**Problem Set 5**

**Due: 4:30PM, Friday March 22, 2002**

**Problem 1 Swing, Interfaces, and Inner Classes. [15%]**

To help you prepare for Problem 2, you are to write a simple Swing application which uses an anonymous inner class to control the application.

![Powers of Two](image)

1. Create a Pow2Frame class that extends JFrame
   a. Define the data members you’ll need:
      - a “Double” button
      - a “Value” field.
      - A panel to hold them both.
   b. A three line main() method, as in lecture notes
   c. Constructor, which:
      - Gets the contentPane
      - Creates the panel holding buttons, etc. and adds it to the pane
      - Creates the label, text field, and button and adds them to their panel
      - Packs the frame to a decent size.
   d. Within the constructor, you should write an anonymous inner class:
      i. ActionListener, with an actionPerformed() method for the “Double” button. The method should:
         - Read the current String value of the value field
         - Convert it to an int. (See `Integer.parseInt(String s)`)  
         - Double it.  
         - Convert it back to a String. (See `Integer.toString(int i)`)  
         - Set the value field to the new String.

Do not worry about what happens if the field does not contain a valid integer.

**Problem 2. Swing. [85%]**

You are to write a graphical user interface for the bus service planning system that you coded in homework 1. We have converted your homework 1 program into a
BusSystem class with appropriate fields and methods that you can use for the calculations in this problem set. Download it from the 1.00 Web site.

The graphical user interface must have the following general appearance:

The user interface should prompt the user to enter:

- The change in bus travel time, using a text field. (Use 0 to 10 minutes to test)
- The change in Logan airport parking charges, using a combo box with the values 0.00, 2.00, 4.00, 6.00, 8.00 and 10.00. (Remember to convert these from dollars to cents before using them in the analysis.)
- The objective weight to be used for the analysis, using a text field (Use 1.0 to 4.0 to test)
The interface must also have a 'Calculate' button to do the calculations and display the result, and a 'Quit' button to exit the program. The interface must draw a picture showing:

- The outer ($L_2$) boundaries of the service area, including the radial edges
- The inner ($L_1$) boundary of the service area
- The routes, shown at a scaled length ($L_1 + 5$), radiating from Logan airport, and spaced the proper angle apart, based on the optimal analysis result for route spacing
- Logan airport should be labeled with a String
- The headway, fare and number of routes, as Strings
- The ridership, revenue and cost, as Strings

You are free to choose the colors, fonts, size, etc. Your solution does not need to look elegant; it can be plain as long as it’s correct.

A suggested outline for your program is:

1. Create a BusFrame class that extends JFrame
   a. Define the data members you’ll need:
      - Buttons, combo boxes and text fields
      - Panel to hold the buttons, combo boxes and text fields
      - Reference to a BusDrawing object (you’ll write the class below), which extends JPanel and will hold the drawing
      - Reference to a BusSystem object that will do the calculations
      - Frame size (x, y)
   b. A three line main() method, as in lecture notes
   c. Constructor, which:
      - Gets the contentPane
      - Creates the panel holding buttons, etc. and adds it to the pane
      - Creates the BusSystem object
      - Creates the BusDrawing object and adds it to the pane
      - Creates the text fields, combo boxes, labels and buttons and adds them to their panel
   d. Within the constructor, you should write two anonymous inner classes:
      i. ActionListener, with an actionPerformed() method for the calculation button. The method should:
         - Read the current value of the combo box and text fields
- Invoke the calculation method in the BusSystem object. You probably need to use setParam() and optimize().
- Invoke the busLines() method in the BusDrawing object, which will generate a vector of shapes to be drawn (see below)
- Invoke the repaint() method of the BusDrawing object (which you don’t have to write) to draw it.

ii. ActionListener, with an actionPerformed() method for the quit button. The method should call System.exit(0).

2. Create a BusDrawing class that extends JPanel
   a. Define the data members you’ll need
      - Vector of shapes (arcs, lines) to draw
      - Reference to the BusSystem object, from which you’ll get the route spacing, etc. that you need for your drawing
   b. Constructor, which creates an empty Vector and takes a BusSystem reference as an argument
   c. A busLines() method, which
      - Creates the outer boundary as an Arc2D.Double object. See Javadoc; note that the arc constructor assumes angles in degrees. Assume the outer boundary goes from 64 degrees to 296 degrees. Use the Arc2D.PIE parameter for the arc type.
      - Creates the inner boundary as another Arc2D.Double object. It also goes from 64 degrees to 296 degrees. Assume its radius is half the outer boundary always.
      - Creates the correct number of routes as Line2D.Double objects. You may use a simple rule to draw these. Draw the first route at 64 degrees plus route spacing. If the route spacing were 40 degrees, for example, the first route would be drawn at 104 degrees. Continue generating lines while the angle is less than 296 degrees. Since the number of routes is not an integer, a simple approximation is sufficient. Draw the lines at the correct scale length. You will need to use Math.sin() and Math.cos(); remember their arguments are in radians.
   d. A paintComponent() method, which overrides the JPanel paintComponent() method. You should:
      - Invoke the superclass paintComponent() always
      - Get the 6 outputs and performance measures from the BusSystem object that you need to display. Use the drawString() method to display them
      - Label Logan airport at the center of your circle
      - Loop through the Vector of shapes generated in busLines() and invoke the draw() method on each. Choose colors, line widths, etc. as you wish.
3. We have rewritten the solution to homework 1 for you, to convert that program into a BusSystem class. This is a quick process: what we did was to:
   a. Make all the input parameters (costs, coefficients, distances, etc.) private members of the class. Initialize all of them to the same values used in homework 1, except b, x, and y, which we will vary just as in homework 1. We don’t need yMin, yMax or yIncr, since we won’t be looping over values of y.
   b. Make all the outputs (headway, fare, etc.) and performance measures (riders, revenue, etc.) be private members of the class. Don’t initialize them.
   c. We don’t have to make the intermediate variables (A0, B0, etc.) members of the class; they can be local variables in the optimize() method, described below.
   d. Create a constructor that takes b, x and y as arguments and sets the appropriate private variables. We also wrote a setParameters() method that sets these 3 variables.
   e. Create a method called optimize() that computes all the intermediate values, optimal values and performance statistics for that value of y. This uses the same code that we already wrote for homework 1: Just remove the loop over objective weight y; that control logic will be managed by the class that calls our optimize() method.
   f. We don’t need any of the JOptionPane inputs, etc. This will all be handled by the graphical user interface that you will write in this homework.
   g. Write a set of getXXX() methods to get the optimal values and performance statistics. These are just one line methods that return the appropriate private variable. We’ll need these to display our outputs.
   h. That’s it. This required almost no new code. This class is not as general as we would like, but it’s sufficient for this homework. In a more perfect world, the class would allow all parameters to be set (through set methods) and gotten (through get methods), etc. Doing all this is straightforward but tedious and is not necessary for this homework.

The BusFrame class should be about 100 lines of code, and the BusDrawing class is slightly shorter. We provide the BusSystem class, which is also about 100 lines of code.

Testing: Test on time changes from 0 to 10 minutes, parking changes from $0 to $10, and objectives from 1.0 to 4.0. If you go beyond these ranges, the model may go outside its range of validity. We simplified the model for the homework, so it’s not particularly robust.

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