```java
import java.util.*;

/**
 * This class transforms Grammar rules into Random sentences.
 *
 * <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
     */
    public GrammarSolver(List<String> rules) {
        if (rules == null || rules.isEmpty()) {
            throw new IllegalArgumentException("List is Empty!");
        } else {
            for(String str : rules){
                extractGrammarRule(str);
            }
        }
    }

    /**
     * Helper method used to extract data from rule string
     * @param rule          The Rule String to be Extracted
     */
    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!");
        } else {
            grammarRulesMap.put(key, value);
        }
    }

    /**
     * Check if a Symbol is non-terminal or not.
     * @param symbol        Symbol to be check
     * @return              True if symbol is a non-terminal, False otherwise
```

```
     */
    public boolean contains (String symbol) {
        if (symbol.isEmpty()) {
            throw new IllegalArgumentException("Symbol is Empty!");
        } else {
            return getSymbols().contains(bracketedSymbol(symbol));
        }
    }

    /**
     * Surround Symbols inside Alligator brackets
     * @param symbol        Symbol to be Formatted
     * @return              A Formatted symbol surrounded by '<' and '>'
     */
    private String bracketedSymbol(String symbol){
        return "<" + simplifiedSymbol(symbol) + ">";
    }

    /**
     * Simplify Symbols, pretty handy
     * @param symbol        Symbol to be Simplified
     * @return              A Simplified symbol without '<' or '>'
     */
    private String simplifiedSymbol(String symbol){
        return symbol.replaceAll(NON_TERM_REGEX, "").trim();
    }

    /**
     * @return              A set of Symbols from the maps
     */
    public Set<String> getSymbols() {
        return grammarRulesMap.keySet();
    }

    /**
     * @param bound        The Upper bound of the Random
     * @return              A random int within the bound
     */
    private int randomIndex (int bound){
        Random randomVault = new Random();

        return randomVault.nextInt(bound);
    }

    /**
     * Generate Sentences from the Symbol which indicates the flavor
     * @param symbol        Symbol indicating sentence flavor
     * @return              A random sentence
     */
    public String generate(String symbol){
        if (contains(symbol)){
            String[] values = grammarRulesMap.get(bracketedSymbol(symbol));

            String[] symbols = values[randomIndex(values.length)].trim().split(TOKEN_REGEX);

            String out = generate(symbols[0]);

            for (int i = 1; i < symbols.length; ++i ) {
                out += " " + generate(symbols[i]);
            }

            return out;
        } else {
            return symbol;
        }
    }
} // IS29
```