

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, named `GISData.txt`, contains a collection of data records, one per line, that contain information about geographical features.

More precisely, each GIS record will contain the following fields in the indicated order (all are mandatory unless indicated otherwise):

Figure 1: Geographic Data Fields

Significance	Type/Format	Comments
Feature ID number (FID)	non-negative integer	unique identifier for this geographic feature
State alphabetic code	two-characters	US postal code abbreviation
Feature name	string	standard name of feature
Feature type	string	descriptive classification of feature
County name	string	county in which feature occurs
State number code	non-negative integer	numeric code for state
County number 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	integer	altitude above/below sea level, optional
Est. feature population	non-negative integer	estimated population of feature, optional
Federal status	string	???, optional
Cell	string	???

In the GIS record file, each record will occur on a single line, and the fields will be separated by pipe (`|`) symbols. Some sample records are shown in Figure 2 below.

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 that.

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.

1495182	VA	Acre of Rocks	summit		Montgomery		51		121		371636N		0801608W		37.27667		-80.26889			2270			McDonalds Mill							
1674451	VA	Agnew Hall		building		Montgomery		51		121		371329N		0802528W		37.22472		-80.42444					Blacksburg							
1481269	VA	Aiken Dam	Hollow		valley		Montgomery		51		121		372056N		0801741W		37.34889		-80.29472		371932N		0801752W		37.32556		-80.29778			McDonalds Mill
1674452	VA	Airport Acres	(subdivision)		pp1		Montgomery		51		121		371238N		0802427W		37.21056		-80.4075					2125			Blacksburg			
1462389	VA	Albarn High School		school		Montgomery		51		121		370341N		0802633W		37.06139		-80.4425							Riner					
1674453	VA	Alleghany Addition	(subdivision)		pp1		Montgomery		51		121		370744N		0802350W		37.12889		-80.39722					2120			Blacksburg			
1462398	VA	Alleghany Church		church		Montgomery		51		121		370726N		0801610W		37.12389		-80.26944							Pilot					
1462399	VA	Alleghany Church		church		Montgomery		51		121		370925N		0801519W		37.15694		-80.25528							Ironto					
1481276	VA	Alleghany Church		church		Montgomery		51		121		370930N		0802507W		37.15833		-80.41861							Blacksburg					
1674454	VA	Alleghany Heights	(subdivision)		pp1		Montgomery		51		121		371509N		0802447W		37.2525		-80.41306					2220			Newport			
1674455	VA	Alleghany School	(historical)		school		Montgomery		51		121		370812N		0801519W		37.13667		-80.25528							Ironto				
1674456	VA	Alleghany School	(historical)		school		Montgomery		51		121		370921N		0802430W		37.15583		-80.40833							Blacksburg				
1674457	VA	Alleghany Spring School	(historical)		school		Montgomery		51		121		370706N		0801602W		37.11833		-80.26722							Pilot				
1462400	VA	Alleghany Springs		pp1		Montgomery		51		121		370741N		0801555W		37.12806		-80.26528							1399			Ironto		
1481287	VA	Allen Hollow		valley		Montgomery		51		121		372001N		0801939W		37.33361		-80.3275		371849N		0801938W		37.31361		-80.32722			McDonalds Mill	
1462408	VA	Allen Hollow		valley		Montgomery		51		121		370849N		0801614W		37.14694		-80.27056		370937N		0801636W		37.16028		-80.27667			Ironto	
1674458	VA	Altoona School	(historical)		school		Montgomery		51		121		370425N		0801805W		37.07361		-80.30139							Pilot				
1674459	VA	Ambler Johnston Hall		building		Montgomery		51		121		371323N		0802517W		37.22306		-80.42139							Blacksburg					
1481314	VA	Anderson Cemetery		cemetery		Montgomery		51		121		371519N		0802058W		37.25528		-80.34944							McDonalds Mill					
1674460	VA	Apperson Park	(subdivision)		pp1		Montgomery		51		121		371425N		0802413W		37.24028		-80.40361					2220			Blacksburg			
1674461	VA	Apple Acres	(subdivision)		pp1		Montgomery		51		121		370923N		0802508W		37.15639		-80.41889					2130			Blacksburg			
1674462	VA	Asbury Methodist Church		church		Montgomery		51		121		370800N		0802436W		37.13333		-80.41							Blacksburg					
1674463	VA	Atkinson Acres	(subdivision)		pp1		Montgomery		51		121		370712N		0802436W		37.12		-80.41					2060			Riner			
1481363	VA	Austin Hollow		valley		Montgomery		51		121		371621N		0801822W		37.2725		-80.30611		371617N		0801657W		37.27139		-80.2825			McDonalds Mill	
1462695	VA	Bain Chapel		church		Montgomery		51		121		370536N		0803150W		37.09333		-80.53056							2116			Radford South		
1674464	VA	Bangs	(historical)		pp1		Montgomery		51		121		370828N		0802407W		37.14111		-80.40194					2029			Blacksburg			
1674465	VA	Barringer Hall		building		Montgomery		51		121		371334N		0802502W		37.22611		-80.41722							Blacksburg					
1481476	VA	Barringer Mountain		summit		Montgomery		51		121		370817N		0802809W		37.13806		-80.46917					2342			Blacksburg				
1477097	VA	Basham		pp1		Montgomery		51		121		370210N		0802034W		37.03611		-80.34278					2430			Pilot				
1481600	VA	Beeks School		school		Montgomery		51		121		371249N		0802423W		37.21361		-80.40639							Blacksburg					

The second input file, named `Commands.txt`, contains a sequence of search commands that must be processed. The only type of search that must be supported is:

```
record_at<tab><offset>
```

If the offset is valid (see below), write the complete record that occurs at that offset to the output log file.

If the offset is negative, write the error message “Negative offset” 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” to the log file.

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” to the log file.

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`.

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 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.

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.

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 and design, 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.

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. 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 the GIS records that are retrieved.

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 1.6.21.

Submit a single zip file (not an 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 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.

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.

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 pledge statement provided on the Programming Standards page in one of your submitted files.