Evan Buss README 4/17/19
CSC402
Project #3

Javadoc Link

https://unixweb.kutztown.edu/~ebuss376/CSC402/Project3/package-summary.html

Compilation

The program consists of 4 different Java files.

To compile all the files run the following command:

```
javac Encode.java Access.java State.java Date.java
```

or

```
javac *.java
```

Execution

There are two different executable programs; Encode and Access.

Encode

Encode expects a single argument consisting of a properly formatted "state" data file.

```
java Encode [state data file]
```

Default Usage:

```
java Encode States90.txt
```

Access

Access expects two arguments consisting of the index file and then the encoded data file.

```
java Access [index file] [encoded file]
```

Default Usage:

```
java Access states.idx states.enc
```

Evan Buss README 4/17/19
CSC402
Project #3

Encoding State Data

Encode.java completes the file encoding in two steps. First, it parses the provided states data file. It reads each state and its data fields using a Scanner object. The text data is then converted into a State object and put into an ArrayList of State objects. Each State in the data file is present in the ArrayList.

The program then goes through each State object in the ArrayList and retrieves the values of each field. It writes these fields in a specific order into the encoded file. Between each of the State's fields, it separates them using a '#' character. This was necessary because some of the fields are multi-word and you need to know where one field stops and the next starts. As each State is written to the encoded file, there is a counter that keeps track of the total number of characters that have been inserted. The program creates and writes to an index file while this process is occurring. The index file has a single line for each State that is inserted into the encoded file. As the encoded file is being written to, each State name is written to the index file along with its starting and ending location in the encoded file. Once finished, the program will have created two new files, "states.idx" and "states.enc".

Data File Format:

```
OHIO
10847115 7 264.9 41330 35 3 1 1803 17 COLUMBUS
```

The above state data would be written to the encoded file like this:

```
OHIO#10847115#7#264.9#41330#35#3#1#1803#17#COLUMBUS
```

And the corresponding entry in the index file would look like this:

```
OHIO 0 51
```

Evan Buss README 4/17/19
CSC402
Project #3

<u>Access Data from Encoding</u>

Access.java receives the index file and the encoded file as command line arguments.

The program reads from the index file line by line when started. For each line, it splits on the last space to get the ending index, then it splits again to to get the starting index and the state name. The program now has the state name and it's start and end location in the encoded data file. A Java HashMap object is created with the state name as it's key and an Integer array of length 2 to store the start and end positions of each state.

When the user selects that they want to search for a specific state, they enter a state name and it is searched for in the HashMap. The HashMap returns the locations array, and then we are ready to retrieve the data from the encoded file.

The encoded file is set as a RandomAccessFile that allows you to access pieces of it non-sequentially unlike a normal file reader. The program seeks in the RandomAccessFile to the location returned by the HashMap search. The program then creates a buffer array of "end index – start index" length. The program reads from the RandomAccessFile and fills the byte array. The byte array is then converted to a string. The resulting string is split into an array on the FIELD_DELIMITER character (#) and the individual fields are loaded into a State object. The state object can than be accessed to get individual values or the State's entire contents.

<u>Why HashMap?</u>

Java's HashMap is a non-synchronized hash table. I could have used the HashTable class but it is generally preferred to use the HashMap if you don't have to worry about synchronization issues. I chose to use a hash table because it allows for very quick data retrieval.

<u>Notes</u>

- The docs are generated with Javadoc
- The files are supplied via command line arguments
  - Encode expects the States90.txt as its only argument
  - Access expects states.idx and states.enc in that order.