Text File Navigation and Parsing in Java

This assignment involves implementing a smallish Java program that performs some basic file parsing and navigation tasks.

The program will deal with two input files. The first input file, obtained from the Lahman Baseball Database, contains records related to MLB players. A player record will contain the following information:

Description of field	Format
Lahman ID; unique to each player.	positive integer
A unique code asssigned to each player. The playerID links the data in this file with records in the other files.	alphanumeric string
An ID for individuals who served as managers	alphanumeric string
An ID for individuals who are in the baseball Hall of Fame	alphanumeric string
Year player was born	four-digit positive integer
Month player was born	positive integer in range 1-12
Day player was born	positive integer in range 1-31
Country where player was born	alphabetic string
State where player was born	alphabetic string
City where player was born	alphabetic string
Year player died	four-digit positive integer
Month player died	positive integer in range 1-12
Day player died	positive integer in range 1-31
Country where player died	alphabetic string
State where player died	alphabetic string
City where player died	alphabetic string
Player's first name	alphabetic string
Player's last name	alphabetic string
Note about player's name (usually signifying that they changed their name or played under two different names)	alphabetic string
Player's given name (typically first and middle)	alphabetic string
Player's nickname	alphabetic string, perhaps with quotes
Player's weight in pounds	positive integer
Player's height in inches	positive integer
Player's batting hand (left, right, or both)	'R' or 'L'
Player's throwing hand (left or right)	'R' or 'L'
Date that player made first major league appearance	mm/dd/yyyy
Date that player made first major league appearance (blank if still active)	mm/dd/yyyy
College attended	alphabetic string
ID used in Lahman Database version 4.0	alphabetic string
ID used in Lahman database version 4.5	alphabetic string
ID used by retrosheet	alphabetic string
ID used by Sean Holtz's Baseball Almanac	alphabetic string
ID used by Baseball Reference website	alphabetic string

Figure 1: Player Data Fields

Each record will occur on a single line. The record fields will be presented in a comma-separated-values format. The last field in each record will be followed immediately by a newline character. Here is a sample player record in comma-separated format:

```
1,aaronha01,,aaronha01h,1934,2,5,USA,AL,Mobile,,,,,,,Hank,Aaron,,Henry Louis,"Hammer,Hammerin'
Hank,Bad Henry",180,72,R,R,4/13/1954,10/3/1976,,aaronha01,aaronha01,aaroh101,aaronha01,aaronha01
```

Note that the record is shown here on multiple lines because of its length, but that the record would be on a single line in the input file.

The player record files are guaranteed to conform to this syntax, so there is no explicit requirement that you validate the files. On the other hand, some error-checking during parsing may help you detect errors in your parsing logic.

Note that it is also logically possible that some data fields will be left empty. Worse, some fields may not be empty, but may consist only of spaces. Your implementation must deal gracefully with records in which some fields are, in fact, empty or just spaces. The player ID and name fields will never be empty.

Complete sample data files will be supplied on the course website.

The bytes that make up the file can be thought of as a sequence of bytes, each at a unique offset from the beginning of the file, just like the cells of an array. So, each player record begins at a unique offset from the beginning of the file.

The second input file, whose name will also be supplied on the command line, contains a sequence of search commands that must be processed. The only types of search that must be supported are:

`report_name_at<ws><offset>`

> Write the player name for the record that occurs at that offset to the output log file, listing the first name and then the second name. For example: Hank Aaron.

`report_vitals_at<ws><offset>`

> Write (to the log file) the player name, height, weight, date of birth and date of death, formatted as follows:
>
> ```
> Name:   Hank Aaron
> Height: 6ft 0in
> Weight: 180lbs
> Born:   February 5, 1934 at Mobile AL
> Died:   ?? ??, ?? at ?? ??, ??
> ```
>
> Note that you must perform considerable parsing of the player record in order to obtain the pieces of information that are required here. You must also be prepared to deal with missing data, in which case you should print the string "??" to indicate what's missing.

`report_career_at<ws><offset>`

> Write (to the log file) the dates on which the player first and last appeared in MLB, formatted as:
>
> ```
> April 13, 1954 to October 3, 1976
> ```

Each command will occur on a line by itself. Lines beginning with a semi-colon character `';'` are comments and should be discarded. `<ws>` denotes an arbitrary amount of whitespace (spaces or tabs).

