Programming Project 2 - Due: Sunday, October 2 at 11:59 PM

You are a programming intern at the alumni relations office. Having heard you are taking CS 20B, your boss has come to ask for your help with a task.

Quite often she needs to process files containing records for students attending UC schools and find those who are SMC alumni. She needs you to write a program that takes in a file containing UC student records and another file containing SMC alumni records and writes out to a file all the SMC alumni who are attending UC schools, ordered by ID.

This sounds like a very simple task and you agree to work on it. She warns you that the files are pretty large - a few hundred thousand student records. You inform her that with today's computers having processors of a few GHz this task is a piece of cake and can be computed in an instant.

Part 1:

You think about the problem for a minute and you come up with a plan.

- You will create a Student class representing student records. It will have three data members corresponding to the data contained in the student records file

```
int id;     String name;      String school;
```

- **You will have no setters or getters in your Student class and the data members will be private.**
- You will override the equals method to consider two student objects as equal if they have the same ID.
- You will implement the comparable interface to allow sorting of student objects by ID.
- You will override the toString method to print the fields as they appear in the student records file.

The idea is that for every record you read from the file, you will create a Student object. You will store the UC students into a student array and the SMC alumni into another student array. Parts 2 and 3 operate on these arrays.

Part 2:

Now that you have two arrays with one containing UC students and the other containing SMC alumni, you plan to process the arrays and find the students who belong to both arrays. Your data structures are these two arrays. Your algorithm will be: for each student in the UC students array you try to find it in the SMC students array by doing a linear search on the SMC students array (comparing for equality). If you do find it, that UC student is then stored in an array containing students who are SMC alumni attending UC schools. Once you have the array of the common students, it is sorted by ID and printed to a file. We will use the Java library sort for this (already done for you in main()).

Part 3:

After running your program from Part 2 and discovering that it takes ~ 5 - 10 minutes to complete for the given input (much longer for larger files), you decide to really think about the problem in order to find a solution that does not force your boss to go get a cup of coffee every time she needs to run your program. It seems even those powerful processors in today's computers can't really save a bad algorithm.

You remember from class that binary search is much more efficient than linear search for finding an element in an array and decide to try it out. We will use the Java library binary search for this. Your algorithm will be: for each student in the UC array you will binary search the sorted SMC array. If the binary search finds the student, that UC student is stored in an array containing students who are SMC alumni attending UC schools. Once you have the array of the common students, it is sorted by ID and printed to a file. We will use the Java library sort for this (already done for you in main()).

Hints:

- It is always a good idea to test each little piece after you complete it. Trying to debug a whole program is hard and very time consuming.

- First implement the Student class. Do not do anything else until you are sure your Student class is 100% functional. Create student objects in a toy main() and call toString on them, exercise the equals and compareTo methods and verify that your class is implemented correctly.

- Whenever you work with a large data set it is much easier to first work on a small data set to ensure your program functions correctly. Once you are confident that it produces the correct result you can move to the larger data sets. To help get you started, two small sample input files and the expected output have been provided. Do not move on to the large files until you are very sure your program is correct.

Deliverables:

Since this is our first real project, the design and most of the work has been done for you. **You should implement the Student class and FILL OUT THE MISSING CODE in the provided program. DO NOT CHANGE THE REST OF THE GIVEN CODE.**

You should submit a zip file named project2_first_last.zip (where first and last are your first and last name) containing **ONLY the files below.**

Student.java
Project2.java
report.txt : a text (not Word, Power Point, ...) file containing ONLY:
a) A 1 to 5 lines paragraph from you saying "I have tested this program and there are no known issues." if you believe that to be the case, or a brief description of known issues in case your program has known problems or you could not fully implement it.
b) The timing printouts and the first and last 3 students in the file that the program generated (6 students in total, copied and pasted here).

How you get points:

- Student.java

the class	10 points
overriding the equals method	10 points
overriding the toString method	5 points
implementing comparable	10 points

- Project2.java

readStudentsFromFile	10 points
writeStudentsToFile	5 points
findCommonStudents1	25 points
findCommonStudents2	25 points

How you lose points:

- You do not follow the given directions and decide to make changes "for fun". Specifically, **do not change the skeleton code given to you**. Just fill out the missing parts.
- You use ArrayList or any data structure we have not yet seen in this class. **Only arrays are allowed.**
- You have getters or setters in your student class. **Only provide the methods that were asked.**
- If any of your code prints anything at all on the console except for the required output. **Remove all your print outs, debug statements, etc. Clean up your code and do not leave clutter**

behind.

- If your code has no comments where needed. **Comment your code appropriately. Brief and to the point.**
- If you submit your whole workspace or executable files. **Submit only the files the project asks for.**