Homework Turnin

Name:	Akshit K Patel
Email:	akshit@uw.edu
Student ID:	1561387
Section:	DC
Course:	CSE 143 16au
Assignment:	a5
Receipt ID:	6c965cc11fb31d3c85d5257401d21fef

Replacing prior submission from Tue 2016/11/08 02:11am.

Turnin Successful!

The following file(s) were received:

Grammar.java (7044 bytes)

```java
import java.util.*;//maps, lists, arrays & Random.

/**
 * @author Akshit Patel
 * @Date 11/5/2016 CSE 143D DC
 * TA: Melissa Medsker
 * HW #5 Grammar
 *
 * The Grammar class creates random strings of text with equal probability given
 * a set of grammar rules with syntax in Backus-Naur Form (BNF) to describe what
 * constitutes a valid sentence.
 */
public class Grammar {
    /*
     * map of the BNF grammar of the file provided.
     */
    private Map<String, List<String>> bnfRuleMap;
    /*
     * Random number for getRandom.
     */
    private Random num;

    /**
     * Constructs a new grammar over a given list of BNF rules. The new grammar
     * has Nonterminals with given terminals and similar NonTerminals are
     * concatenated to the one already initialized.
     *
     * @param rules List of strings representing the BNF rule each corresponding
     * one line of text.
     * @throws IllegalArgumentException if list is null or empty.
     */
    public Grammar(List<String> rules) {
        if (rules == null || rules.isEmpty()) {
            throw new IllegalArgumentException();
        }
        // initialize map for BNF rules.
        this.bnfRuleMap = new TreeMap<String, List<String>>();
        this.num = new Random();
        // process each BNF rule into the map.
        for (int i = 0; i < rules.size(); i++) {
            String rule = rules.get(i);
            // Separate the non-terminal from terminal.
            String[] pieces = rule.split("::=");
            String nonTerminal = pieces[0].trim();
            // Separate different terminals.
            String[] terminals = pieces[1].split("\\|");
            List<String> valueTerminals = new ArrayList<String>();
            valueTerminals = Arrays.asList(terminals);
            if (!this.bnfRuleMap.containsKey(nonTerminal)) {
                this.bnfRuleMap.put(nonTerminal, new ArrayList<String>());
            }
            this.bnfRuleMap.get(nonTerminal).addAll(valueTerminals);
        }
    }

    /**
     * Returns boolean true when a given String is a Nonterminal in the grammar
     * made from the BNF rules provided. The given string check is case
     * sensitive and assumed to have no leading or trailing spaces.
     *
     * <p>
     * Terminal symbols are the elementary symbols of the language defined by a
     * formal grammar. Nonterminal is a symbol from which all the strings in the
     * language may be derived by successive applications of the production
     * rules (terminals).
     * </p>
     *
     * @param symbol String representation of the possible Nonterminal.
     * @return true if given symbol is a Nonterminal false otherwise.
     * @throws IllegalArgumentException if the given string is null or empty.
     */
    public boolean isNonTerminal(String symbol) {
        if (symbol == null || symbol.isEmpty()) {
            throw new IllegalArgumentException();
        }
        return this.bnfRuleMap.containsKey(symbol);
    }

    /**
     * Returns String representation of Nonterminals in alphabetical order from
     * the grammar made using the BNF rules given.
     */
    @Override
    public String toString() {
        return this.bnfRuleMap.keySet().toString();
    }

    /**
     * Returns a random String of text of a given Nonterminal that follows the
     * BNF rules of the grammar created. Random in a sense that the string is
     * made up of terminals chosen in random order when the terminals are
     * separated with a vertical slash(|).
     *
     * @param nonterminal String representation of the Nonterminal in the
     * grammar made.(case-sensitive)
     * @return a random String of text of the given Nonterminal following the
     * rules of grammar.
     * @throws IllegalArgumentException if given string is null or not a
     * Nonterminal.
     */
    public String getRandom(String nonterminal) {
        // check for exception.
        this.errorCheck(nonterminal, 1);
        return this.randomSentence(nonterminal);
    }

    /**
     * Returns a list of strings when given the total number of random strings
     * of text to be created for a given Nonterminal that follows the BNF rules
     * of the grammar made. Length of the list is the total number given.
     *
     * @param number int representation of the total number of random strings of
     * text of the given non-terminal to be created.
     * @param nonterminal String representation of the Nonterminal in the
     * grammar made.
     * @return random string of text of the given nonterminal as a list of
     * strings with each string of text in new index position.
     * @throws IllegalArgumentException if given string is null or not a
     * non-terminal or the given number is negative.
     */
    public List<String> getRandom(int number, String nonterminal) {
        this.errorCheck(nonterminal, number);
        List<String> sentences = new ArrayList<String>();
        for (int i = number; i > 0; i--) {
            sentences.add(this.randomSentence(nonterminal));
        }
        return sentences;
    }

    /**
     * Throws exception when the given string is null or not a Nonterminal or
     * the given number is less than 0.
     *
     * @param nonterminal String representation of the Nonterminal in the
     * grammar.
     * @param number int representation of the total number of random
     * occurrences of the given Nonterminal.
     */
    private void errorCheck(String nonterminal, int number) {
        if (nonterminal == null || !isNonTerminal(nonterminal) || number < 0) {
            throw new IllegalArgumentException();
        }
    }
    /**
     * This method creates a random string of text of a given non-terminal that
     * follows the BNF rules of the grammar created.
     *
     * @param nonterminal String representation of the non-terminal in the
     * grammar.
     * @return a random occurrence of the given nonterminal as a string.
     */
    private String randomSentence(String nonTerminal) {
        // base case: given string is terminal only.
        if (!isNonTerminal(nonTerminal)) {
            return nonTerminal;
        }
        String sentence = "";
        List<String> terminals = this.bnfRuleMap.get(nonTerminal);
        // get a random index position from the possible values.
        int randomNum = this.num.nextInt(terminals.size());
        // get the string from the random index position.
        String terminalValue = terminals.get(randomNum).trim();
        // split the string in to array if spaces in between string.
        String[] terminalSplit = terminalValue.split("\\s+");
        // process the individual strings from the array created.
        for (int i = 0; i< terminalSplit.length; i++){
            sentence += randomSentence(terminalSplit[i]) + " ";
        }
        return sentence.trim();
    }
}
```

