605.202 Data Structures
Programming Assignment 1: Elevator Stimulation using Stacks
Author: Boshika Tara
Fall 2016

Project Name: Elevator Stimulation
Team: Solo Project
Assignment 1 for Data Structures
Term: Fall 2016
Johns Hopkins University

605.202 Data Structures
Programming Assignment 1: Elevator Stimulation using Stacks
Author: Boshika Tara
Fall 2016

TABLE OF CONTENTS

605.202 Data Structures
Programming Assignment 1: Elevator Stimulation using Stacks
Author: Boshika Tara
Fall 2016

Abstract

This application stimulates an elevator, where passengers have certain floors they get on and off at. The application has been designed to demonstrate the usage of stacks data structure, in Java. The algorithm uses the LIFO characteristic of stacks to create an elevator like stimulation, where passengers enter, and exit, and move up and down, using Stack methods.

605.202 Data Structures
Programming Assignment 1: Elevator Stimulation using Stacks
Author: Boshika Tara
Fall 2016

Design and Implementation

The basic premise for this algorithm to efficiently use an elevator, for multiple people was to design it by making people who want to exit as the focal point of the program. This was done by creation of two separate stack objects, one to hold all the passengers that are currently in the elevator, and other to hold passengers that need to temporarily get off to make way for the people exiting the elevator. Some of the additional feature of this program include checking to see if the elevator is full, and if so notifying the passenger, so they can take the stairs.

The program contains three different classes

- Stacks.java: Represents a last-in-first-out(LIFO) stack of generic items. Methods supported are push, pop, peek, test to see if a stack is empty, getting number of items in the stack, and iterating over items in a LIFO order. It is a stack implemented using a linked-list.
- Passenger.java: This class basically sets up methods that returns all the information associated with the people using the elevator, it implements methods like getName(), getEntryFloor(), getExitFloor(), getTemporaryExit().
- ElevatorSimulation.java: This is the main algorithm for the elevator stimulation. It sets the default starting floor to be one. Initially the algorithm is set to check if there are people who want to get off. The way this is done to check the current floor against the target floor for all people in the list, if someone wants to exit then using the LIFO of a stack people standing in front who do not want to get off are removed, this done by using the pop() method. The people removed are then stored in the a temporary Stack, while people who have reached their target floor are removed. Once this is achieved, the people from the temp stack are put back in the elevator using the pop(), and then push() method. At this point the algorithm checks to see if there is more space in the elevator or if the elevator is full, if the elevator has space, and there are more people waiting to get on, then they are pushed as well using the push() method, till the elevator is at capacity of 5. Once at capacity, people who are still waiting are told to take the stairs.

  The algorithm makes certain basic assumptions:
  - Each person has a fixed entry and exit point
  - Each person only rides the elevator once
  - Elevator stimulation is linear
  - People who have to temporarily exit are loaded first in the elevator, before addition of new people from the passenger list
  - People who cannot get into the elevator because it is full, will always have to take the steps

Use of Stacks

The implementation basic premise is to use stacks to keep track of passengers. One of the main reasons why stacks is a good way to do this is because of LIFO. A scenario to further explain this, would be say if three people are riding in the elevator, and person in the middle wants to get off, the most efficient way to do this would be for the passenger standing closest to the door, to exit to make room for the person who wants to get off, and since the first passenger to exit, was also the last one to enter the

elevator, it would be perfect for us to use a stack like method to implement this. Also, this goes to the point of scalability, using a stack would also be helpful, if more passengers start taking the elevator.

Recommendations

As noted above, this specific stimulation makes certain assumptions. If those assumptions were removed that would further complicate the scenario. For example, in cases where people has multiple exits and entry points, this would create a logjam, and the algorithm the way it is currently designed is not able to handle that scenario. If such a thing arises, the best way to tackle this would be to limit elevator usage. This can be done by setting a threshold for how many times a passenger can ride, if the passenger reaches the threshold, he/she is removed from the Stack(elevator), and asked to take the stairs. Another way to handle this would be the second elevator, since this is an only five floor building, it might not make an impact, but if it was more than five floors, two separate stacks could be set up. Where one stack(elevator 1) only services floors 1-5, the second stack(elevator 2), service floor 5-10. This would create more space for more people to ride, this can also be done with five floors.

If I had to enhance this stimulation, I would build the algorithm in a way that checks also for the number of floors people have to move through before they reach there exit floor, for example if a passenger only has one floor between the entry and exit point, and other passengers behind them have more than one floor between the entry and exit, I would pop this passenger of the list, and ask them to use the stairs, and add other passengers to the elevator. Setting up this sought of a checkpoint would help ensure passengers who have to travel longer distances get priority.