File Navigation and Text Parsing in Java

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

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). 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:

Figure 1: Geographic Data Record Format

Name	Type	Length/ Decimals	Short Description
Feature ID	Integer	10	Permanent, unique feature record identifier and official feature name
Feature Name	String	120	
Feature Class	String	50	See Figure 3 later in this specification
State Alpha	String	2	The unique two letter alphabetic code and the unique two number code for a US State
State Numeric	String	2	
County Name	String	100	The name and unique three number code for a county or county equivalent
County Numeric	String	3	
Primary Latitude DMS	String	7	The official feature location *DMS-degrees/minutes/seconds* *DEC-decimal degrees.* *Note: Records showing "Unknown" and zeros for the latitude and longitude DMS and decimal fields, respectively, indicate that the coordinates of the feature are unknown. They are recorded in the database as zeros to satisfy the format requirements of a numerical data type. They are not errors and do not reference the actual geographic coordinates at 0 latitude, 0 longitude.*
Primary Longitude DMS	String	8	
Primary Latitude DEC	Real Number	11/7	
Primary Longitude DEC	Real Number	12/7	
Source Latitude DMS	String	7	Source coordinates of linear feature only (Class = Stream, Valley, Arroyo) DMS-degrees/minutes/seconds DEC-decimal degrees. *Note: Records showing "Unknown" and zeros for the latitude and longitude DMS and decimal fields, respectively, indicate that the coordinates of the feature are unknown. They are recorded in the database as zeros to satisfy the format requirements of a numerical data type. They are not errors and do not reference the actual geographic coordinates at 0 latitude, 0 longitude.*
Source Longitude DMS	String	8	
Source Latitude DEC	Real Number	11/7	
Source Longitude DEC	Real Number	12/7	
Elevation (meters)	Integer	5	Elevation in meters above (-below) sea level of the surface at the primary coordinates
Elevation (feet)	Integer	6	Elevation in feet above (-below) sea level of the surface at the primary coordinates

Map Name	String	100	Name of USGS base series topographic map containing the primary coordinates.
Date Created	String		The date the feature was initially committed to the database.
Date Edited	String		The date any attribute of an existing feature was last edited.

Notes:

- See https://geonames.usgs.gov/domestic/states_fileformat.htm for the full field descriptions.
- The type specifications used here have been modified from the source (URL above) to better reflect the realities of your programming environment.
- Latitude and longitude may be expressed in DMS (degrees/minutes/seconds, 0820830W) format, or DEC (real number, -82.1417975) format. In DMS format, latitude will always be expressed using 6 digits followed by a single character specifying the hemisphere, and longitude will always be expressed using 7 digits followed by a hemisphere designator.
- Although some fields are mandatory, some may be omitted altogether. Best practice is to treat every field as if it may be left unspecified. Certain fields are necessary in order to index a record: the feature name and the primary latitude and primary longitude. If a record omits any of those fields, you may discard the record, or index it as far as possible.

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 the syntax described above, 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.

A 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.

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, `0X0D0A`). In Unix systems, the line terminator is a single ASCII character (LF). Other systems may use other line termination conventions.

Why should 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 scheme.

All that being said, as project is auto-graded, all input files will have the same line termination, and the grading of correctness will depend on whether you report the correct search results, not on the file offsets you report.

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|
. . .
```

We will execute your program using the following syntax:

```
java GISParser <GIS record file name> <commands file name> [-offsets]
```

Note this implies your main class (the one that implements `public static void main()`) must be named `GISParser`. If you execute your program from within Eclipse, you can still specify command-line parameters; see the Eclipse for 3114 notes for an example.

The first input file will be a GIS record file, as described above. 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 field for the record that occurs at that 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".

The Primary Latitude and Longitude fields are given in the GIS records in two formats, DMS and DEC. You should parse the DMS version. Your output must reformat the latitude or longitude in a human-friendly manner. Specifically:

1090224W → 109d 2m 24s West

Note that latitude values will always consist of 6 decimal digits, followed by a hemisphere designator ('W' or 'E'), and longitude values will always consist of 7 decimal digits, followed by a hemisphere designator ('S' or 'N'). The Java String and Scanner classes both have useful methods for breaking out the relevant parts.

`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".

What about invalid offsets?

- If the offset is not positive, write the error message “Offset is not positive”.
- If the offset is larger than the length of the data file, write the error message “Offset is too large”.
- If the offset is non-negative but does not correspond to the first character on a line of the file that contains a GIS record, write the error message “Unaligned offset”.

Note that since this assignment is autograded by the Curator, spelling, punctuation and capitalization must be exactly correct.

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. Blank lines are possible. 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.

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

When your program is invoked, there is an optional switch: `-offsets`. If the switch is present, your program will parse the given GIS record file and write 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, and write its output to a file named `Offsets.txt`. Your program will not process the commands file in this case. See the posted log files for examples. The motivation for this option is to make it easy to check what offsets your program is determining for the records, which will affect the correctness of the searches. This feature will not be tested.

If the optional switch is omitted, your program will process the given commands file (and NOT write out a list file offsets and FIDs), and write its output to a file named `Results.txt`. Each command must be echoed into the output 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, nor is your program allowed to store a set of file offsets for the records in the file.

Testing

Sample GIS and command files will be supplied on the course website, along with the corresponding correct results. You must test your solution with each of those samples. There is no guarantee these will cover all the logical cases.

Beyond that, 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.

Solutions to this assignment will be graded on CentOS 7, as installed on the rlogin cluster. The evaluation will be performed using command-line tools only. The resources page includes instructions for using command-line tools to compile and execute Java programs. You should consult those, as well as the course staff when you have questions.

You may develop your solution on Windows or Linux, as you like, and you may use Eclipse or any other IDE you like. But, it is your responsibility to ensure that you have tested your code with the posted testing/grading code on CentOS. Failure to do that may result in a score of zero on this assignment.

Grading Code

The website has a link to a zip file containing tools you can use to test and grade the correctness of your solution. In fact, these are the same tools that we will use to do the grading of your solution (instead of the Curator's built-in tools).

Since this is the exact code that will be used when we grade your solution, it is imperative that you make good use of this code in your own testing. See the `readme.txt` file for detailed instructions.

Documentation

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.

You should consider what possible errors might be encountered, and which ones you should check for. One common example is the validation of command-line parameters. We will not test your solution with invalid parameters, but in reality it's fairly common for a user to supply incorrect, or no, parameters. It's simple to test for the existence of files, and log an error message and exit cleanly if expected files do not exist. As mentioned earlier, a GIS record may not specify a value for every field. In other cases, but not this assignment, logically invalid values may be supplied; for example, a character string (e.g., "Fred") where a numeric value is expected. A program that crashes, or even just computes nonsensical results, in such a situation looks very unprofessional.

Finally, you should be careful about designing your solution to catch exceptions and write useful diagnostics to standard output if an exception is caught, especially if it cannot be handled internally.

What to Submit

This assignment will be auto-graded under CentOS (which should not matter at all) using Java version 1.8u20 or later, and the posted testing/grading code.

For this assignment, you must submit a zip file containing all the Java source code files for your implementation (i.e., `.java` files). Submit only the Java source files. Do not submit Java bytecode (class) files. If you use packages in your implementation (and that's good practice), your zip file must include the correct directory structure for those packages. That's easy to verify by copying and unzipping the file you are planning to submit.

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, in order to make corrections. The Curator will not be used to grade your submissions in real time, but you will have already done the grading using the posted code.

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, including the Internet, 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 supplied grading code.
//
//    <Student's Name>
```

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

Change Log

Version	Date	Change(s)
4.00	May 21	Base document