grammar.txt (1562 bytes)

```
<ClassGenerator>::= <courseDetails> <Course> <difficulty> <Availability>
<CourseIntro>::= <name> <CourseNum>
<CourseNum>::= <num> <num> <num> | <num> <num>
<num>::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
<subject>::= Turntablism | Raptor Natural History, Conservation, and Captive Management
<subject>::= Elvish | Circus Arts | PHYSICS | Computer Science | Chemistry | Zombification
<courseDetails>::= SLN ( <num> <num> <num> <num> <num> ) <CourseIntro> | <CourseIntro>
<Course>::= <CourseType> <ver> <CourseName>  | <CourseName> <ver> <subject>
<CourseType>::= Intro | Introduction| Knowledge | <subject>
<ver>::= to | by | of | in
<CourseName>::= Zombies | Data Abstraction | Organic Chemistry | Mechnaincal Failure | POKEMON STUDIES
<CourseName>::= Speed Of LIGHT | Shooting Ghosts | Composition | Street Fighting Mathematics | The meaning of life | POKEMON
<name>::= <Arts> | <sciences> | <engr> | <Built> | <enviornment> | <information> | <buisness>
<Arts>::= AES | AIS | ANTH | ART H | AS ST | LANG | COMM | DANCE | DRAMA | GEN ST | MUSIC | HIS | POL
<sciences>::= PHYS | CHEM | CSE | POL SC | BIO | MED | M SCI | STAT
<engr>::= M E | EE | CE | Mol E | BIO ENG | IND ENGR | HCDE | MSE | CIVIL
<Built>::= ARCH | LAND |COMM ENV | CM | ARCH D
<enviornment>::= FISH | ATMOS SCI | BioRES SCI| ENVR
<information>::= INFO
<buisness>::= ECON | BA | ADMIN | ACCTG | HRMOB | FIN
<difficulty>::= (COMPTETIVE) | (OPEN) | (MIXED) | (MINIMUM)
<Availability>::= Freshmen & Sophomore only | Junior only | Everyone | Senior only| Junior & Senior only
```