CSC171 — Lab 6

Collections

The goal of this assignment is to give you experience working with the Java Collection Framework and more familiarity with basic concepts of data structures. You will implement two different classes in one source file named `Transform.java`, which you should submit to blackboard before the deadline. Example inputs with their correct outputs are available as a zipfile on blackboard.

This assignment is due Sunday November 20th 1159pm. Late submissions will be accepted with a 5 point per day penalty. This is a solo assignment - keep collaboration at discussion level. It will be good practice; everyone should do it.)

Program Requirements

1. Create a non-public (and non-private) top-level class named `Entry` with instance variables for the last name, first name, course number, and semester values. You may choose whether to make your instance variables private, public, or default.
   (a) You should declare a `public Entry(String line)` method which can construct the entry directly from a given line of input. This method can split the string by the separator and store the resulting fields in their respective instance variables. Hint: be sure to check out the `String.split()` method for parsing the entry.
   (b) This class must implement the `Comparable` interface by overloading the `public int compareTo(Entry other)` method.
   You must define a `compareTo` function that sorts the entries into ascending order by semester, then course, then last name, then first name. This means `a.compareTo(b)` should return a negative number when a should appear earlier than b.
   (c) Overload the `toString()` method in order to print your processed entries according to the required format. Note that the format matches the order of the sort keys.
2. Create a public top-level class named `Transform` (and put it and your `Entry` class from above into a source file named `Transform.java`). This class should contain your main method. Write your main method to create a Scanner and read the data from the user until the first blank, then read the filter course, then process the data.
   Depending on your choice of data structure, you may or may not need to use `Collections.sort()` to sort by the required key sequence. If you create a `List`, you will have to sort it. If you create a `TreeSet`, it will sort automatically when you add to it.
   Finally, you should iterate through the sorted elements of the collection and display those which pass the filter to standard out.

Example Transformation

Program Input

The data input format will consist of comma-separated tokens: lastName, firstName, semester, course. There will be one entry per line. There are no spaces between the values and the commas. A blank line will be entered to signify the end of the database. A final token will be issued to indicate the query (in this case, the query is 171.)

Here is the example data:

```
Mcclaine,Kendria,F16,171
Windsor,Christalyn,S21,257
Widman-Pinheiro,Chantay,S20,171
Widman-Pinheiro,Chantay,F18,257
Hewlett-Clawson,Mahyar,F18,171
Hewlett-Clawson,Mahyar,F21,257
Hawkins,Desmund,F19,257
Hawkins,Desmund,F21,246
Yingling,Oswald,S18,257
Faulk,Joon,F16,257
Faulk,Joon,F16,246
Faulk,Joon,F16,171
Millan,Shekinah,F17,257
Millan,Shekinah,S21,246
Stallcup,Andriana,S16,257
Rodman,Jania,F16,246
Rodman,Jania,S20,171
Rodman,Jania,F20,257

171
```

Program Output

```
F16,171,Faulk,Joon
F16,171,Mcclaine,Kendria
F18,171,Hewlett-Clawson,Mahyar
S20,171,Rodman,Jania
S20,171,Widman-Pinheiro,Chantay
```

Grading and Other Notes

We will use the following approximate rubric:

- Entry class
  - instance variables set by constructor – 10pts
  - overloaded toString for printing columns – 10pts
  - columns printed in correct order – 10 pts
  - implements Comparable interface – 10 pts
  - compareTo sorts semester correctly – 10 pts
  - compareTo sorts courses correctly – 10 pts
  - compareTo sorts names correctly – 10 pts
  - filtering works correctly – 10 pts
- Transform class
  - main method control flow – 10 pts
  - main method input and output – 10 pts

All test cases for this assignment are based on randomly generated names and enrollments. (The instructor found a list of common surnames and a list of common first names on the internet, and merged them.)