CSE 142 - Su 02
Homework 7
Assigned:  Wednesday, August 14
Due:  Wednesday, August 21, BEFORE MIDNIGHT

This file describes the programming project.  There are other files that describe the practice problems and the graded problems.

This project is based on the King County Metro Transit bus system.  In this homework project, you will write a Java class that we can use to create visual displays of the buses and their locations, using real-time data or using a local copy of a small set of the data.

As provided to you, the BusDisplay program cannot be run because it is missing the class to build the map symbols indicating a bus location.  Your task is to write the TransitBusSymbol class that is used to display a bus symbol on the map.

There are two programs involved in this project.  One program is the one that draws the map.  Your new class is part of this program.  The map drawing program can read a disk file for its data, so you can do all the development you need without connecting to the net.  The other program is used when you want to display realtime data off the Internet.  This program is completely written, you don't need to do anything with it except start it running.

** Source of the data **

There is a file (UDistrict.txt) containing a snapshot of data that you can use during development.  To run the display program using this data, use the batch command file runBusDisplayFile.bat.  This program is a little more difficult to run from BlueJ, because BlueJ does not necessarily start in the right directory, so I suggest that you use BlueJ as an editor and compiler if you like, but that you run the program using the batch file.  Just double-click the file to start the program running.

When you want to connect to the network, there is a program called BusRcv that you can run on your machine to connect to the actual bus information server, available at sdd.its.washington.edu, port 8412.  When you are ready to use it, make sure that you are connected to the Internet, and then run BusRcv by double-clicking on runBusRcvNet.bat.  After BusRcv is running, then start BusDisplay running by double-clicking on runBusDisplayRcv.bat.  You only need to start BusRcv once and then leave it running.

** Destination of the data **

BusDisplay.java is the main program.  It contains the "static void main(String [] arg)" method where Java starts running the program.  This method creates a new TransitMap object and then starts reading bus events from the input stream, either a network connection or a local disk file.  BusDisplay.java is complete as provided to you and does not need to be modified.

* TransitMap.java *

This class loads the image file that is the University District map, creates the graphics window to show it in, and then listens for updates to bus positions and manages the resulting list of buses. When new TransitBusEvents are reported by BusDisplay, TransitMap checks to see if it already has a symbol for this bus.  If necessary, it creates a new TransitBusSymbol object (that's you) and tells the symbol where it should be on the map.

* TransitBusSymbol.java *

TransitBusSymbol is the primary class of interest in this assignment. A TransitBusSymbol object takes information about a bus and its location, and creates a little display symbol for the map. The symbol is clickable, and so the object also can do things in response to being clicked. Your job (described below) is to implement a functioning TransitBusSymbol class.

* BusReader.java *

This class reads and interprets events from the network or from a disk file. This class is complete as written and does not need to be modified.

++ Project Requirements ++

1. Download the cse142-hw7.zip file and unzip it. The project skeleton is in the directory hw142\hw7. Your task is to create TransitBusSymbol.java to provide the required capabilities described below. The documentation for the code in the hw7 directory is in subdirectory doc. Double-click on the file index.html to get started. Among other things, you may want to look at methods pixelX(lon) and pixelY(lat) in class TransitMap, since this is how your symbol figures out where to place itself on the map image.

The rest of the code I am providing to you for this project is bundled up in the file hw7.jar. The documentation for the files in hw7.jar is in subdirectory doc-hw7. Double-click on the file index.html to get started. Among other things, you will want to look at hw7.ClickableProp, since this is an interface that your symbol class must implement.

2. In order to compile the program with BlueJ, you need to tell BlueJ about the jar file, just like we did for the hw6.jar file. In BlueJ, select menu item Tools->Preferences, then the Libraries tab. Next to the area labeled "User Libraries" there is a button "Add". Click on this button, then navigate to the hw7.jar file that you just received. Select it and close up the dialog, then restart BlueJ.

After you have set the library file and restarted BlueJ, you should be able to edit and compile the code. Of course, it won't compile correctly until you write TransitBusSymbol.

If you are using JEdit or some other development environment, be sure to tell it where the hw7.jar file is so that it can do the compilation.

3. Once you have written the TransitBusSymbol class, you can compile it and start running the program. To run the BusDisplay program and show a map based on the little snapshot of data on disk, you should use the batch command file runBusDisplayFile.bat. Double-click on the file and it will start the program, create a map window, and add a bunch of buses to the display. The display should look like the one provided on the homework assignment page.

If you expand the size of the window by dragging it in the lower right corner, but the map doesn't expand to fill the window, try clicking in the window. That should cause the window to be redrawn completely.

