**Gibberish**

This project will allow you to practice what you have learned about the String class.

Your application will play a word game, much like MadLibs. You will read a template string that contains marked nouns, verbs, adverbs, etc. that you will replace with random other words with the same part of speech. The supplied library defines three classes that will help you implement your gibberish generator. The first class is Template. The template class provides your Gibberish program with a template string. The second class is WordList5000. A method from that class will allow you to select a random word to put in your template. You will find the documentation for these two classes at the end of these instructions.

To begin, copy the folder, “Gibberish”, from the shared drive to your working directory. In that folder you will find a BlueJ project file, package.bluej. Open that file. BlueJ will start. You should see a BlueJ project window. The project does not contain any class files but it does contain a library that you will need to complete the project.

Step 1: Create a runner class

In BlueJ, create a new class, GibberishRunner. Define a main (or run) method in GibberishRunner, just as you have done in other labs and projects. The body of the main method should contain four statements:

1. Define a local variable, wordList5000, and initialize it with a new object by calling the WordList5000 constructor.

2. Define a local variable, gibberish, and initialize it in a similar way, using the Gibberish contstructor.

3. Create your gibberish string by calling the method, makeGibberish, in the Gibberish class. You will write the Gibberish class, including this method, in Step 2. The signature for makeGibberish is:

*public String makeGibberish(WordList5000 wordlist5000)*

4. Using System.out.println, print out the string returned by makeGibberish.

Step 2: Create the Gibberish class

In BlueJ, create the new class, Gibberish. Define one instance variable, sourceString, of type String. Define a contstructor for Gibberish that takes no arguments. The body of the constructor will initialize the sourceString instance variable, using the following two statements:

1. Define a local variable, template, and initialize it with a new object by calling the Template constructor.

2. Initialize the instance variable, sourceString, by calling the method in the Template class, getSourceString.

Step 3: Implement the makeGibberish method

The method, makeGibberish, performs all the interesting work. Remember from above it takes one parameter and returns a string result:

*public String makeGibberish(WordList5000 wordlist5000)*

makeGibberish will create the result by scanning through the sourceString looking for places to replace placeholders in the template with random words. Here is an example of a placeholder:

<t>find</>

The placeholder begins with a left angled bracket. After that comes a character that defines the part of speech of the placeholder. In the example above, t indicates the placeholder is a transitive verb. Next comes a right angled bracket. After that there is arbitrary text until you reach the placeholder termination string, “</>” Your goal with makeGibberish is to find all of these placeholder strings and replace them with random words that match the specified part of speech. Here is a guide to the implementation of makeGibberish:

1. Define two local variables. The first, currentSourceIndex, keeps track of how much of the source string you have scanned, so far. Initialize it to 0. The second, resultString, contains the gibberish string the method constructs.

2. Create an indefinite loop (a while loop) that executes the statements in its body until the entire the entire sourceString has been scanned. You will use currentSourceIndex to keep track of how much of the source string has been processed. As long as it is less than the length of the source string, there is more work to do. Here is a guide to the body of the loop:

a. Search for the index of the next occurrence of “<” after currentSourceIndex. (Use the String method, indexOf).

b. If no “<” is found then concatenate what remains of the source string to the resultString (Use the String substring method and the + operator. Set currentIndex to the length of sourceString to indicate it is time to exit the while loop.

c. Otherwise (i.e. a “<” was found):

i. Find the part of speech string. You already know where it starts – at the index 1 greater than the index for “<”. It ends at 1 less than the next occurrence of “>”. Search for that “>” and then declare a local variable, placeHolderString, to hold the placeholder. Initialize it using the substring method of String.

ii. Append a random word to resultString. Use the method, getRandomWordString, from the WordList5000 class.

*Be careful! getRandomWordString may return null. Check for that before you append to resultString. If you don’t you may get a NullPointerError!*

iii. Skip over the rest of the placeholder. To do this, search for the string, “</>”. Update currentSourceIndex to the index for the character after that placeholder terminating string.

3. Return resultString

Compile and run your program!