operations such as opening and closing the elevator doors and moving the elevator between floors. The elevator threads should be coordinated so that exactly one elevator picks up each set of passengers whenever possible.

Your program will simulate a building containing a set of elevators, with people entering and exiting the elevators at various times. All the elevators should run in parallel, as should all the people (so the total number of threads in your system should be at least the number of elevators plus the number of people).

`You should have variables:`
`n` is the number of elevators (numbered consecutively starting from 0),
`f` is the number of floors (numbered consecutively from 0),
and *`file`* is the name of a text file describing the people's behavior. The simulation ends when every person has visited all the floors they wanted to visit. The file should consist of a sequence of lines each of the form

`name [f0; f1; ...; fn] t s`

where

- *`name`* is a string containing the person's name
- *`[f0; f1; ...; fn]`* contains a list of floor numbers that the person wishes to go to, in order from *`f0`* to *`fn`*
- *`t`* is the time, in seconds, that the person spends on each floor after they leave the elevator
- *`s`* is the floor the person starts on.

For example, a file containing

```
bill [1; 2; 0] 2 0
jane [2; 1; 0] 4 0
```

would mean that both `bill` and `jane` start on floor 0, `bill` goes to floor 1, waits 2 seconds, then goes to floor 2, waits 2 seconds, and then goes to floor 0 and waits 2 seconds; and `jane` goes to floors 2, then 1, then 0, waiting 4 seconds upon arriving at each floor.

As the people participate in the simulation, they should print out messages to standard output saying what they are doing:

- *name* `waiting on` *f* `for floor` *g* should be printed when *name* is currently on floor *f* and is waiting to board an elevator to go to floor *g*
- *name* `taking elevator` *e* should be printed when *name* boards the elevator numbered *e*
- *name* `arrived at floor` *f* should be printed when *name* disembarks from the elevator at floor *f*

All of these messages should each be printed on their own line. Collectively, this trace output captures what happened during the simulation.

Each elevator may hold at most 3 people. When an elevator arrives at a floor, it should print

`Elevator` *e* `arrived at floor` *f*

where *e* is the elevator number and *f* is the floor number. Then the elevator should sleep for 1 second, during which time people may enter or exit the elevator. After the elevator wakes up, it should print

`Elevator` *e* `serviced floor` *f*

An elevator should service *every* floor it reaches, and the floors must be serviced in order (i.e., after servicing floor 1, the elevator should either service floor 2 or 0 next, depending on its direction.)