```java
import java.util.*;

/**
 * This class transforms Grammar rules into Random sentences.
 *
 *  It first read in the rule file, extract data into a map
 *  with non_terminal as key, and values as associated rules
 *  for each non_terminal.
 *
 *  Upon receiving a symbol, the object pre-process the symbol,
 *  strip it from non-rule characters, then compares it with
 *  the set of key to generate the random sentence.
 *
 *  User can generate sentence without the '<' and '>' brackets
 * <ul>
 * <li> Name: GrammarSolver.java
 * <li> Description: Grammar Solver
 * <li> Class: Java 145
 * <li> Instructor: Ken Hang && Janet Ash
 * <li> Date: March 1 2015
 * </ul>
 * @author  Hai H Nguyen (Bill)
 * @version Winter 2015
 */
public class GrammarSolver {
    private static final String TERM_SEPARATOR = "::=";

    private static final String NON_TERM_REGEX = "(<|>)+";

    private static final String RULE_REGEX = "(\\|\\B)|(\\|\\b)+";

    private static final String TOKEN_REGEX = "[ \t]+";

    private Map<String, String[]> grammarRulesMap = new TreeMap<String, String[]>();

    /**
     * Constructor, given a rule set, does the Following:
     * <ul>
     * <li> If rule set is null or empty, throw IllegalArgumentException
     * <li> If the grammar definition is duplicated, throw IllegalArgumentException
     * <li> Initializes a new grammar solver
     * <ul/>
     * @param rules                         BNF grammar rules
     * @throws IllegalArgumentException     If rules List is Empty or Duplicate Found
     */
    public GrammarSolver(List<String> rules) {
        if (rules == null || rules.isEmpty()) {
            throw new IllegalArgumentException("List is Empty!");
        } else {
            for(String str : rules){
                extractGrammarRule(str);
            }
        }
    }

    /**
     * @param rule                          A line from the Grammar file
     * @throws IllegalArgumentException     If Duplicate Found
     */
    private void extractGrammarRule(String rule){
        String[] data = rule.split(TERM_SEPARATOR);

        String key = bracketedSymbol(data[0]);

        String[] value = data[1].replaceFirst("^\\|", "").trim().split(RULE_REGEX);

        if(contains(key)){
            throw new IllegalArgumentException("Duplicated Non-terminal!");
```

```
69          } else {
70              grammarRulesMap.put(key, value);
71          }
72      }
73
74      /**
75       * Check if a Symbol is Non-terminal or not.
76       * @param symbol                        Symbol to be check
77       * @return                              True if symbol is a non-terminal
78       * @throws IllegalArgumentException     If Symbol is empty
79       */
80      public boolean contains (String symbol) {
81          if (symbol.isEmpty()) {
82              throw new IllegalArgumentException("Symbol is Empty!");
83          } else {
84              return getSymbols().contains(bracketedSymbol(symbol));
85          }
86      }
87
88      /**
89       * @return              A set of Non-terminals from the maps
90       */
91      public Set<String> getSymbols() {
92          return grammarRulesMap.keySet();
93      }
94
95      /**
96       * Generate Sentences with the given Non-terminal
97       * @param symbol        Non-terminal to look for rules
98       * @return              A Random sentence
99       */
100     public String generate(String symbol){
101         if (contains(symbol)){
102             // Get the Rules associated with the symbol
103             String[] values = grammarRulesMap.get(bracketedSymbol(symbol));
104             // Extract one random Rule and Split it into symbols
105             String[] symbols = values[randomIndex(values.length)].trim().split(TOKEN_REGEX);
106             // Initialize output using the first symbol
107             String out = generate(symbols[0]);
108             // Implement the sentence, treating each symbols as non-terminal
109             for (int i = 1; i < symbols.length; ++i ) {
110                 out += " " + generate(symbols[i]);
111             }
112             // Return the sentence
113             return out;
114         } else { // If it is a Terminal, Return it directly
115             return symbol;
116         }
117     }
118
119     /**
120      * @param symbol    Symbol to be Bracketed
121      * @return          Symbol with brackets
122      */
123     private String bracketedSymbol(String symbol){
124         return "<" + simplifiedSymbol(symbol) + ">";
125     }
126
127     /**
128      * @param symbol    Symbol to be simplified
129      * @return          Symbol without non-words
130      */
131     private String simplifiedSymbol(String symbol){
132         return symbol.replaceAll(NON_TERM_REGEX, "").trim();
133     }
134
135     /**
136      * @param bound     Upper bound Exclusive bound
```

```
137      * @return          Random Index within the bound
138      */
139     private int randomIndex (int bound){
140         Random randomVault = new Random();
141 
142         return randomVault.nextInt(bound);
143     }
144 } // IS29
```

```
act  ::= hero verb | verb | hero verb hero symb | hero verb trea symb
verb ::= die | attack | poison| tackle | swing | shoot| adj verb | got | acquired
adj  ::= green | red | purple | powerful | soulful | beautiful
hero ::= hermit | warrior | pikachu | kratos | Layla | Adam | Jojo | adj hero
symb ::= !|?!| !!! | ??
trea ::= ruby | saphire | topaz | lama | adj weap
weap ::= sword | spear | soul | book | hero
plac ::= Egypt | London | Shibuya | Hanoi | Shanghai | New York
vehi ::= car | adj vehi | bike | cloud | plane
```