FLORIDA SOUTHERN COLLEGE

Analysis of Algorithms
CSC 3380

Fall 2018

Program 1: *FSC Grade Book*
Assigned: Monday, August 27th, 2018
Due: Friday, September 7, 2018 by 11:55 PM

Purpose:

1. The first purpose of this program is for you to REVIEW your JAVA (CSC 2281), including arrays, classes and objects, and arrays of objects.
2. The second purpose of this program is for you to review file I/O (input/output).
3. Finally, you are to implement binary search in the program.

Read Carefully:

- This program is worth 5% of your final grade.

- **WARNING**: This is an individual assignment; you must solve it by yourself. Please review the definition of cheating in the syllabus, the FSC Honor Code, and the serious consequences of those found cheating.
    - **The FSC Honor Code pledge MUST be written as a comment at the top of your program**.

- The deadline for this project is by **Friday, September 7, 2018 by 11:55 PM.**
    - **Note:** once the clock becomes 11:55 PM, the submission will be closed! Therefore, in reality, you must submit by 11:54 and 59 seconds.

- LATE SUBMISSION: you are allowed to make a late submission according to the rules defined in the syllabus. Please see course syllabus for more information.

- **Portal Submission**:
    - This assignment must be submitted online via Portal.
    - If your file is empty or you upload wrong the file, it will be solely your responsibility, and your grade will be **zero**.
    - Your java files should all be zipped up. Your zip file should be named as follows:
        - `CSC3380_YourFSCIDnumber_Program1.zip`
        - Example: `CSC3380_2351345_Program1.zip`

Program 1: FSC Book

Objective

Practice all concepts from CMS 2280 & CSC 2281, with a focus on making classes and creating objects from these classes. Specifically, you will practice making an array of objects and manipulating/using the data inside the array. Finally, and VERY important is that this program requires you to read from a file and write to a file (practice with File I/O). This is very important because all programs during the semester will use File I/O.

Program Description

For this program, you will implement a basic grade book. You will use File I/O to read input from a file and then print the output to a file. To be clear, you will read COMMANDS from an input file. Example commands are ADDRECORD, SEARCHBYID, etc. Then, depending on the command, you will either add a student record, search for a student, display results, etc. But instead of printing to the console window (screen), you will print to an output file. **Sample input and output files have been provided for you on the website.*

For this program, you will create a Class called `Student`. This class, `Student`, will be used to create objects of type `Student`. Each `Student` object will store all the needed information for one student (ID, first name, last name, exam grades, final grade, etc). Then, in the main program, you will create an array of `Student` objects, and you will add student objects to this array as needed (whenever you see the ADDRECORD command).

The first three lines of the input file are as follows:
Line 1: the course number
Line 2: the name of the instructor
Line 3: the maximum possible number of students allowed in the course

You must read these values into appropriate variables.

Once you read the maximum number of students allowed, you will then make an array of student object ***references***, and you will use this input (max possible number of students in the course) to make the correct size of the array.

As an example, if you want your array variable to be called `students`, you can declare and create your array of `Student` object references as follows:

```
Student[] students = new Student[maxNumStudents];
```

- **`maxNumStudents`** is the value from line 3 of the input file

What does this line do? It creates an array of **references**. Note: each reference has a default value of **`null`**. Until now, we have NOT created any objects. Student objects are only created when we see the command ADDRECORD. At that time, a new Student object will be created and the reference for that object will be saved in the appropriate location of the array.

The Commands to be implemented are as follows:

1. **ADDRECORD**
   This command will be followed by the following information in the input file: student ID (a non-negative integer), first name, last name, and then three exam grades, each being a non-negative integer. When you process this command, you should make a new student object, and then you should scan, from the file, the student ID, the first and last name of the student, and the three exam grades (see output). All of this information should be stored inside the newly created student object. Finally, you must save the reference of this object inside the appropriate index of the student array.

