Homework Turnin

Name:	Xuqing Wu
Account:	xw88 (xw88@uw.edu)
Student ID:	1933202
Section:	AD
Course:	CSE 143 20wi
Assignment:	a5
Receipt ID:	c5a28ae6f0f90766e92bb33f40eeaa37

Warning: Your turnin is 1 day late. Assignment a5 was due Thursday, February 13, 2020, 9:00 PM.

Turnin script completed with output:

Turnin Successful!

The following file(s) were received:

GrammarSolver.java (3501 bytes, sha256: 0b6fe594ba1fe298232c5be3bbfedb8d)

```
1.  // Xuqing Wu
2.  // 2/9/2020
3.  // CSE143
4.  // TA: Eric Fan
5.  // Assignment #5
6.  //
7.  // Class GrammarSolver allows a client to randomly generate
8.  // different grammar types sentence for as many times they want
9.  // by reading an input file with a grammar in Backus-Naur Form.
10.
11. import java.util.*;
12.
13. public class GrammarSolver {
14.    private SortedMap<String, List<String[]>> nonterminalToRules;
15.    //key of map is nonterminal symbol and value of map is
16.    //rules which belong to the symbol
17.
18.    //pre: the list of string passed is not empty and there
19.    //     is only one entry in the list for one nonterminal
20.    //     symbol(throw an IllegalArgumentException if not)
21.    //post: store the list of grammar passed in a convenient way
22.    //      through splitting strings. The list passed is not changed.
23.    public GrammarSolver(List<String> grammar) {
24.       if(grammar.isEmpty()) {
25.          throw new IllegalArgumentException("Grammar is empty!");
26.       }
27.       nonterminalToRules = new TreeMap<>();
28.       for(String form: grammar) {
29.          String[] parts = form.split("::=");
30.          if(nonterminalToRules.containsKey(parts[0])) {
31.             throw new IllegalArgumentException
32.                ("Two or more entries for the same nonterminal!");
33.          }
34.          List<String[]> ruleElement = new ArrayList<>();
```

```
35.            String[] rules = parts[1].split("[|]");
36.            for (int i = 0; i < rules.length; i++) {
37.               ruleElement.add(rules[i].trim().split("[ \t]+"));
38.            }
39.            nonterminalToRules.put(parts[0], ruleElement);
40.         }
41.      }
42.
43.      //post: return true if the passed string is a nonterminal of the grammar
44.      public boolean grammarContains(String symbol) {
45.         return nonterminalToRules.containsKey(symbol);
46.      }
47.
48.      //post: return a string to represent the nonterminal symbols from the
49.      //      grammar. its form should be sorted, comma-separated list enclosed
50.      //      in square brackets
51.      public String getSymbols() {
52.         return nonterminalToRules.keySet().toString();
53.      }
54.
55.      //pre: the grammar contains the passed string and the
56.      //     number of times passed is more than or equal to 0
57.      //     (throw an IllegalArgumentException if not)
58.      //post: return the sentences being asked to generate as an array of strings
59.      //      by randomly generating sentences of the passed symbol for the
60.      //      passed number of times, each rule of nonterminal symbol should
61.      //      have equal probability to be chosen
62.      public String[] generate(String symbol, int times) {
63.         if(times < 0 || !grammarContains(symbol)) {
64.            throw new IllegalArgumentException();
65.         }
66.         String[] result = new String[times];
67.         for(int i = 0; i < times; i++) {
68.            result[i] = getString(symbol);
69.         }
70.         return result;
71.      }
72.
73.      //post: generate one sentence of the symbol passed using recursing method
74.      //      and return that string, there is one space between each terminal
75.      //      and no leading or trailing spaces in the string
76.      private String getString(String symbol) {
77.         Random rand = new Random();
78.         if(!grammarContains(symbol)) {
79.            return symbol;
80.         }
81.         int range = nonterminalToRules.get(symbol).size();
82.         int num = rand.nextInt(range);
83.         String str = "";
84.         for(int i = 0; i < nonterminalToRules.get(symbol).get(num).length; i++) {
85.            String added = nonterminalToRules.get(symbol).get(num)[i];
86.            str = str + getString(added) + " ";
87.         }
88.         return str.trim();
89.      }
90.   }
```