05-630 Programming Usable Interfaces Assignment 2a:
Use all the components, and elevator simulation
Due: 3:00pm (before class), Tuesday, Feb 17

Objective

The goal of this assignment is to make sure you can use all the common components, get you to do some simple design of an interface, and give you experience with a non-passive (real-time) interface.

To Hand In:

You must write two programs, which you will hand in like last time (submit .zip files to the digital dropbox on blackboard). Please include comments in all of your code, otherwise we won't be able to understand it, and you will lose points.

Note: your .zip files should include an executable (.swf) file of your programs. Flex Builder will do this automatically, and put it in the debug-bin folder. (Note that this allows you to run your code without opening up Flex Builder). When you submit your project be sure to name the .zip files like this: [firstname]-[lastname]-hw2a-[projectname].zip Zhiquan Yeo, would submit these files:

zhiquan-yeo-hw2a-useall.zip
zhiquan-yeo-hw2a-elevator.zip

Useall.zip -- Use (Almost) All the Components:

For **useall**, the components should be in a clear order, so that we know which one to interact with first, then second, etc. As we interact with the first, it should clearly change something about the second, then interacting with the second clearly changes something about the third, and so on. If you have multiple components that all want to be acted upon, but not act on anybody else, make them all be at the end of the chain, affected by the last one that handles input. You can do this in a mundane way, or you can try to be clever. However, you probably do not want to invest too much time in it, as this part of the assignment is basically pass/fail. Components you must include are:

- Label
- TextInput
- Button
- CheckBox
- RadioButton
- ComboBox
- List
- HSlider
- VSlider
- Image

elevator -- An Elevator Simulation

This program requires you to write an interface that isn't "passive," to use arrays, to use event handlers for multiple controls, to write a fun simulation, to handle boundary conditions (no elevators going through the roof, please!), and to use subroutines. Your program should show the user the current state of, and allow the user to interact with, a running elevator simulation. The user is given a "God's eye view," meaning that they are able to generate events that all the people in the building would generate. Specifically, the rules of the simulation are:

- There are three elevators.
- There are ten floors.
- Each elevator is always on a specific floor, and always "headed up" or "headed down"
- All elevators start on floor four, headed down.

- On each floor, there is a single "call button," and a light that shows if it has currently been pressed, but not serviced. When the user presses that button, the light goes on. When the floor is "serviced" the light goes off. If the user presses a light that's on, nothing happens (just like in real life :-) The user can press buttons at any time.
- Once per second, a single elevator should run the following algorithm. The leftmost elevator should go first, then a second later the middle one, then the rightmost one, then the leftmost one goes again, and so on.

IF the elevator is headed up AND there are pending calls above the elevator THEN
move up one floor
turn off the light on the new floor, if it's on (since it has now been serviced)
ELSEIF the elevator is headed down AND there are pending calls below the elevator
move down one floor
turn off the light on the new floor, if it's on (since it has now been serviced)
ELSE
keep elevator on the current floor
switch the direction it's going
turn off the light on the current floor, if it's on (since it has now been serviced)
ENDIF

Until you press a button, your elevators should happily sit on the floor they start on; each second, one of the elevators will switch its direction (which should be visible as your simulation runs – how you make this apparent is up to you). You will be graded primarily on the correct functioning of your code, but also on the appearance and usability of your interface. Note that design does not make up for functionality on this assignment. If your elevators do not function **exactly** as specified in this handout, you will receive a poor grade. The algorithm must be implemented exactly as described above. See the instructor or one of the TAs if you are not sure about how to do that. Once you have the basics up and running, feel free to embellish the simulation in reasonable ways. The entire state of the simulation should be visible at all times.

Note: although functionality is strictly graded, you shouldn't necessarily code first and design later. If you design for usability first and plan your implementation around your design, you're much more likely to have a functional and usable interface.

As Always….

Make sure your code is written in a general way, allowing easy modification at a later time (part B of the assignment will ask you to modify your program in some way). Starting early is highly recommended; if you have questions, email the instructor and/or TAs using Blackboard.