

Lab 3: Word Generator

DUE: Monday, 27 Feb, 9:00am

Lab Preparation

Always read each lab handout all the way through *at least once* in preparation for lab. Also try to sketch out an approach you will take to the problem. This will save you considerable time in lab and make lab much more productive for you.

To turn in for Wed: Short Answer Problems (SC stands for Self Check)

Take a look at the following questions (nothing to turn in): SC 3.2—SC 3.5.

Complete the following problems from the book and use the `turnin` program to turn them in as a file `lab3.txt` by the **start** of lab on Wed: SC 3.7, SC 3.10, 3.6 (not the self check, the one on page 65)—DON'T WRITE the class, just answer the question.

Lab Description

This week, you will be developing an implementation of the word generator program described below.

This week's lab can be thought of as an experiment in Artificial Intelligence. The idea is to write a program which will read in text and then use that text to generate some new text. The method for generating the text uses simple probability—we read the text character by character and keep track of how often each three-character sequence appears. From this, we can compute the probability that a certain character will immediately follow two given characters. For example, if the text is: “the theatre is their thing”, e appears after th 3 times, and i appears after th 1 time; no other letters appear after th. So the probability that e follows th is .75; the probability that i follows th is .25; the probability that any other letter follows th is 0.

Once we have the text processed and stored in a structure that allows us to check probabilities, we then pick two letters (for example, the first two in the input text) to use as a beginning for our new text. Then we choose subsequent characters based on the preceding two characters and the probability information.

During this lab, you will become proficient in using existing structures to build more complex ones, you will learn the importance of careful class design, and you will learn about the Vector and Association classes in the structure package.

Important Considerations

You should think about the design of this program carefully before entering the lab. What would constitute a good data structure for this problem? The table of information should support requests of the form:

- update the probabilities in your table, given a new triple of characters.
- select a next character, given a pair of characters and the probabilities stored in your table.

A three-dimensional array might seem reasonable at first, but its size would be quite large (approximately 27,000 entries if you include blanks and punctuation). Instead, develop a `Table` class which is implemented as a `Vector` of `Associations`. Each `Association` would have a 2-character pair (stored as a `String`) as its key, along with a value which is a frequency list. The frequency list would keep track of which characters appeared after the given 2-character pair, along with a frequency.

Each frequency list should be an object of another class, `FrequencyList`, that you will define. There are many ways to implement the frequency list. A good possibility is . . . another `Vector` of `Associations`. Thus the frequency list's `Vector` would consist of `Associations` in which the key is a single character (stored either as a `Character` or a `String`) and the value is a count of the number of times that letter occurred after the pair with which the list is associated (stored as an `Integer`). Think carefully about what methods the frequency list needs to support and any other instance variables that might be useful.

The data structure design built from these two classes has the benefit of having only as many entries as necessary for the given input text. You will find it helpful to look carefully at the word frequency program on page 48 of Bailey.

Your `main` method for the program should be written in a third class, `WordGen`, which reads the input text, builds the table, and prints out a randomly-generated string based on the character sequence probabilities from the input text. All I/O (input and output) should happen in this method.

Warning: When you import the package with the classes `Random` and `Scanner`, use

```
import java.util.Random;
import java.util.Scanner;
```

If you write `import java.util.*;`, the program will get confused as to which version of the `Vector` class it should use as there is one in `java.util` as well as one in `structure`. To ensure that your program can find the `structure` package for the implementations of `Vector` and `Association`, you will need to add this to the top of your Java files:

```
import structure.*
```

Input First debug your program using as input a `String` constant (e.g., “the theater is their thing”) until it works properly. I recommend creating a method which will print out your data structure so you can see that it is indeed working. After it works, you can take input from the keyboard using the `Scanner` class. To use the `Scanner`, you will want to build up a string of the entire input line by line, using the methods `hasNextLine()` (c.f. `SudokuVerifier.java` example) to find out if there is another line of input ready, and if there is, to read it with the method `nextLine()`. End of the input is signalled for Java on the Mac (and, indeed, on any Unix system) by typing “Control-D” on a new line.

You may also read your input from any text file, e.g. “whosonfirst.txt”, with this command:

```
java WordGen < whosonfirst.txt
```

Output After the input has been processed you should generate and print out new text using the frequencies in the table. You may start with a fixed pair of letters that appears in the table or choose starting characters randomly. Generate and print randomly-generated text up to about 2000 letters (about a screenful in a terminal window) so that we can see how your program works.

Submitting Your Work

Create a gzipped tar file of the directory which contains you lab 3 files, named with your userid followed by “lab3.tar.gz”, i.e. `00skw_lab3.tar.gz`, and submit using the `turnin` utility. As in all labs, you will be graded on design, documentation, style, and correctness. Be sure to document your program with appropriate Javadoc comments, including a general description at the top of the file, a description of each method with pre- and post-conditions, and `@return` tags where appropriate.