Problem Set 2

Due: 4:30PM, Friday February 22, 2002

In this problem set you will describe the operation of an elevator in several situations.

Problem 1. Single Elevator [35%]

Write an elevator class with the following methods:

1. Constructor taking 3 arguments:
   a. Lowest floor served (usually floor 1)
   b. Highest floor served
   c. Current floor of elevator
2. Constructor taking 2 arguments:
   a. Lowest floor served
   b. Highest floor served
   c. Assume current floor is the lowest floor at the start
3. Constructor taking 1 argument
   a. Highest floor served
   b. Assume lowest floor= 1, and current floor= lowest floor to start
4. Request, which takes 1 argument, the requested floor, and returns nothing:
   a. If the requested floor is greater than the highest floor, lower than the lowest
      floor, or equal to the current floor, do nothing
   b. Else if the requested floor is above the current floor, move the elevator to the
      requested floor. Print out a line stating the current floor, then (possibly
      multiple) lines stating the elevator is going up past each intermediate floor,
      and last a line saying the elevator stopped at the requested floor.
   c. Else if the requested floor is below the current floor, provide similar output
      showing the elevator's progress.

Write a main() method in an ElevatorTest class:

1. Using a JOptionPane, ask the user to enter the maximum floor served by the elevator.
2. Using the one-argument constructor, create an Elevator object.
3. In a loop, using a JOptionPane, ask the user to enter the floors requesting the
   elevator. Tell the user to enter –1 to terminate the program.
4. Invoke the request method to move the elevator and produce output.

Problem 1A. Multiple Elevators, Without Coordinated Control  [0%]

(This problem is optional; do not hand it in, but it's worth doing to make sure you are ready to go on to problem 2)

Write a revised main() method to create an elevator bank of two or three elevators, using the Elevator class you wrote above. Have each elevator operate separately, responding to requests only to it. (There is no coordination among elevators.) Make sure you understand how multiple objects of a class are created, named and accessed.

Problem 2. Multiple Elevators with Integrated Control [65%]
You are to design and implement a program to allow a user to request an elevator at a floor. The request will be met by the elevator from an elevator bank that can respond by moving the fewest floors. The number of elevators in the elevator bank is exactly 3 in this problem, and all of them serve the same floors. We haven't covered arrays yet, so you'll have to keep track of the elevators individually. Your "integrated controller" class needs only to work with three elevators; it does not need to be more general.

This problem is a little less structured than the problems you've seen so far. You must design an additional class (Controller), its methods and member variables. Its member variables must be private. The class must have a constructor and methods. You will also write an ElevatorTest2 class that has only a main() method, handling input via JOptionPane, creating objects and invoking methods on them, but not implementing any detailed program logic. You will also make some minor extensions to your existing Elevator class to support the controller. You may copy your program from problem 1 into a new Java source file as a starting point for this problem.

The requests are entered via a JOptionPane as before. When a request is made:

1. Display the current location of the three elevators
2. Display the elevator chosen to fulfill the request
3. Have the elevator show its movement as in problem 1

Use the "2-argument" Elevator constructor in your main() method. You will probably need to add an argument to all 3 constructors with a name for each elevator, so that you can output which elevator is handling the request, etc. Thus, your "2-argument" constructor will turn into a 3-argument constructor, etc.

The Controller class models the integrated control of the three elevators. It will have one key method that will obtain the current floor of each elevator (you'll need to add an accessor method to Elevator), compute the closest elevator, and invoke the chosen elevator's request() method to actually service the user request.

At the end of the program, display the total number of requests, the number of requests served by each elevator, the total number of floors moved by each elevator, and the average number of floors moved per request by each elevator. Modify the Elevator class you wrote in problem 1 so that each elevator keeps track of its own number of requests and floors moved, and can report these totals when asked. Ignore invalid requests (requests for floors that are too high, too low, or the current floor) in the totals.

**Turnin**

**Turnin Requirements**

- Problem 1: Email only. No hardcopy required.
- Problems 2 & 3: Hardcopy and electronic copy of ALL source code (all .java files).
- Place a comment with your name, username, section, TA's name, assignment number, and list of people with whom you have discussed the problem set on ALL files you submit.
- DO NOT turn in electronic or hardcopies of compiled byte code (.class files).

**Electronic Turnin**
To electronically turn in your problem sets, run Netscape

Then go to the 1.00 web page at:

http://command.mit.edu/1.00Spring02

Click on the "Submit Assignment" button. Be sure to set the Selection Bar to Problem Set 1 or your files may be lost. Finally, go back to the home page and click on the "View" section and be sure that your files were received. If you submit a file twice, the latest version will be graded.

**Penalties**

- Missing Hardcopy: -10% off problem score if missing hardcopy.
- Missing Electronic Copy: -30% off problem score if missing electronic copy.
- Late Turnin: -20% off problem score if 1 day late. More than 1 day late = NO CREDIT