About Lab 6
In Lab 6 you will create a Concordance -- an index of all of the words in a text file and a list of the line numbers on which each word appears.
The primary data structure we will use is TreeMap\(<K, V>\) which associates \textit{values} with \textit{keys}. If you give a map a key it returns the value associated with it. In the class TreeMap\(<K,V>\) K is the type of the keys and V is the type of the values. A TreeMap stores this data in a BinarySearchTree based on the keys.

Here are some methods of the TreeMap class:

- \(V \text{ get}(K \text{ key})\) This gives the value associated with the key. It returns \textit{null} if \textit{key} is not actually a current key of the map.
- \(\text{Boolean put}(K \text{ key, V value})\) This adds the key-value association to the map.
containsKey(K x) returns true if x is currently a key of the map
size() gives the number of keys in the map.
Set<K> keySet() gives the current set of keys.
We have used Strings many times but in this lab we will look at their elements. A String in Java is a sequence of char. The primitive type char has a wrapper class Character.

One method of class String is

```java
char charAt(int index)
```

So if String S is "abcd" then S.charAt(0) is 'a', S.charAt(1) is 'b' and so forth. Note that 'a' is a char and not a String. In Python 'a' and "a" are the same; in Java they are not.
We will also use the substring method of the String class. 
S.substring(int i, int j) is the substring of String S starting at index i 
and going up to but not including index j.

"abcdefg".substring(2, 4) is "cd"
The concordance we build stores data in a TreeMap<String>, ArrayList<Integer>> structure. The keys are words of the file; the values are lists of the line numbers on which the words appear.

To build this structure we open a text file with a Scanner. We get each line of the file and open a second scanner on that line to get the individual words of the line. Each time we get a new line we increment a lineCount variable; we add each word with this line number to the structure.
The words that come from the scanner have all sorts of punctuation marks attached to them. For example, if we open a scanner on the line

"How are you?" he asked.

we get the words

"How
are
you?"
he
asked.

You want to strip the punctuation to get the simple words *How are you* and so forth
We will remove punctuation marks from the beginning and ends of words. We won't remove internal punctuation marks, such as the apostrophe in *isn't* and the hyphen in *life-size*.

For this lab we will regard everything except letters and digits to be punctuation.
To remove the punctuation start a variable i at index 0 and variable j at the last index of the string. While the character at index i is punctuation increment i; while that at index j is punctuation decrement index j.

The substring between i and j+1 is the word we want. Of course, if i and j cross just stop; there is no word left.
Once we have stripped the word of punctuation we reduce it to lowercase (there is a String method for that) and if it has positive length we add it and its line number to the TreeMap. We don't want to add empty strings as keys to the map.
The Concordance constructor builds a TreeMap from the file whose name is the constructors’ argument. Beyond that there are three public Concordance methods:

- `printAlphabetically()` This prints the keys of the Concordance (i.e. the words of the file) in alphabetical order, and after each the line numbers on which it appears. If a word appears on more than 10 lines (i.e. the value associated with the word is an ArrayList longer than 10), just print the number of entries rather than the entries themselves.

- `printByFrequency()` This also prints the keys of the Concordance with the associated values, but this time in order from smallest frequency to largest frequency. You will need to make a Comparator that compares two strings based
on the length of their associated value lists.

- **Search(String word)** This prints the Concordance entry for the given String. For example if the argument is “apple” it might print
  
  apple [ 3 5 5 8 ]
  
  meaning that the word “apple” appeared on line 3, twice on line 5, and again on line 5.