COP5555 Spring 2012
Project 1

Assigned Jan 17
Due: Jan 31 at 3pm

This semester, we will implement a compiler for a simple programming language. The most interesting feature of the language is support for a Map type similar to that found in many scripting languages.

The first part of the project is to implement a scanner for the given lexical structure.

Use the given TokenStream , Scanner, and Kind classes as starting a starting point. These classes has a structure that is similar to that used by eclipse IDEs. A slightly simpler version was described in the Lexical Analysis lecture.

The basic idea is that TokenStream class is initialized with char array containing the input characters. It also contains an initially empty list of Token objects called tokens. Your Scanner takes a TokenStream instance and scans its input characters, inserting Token objects into the tokens list. The Token class is an inner class of the TokenStream class.

The way these classes would be used in a parser is as follows

```
TokenStream stream = new TokenStream(input);
Scanner s = new Scanner(stream);
s.scan();  //create the Token array
//loop over the token list and process tokens
TokenStreamIterator iter = stream.iterator();
while (iter.hasNext()) {
      process iter.next();
}
```

An almost complete version of the TokenStream class has been provided. For your convenience, it has several constructors that get the input from different sources. (For example, if you want to read from a file, you can pass it a FileReader). Implement the missing methods.

One thing you need to implement in this class is the getLineNumber method in the Token inner class. This will require adding appropriate fields and methods. The preferred way to do this is to make a pass over the input array and store the indices of the eol characters in an array. Then, when you want to find the line number of a token, do a binary search on the array. The java.util.Arrays class has useful methods. Another alternative is to keep track of line numbers during scanning and store the line number for each token.

The Scanner class has more to do. Use the given class as a starting point.

You may add fields and methods to the class but do not change the names or signatures of the public methods. Do not use any static methods or variables, as doing so may not work with our test scripts.

A class with a few JUnit tests has also been provided so that you can ensure that things are set up properly and to define the correct answer for a few tricky cases. This assumes the class is in a package called tests. You can run it in eclipse by right clicking on the file name, selecting Run As in the menu, and then choose JUnit Test. If JUnit is not in your build path, an easy way to get it there is select new (in the upper left corner) and try to create a new JUnit test. In the process, eclipse will ask if you want to update your build path. Using JUnit is optional, but a good thing for Java programmers to know how to do. If you don't want to use JUnit, just write a main method that calls the test methods and replace assertEquals(a,b) with assert a.equals(b). Make sure to execute your test with assertions enabled (i.e. command line switch –ea).

Turn in a jar file called P1.jar containing the sources Kind.java TokenStream.java and Scanner.java. We will recompile and call the public methods in both classes from our test program. Do not change the signatures of these methods or the package declarations or you will break the test script.

Hints:

Work incrementally. First handle and test the single character tokens and white space. Then the two character tokens, then INTEGER_LITERALS, etc. Make sure with multiple character tokens that you test for EOF and handle it appropriately; sometimes it is an error, other times, it just terminates the current token. Comment out the tests in the sample test program until you are ready to test them.

Useful methods **:**

- Character.isJavaIdentifierStart(char ch) == char is in A..Z,a..z,$,_
- Character.isJavaIdentifierPart(char ch) == char is in A..Z,a..z,$,_,0..9
- Character.isDigit(char ch) == char is in 0..9