CS 245 - Assignment 2 Fall, 2023

Assignment 2 - Elevator

This assignment tests your ability to use the data structures we studied so far to create a simulation of one or more elevators.

Background

We (will have) discussed many general-purpose data structures, including:

- Lists (with LinkedList and ArrayList implementations)
- Queues
- Stacks
- Heaps / Priority Queues

This simulation should be focused on an object-oriented solution — i.e. objects should be instantiated, should be stored in various collections / containers (Lists, Stacks, Queues, Trees, etc.) and should be de-referenced[1] when appropriate.

There is no starter code for this assignment.

Requirements

There are several requirements for this assignment. These requirements are interdependent, meaning that a proper implementation requires that all the requirements be implemented and used in a cooperative system fusing each of the individual requirements.

Requirement 1: Read the (possibly missing) property file.

A property file will indicate the correct values of variables in your system. Each line in the property file contains a KEY=VALUE pair, for example:

```
structures=linked
```

… where “structures” is the key and “linked” is the value. The meanings of the keys and values are specified in Table 1. If a property file is present, it will be passed to your implementation as the first argument and available as `args[0]`. The choices in this property file include everything listed in Table 1.

[1] Because Java has a garbage collector, any object that is de-referenced (eg. by assigning an object to null) will be effectively destroyed by the garbage collector. In languages without garbage collectors, like C++, objects must be actively destroyed (eg. with the “`delete`” keyword).

If the property file is missing or is not provided, your implementation must use the values found under the column “Default” in Table 1.

Your implementation may [read: is encouraged to] use the `java.util.Properties` class to manage the property file. (This allows you to get the correct key-value pairs even if the property file contains irregularities, such as spaces, separations other than the “=” sign, etc. You can read more about the Properties class at GeeksForGeeks (https://www.geeksforgeeks.org/java-util-properties-class-java/).

Key	**Valid values & Meanings**	**Default**
structures	• *linked*: All collections / containers must use linked data structures (eg. LinkedList, Linked-based Queue / Linked-based Stack) for containing objects. • *array*: All collections / containers must use array-based structures (eg. ArrayList, Array-Based Queue, etc.) for containing objects. Some data structures (trees and heaps) have only one implementation (linked and array, respectively). These data structures are exempt from this requirement.	linked
floors	Valid: any integer >= 2. This is the number of floors in the building serviced by the elevator(s).	32
passengers	Valid: any real number > 0 and < 1.0. This represents the probability that a passenger appears on any floor and requests an elevator in order to go to a different floor. This probability applies to each floor in the simulation and is applicable for each “tick.” Any passenger appearing on a floor must have a *destination* floor different from the floor on which the passenger appears. Passengers only get on an elevator going in the desired direction — i.e. a passenger going “up” will never get on an elevator going “down.”	0.03
elevators	Valid: any integer >= 1. This represents the number of elevators in the simulation. An elevator moving in one direction (eg. “up”) will continue moving in that direction while there are requests for passenger pick-ups or drop-offs in that direction.	1
elevatorCapacity	Valid: any integer >= 1. This represents the maximum number of passengers per elevator.	10
duration	Valid: any positive integer >= 1, representing the number of “ticks” in the simulation.	500

Table 1: Property file keys, values & meanings and defaults

Requirement 2: Run the simulation.

Running the simulation requires instantiating objects which span the duration of the simulation (eg. elevators) at the start of the simulation. Other objects are created only as needed and de-referenced when no longer needed. For example: passenger objects should appear only when requesting transport and should be de-referenced once an elevator takes them to their destination floor.

The simulation runs for a number of time units, called “ticks,” as specified in the property file with the “duration” key. During a “tick,” any the following items may occur:

- *Elevator unload & load*: An elevator may stop at a floor and unload all passengers in the elevator bound for that floor. Additionally, during the same “tick”, any passengers on the floor waiting for an elevator going in the desired direction (up or down). You may assume that passengers never enter an elevator going in the wrong direction.
- *Elevator travel*: An elevator may travel between no more than 5 floors (eg. from the 8th floor to the 13th floor).
- *New passengers*: Given “passengers” — the probabilities in the property file — a new passenger may appear on a floor and request transportation to another floor.

Requirement 3: Report the results of the simulation.

Once the simulation concludes, your implementation is required to report the following items:

- Average length of time between passenger arrival and conveyance to the final destination
- Longest time between passenger arrival and conveyance to the final destination
- Shortest time between passenger arrival and conveyance to the final destination

Submission

You are required to submit your complete implementation for this assignment:

1) One or more Java classes with code necessary to complete all the requirements above.
2) Optionally, you may submit a PDF or markdown (eg. README.md) document explaining any of the following:
    a) Design choices and OO decomposition of the problem
    b) Data structure choices
    c) Instructions on how to get your implementation compiled and running correctly
    d) Anything else you believe might be useful

On Canvas, submit the link to your github repository containing these items.

Grading

Grades will be determined as follows:

- 50% = Implementation
    - 10% Requirement 1 = Read the property file
    - 30% Requirement 2 = Run the simulation

    - 20% Requirement 3 = Report the results of the simulation
- 20% = Design, including use of data structures to create this simulation
- 10% = Efficiency of code and algorithm
- 10% = Style, including consistent indentation, variable naming, etc.
- 10% = Documentation, specifically:

Please submit your assignment before the due date. The syllabus contains information on policies for late assignments.