Homework Turnin

Name:	Ameya Singh
Account:	ameyas (ameyas@uw.edu)
Student ID:	1868457
Section:	AQ
Course:	CSE 143 18au
Assignment:	a5
Receipt ID:	93d5a4da7a4cb46f07c06c88bc71abd0

Turnin Successful!

The following file(s) were received:

GrammarSolver.java (5016 bytes)

```
1. /*
2.  * Author: Ameya Singh
3.  * CSE 143 AQ
4.  * TA: Soham P.
5.  * Homework 5: GrammarSolver
6.  */
7.
8. import java.util.*;
9.
10. /**
11.  * GrammarSolver facilitates the manipulation of a grammar. Given rules in
12.  * Backus-Naur Form the class will allow the user to preform certain tasks,
13.  * notably generating elements of the grammar.
14.  */
15. public class GrammarSolver {
16.     /**
17.      * Represents all of the grammar rules.
18.      */
19.     private SortedMap<String, List<String[]>> grammar;
20.
21.     /**
22.      * Constructs a new GrammarSolver given a list containing strings that
23.      * represent the grammar rules in the BNF format. Throws an exception if the
24.      * passed passed grammar is empty or if it contains more than one entry for
25.      * the same non-terminal.
26.      *
27.      * @param grammar List of strings containing grammar rules in BNF format.
28.      * @throws IllegalArgumentException Thrown if passed grammar is empty.
29.      * @throws IllegalArgumentException Thrown if the passed grammar contains
30.      *                                  more than one entry for the same non-
31.      *                                  terminal.
32.      */
33.     public GrammarSolver(List<String> grammar) {
34.         if (grammar.isEmpty()) {
35.             throw new IllegalArgumentException("Grammar is empty.");
36.         }
37.
38.         this.grammar = new TreeMap<String, List<String[]>>();
39.
40.         for (String s : grammar) {
41.             String[] symbol = s.split("::=");
42.
43.             String nonTerminal = symbol[0];
44.             if (this.grammarContains(nonTerminal)) {
45.                 throw new IllegalArgumentException("Grammar contains " +
46.                         "more than one entry for the same non-terminal.");
47.             }
```

```
48.
49.             String[] rules = symbol[1].split("\\|");
50.
51.             List<String[]> ruleList = new LinkedList<String[]>();
52.             for (String rule : rules) {
53.                 ruleList.add(rule.trim().split("\\s+"));
54.             }
55.             this.grammar.put(nonTerminal, ruleList);
56.         }
57.     }
58.
59.     /**
60.      * Returns whether or not the passed symbol is a non-terminal part of the
61.      * current grammar.
62.      *
63.      * @param symbol String to be checked as a valid non-terminal in the grammar.
64.      * @return Returns true if the symbol is a non-terminal of the grammar,
65.      * returns false otherwise.
66.      */
67.     public boolean grammarContains(String symbol) {
68.         return grammar.containsKey(symbol);
69.     }
70.
71.     /**
72.      * Randomly generates the given number of occurrences of a given symbol.
73.      * For any given non-terminal symbol, each rule has an equal probability of
74.      * being applied. Throws an exception if the passed symbol is not part of
75.      * the grammar or if the passed number of times is less than 0.
76.      *
77.      * @param symbol Symbol whose rules will be used to generate the output
78.      *               strings.
79.      * @param times  Number of occurrences of the symbol to generate.
80.      * @return Returns an array of strings that conform to the rules within the
81.      * grammar for the passed symbol.
82.      * @throws IllegalArgumentException Thrown if the grammar does not contain
83.      *                                  the passed non-terminal symbol.
84.      * @throws IllegalArgumentException Thrown if passed number of times to
85.      *                                  generate symbol is less than 0.
86.      */
87.     public String[] generate(String symbol, int times) {
88.         if (!grammarContains(symbol)) {
89.             throw new IllegalArgumentException("Symbol not present in " +
90.                     "grammar.");
91.         }
92.         if (times < 0) {
93.             throw new IllegalArgumentException("Times cannot be less than 0.");
94.         }
95.
96.         String[] out = new String[times];
97.         Random random = new Random();
98.         for (int i = 0; i < times; i++) {
99.             String terminal = getTerminal(symbol, random);
100.            out[i] = terminal.substring(0, terminal.length() - 1);
101.        }
102.
103.        return out;
104.    }
105.
106.    /**
107.     * Private helper that will generate a string given a non-terminal symbol.
108.     *
109.     * @param symbol Non-terminal to generate string for.
110.     * @param random Random object used to pick random rule.
111.     * @return Returns a terminated string from the non-terminal.
112.     */
113.    private String getTerminal(String symbol, Random random) {
114.        if (!grammarContains(symbol)) {
115.            return symbol + " ";
116.        } else {
117.            List<String[]> ruleList = grammar.get(symbol);
118.            String[] rules = ruleList.get(random.nextInt(ruleList.size()));
119.
120.            String out = "";
121.            for (String rule : rules) {
122.                out += getTerminal(rule, random);
123.            }
124.            return out;
125.        }
126.    }
127.
```

```
128.     /**
129.      * Returns a string representation of the non-terminal symbols contained in
130.      * the grammar.
131.      *
132.      * @return Returns all available non-terminal symbols as a sorted
133.      * comma-separated list enclosed in square brackets.
134.      */
135.     public String getSymbols() {
136.         return grammar.keySet().toString();
137.     }
138. }
139.
```