The command file is guaranteed to conform to this specification, so you do not need to perform error-checking when reading it.

Here is a sample command file:

```
; Test script for MLB Parser Program
;
; Case 1
; Display player's career dates:
report_career_at 0
; Case 2
; Display player's vital statistics:
report_vitals_at 16912
; Case 2
; Display player's name:
report_name_at 16414
; Case 3
; Display player's career dates:
report_career_at 10688
; Case 4
; Display player's name:
report_name_at 2370
; Case 5
; Display player's vital statistics:
report_vitals_at 158
; Case 6
; Display player's name:
report_name_at 9508
; Case 7
; Display player's career dates:
report_career_at 5916
; Case 8
; Display player's name:
report_name_at 4978
; Case 9
; Display player's name:
report_name_at 5738
; Case 10
; Display player's career dates:
report_career_at 1175
```

See the posted command files for additional examples.

The program will begin by making a pass through the specified data file and writing a list of the file offsets and player ID fields to a text file, whose name will be the third command line parameter to the program. Here is the beginning of an offsets file:

```
    0: adamsmi02
  158: aldrico01
  305: almonbi01
  470: anderge01
  657: aragoan01
  836: asselbr01
  . . .
```

The program will write results from executing commands to a log file, whose name will be the fourth command line parameter to the program. Each line of output must be properly terminated. Since this assignment will be graded automatically, your log file must be formatted in a certain way in order to pass testing. You should consult the *Student Guide to the Curator* for details on what is important and what is not.

The format of the log file is best illustrated with an example:

```
MLBPlayer dB Parser

dbFile: DB.csv
script: Script.txt
log:    Log.txt
----------------------------------------

Command  0:  report_career_at 0

September 10, 1972 to July 25, 1978
----------------------------------------

Command  1:  report_vitals_at 16912

Name:   Charlie Hodnett
Height: ??
Weight: ??
Born:   ?? ??, 1861 at ?? IA, USA
Died:   April 25, 1890 at St. Louis MO, USA
----------------------------------------

Command  2:  report_name_at 16414

Ramon Hernandez
----------------------------------------

Command  3:  report_career_at 10688

May 14, 1945 to April 26, 1946
----------------------------------------

Command  4:  report_name_at 2370

Dave Berg
----------------------------------------

Command  5:  report_vitals_at 158

Name:   Cory Aldridge
Height: 6ft 0in
Weight: 210lbs
Born:   June 13, 1979 at San Angelo TX, USA
Died:   ?? ??, ?? at ?? ??, ??
----------------------------------------

Command  6:  report_name_at 9508

Pat Dodson
----------------------------------------

Command  7:  report_career_at 5916

September 17, 1906 to September 2, 1918
----------------------------------------
```

```
Command  8:  report_name_at 4978

Ben Caffyn
----------------------------------------

Command  9:  report_name_at 5738

George Caster
----------------------------------------

Command 10:  report_career_at 1175

September 11, 1976 to June 8, 1977
----------------------------------------
```

Your main class (the one that implements `public static void main()`) must be named `P1`. If not, your submitted code will fail to execute properly. We will execute your program using the following syntax:

```
java P1 <MLB record file> <commands file> <offsets file> <log file>
```

If you execute your program from within Eclipse, you can still specify command-line parameters; see the Eclipse for 3114 notes for an example.

When your program begins execution, it will parse the given commands file. Each command must be echoed into the log file, on a line by itself, numbered as shown in the sample log file. Following each echoed command, your program will write a blank line, followed by one or more lines showing the results of carrying out that command, as described above. That output will be followed by a line of delimiters and another blank line.

Under no circumstances may your program store more than one MLB player record in memory at any given instant.

Note on line termination:

Each line of a text file ends with a particular marker (known as the line terminator). In MS-DOS/Windows file systems, the line terminator is a sequence of two ASCII characters (CR + LF). In Unix systems, the line terminator is a single ASCII character (LF). Other systems may use other line termination conventions.

