CSE 114 Fall 2023: Assignment 8

Due: November 14, 2023 at 11:59 PM [KST]

Read This Right Away

Directions

- At the top of every file you submit, include the following information in a comment
    - Your first and last name
    - Your Stony Brook email address
- Please read carefully and follow the directions exactly for each problem.
- Your files, Java classes, and Java methods must be named as stated in the directions (including capitalization). Work that does not meet the specifications will not be graded.
- Your source code is expected to compile and run without errors. Source code that does not compile will not be graded.
    - You can get partial credit if your program is incomplete but does compile and run.
- You should create your program using a text editor (e.g. emacs, vim).
- You must use the command-line interface to run your programs. That is, you must use the `javac` and `java` commands to do this assignment. Do not use IntelliJ IDEA yet.
- Your programs should be formatted in a way that is readable. In other words, indent appropriately, use informative names for variables, etc. If you are uncertain about what a readable style is, see the examples from class and textbook as a starting point for a reasonable coding style.

What Java Features to Use

For this assignment, you are *not* allowed to use more advanced features in Java than what we have studied in class so far.

Problem 1 (10 Points)

Write an application to read a file with names and bowling scores and write a file with the averages of each players series of games.

You will need a class to compute the averages (**BowlingAverage**) and one to hold bowler information (**Bowler**). The Bowler class will hold a String with the name of the bowler and an array of integers holding the scores of that player's games. The array must be sized correctly for each bowler. Bowlers each may have a different number of games played! Bowler will also have a constructor taking 2 arguments. Finally, it will provide a getter method that returns the average of the scores as a double. However, do not compute sand store the average in the object! Only compute it from the scores array when `getAverage()` is called on the object.

The **BowlingAverage** class should have a `main()`. The main does the following:

1. Prompts the user on the console for an input file name.
2. Opens the file and wraps it in a Scanner object (so we can use next() and `nextInt()` for the fields!)
3. Reads the bowler information into an array of Bowlers (more on this step shortly.)
4. Open an output file. The file's name should be the same as the input file with '.out' appended to it. [So for an input file of bowlers.in, write a file bowlers.in.out]
5. Loops through the array of Bowlers calling `getAverage()` to compute the average and then writing a line to the output file that includes the bowler's name and the average as a double.
6. Close the output file.
7. Close the input scanners for the console and the input file.

For step 3, use a separate method to do the task of reading all the bowler information and creating the array of `Bowler` objects (This is a great way to structure your code so it is manageable). Call the method `readBowlers`. Pass a scanner to it (for the bowler input file) and have it return an array of Bowler objects. So the prototype should be:

```
public static Bowler[] readBowlers(Scanner bowlerInput);
```

Notice that this is static! We are calling it from a static context (main()) so it must also be a static method.

The format of the input file is as follows:

Namestring CountOfGames Game1Score Game2Score…etc

The bowler name is only a first name so next() will work fine to read it.

The next field is an integer count of games played. Read this and that will tell you how large to make the scores array inside the Bowler object you will create. Then there is one score (between 0 and 300) for each game played.

As an example:

Tom 3 270 180 222
Dick 5 190 180 101 199 270
Harry 4 222 233 247 201

With that input data, your output file should contain the following after the program is run:

Tom: 224.0
Dick: 188.0
Harry: 225.75

You will not know how many bowler records there are! So as we did with an earlier assignment, use a temporary work array with 10 elements (assume the maximum number of bowlers will never be over 10). Then, once all the bowler objects are created, create a new array with the right number of elements for the bowlers read and copy the bowlers to the final array which will be returned by `readBowlers()`.

Important! When you open files (for input or output) and do file based I/O, you will need to write try/catch blocks to catch possible errors like `FileNotFoundException` or `IOException`. Java will complain and remind you to do this when you compile the code.

Problem 2 (15 Points)

Suppose you are writing a program that will help our Admissions Office process the applications into the University. In this problem your program will open an input file containing applicant information and read it into an array of application objects. Once you have them read into an array, you can do some interesting operations with the data that you just read in. See a sample input file named `a0.txt` included in this assignment dropbox. I added multiple input files of various sizes you're your consumption if you need them. I also provided sample output for each file. An entry (an application record) in the input file has six attributes and is formatted as follows:

- The very first line of the input file contains a number indicating the number of records contained in the entire input file.
- The first line is the id number of the application.
- The next line of an application record is the name of an applicant consisting of first name, middle name (or middle initial with a period), and last name in that order in a single line, all separated by a blank space.
- Next line is the street address in one line, city name in the next line, state in another line, and finally zip code in a separate line.
- The next line is a phone number consisting of the area code, prefix, and last four digits all separated by a blank space.
- The next line is the intended major, e.g., Computer Science, Economics, etc. A multi-word name should be allowed.
- The next line is the applicant's high school GPA, e.g., 3.96.
- The next line is an indication of whether the applicant is applying for a scholarship or not.
- If there is another application record in the file, it will follow next in the same format.

Your task is to read a file containing application records and do what is asked to be done in `ProcessApplications.java` that is provided with this assignment description.

- Your solution should consist of the following files:
- `Application.java`: This file contains a class definition that represents an application. It holds all the data about the student's application read from the input file.
- `ProcessApplications.java`: This file contains a `main` that reads in an input file and processes the application data. (A skeleton is provided. Look for the word **FILL_IT_IN**)
- It is likely that you will want to add another file, perhaps `State.java` to represent a state and use it in the `main`.

Hand in your Java files and at least one input file that works with your program. I assume that your input file would be of the same format as mine, but I would still like to have one of your input files that for sure works with your program.
When you test your program as you develop it, use a small input file, for example start with one line and make sure it works; then two lines and make sure it works, and so on. Incremental development, right? It is much easier to deal with a smaller input file. After you think you are done debugging your program, use a large input file to test it again before you hand it in.
Note: I included a program that you can use to generate an input file of any arbitrary size. The file is called `RandomApplications.java`. Take a look if you are interested. Additionally, I provided a few files you can use as test input (**a0.txt – a3.txt**).

Submission Instructions

Please follow this procedure for submission:

1. Place the deliverable files (`BowlingAverage.java, Bowler.java, ProcessApplications.java,` and `Application.java`) into a folder by themselves. The folder's name should be CSE114_PS8_<yourname>_<yourid>. So if your name is Joe Cool and your id is 12345678, the folder should be named 'CSE114_PS8_JoeCool_12345678.

2. Compress the folder and submit the zip file.
    a. On windows, do this by pressing the right-mouse button while hovering over the folder. Select 'Send to -> Compressed (zipped) folder'. The compressed folder will have the same name with a .zip extension. You will upload that file to the Brightspace dropbox for Assignment8
    b. On mac, move the mouse over the folder then right-click (or for single button mouse, use Control-click) and select **Compress**. There should now be a file with the same name and a .zip extension. You will upload that file to the Brightspace dropbox for Assignment8.
3. Navigate to the course Brightspace site. Click **Assignments** in the top menu. Click **Assignment8** in the content area. Under **Submit Assignment**, click **Add a File**. Click **My Computer** at the top of the option list. Find the zip file in your file browser and drag it to the Upload area of the dialog box that appears. Click the **Add** button at the bottom left of that dialog.
4. ***Important!!*** Click the **Submit** button! If you fail to click **Submit**, the assignment will not be downloadable and will not be graded.