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

** Homework 6 Programming Project **

This project is based on the King County Metro Transit bus system. As you know, Metro and the University of Washington make quite a bit of information about the buses and their locations available in real-time over the network. In this homework project, you will write a Java class that we can use to create reports about the buses and the routes they are on, using real-time data or using a simple local copy of a small set of the data.

As provided to you, the Metro program runs and collects data about the the bus system, but doesn't have any way to generate reports based on that data.

This is typical in the real world, by the way. People often get excited about collecting data from a real-time source and write an elaborate program to collect it, but then neglect to provide any way for a user to see a report of the data.

Your role in this homework project is to write a TransitReporter class that can get a snapshot of the buses that are currently defined in the TransistSystem and then print out a couple of different reports about them depending on user commands.

There are two programs involved in this project. One program is the source of the data, the other program is the Metro program that reads the data. The source program is completely written, you don't need to do anything with it except start it running.

** Source of the data **

There is a simple program called BusXmt that you can run on your machine to be a local source of bus location data. This is useful for two reasons. First, you don't need to be connected to the network in order to run. Second, there is a controlled script that governs the data that the BusXmt program sends out, and so you can check your program against known data before you turn it loose on the real Metro King County bus system.

To run BusXmt, use the batch file runBusXmtShort.bat. This will start a data server running on your machine. Once you've started it, you can leave this program running continuously. The output from this program looks like this when it first starts up:

```
>java -classpath hw6.jar bus.BusXmt bus/ShortScript.txt
Usage: BusXmt filename [requestPort]
  filename is the file that contains position update records.
  See BusRcv for information about the format.
  requestPort defaults to 8412 if not specified.
NOTE: This program runs until terminated with control-C.

BusXmt listening for connections on port 8412
BusXmt transmitting data from script file bus/ShortScript.txt
```

The other possible source of data for your program is the actual bus information server, available at sdd.its.washington.edu, port 8412. You don't

need to do anything in order to start this server running; it is up all the time. When you are ready to use it, make sure that you are connected to the Internet, and then run the Metro program with a net connection as described below.

** Destination of the data **

Metro.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 TransitSystem object and a new TransitReporter (that's you), and then tells the TransitSystem to start listening for information about the buses. After the system is started, Metro prompts the user for commands, and passes each command on to the TransitReporter for action.

Metro.java is complete as provided to you and does not need to be modified. Most of the code for the Metro program is stored in the archive file hw6.jar and does not need to be modified.

* TransitSystem.java *

This class listens for updates to bus positions and manages the resulting list of buses. Your TransitReporter can ask the TransitSystem for a list of known buses at any time. TransitSystem.class is part of the hw6.jar and does not need to be modified.

* TransitReporter.java *

TransitReporter is the primary class of interest in this assignment. A TransitReporter object interprets commands from the user and prints out the requested reports. Your job (described below) is to implement a functioning TransitReporter class.

* CompareByRoute.java *

This Comparator implements a method to compare two TransitBus objects and identify which one has the lower route number. It is provided so that you can use Collections.sort(...) to sort a List of TransitBuses by route number, rather than vehicle identification number. This class is complete as written and does not need to be modified.

++ Project Requirements ++

1. Download the cse142-hw6.zip file and unzip it. The project skeleton is in the directory hw142\hw6. Your task is to open the project and update TransitReporter.java to add the required capabilities described below.

Most of the code I am providing to you for this project is bundled up in the file hw6.jar. The documentation for the files in hw6.jar is in subdirectory hw6-doc. Double-click on the file index.html to get started. One thing you may want to look at is method getBusList() in class TransitSystem, since this is the method you call to get a current list of buses.

2. Before you run the Metro program to generate reports, you need to start the local data server program (or use the remote server and get the data from the Internet). To run the local data server, use the batch file runBusXmtShort.bat by double clicking or running it from a command window.

3. To run the Metro program and generate reports, you can use BlueJ if you like, or you can use any other program editor and compiler combination. I have provided simple batch files that compile and run this project outside of BlueJ.

In order to run the program from BlueJ, you need to tell BlueJ about the jar file, just like we did a month ago for the uwcse library 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 hw6.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, just right-click on the Metro class and select the main method. This will start the program running and it will produce some printed output, then wait for your command. If you want to connect to the remote server, you have to specify the name of the remote server when you start the program. You do this by putting it between the curly braces that BlueJ shows when you call the main method. So you would specify {"sdd.its.washington.edu"} in order to run with remote data in BlueJ.

If you run the program from the command line using runMetroLocal.bat or runMetroNet.bat, the proper arguments are specified in the batch file, and the program will print the output to the screen window.

4. All the new methods that you will write are to be added to TransitReporter.java. The javadoc comments for TransitReporter methods are given in TransitReporterComments.txt. The specific implementation requirements are as follows.

4a. Implement a constructor for the TransitReporter class. The constructor initializes the instance variables. It should keep a reference to the TransitSystem so that it can ask for a List of buses later on, and should also initialize an instance variable for the List of buses and create and remember a new CompareByRoute object for later use.

4b. Update the doCommand method so that it can do something more than just recognize the exit command. This requires that you rewrite the "if" statement so that it can recognize several different commands and call the appropriate private method for each command, as described below.

