```java
package proj5;

/**
 * Data structure that holds words and their associated synonyms.
 * You can look up a word and retrieve a synonym for it.
 * author: Son Nguyen (Kyrie)
 * version: 6/3/2020
 */
public class Thesaurus {

    // empty string array constant
    private String[] EMPTY = new String[]{};

    // instance variable
    private LineReader thesaurusReader;
    private String[] currentLine;
    private String currentKeyword;
    private String[] currentList;
    private BinarySearchTree<SynonymsList> myThesaurus;

    /**
     * Default constructor. Creates an empty thesaurus.
     */
    public Thesaurus() {
        myThesaurus = new BinarySearchTree<SynonymsList>();
    }

    /**
     * Builds a thesaurus from a text file.
     * Each line of the text file is a comma-separated list of synonymous words.
     * The first word in each line should be the thesaurus entry.
     * The remaining words on that line are the list of synonyms for the entry
     * @param file path to comma-delimited text file
     */
    public Thesaurus(String file) {
        thesaurusReader = new LineReader(file, ",");
        currentLine = thesaurusReader.getNextLine();
        myThesaurus = new BinarySearchTree<SynonymsList>();
        buildThesaurus();
    }

    /**
     * helper method to construct the non-default thesaurus
     */
    private void buildThesaurus() {
        while (currentLine != null) {
            currentList = new String[currentLine.length - 1];
            currentKeyword = currentLine[0];
            for(int i = 1; i < currentLine.length; i++) {
                currentList[i - 1] = currentLine[i];
            }
            myThesaurus.insert(new SynonymsList(currentKeyword, currentList));
            currentLine = thesaurusReader.getNextLine();
        }
        thesaurusReader.close();
    }

    /**
     * removes entry (and its associated synonym list) from this thesaurus.
     * If entry does not exist, do nothing.
     * @param entry word to remove
     */
    public void delete(String entry) {
        myThesaurus.delete(new SynonymsList(entry, EMPTY));
    }

    /**
     * Gets a random synonym for the given keyword.
     * If keyword does not exist, return the empty string.
     * @param keyword word to find a synonym for
     * @return a random synonym from the synonym list
```

```java
 of that word,
     * or empty string if keyword doesn't exist.
     */
    public String getSynonymFor(String keyword) {
        SynonymsList database = myThesaurus.search(new SynonymsList(keyword, EMPTY));
        if (database != null) {
            return database.getSynonym().toString();
        }
        return "";
    }

    /**
     * inserts entry and synonyms into thesaurus. If entry does not exist, it creates one.
     * If it does exist, it adds the given synonyms to the entry's synonym list
     * @param entry keyword to be added
     * @param syns array of synonyms for keyword entry
     */
    public void insert(String entry, String[] syns) {
        SynonymsList toInsert = new SynonymsList(entry, syns);
        SynonymsList counterpart = myThesaurus.search(toInsert);
        if (counterpart == null) {
            myThesaurus.insert(toInsert);
        }
        else {
            for (String synonym: syns) {
                counterpart.add(synonym);
            }
        }
    }

    /**
     * @return this thesaurus as a printable string.
     * Each keyword and synonym list should be on its own line. T
     * he format of each line is: <keyword> - {<syn1>, <syn2>, ..., <synN>}
     * For example,
     * happy - {glad, content, joyful}
     * jump - {leap, bound}
     *
     * The thesaurus keywords will be in alphabetical order.
     * The order of the synonym list words is arbitrary.
     */
    public String toString() {
        return myThesaurus.toString();
    }
}

```