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, whose name will be supplied from the command line, contains a collection of data records pertaining to geographical features, obtained from the website for the USGS Board on Geographic Names (geonames.usgs.gov States, Territories and Associated Areas of the United States). The file begins with a descriptive header line, followed by a sequence of GIS records, one per line, which contain the following fields in the indicated order (all are mandatory unless indicated otherwise):

Significance	Type/Format	Comments
Feature ID (FID)	non-negative integer	unique identifier for this geographic feature
Feature name	string	standard name of feature
Feature class	string	descriptive classification of feature
State alphabetic code	two-characters	US postal code abbreviation
State numeric code	non-negative integer	numeric code for state
County name	string	county in which feature occurs
County numeric code	non-negative integer	numeric code for county
Primary latitude (DMS)	DDMMSS['N' | 'S']	feature latitude in DMS format or UNKNOWN
Primary longitude (DMS)	DDDMMSS['E' | 'W']	feature longitude in DMS format or UNKNOWN
Primary latitude (dec deg)	decimal number	feature latitude in decimal format or UNKNOWN
Primary longitude (dec deg)	decimal number	feature longitude in decimal format or UNKNOWN
Source latitude (DMS)	DDMMSS['N' | 'S']	latitude of feature source in DMS format, optional
Source longitude (DMS)	DDDMMSS['E' | 'W']	longitude of feature source in DMS format, optional
Source latitude (dec deg)	decimal number	latitude of feature source in decimal format, optional
Source longitude (dec deg)	decimal number	longitude of feature source in decimal format, optional
Feature elevation in meters	integer	altitude above/below sea level, optional
Feature elevation in feet	integer	altitude above/below sea level, optionsl
Map name	string	name of USGS topographic map including feature
Date created	string	date feature was initially committed to the database
Date edited	string	date feature record was last updated, optional

In the GIS record file, each record will occur on a single line, and the fields will be separated by pipe (`'|'`) symbols. Empty fields will be indicated by a pair of pipe symbols with no characters between them. See the posted `VA_Monterey.txt` file for many examples.

GIS 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.

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 GIS record begins at a unique offset from the beginning of the file.

Note:

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 the line termination.

Figure 2: Sample Geographic Data Records

Note that some record fields are optional, and that when there is no given value for a field, there are still delimiter symbols for it.

Also, some of the lines are "wrapped" to fit into the text box; lines are never "wrapped" in the actual data files.

```
FEATURE_ID|FEATURE_NAME|FEATURE_CLASS|STATE_ALPHA|STATE_NUMERIC|COUNTY_NAME|COUNTY_NUMERIC|PRIMARY_LAT_DMS|PRIM_LONG_DMS|PRIM_LAT_DEC|PRIM_LON
G_DEC|SOURCE_LAT_DMS|SOURCE_LONG_DMS|SOURCE_LAT_DEC|SOURCE_LONG_DEC|ELEV_IN_M|ELEV_IN_FT|MAP_NAME|DATE_CREATED|DATE_EDITED
1479116|Monterey Elementary School|School|VA|51|Roanoke (city)|770|371906N|0795608W|37.3183753|-
79.935857|||||323|1060|Roanoke|09/28/1979|09/15/2010
1481345|Asbury Church|Church|VA|51|Highland|091|382607N|0793312W|38.4353981|-79.553807|||||818|2684|Monterey|09/28/1979|
1481852|Blue Grass|Populated Place|VA|51|Highland|091|383000N|0793259W|38.5001188|-79.5497702|||||777|2549|Monterey|09/28/1979|
1481878|Bluegrass Valley|Valley|VA|51|Highland|091|382953N|0793222W|38.4981745|-79.539492|382601N|0793800W|38.4337309|-
79.633383|759|2490|Monterey|09/28/1979|
1482110|Buck Hill|Summit|VA|51|Highland|091|381902N|0793358W|38.3173452|-79.5661577|||||1003|3291|Monterey SE|09/28/1979|
1482176|Burners Run|Stream|VA|51|Highland|091|382509N|0793409W|38.4192873|-79.5692144|382531N|0793538W|38.4252778|-
79.593889|848|2782|Monterey|09/28/1979|
1482324|Mount Carlyle|Summit|VA|51|Highland|091|381556N|0793353W|38.2656799|-79.5647682|||||698|2290|Monterey SE|09/28/1979|
1482434|Central Church|Church|VA|51|Highland|091|382953N|0793323W|38.4981744|-79.5564371|||||773|2536|Monterey|09/28/1979|
1482557|Claylick Hollow|Valley|VA|51|Highland|091|381613N|0793238W|38.2704021|-79.5439343|381733N|0793324W|38.2925|-
79.556667|573|1880|Monterey SE|09/28/1979|
1482785|Crab Run|Stream|VA|51|Highland|091|381707N|0793144W|38.2854018|-79.528934|381903N|0793415W|38.3175|-79.5708333|579|1900|Monterey
SE|09/28/1979|
1482950|Davis Run|Stream|VA|51|Highland|091|381824N|0793053W|38.3067903|-79.5147671|382057N|0793505W|38.3491667|-79.5847222|601|1972|Monterey
SE|09/28/1979|
1483281|Elk Run|Stream|VA|51|Highland|091|382936N|0793153W|38.4934524|-79.5314362|383121N|0793056W|38.5226185|-
79.5156027|757|2484|Monterey|09/28/1979|
1483492|Forks of Waters|Locale|VA|51|Highland|091|382856N|0793031W|38.4823417|-79.5086575|||||705|2313|Monterey|09/28/1979|
1483527|Frank Run|Stream|VA|51|Highland|091|382953N|0793310W|38.4981744|-79.5528258|383304N|0793341W|38.5512285|-
79.561438|780|2559|Monterey|09/28/1979|
1483647|Ginseng Mountain|Summit|VA|51|Highland|091|382850N|0793139W|38.480675|-79.527547|||||978|3209|Monterey|09/28/1979|
1483860|Gulf Mountain|Summit|VA|51|Highland|091|382940N|0793103W|38.4945636|-79.5175468|||||1006|3300|Monterey|09/28/1979|
1483916|Hamilton Chapel|Church|VA|51|Highland|091|381740N|0793707W|38.2945677|-79.6186591|||||823|2700|Monterey SE|09/28/1979|
1484097|Highland High School|School|VA|51|Highland|091|382426N|0793444W|38.4071387|-79.5789333|||||879|2884|Monterey|09/28/1979|09/15/2010
1484099|Highland Wildlife Management Area|Park|VA|51|Highland|091|381905N|0793439W|38.3181785|-79.577547|||||954|3130|Monterey SE|09/28/1979|
. . .
```

