Lab 6 Files

Purpose

Purpose is to practice using file input and output, and array list of objects. Also, this document tells you only what to do, you now have more responsibility to design how to do it.

Problem description

You are given a text file called '`Students.txt`' that contains information on many students. Your program reads the file, creating many `Student` objects, all of which will be stored into an array list of `Student` objects, in the `Students` class. The `Tester` class controls everything, calling many `Students` class methods to produce lots of different outputs. The program must write the output to an output file and to the Terminal Window.

File and class specifications

`Students.txt` file format

Information for each student is stored in the file as 3 lines of text:

name
age
GPA

e.g. the following shows data for two students:

```
Name0
22
1.2
Name1
22
2.71
```

`Student` class

The `Student` class has instance variables and methods to represent one single student.

Your `Student` class must have exactly and only the following instance variables:

```
private String name;
private int age;
private double gpa;
```

Design appropriate constructors and other methods for your `Student` class, including:

`+ toString()` – returns a `String` containing the 3 instance variables e.g.

```
Name0       22    1.2
```

`Students` class

Very importantly, the `Students` class is used to store and process many `Student` objects. It will have an instance variable to store many `Student` objects. Methods intended to process many `Student` objects belong in this `Students` class.

Your `Students` class must have exactly and only the following instance variable:

```
private ArrayList<Student> students;
```

`students` here is an array list of `Student` objects, in which all of the `Student` objects are stored.

`Students` must have appropriate constructors and methods, including the following:

`+ readFile()` – opens the data file, reads the data, creates `Student` objects, and adds them to the `students` array list

`+ toString()` – returns a `String` containing a line of information for each `Student` in the `students` array list. Will call `Student`'s `toString()` to do this. For example:

```
Name0       22    1.2
Name1       22    2.71
```

Many other methods for processing a `Students` object. Most of the code you write will be in this class.

Reading the data file

Your program will use the `Scanner` class to read from the data file, as demonstrated during the Week 13. Files lecture.

Writing the output file

Your program must use the `PrintWriter` class to save all its output to the `output.txt` file, as demonstrated during the Week 13. Files lecture. It will also send the same output to the BlueJ Terminal Window, as usual.

`Tester` class

The `Tester` class controls everything. `Tester` does not have any instance variables. You have to write the `Tester` class.

`Tester` contains only a `main()` method, which first creates a single `Students` object (and a `PrintWriter` object). The `Students` object then calls a separate `Students` method to do each of the 6 different tasks below. You must design appropriate parameters and return values, in particular so that all program output to Terminal Window and `output.txt` is done from `main()`. In pseudocode:

```
+ main()
```
create an empty `Students` object
create a new `PrintWriter` object to create the '`output.txt`' output file

read data file into `Students`
print all `Student` objects from `Students`
print the `Student` with the best GPA
calculate and print the average GPA
print the youngest `Student` who has a GPA below average
print just the names of all the students

Hints

- the `Students.txt` data file is available at Blackboard, Course Documents, Week 14 folder, Example programs. You must copy it into your Lab 6 BlueJ project folder

- you will need to add `throws IOException` to your `Tester` class `main()` method header and the `Students` class `readFile()` method header. For example, in `Tester`:

  ```
  public static void main(String args[]) throws IOException
  ```

  This prevents a file handling syntax error: "unreported exception `java.io.IOException`; must be caught or declared to be thrown"

- (you may want to review array lists from Blackboard, Course documents, Week 7 Arrays and array lists)

- it is important in this lab always to keep in mind that each element in the `Students` class array list is itself an entire `Student` object...

- it is essential that you draw pictures of the objects involved in your program, so that you are always aware of the data type you are working with

- the 6 different actions above will each be implemented as a separate public `Students` method, called from `main()` by the `Students` object, using appropriate parameters and return types

- (by the way, there is no inheritance in this lab)

Syntax for processing an array list of objects

- the syntax for processing an array list of `Student` objects is exactly as you would expect. For example, here's a method that prints all the `Students` with GPAs above the average

  – first, the method call in `main()`. Since we must do all printing from `main()`, we design `aboveAverage()` to return a new `Students` object:

```
System.out.printf("\nGPAs above the average of %.2f\n", gpa);
ot.printf("\nGPAs above the average of %.2f\n", gpa);
System.out.println(students.aboveAverage(gpa).toString());
ot.println(students.aboveAverage(gpa).toString());
```

  – since `aboveAverage()` deals with many students, it would be part of the `Students` class. It creates a new `Students` object in which to return many `Student` objects:

```
public Students aboveAverage(double avgGPA)
{
    Students aboveAverage = new Students();

    for (int i = 0; i < students.size(); ++i) {
        if (students.get(i).getGPA() > avgGPA)
        aboveAverage.add(students.get(i));
    }
    return aboveAverage;
}
```

syntax to call a method on a `Student` object in the `students` array list

  – see that it calls a method `getGPA()`, that must directly return the GPA of a student. So `getGPA()` would be part of the `Student` class:

```
public double getGPA()
{
    return gpa;
}
```

  – it also calls a method `add()`, that takes a `Student` object and adds it to a `Students` object. So `add()` would be part of the `Students` class:

```
public void add(Student s)
{
    students.add(s);
}
```

Designing method return types

- the natural unit of an object-oriented program is an object. So methods returning results tend to return entire objects. Some hypothetical examples to illustrate this:
  - e.g. `bestStudent()` returns a single student, so would return a `Student` object. Method header would be something like:

    ```
    public Student bestStudent(~~~~~~)
    ```

  - e.g. `studentsAboveAverage()` returns many students, so would return a `Students` object. Method header would be something like:

    ```
    public Students studentsAboveAverage(~~~~~~)
    ```

  - (of course methods can also return non-object data types if that is appropriate) e.g. `averageGPA()` returns the average GPA, so method header would be something like:

    ```
    public double averageGPA()
    ```

Required

- your program must work for a file containing any number of students
- you are required this time to use `PrintWriter` to create your `output.txt` output file. Cannot just save the Terminal Window output as usual
- your program must clearly label each part of the output e.g. "Student with best GPA is:", "Average GPA is: ", "Youngest student below average GPA is:", etc
- use good programming practices regarding encapsulation of class instance variables i.e. all must be declared `private` as shown above
- every method must have a clear, meaningful Javadoc comment
- automatically and routinely use all the other components of simplicity and clarity, as listed in Blackboard, Course Information, “How labs are graded”

Lab 6 submission

- deadline for this lab is 3 weeks, by end of Sunday 5/15

- zip your BlueJ project plus `output.txt` output file and email to me at awsmith@palomar.edu

  – you will lose points if you do not include a file named `output.txt` containing the output of your program

- your email Subject line must say 'CSCI 114 Lab 6' followed by your full name, so that it filters to the correct email folder for grading

  – you will lose points if you format your email Subject incorrectly

  – e.g. my email Subject would be:

  ```
  CSCI 114 Lab 6 Anthony W. Smith
  ```

- this is a graded lab, so a reminder that you may not copy code from other people

- reminder that late labs will be penalized 2 points per week or part of week late