ICT 4361 — Java Programming Exercise 6

Purpose:

This exercise will familiarize you with file processing, and provide additional experience in text processing in Java.

Our approach will be based on a simple Mad-Libs exercise, suggested by the MIT introductory course to Java, (MIT OpenCourseWare Java Preparation Course), as begun in the previous exercise.

In this week's exercise, we add a file-parsing front-end on the basic MadLib construct.

There are many ways to extend this exercise.

You can check out 

Madlibs.org to see how a web front end is added to a similar capability.

What To Hand In:

Please hand in a listing for each program requested, formatted in an easy-to-read style.

Ensure your name, and the name of the file is available in a comment at the top of the file.

Also, ensure that you have a sample of the output from the program.

If your program fails to compile, hand in your error listing as your output.

For each question asked, provide one or two sentences summarizing your answer. Please be both complete and succinct.

Problems:

Begin with your solution—or the sample solution—from last week.

1. Create a class called MadlibFileReader which has the following methods:
    - A no-parameter constructor
    - A constructor which takes a file name
    - A setFile method which takes a file name
    - A readLine method which returns one line read from the file
    - A getTokens method which returns an array of tokens found on the line. A token is either a buffer of text, or a replacement slot. These slots are recognized by starting with a < and ending with a >
    - A way to test the class to ensure it works properly (e.g., read a file, and output the resulting tokens).
2. Create a class called MadlibFile which encapsulates a simple main program (not very different than MadHello in many ways):

- Gets a filename from the command line or by prompting the user (implement one of the choices)
- Creates a new MadlibFileReader with this filename
- Creates a MadLib instance with the filename as the title
- While it can read a line from the MadlibFileReader:
    - Break the line into tokens
    - For each token, if it is a simple string (i.e., doesn't begin with a <, add it as a string.
    - Otherwise, add it as a slot

  Note that various text processes, such as trimming and substrings will be needed to make this go smoothly.

3. Create your own MadLib template file, and test your program by running MadlibFile with it

Notes:

- The no-parameter constructor and the one-parameter constructor should call `setFile`. The first with a default file name, and the second with the parameter.
- The `setFile` method needs to arrange for the file to be read. It would be best, so the file can be ingested a line at a time, for the input to be setup as a `BufferedReader` object. A `BufferedReader` requires a `FileReader` to construct it. A `FileReader` is constructed from a `File` Also, note that it is possible the file does not exist, or perhaps cannot be read. Thus, your `setFile` may want to call a `setInput` function like so:

```
private void setInput(String filename) {
try {
FileReader f = new FileReader(filename);
input = new BufferedReader(f); // Assumes input is the field name
} catch (FileNotFoundException fnfe) {
System.err.println("File "+file+" not found");
System.exit(0); // Quit the program
}
}
```

- The `readLine` method can return `input.readLine()`, just like any other `BufferedReader`. However, you may also need to catch a possible `IOException` it may raise.
- The class `StringTokenizer` provides a very flexible way to parse input strings. Note that, each time you find a "<" token, you next need look for a ">" token, to find the end of the `Slot` name.
- The basic `StringTokenizer` methods are `hasMoreTokens()`, which returns `true` when there is another token to read, and `nextToken(delimiter)`, which returns the next String bounded by that delimiter.
- When constructing a `StringTokenizer`, you may provide the default token delimiter, and a boolean indicating whether you'd like to get the delimiters themselves back as tokens.
- When accumulating your array of results, you may find it useful to temporarily store them in a `List<String>`, since it is easy to add `String`s to it. A `LinkedList` of `String` is a

good implementation class. To turn a `List` into an array, remember to use the `toArray` method of the collection object, and pass a `new String[0]` as a parameter to coerce the return type.

Evaluation Criteria:

Criteria	Weight
MadlibFileReader, test, and output	35
MadlibFile program and test output	35
Your own MadLib template, and its output run	30