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:

`show_name<tab><offset>`

> If the offset is valid (see below), write the Feature Name for the record that occurs at that offset. If the offset is not positive, write the error message "Offset not positive" to the log file.
> If the offset is larger than the final valid offset within the data file, write the error message "Offset too large".
> If the offset is non-negative but does not correspond to the first character on a line of the file, write the error message "Unaligned offset".

`show_latitude<tab><offset>`
`show_longitude<tab><offset>`

> If the offset is valid (see below), write the primary latitude or primary longitude, as specified, for the record that occurs at that offset. The specified fields should be separated by whitespace (your choice as to what). If the specified field is not included in the record, write "Coordinate not given". See the posted logs for any additional formatting requirements.
> If the offset is not positive, write the error message "Offset not positive".
> If the offset is larger than the final valid record offset within the data file, write the error message "Offset too large".
> If the offset is non-negative but does not correspond to the first character on a line of the file, write the error message "Unaligned offset".

`show_elevation<tab><offset>`

> If the offset is valid (see below), write the elevation in feet field for the record that occurs at that offset. One complication is that the elevation fields are optional; if the elevation is not given, write "Elevation not given".
> If the offset is not positive, write the error message "Offset not positive".
> If the offset is larger than the final valid offset within the data file, write the error message "Offset too large".
> If the offset is non-negative but does not correspond to the first character on a line of the file, write the error message "Unaligned offset".

The only other command is:

`quit<tab>`

> Cease processing the commands file, log the message "Exiting", close all files and exit the program.

Each command will occur on a line by itself. Lines beginning with a semi-colon character `';'` are comments and should be ignored. The command file is guaranteed to conform to this specification, so you do not need to worry about error-checking when reading it. See the posted command files for examples.

The program will write results to a single output file, named `Results.txt`. Each line of output must be properly terminated.

When your program begins execution, it will parse the given GIS record file and report the file offset and FID field for each of the records found in the file, listed in the order the records occur in the file. This section of the output file will consist of a header, followed by one line for each GIS record containing the file offset, followed by some whitespace (your choice as to what), followed by the FID field for that GIS record, and then a newline character. See the posted log files for examples.

Your program will then process the given commands file. Each command must be echoed into the log file, on a line by itself, numbered as shown in the posted log files. Following each echoed command, your program will write one line reporting the results of carrying out that command, as described above.

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

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 for locking in your grades.

Correctness 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.

Note that the evaluation of your project may depend substantially on the quality of your code and documentation.

Design Considerations:

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 8 distinct classes and 2 enumerated types, all of which play important roles 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 GIS database system will need to actually do something with various fields of the GIS records that are retrieved in this assignment.

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 1.8u25 or later.

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.

Your main class (the one that implements `public static void main()`) must be named `Project1`. If not, your submitted code will fail to execute properly. We will execute your program using the following syntax:

```
java Project1 <GIS record file name> <commands file name>
```

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 other unauthorized source, 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.