To stop the program, close the map display window by clicking the close box in the upper right corner, then highlight the DOS command window and press any key.

4. Your task is to implement the TransitBusSymbol class. The javadoc comments for TransitBusSymbol methods are given in TransitBusSymbolComments.txt. The specific implementation requirements are as follows.

4a. The TransitBusSymbol class implements the ClickableProp interface. This interface extends the uwcse.animation.Prop interface. The specific methods that they require are described below.

4b. Implement a constructor for the TransitBusSymbol class that takes a single TransitBusEvent as a parameter variable.

Initialize an instance variable for the display window to null, since your symbols hasn't been added to a window yet. Remember the vehicle id number, lat, lon, and route information from the TransitBusEvent object by storing them in instance variables, either individually or as part of a TransitBus object.

Create and remember the various shapes (Rectangle, TextShape) from the UWCSE library that you will use to display this bus on the map. At minimum, you should have a plain rectangle for a background, a TextShape for the route number, and a TextShape for the vehicle ID number. You might want to save some offsets here to tell you where each of the text elements go relative to the background rectangle. Feel free to get fancier with this once you get the basic programming running correctly.

Initialize any other instance variables that you need in the other methods.

Information about the TransitBusEvent passed to the constructor is available in the doc-hw7 directory, along with information about TransitBus if you would like to use that to store information about your bus.

4c. Implement the method addTo(GWindow gw). This method is specified by the uwcse.animation.Prop interface.

If your symbol has already been added to a window (ie, the instance variable that remembers the window already has a non-null value) then just return and do nothing.

If your symbol has not yet been added to the window, then store the window reference provided to you (ie, gw), and add all the shapes of your symbol to that window. Use the GWindow method window.add(shape) to do this. Remember, the shape objects were created in your constructor, you are just telling the window about them at this point. The first shape that you add is the background rectangle, then the following shapes are drawn on top of that.

4d. Implement the method removeFromWindow(). This method is specified by the uwcse.animation.Prop interface.

If your symbol is not in a window (ie, the instance variable that remembers the window has a null value) then just return and do nothing.

If your symbol is in a window, then use the GWindow method window.remove (shape) to remove all of your shapes from the window. You still have references to these shapes in your instance variables, you are just telling the window not to bother displaying them at this time.

Set the instance variable you are using to remember the window back to null to indicate "no current window".

4e. Implement the method intersects(Shape other). This method is specified by the ClickableProp interface.

This method uses the Shape method shape.intersects(other) to decide if there is an intersection between the other Shape and our background shape, assuming that the background Shape covers all the area that the rest of this symbol

might fall onto.  Return true if the given Shape intersects us, else false.

4f.  Implement the method setVehicleLocation(TransitMap m,double lat,double lon).

This method sets the Location of the TransitBus we are tracking, based on new latitude and longitude values.  Move the symbol to that location on the map by moving all of the Shapes to their new locations.  The supplied TransitMap object has methods pixelX(lon) and pixelY(lat) that can convert longitude and latitude into the proper x,y pixel coordinates for you.  Use the Shape method shape.moveTo(x,y) to actually move each shape.

Remember to save the new lon/lat location in whatever instance variables you are using, and also move all the shapes in your symbol to the corresponding pixelX/pixelY spot on the map.  Also remember that some of the shapes in your symbol are probably offset a little from the others (for example to put the route number and vehicle number in different parts of the symbol) and so you don't necessarily want to move them all to exactly the same pixelX and pixelY values.  Some small offsets between the various elements may be required.

4g.  Implement the method doClickAction().  This method is specified by the ClickableProp interface.

This method is called whenever this symbol is clicked on.  The method prints out some information about the bus this symbol represents.  Using System.out.println, you should print out the bus vehicle id number, the route, and the lat/lon location.  Sample output is:

```
Sym: hw7.TransitBus 2427 at (47.666431, -122.317397) on route 67
Sym: hw7.TransitBus 1003 at (47.664906, -122.314218) on route 70
```

Your output should resemble this, but does not have to be exactly the same. This output was created using the toString methods of TransitBusSymbol and TransitBus.

4h.  Depending on how you implement the doClickAction() printout, you may want to implement method toString() to print information about this object, perhaps calling the toString() method of a TransitBus.

5.  Test your new class using the local data file and running the program using runBusDisplayFile.bat.  The resulting map should look like the example posted.

6.  Once the program is displaying shapes, make sure that they are in the right places on the map and that the route number and vehicle id numbers are displayed correctly.

7.  To run the program with real data, you can use runBusRcvNet.bat, followed by runBusDisplayRcv.bat.  Your map should be updated in real time with the actual bus location information.