4c. Write the doHelp method. This method is called from doCommand when the user enters the "help" command. The method prints a simple list of valid commands on the console and then returns to doCommand.

4d. Write the doSnap method. This method is called from doCommand when the user enters the "snap" command to request that a new data snapshot be retrieved from the TransitSystem. This is the point at which you use the reference to the TransitSystem that was saved by the constructor in an instance variable. You call the TransitSystem method getBusList(), and it returns to you an ArrayList containing one TransitBus object for each bus currently known to be running. The doSnap method updates an instance variable with a pointer to this ArrayList of buses, prints out a brief message about the number of buses (ie, the size of the List) and then returns to doCommand.

4e. Write the doBus method. This method is called from doCommand when the user enters the "bus" command to request that a list of buses in the current snapshot be displayed.

You need to make sure that there is a valid snapshot before doing this command. If there is no snapshot, get one before doing the following printout. If there is a snapshot, don't get another one, just use what you've got.

Sort the list using the sort method in the java.util.Collections class that

sorts the specified list into ascending order, according to the natural ordering of its elements. This will sort the list according to the vehicle id number. Then go through the entire List and print out a line for each bus. You can use simple calls to System.out.println(...) for this, since the toString() method of TransitBus provides a good description of a bus. Remember that the documentation for TransitBus is included in the hw6-doc subdirectory.

After you print one line for each bus, print a summary line that tells how many buses are in the system.

4f. Write the doRoute method. This method is called from doCommand when the user enters the "route" command to request that a list of all the known routes be displayed.

As with the bus command, you need to make sure that there is a valid snapshot before doing this command. If there is no snapshot, get one before doing the following printout. If there is a snapshot, don't get another one, just use what you've got.

Sort the list using the sort method in the java.util.Collections class that sorts the specified list according to the order induced by the specified comparator. Use the comparator object that you created in the constructor for this purpose. This will sort the list according to the route number.

Then go through the entire List and print out a line for each route. You need to be careful how you do this, and you need a set of nested while loops to implement it. The outer loop executes once per route. The inner loop counts up all the buses on a particular route. One implementation would use the number of buses already processed as the control variable for both the inner and outer loops, and would increment the number every time we counted a bus as being on a route.

After you print one line for each route, print a summary line that tells how many routes are active in the system.

4g. Write the doSummary method. Print a line telling how many buses and how many routes. This method needs the same information as was generated in doRoute, so you may want to design a separate little helper method that can count the routes and optionally print a line for each one. This helper could be called from doRoute or doSummary.

5. Test your new methods after you get each one written. You should use the local data server (see step 2) during checkout so that you there are only a few buses being reported and you can check your code easily. Remember that you can leave the local data server running the whole time you are working. The Metro program connects to the server when it starts, and then disconnects when it exits. Don't start multiple copies of the data server; you don't need them.

A sample run using the runBusXmtShort.bat data server looks like this:

```
>java -classpath .;hw6.jar Metro localhost 8412
Program arguments: hostname portnumber
hostname : localhost
portnumber : 8412

Type help for information about running this program.
```

```
? snap
Snapshot captured data for 6 buses.
? help

This program accepts data from a source of Transit Bus position data and
makes it available for analysis by the user of the program.  Use the snap
command to take a snapshot of the current state of the bus system, then
use the other commands to print information based on that snapshot.

The following commands are available:
bus         list all the buses in the most recent snapshot
exit        shut down the program.
help        print this message
quit        same as exit
route       list all the routes in the most recent snapshot
snap        take a new snapshot of the buses in the system
stop        same as exit
summary     print a brief summary of the most recent snapshot
? bus
hw6.TransitBus 1142 at (0.0, 0.0) on route 74
hw6.TransitBus 2337 at (0.0, 0.0) on route 71
hw6.TransitBus 2364 at (0.0, 0.0) on route 0
hw6.TransitBus 4012 at (47.508509, -122.253559) on route 7
hw6.TransitBus 4014 at (47.601486, -122.329767) on route 7
hw6.TransitBus 5101 at (47.53381, -122.269995) on route 7
6 buses counted
? route
Route 0 has 1 buses assigned.
Route 7 has 3 buses assigned.
Route 71 has 1 buses assigned.
Route 74 has 1 buses assigned.
4 routes counted.
? summary
6 buses on 4 routes.
? exit
```

---

6. After you get your program running at least partially, you might want to run it using real data from the real Metro system. To do that, you specify the name of the server host in the arguments to the program. The name of the host is "sdd.its.washington.edu". You can specify that in the argument list for main in BlueJ, or you can just double click on the batch file runMetroNet.bat and it will specify it for you. When you are connected to the real data server over the Internet, you can do repeated snap commands and see how the number of buses reported grows until you have a list of all the Metro buses running at the time. There are several hundred buses at most times of the day.

A sample run using runMetroNet.bat to connect to the real data server looks like this:

```
>java -classpath .;hw6.jar Metro sdd.its.washington.edu 8412
Program arguments: hostname portnumber
hostname : sdd.its.washington.edu
portnumber : 8412

Type help for information about running this program.
```

```
? snap
Snapshot captured data for 233 buses.
? summary
233 buses on 80 routes.
? exit

--------------
```