   ***Note**: this array **must** stay in sorted order based on the ID of the student. So the student with the smallest ID value will be at index 0, the student with the next smallest at index 1, and so on. Therefore, when you save the Student object reference into the array, you must first find the correct sorted index. After you find the correct insertion location, if there is no object at that location, then you can easily save the object reference at this index. BUT, IF there is an object at the index that you find, you must SHIFT that object, **AND** all objects to the right of it, one space to the right. This is how you make a new space in the array for the new object.

   Example: if your array already has 6 student object references, from index 0 to index 5. And now you want to add a new student object reference at index 4 (based on the sorted location). Before you can add the new student object reference at index 4, you must SHIFT the student object references at index 4 and index 5 to index 5 and index 6. Basically you need to move them one position over to the right. Now, you have an empty spot at index 4 for the new student object reference.

   So, once the first student is added, that student's information (ID, name, exam grades, etc) can be found inside the `Student` object at index 0 of the `Student` object array (because it is the first object in the array). Then, as you add additional objects to the array, they will go at the appropriate sorted index based on the student ID.

   Finally, the final grade and letter grade of the student will be calculated and saved into the Student object, and a message is printed to the file. The final grade is calculated as follows: Exam 1 is worth 30%, Exam 2 is worth 30%, and the Final Exam is worth 40%.

   Example:
   If the following line was in the input file:
   ```
   ADDRECORD 111 Joe Smith 90 85 94
   ```

   This would be the output:
   ```
   Command: ADDRECORD
   Joe Smith (ID# 111) has been added to the FSC Grade Book.
      Final Grade: 90.10 (A).
   ```

   **So the final grade of 90.10 was calculated by: 90*(0.3) + 80*(0.3) + 100*(0.4)

2. **SEARCHBYID**
   This command will be followed by a student ID, a non-negative integer. You must use a **binary search** to search for the student using his ID. If the record is found, it is printed to the file (see sample output file). If it is not found, an error message should be printed instead (see sample output file). You should notice, in the output file, that you are required to print the indices searched, which will demonstrate that you did correctly use binary search.

3. **SEARCHBYNAME**
   This command will be followed by a student first and last name. If the record is found, it is printed to the file (see sample output file). If it is not found, an error message should be printed instead (see sample output file). This should be implemented using linear search.

4. **DISPLAYSTATS**
   This command will not have any other information after it (in the input file). When you scan this command, the statistical results of the class should be printed to the file (see sample output file).

5. **DISPLAYSTUDENTS**
   This command will not have any other information after it (in the input file). When you scan this command, all students, and their details, should print to the output file. (see sample output file).

6. **QUIT**
   This command will not have any other information after it (in the input file). When you scan this command, the program should print a message to the file and then quit.

Input File Guarantee/Promise

You are guaranteed that no other commands will be in the input file. You are also guaranteed that the input will be correct, which means you do not have to worry about mistakes in the input. Finally, you are guaranteed that each ADDRECORD command will be independent of all other commands (there will not be duplicate ADDRECORD commands in the input).