Why do you care? Which line termination is used has an effect on the file offsets for all but the first record in the data file. As long as we're all testing with files that use the same line termination, we should all get the same file offsets. But if you change the file format (of the posted data files) to use different line termination, you will get different file offsets than are shown in the posted log files. Most good text editors will tell you what line termination is used in an opened file, and also let you change that.

Testing:

It is your responsibility to design and conduct thorough and sensible tests of your implementation before submitting it. For that purpose, you may share test input and output files (but absolutely no solution code!!) via the class Forum. You should **not** use the Curator for your own testing and debugging purposes. The Curator will only accept a limited number of submissions by each student, so use them wisely.

Implementation and Design Evaluation:

You should document your implementation in accordance with the *Programming Standards* page on the course website. It is possible that your implementation will be evaluated for documentation, as well as for correctness of results. If so, your submission that achieved the highest score will be evaluated by one of the TAs, who will assess a deduction (ideally zero) against your score from the Curator.

You should apply good object-oriented design principles in your project. Think through object responsibilities and interactions, and sketch out your design before you start coding. The most common design shortcomings with an assignment like this are to identify a too-small set of candidate classes, or to adopt a minimal design in order to reduce coding time. As inspiration, I will tell you that my solution incorporates 4 distinct classes, each of which plays an important role within the requirements of the assignment.

Keep in mind that later projects in this course may build on this one. For example, it is likely that in a later project some other part of the MLB database system will need to actually do something with various fields of the MLB player records that are retrieved in this assignment.

There are no restrictions on your use of the Java Library classes. I made use of (at least) the following in my solution:

`String`, `Integer`, `Long`, `RandomAccessFile`, `Scanner`, `FileWriter`, `Vector`

You may not use Java code from any other sources than the course notes, the Weiss text, and your own mind. It may be tempting to Google for code; aside from personal honesty, remember that other students may very well find and adopt the same code you find online, and that the likely result will be that your solution will match theirs in ways that will attract the attention of MOSS.

What to turn in and how:

This assignment will be auto-graded on the Curator system. The testing will be done under Windows (which should not matter at all) using Java version 1.7.0 update 55.

Submit a single uncompressed `jar` file (not a zip or RAR or other compressed format) containing the source code for your solution to the Curator System. Submit nothing else. Your solution should not write anything to standard output (i.e., `System.out` in Java).

Your source code submission for this assignment must be "flat". That is, you must not place code in subdirectories or use Java packages. Doing so will ensure that your submitted code does not compile.

If you have installed the JDK on your system, you can create the necessary jar file by executing a command like

```
jar -cvf MySubmissionFile.jar *.java
```

in the directory containing your `java` files. If not, you can create a suitable jar file from Eclipse; see the website for more information.

We will compile your Java source code with the following command:

```
javac *.java
```

where `javac` will invoke version 1.7.0 of the Java compiler.

Your main class (the one that implements `public static void main()`) must be named `P1`. If not, your submitted code will fail to execute properly. We will execute your program using the following syntax:

```
java P1 <MLB record file> <commands file> <offsets file> <log file>
```

If you execute your program from within Eclipse, you can still specify command-line parameters; see the Eclipse for 3114 notes for an example.

Instructions, and the appropriate link, for submitting to the Curator are given in the *Student Guide* at the Curator website:

http://www.cs.vt.edu/curator/.

You will be allowed to submit your solution multiple times, up to a limited number as indicated in the Curator system; the highest score will be counted.

Pledge:

Each of your program submissions must be pledged to conform to the Honor Code requirements for this course. Specifically, you **must** include the following pledge statement at the beginning of the file that contains `main()`:

```
//    On my honor:
//
//    - I have not discussed the Java language code in my program with
//      anyone other than my instructor or the teaching assistants
//      assigned to this course.
//
//    - I have not used Java language code obtained from another student,
//      or any source other than the course notes or textbook, either
//      modified or unmodified.
//
//    - If any Java language code or documentation used in my program
//      was obtained from another source, such as a text book or course
//      notes, that has been clearly noted with a proper citation in
//      the comments of my program.
//
//    - I have not designed this program in such a way as to defeat or
//      interfere with the normal operation of the Curator System.
//
//    <Student's Name>
```

We reserve the option of assigning a score of zero to any submission that does not contain this statement.