CS310: Advanced Data Structures and Algorithms, PA0

Suggested steps, more discussion

Adapted from Prof. Betty O'Neil

First get organized in your filesystem for this class. We are not using a VM, so the files for this class competes for attention with all your other computer activities. Decide on one top-level directory for all the projects, say `c:/cs310` on Windows or `~/cs310` on Mac or Linux, but it's up to you. Then make a `src` directory inside this top-level directory for this project, and a `pa0` subdirectory inside `src`.

Decide what IDE, if any, you want to use for this class. I prefer eclipse for Java, but also use Netbeans for other languages and could use it for Java. These are both free and in common use by working programmers. IDEs know the syntax of the language, and what methods each class has, for easier programming. They automatically compile as you edit, for immediate feedback on syntax problems. You can also use a simple text editor and compile and run through the command line.

Make sure you understand all the terminology that goes with a Java class. First look at page 9 in the text: all those red notations on parts of this basic program. This program does not create an object, however. Then look at page 85 for the case of a Java source for an object class, again with red annotations showing terminology.

LineUsage: a program using JDK Collection classes

Suggested steps: a bottom-up approach, so we can execute something for each step. All the java files for this part will be in the src directory, and in the `pa0` package, so they need a *package statement* at the top. The JDK classes are all available by default, so no .jar file is needed in build commands.

1. Design and implement the `Usage` class. This has an integer count and a String username in it as instance variables, a constructor taking both count and username, and getters for the two instance variables (setters are not needed). You can add an increment method if you want. See page 84 for class terminology and make sure you know all these concepts. For getters and setters, see dzone.com article.

2. Test `Usage.java` as follows. In main of `Usage.java`, create a `Usage` object, add one to its count, and then print out its username and count using the getters. Putting test code in main is convenient, but is not as hands-off as test code in another class, because main is part of the class code, giving it rights to access private members. It is often done just temporarily during development, but leave it in this case. Note that test code in main is not meant to be part of the API of the class and can't be put in an interface for the class because main is a static method.

3. Now start on `LineUsage`. It is supposed to use a Map of `String` to `Integer` to hold the counts so far for each user seen on a particular line. Thus if OPERATOR was seen on the line 34 times and USERMGR seen 12 time, the map would hold "OPERATOR" $\rightarrow$ 34 and "USERMGR" $\rightarrow$ 12. You could find the 34 by calling map's get method, `get("OPERATOR")`. Use the Map type for the type of the instance variable, but of course use you need to use the concrete type `HashMap` or `TreeMap` for creating the container object.

4. Study the supplied program `FrequencyCounter.java` (from HW1) to see the details of repeatedly adding one to the count for a certain string. Here the map itself is held in an instance variable of `LineUsage`, private of course, not a local variable of method main as you see in `FrequencyCounter.java`. Thus each `LineUsage` object has its own map for its own data, separate from the data for a different line number. The constructor of `LineUsage` creates an instance of the map for the value of this instance variable. `LineUsage` should have methods `addObservation(String username)` and `Usage`

`findMaxUsage()`, which returns the `Usage` object for the user with the highest count. Note that `Usage` is not used in the map, only in the delivery of results once the Map is full of data.

5. Try out your `LineUsage` object: Write a test program `TestLineUsage.java` that, in main, creates one such object, and adds an observation of OPERATOR to it, then an observation of USERMGR, then a second observation of OPERATOR, then finds the most frequent from the container by calling `findMaxUsage` and prints the results out. Note that putting the test code in a different source file provides a better test than putting test code in main, and clearly removes it from API considerations.

6. Write the main program with the big array of `LineUsage` objects in `LineReport.java`. This code , working just one of its elements (itself a `LineUsage` object) as appropriate for each line of input. For a small example of an array of objects, see example, but add "private" to the Employee instance variables in that example. To parse the input file, use Scanner. With line-oriented input like this, it's best to pull each line into a String (using `hasNextLine` and `nextLine`), and then process the line, now in a `String`. That `String` for one line can be processed by `String.split` or another `Scanner` for this `String`. Put this code that reads the file into the array in its own method: this can be a static method or an object method. Once the input line is parsed, you have a terminal line number and username. Find the right `LineUsage` object in the big array and add the username observation to it.

7. Continuing in `LineReport`, write a method that outputs the data as specified.