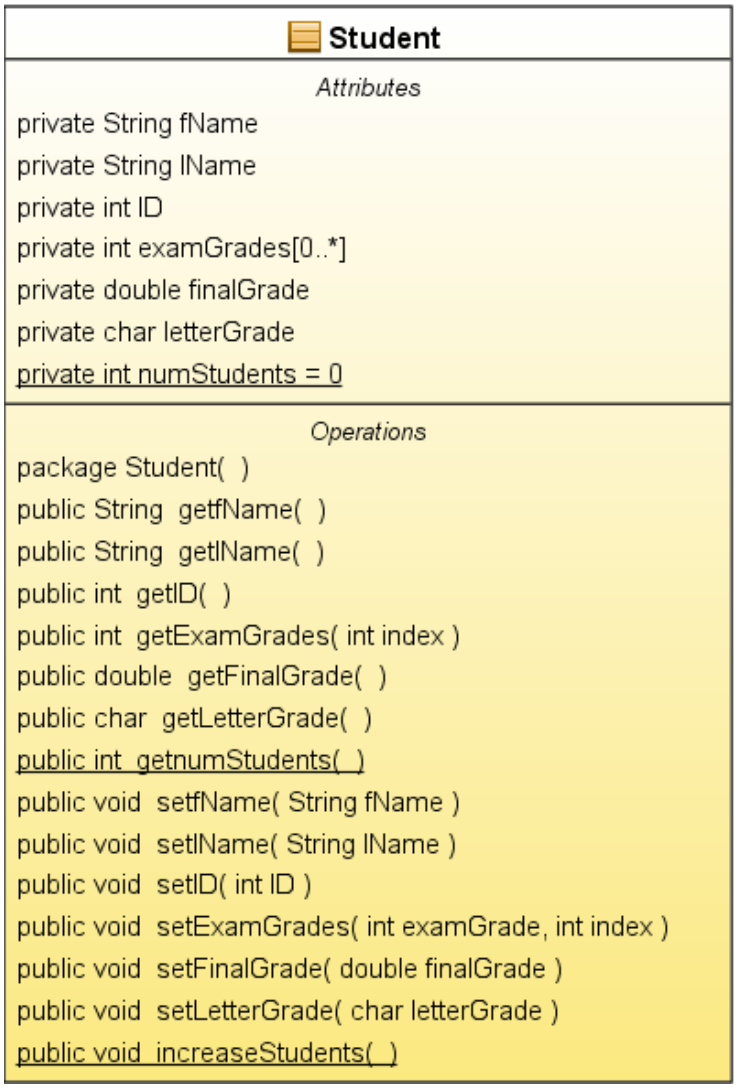
Array Required in Your Program

Your program should have ONLY ONE array to store all the student objects. This array MUST BE an array of Student objects. You will save all information, for each student, in one Student object.

> * *Of course, inside the Student object, you have a small array (length 3) to store the three exam grades.*

Description of the two Java Files Needed

1. Here is the UML diagram for the Student Class (**`Student.java`**)



*All members of the Student class must be private. As a result, you must use accessor and mutator methods (get and set) methods to access and change/set the values.

** Pay close attention to the getExamGrades() and the setExamGrades() methods. For the getExamGrade() method, you send one parameter, the index of the exam you want to get. For the setExamGrades() method, you send two parameters: the grade of the exam and the index where you want to set it.

***The member `numStudents` should be static (this is why it is underlined in the diagram). This variable is used to store the current number of students (records) that have been added to the `students` array. Also the methods `getnumStudents()` and `increaseStudents()` should be static.

2. The other Java file is your main program (**`FSCgradebook.java`**)

   The following methods must be used in your program file:

   1. **public static int addRecord(***parameters here***)**
      Inside this method, you must:
      1. Scan from file all required information.
      2. Save all information into a new `Student` object.
      3. Calculate (and save) the final grade of the student.
      4. Determine (and save) the final letter grade of the student.
      5. Insert the new `Student` object into the correctly sorted index of the array of students.
      6. Return the index of the new insertion position (the position that you just inserted the student at. You will n

   2. **public static void searchStudentsByID (***parameters here***)**
      Inside this method, you must:
      1. Scan from file the ID of the student to search for.
      2. Search for this student ID using a **binary search**.
      3. If the student ID is not found, an error message should print (see output)
      4. If the student ID is found, the student's record should print (see output)

   3. **public static void searchStudentsByName(***parameters here***)**
      Inside this method, you must:
      1. Scan from file the first and last name of the student to search for.
      2. Search for this student using first and last name (linear search)
      3. If the student is not found, an error message should print (see output)
      4. If the student is found, the student's record should print (see output)

   4. **public static void displayClassStatistics(***parameters here***)**
      Inside this method, you must:
      1. Display the statistical results for the class (see output)

   5. **public static void displayAllStudents (***parameters here***)**
      Inside this method, you must:
      1. Display the record of all students, along with their grades.
      2. If there is no student, an error message should print (see output)
      3. Else, all students should be printed.

