**Lab 3: Silly Sentences and Fancy Fractals**

In this lab, you will be working with recursion. In the first part of the lab, you will write a program that generates random sentences based on a set of syntax rules. In the second, you will use recursion to create pictures of fractals.

**Part 1: Recursive Syntax**

The grammar of natural languages such as English exhibits a recursive structure. This structure can be expressed in syntax rules written in the format known as BNF (Bachus-Naur Form, named after the people who invented it). You have probably seen BNF used to specify the syntax of programming languages. While BNF is ordinarily used as a guide for parsing (that is, determining whether and how a given string follows the syntax rules), it can also be used a guide for generating strings that follow the syntax rules. An example of this can be found in the sample program SimpleRandomSentences. In this example, each syntax rule -- except for the most basic ones -- is represented by a method that generates strings that follow that rule. Where one syntax rule refers to another rule, the method that represents the first rule calls the method that represents the second rule.  
  
For the first exercise of the lab, you should write a similar program that implements the following rules:  
  
<sentence> ::= <simple\_sentence> [ <conjunction> <sentence> ]  
  
<simple\_sentence> ::= <noun\_phrase> <verb\_phrase>  
  
<noun\_phrase> ::= <proper\_noun> |  
<determiner> [ <adjective> ]. <common\_noun> [ who <verb\_phrase> ]  
  
<verb\_phrase> ::= <intransitive\_verb> |  
<transitive\_verb> <noun\_phrase> |  
is <adjective> |  
believes that <simple\_sentence>  
  
<conjunction> ::= and | or | but | because  
  
<proper\_noun> ::= Fred | Jane | Richard Nixon | Miss America  
  
<common\_noun> ::= man | woman | fish | elephant | unicorn  
  
<determiner> ::= a | the | every | some  
  
<adjective> ::= big | tiny | pretty | bald  
  
<intransitive\_verb> ::= runs | jumps | talks | sleeps  
  
<transitive\_verb> ::= loves | hates | sees | knows | looks for | finds  
  
As in SimpleRandomSentences.java, you can use arrays to implement the last seven rules in this list. (You might improve on that program by writing a single method "void String randomItem(String[] listOfStrings)" for picking a random item from an array of strings.) You are welcome to add to or modify the items in the lists given here.  
  
For each of the first three rules, you should write a subroutine to represent that rule. Note that a choice of alternatives (represented in the rules by "|") can be implemented using a switch or if..else statement; the various choices don't necessarily have to have the same probability. An optional element (represented by brackets, "[ xxx ]") can be implemented by a simple if. And a repeated optional element (represented by brackets with dots, "[ xxx ]...") can be represented by a while loop. You should implement the first four rules exactly as stated here. The main routine should call the <sentence> subroutine to generate random sentences.  
  
You have to be careful in this program to avoid infinite recursion in this program. Since it will use random choices, there is no guarantee that the recursion will ever end. If your probabilities of doing recursion and continuing loops are too high, it is possible for the program to get lost in recursive calls forever -- or to produce some finite but ridiculously long sentences. You should adjust your probabilities to make sure that this doesn't happen, but that you still get some interesting sentences.