Run this program with jGRASP. There should be no installation or setup necessary. Included are the files Dictionary.txt, Dracula.txt, Frank.txt, and Seuss.txt, all of which can be used for sentence generation. All the generators can be run using these files, but you may want to make your own for fun. The program GeneratorClient is the only one that has a main method, so run it to interact with the program. The program guides you through how to use it, so it should be completely self-explanatory, but remember that anything that starts with a Y or y means “yes” or “continue”, while anything else means no. GeneratorClient directly interacts with four classes, SentenceGenerator, WordGenerator, MarkovGenerator, and SyntaxTreeGenerator. SentenceGenerator and WordGenerator are very basic programs, which pick complete random sentences from the text and compile random words from the text respectively. I made these two for comparison, to see how far the sentences the other two generated were from English or random words. SentenceGenerator, WordGenerator, and MarkovGenerator take a “written work” such as a song, a novel, or a poem as input. Use these with Dracula.txt, Frank.txt, or Seuss.txt, or any similar thing. SyntaxTreeGenerator only works with dictionaries. If the file is not formatted like a dictionary, it will not let you generate sentences.

You should look at the code in this order: GeneratorClient to get a sense of how the Generators are used and what they generally do (Main method is important, getWords/getSentences/getDictionary less important), then Generator.java to get a better sense of how Generators work (Generator superclass – note getSentence method!). Then look at how the generators work individually. If you’d like, look at WordGenerator and SentenceGenerator, but they’re very simple, and not important. Then look at SyntaxTreeGenerator. Finally, look at MarkovTree, then MarkovGenerator.

Some good tests to do are running it with small files or files with strange input that might cause errors. I think I protected from throwing exceptions in most cases, but I might have missed some. You also might want to try a very large file, but it will probably take too long loading to ever run into an error. Also, just try to give it wrong input to break it. More fun (but less error-finding) tests involve creating your own dictionary to make different sentences with a SyntaxTreeGenerator, or putting in poems or lyrics (try rap lyrics!) to a MarkovGenerator.

How to make your own Syntax Tree Dictionary:

Make a .txt file that contains a set of words, labeled with their part of speech. They don’t have to be organized, but you have to make sure you use the correct label for each part of speech and have at least one of each.

Parts of speech needed: subordinate conjunctions (SUBC), conjunctions (CONJ), Transitive verbs (TRAN), intransitive verbs (VERB), nouns (NOUN), adjectives (ADJE), adverbs (ADVE), and prepositions (PREP). Format the file the same as the Dictionary file:

“PART word”

GRADING RUBRIC (Nothing needed to change from the spec)

10 points – External Correctness

Does the program create some sort of sentence approximations (7)? Does it throw errors when it shouldn’t/is user input robust (3)? Note: This shouldn’t be graded on whether or not it makes grammatically correct sentences, it should be graded on whether or not it uses the algorithms correctly. Some of these algorithms don’t make great sentences all the time.

20 points – Internal Correctness

Does the program correctly use the algorithms specified in the program’s introduction (12)? Does it use the right data structures to store information (3)? Is there redundant code (2)? Is it efficient/Does it take longer than a few seconds to generate a sentence (3)?

10 points – Style and documentation

Is the program readable (4)? Without prior knowledge of how these algorithms work can you understand how this program works (6)?