   6. **public static char getLetterGrade(***parameters here***)**
      Inside this method, you must:
      1. Determine the letter grade of the student from their final, numerical grade. *You will use the standard scale: 90 or greater is an A, 80 or greater is a B, 70 or greater is a C, 60 or greater is a D, and under 60 is an F.
      2. Return the letter grade.

   NOTE: You can have other methods as you feel are needed and appropriate. Additionally, you can change the return values of the aforementioned methods if needed.

Input File & Output File

You must read from an input file called **`FSCgradebook.in`**. The first line will have the course number (example: CSC 3380). The second line of the input will have two strings, which represent the first and last name of the instructor. The third line will have a non-negative integer, which represents the maximum possible number of students that can be added (you will use this number to create your array of student object references. These three lines will be followed by k lines, with each line having a command and information after the commands (as described above). You must print to the output file, **`FSCgradebook.out`**.

Sample Input & Output File

We have provided you a sample input with correct output. See the website to download files.

*****WARNING*****

Your program MUST adhere to the EXACT format shown in the sample output file (spacing capitalization, use of dollar signs, periods, punctuation, etc). The graders will use very large input files, resulting in very large output files. As such, the graders will use text comparison programs to compare your output to the correct output. If, for example, you have two spaces between in the output when there should be only one space, this will show up as an error even through you may have the program correct. You WILL get points off if this is the case, which is why this is being explained in detail. Minimum deduction will be 10% of the grade, as the graders will be forced to go to text editing of your program in order to give you an accurate grade. Again, your output MUST ADHERE EXACTLY to the sample output.

Grading Details

Your program will be graded upon the following criteria:

1) Adhering to the implementation specifications listed on this write-up.
2) Your algorithmic design.
3) Correctness.
4) **Use of objects and classes. Use of an array of Student objects. If you do not use objects, you will receive a zero. Additionally, you must use binary search.**
5) The frequency and utility of the comments in the code, as well as the use of white space for easy readability. (We're not kidding here. If your code is poorly commented and spaced and works perfectly, you could earn as low as 80-85% on it.)
6) Compatibility to the **newest version** of NetBeans. (If your program does not compile in NetBeans, you will get a large deduction from your grade.)
7) Your program should include a header comment with the following information: your name, **email**, course number, section number, assignment title, and date.
8) Your output MUST adhere to the EXACT output format shown in the sample output file.

Deliverables

You should submit a zip file with TWO files inside:

1. `FSCgradebook.java`
2. `Student.java`

NOTE: your name, ID, section number AND EMAIL should be included as comments in all files!

*****Hint*****

Getting and setting the exam grades may be tricky at first. So here is a bit of code from the addRecord() method:

```
for (int i = 0; i < 3; i++) {
    tempStudent.setExamGrades(input.nextInt(), i);
}
```

So as you can see, we are saving the three exam grades into the `examGrades` array. How are we doing this? We invoke the `setExamGrades()` method with two parameters:

1. The first parameter is the grade received from the input file
2. The second parameter is a counter variable, `i`. So when '`i`' is zero, this means that the inputted exam grade will get "set" at index zero of the `examGrades` array. When '`i`' is 1, the inputted exam grade will get "set" at index 1…and so on.

Suggestions:

- Read AND fully understand this document BEFORE starting the program!
- Next, start by making the Student class to create objects of type student.
- Now make your main program file:
  - `FSCgradebook.java`
- This is your main program that will read the commands from the file, process the commands, and print to the output file.
- Use the `IOexample.java` program to help you create your main to read from a file and write to a file.
- Finally, one by one, implement the commands (ADDRECORD, SEARCHBYID, etc).

Hope this helps.

Final suggestion: START EARLY!