CSc 227 — Program Design and Development
Spring 2014 (McCann)

http://www.cs.arizona.edu/classes/cs227/spring14/

Program #3: The WordHelper Class

Due Date: February 11th, 2014, at 9:00 p.m. MST

Overview: The English language is generally acknowledged to be one of the hardest languages to learn to speak. Nearly all rules of pronunciation work for some words, but not others. Even determining where a word divides into syllables is difficult. For this assignment, you will be writing a Java class whose instances each manage one word, including determining syllables and pronunciation.

Assignment: You will create a new Java class called `WordHelper` in a file named `WordHelper.java`. Each `WordHelper` object will have instance variables that hold state information about the word, such as a reference to a `String` object holding the word. Exactly which and how many instance variables you'll need will depend on how you choose to implement the constructor and other methods of the class.

`WordHelper` will have one constructor:

- `public WordHelper (String)` — sets the state of the new object to match that of the word referenced by the parameter, in lower-case. Thus, if the given word is 'CompoundWord,' the object will represent the state of the word 'compoundword'. If the parameter includes non-letters, retain only the letters.

Your `WordHelper` class will also contain implementations of the following instance methods. Details about how to implement some them follow this list:

- `public String getWord ()` — returns a reference to a **copy** of the object's current word.
- `public void setWord (String)` — resets the state of the object to correspond to this new word.
- `public int numberOfSyllables ()` — returns an estimate of the number of syllables in the word.
- `public String syllablize ()` — examines the word, applies some very simple heuristics to try to identify the syllables within the word, and returns a `String` reference to a 'divided' version of the word. For example, if the word is "identify", the returned string will be "i/den/ti/fy", including the forward slashes but without the quotes. Our crude rules for identifying syllables are given below.
- `public String pronounce ()` — examines the 'syllablized' version of the word, one syllable at a time, and returns a string representing its phonetic equivalent. For example, if the original word was written as "ma/jor", this method would return "may–jor"; note that the slashes are replaced with hyphens. Our rules for phonetic conversions are given below.

  `pronounce()` should call a private instance method called `translateSyllable()` (see next description) to phonetically translate each syllable.
- `private String translateSyllable (String)` — given a syllable, it determines and returns the phonetic version of that syllable. The rules for performing the conversion are provided below.

(Continued...)

The implementation details:

1. *How to approximately count syllables:* Syllables have to have at least one vowel (a,e,i,o,u, and y, for our purposes). But, many words have pairs of vowels that represent just one vowel sound, such as the 'ou' in "sound". Also, many words end in a silent 'e', such as the word "before". With that in mind, here's how you are to estimate the number of syllables in a word: Count the vowels. Subtract one from that count for each pair of vowels in the word. If the word ends with an 'e', subtract one more. The remaining amount is the estimate of the number of syllables in the word. For example, "instantiable" has 5 vowels, including one pair of vowels and a trailing 'e'. We'd estimate its number of syllables to be 5 - 1 - 1 = 3. That's not the actual number of syllables, but that's what your class' estimate should be. If your estimate is less than one, return one. Words with three vowels in sequence (e.g., "quail") have two pair: "ua" and "ai".

2. *How to identify syllables:* We'll confine ourselves to just two situations, commonly known as the 'vccv' and 'vcv' rules. As you scan a word from left to right, if you encounter a vowel followed by two consonants (non-vowels) followed by a vowel, divide between the consonants. For example, "hidden" contains this pattern and is divided into two syllables: 'hid/den'. If you see the 'vcv' pattern, divide before the consonant. Thus, "consonant", which exhibits both patterns, would be divided like this: 'con/so/nant'.

   Note that your approximate count could be different than the number of identified syllables. For this program, that's OK.

3. *How to perform phonetic syllable conversion:* To write `translateSyllable()`, you need to know some rules for phonetic modification. Our list is far (far!) from complete, but will suit our purposes.

   The rules you are to follow are divided into two sets:

   (a) These rules should only be used when characters appear at the **end** of syllables:
      i. If a syllable ends in 'o', replace the 'o' with "oh".
      ii. If a syllable ends in 'i', replace the 'i' with "ee".
      iii. If a syllable ends in 'a', replace the 'a' with "ay".
      iv. If a syllable ends in "ce", replace the "ce" with "ss".
      v. If a syllable ends in "es", replace the "es" with "ez".
      vi. If a syllable ends in "vy", replace the "vy" with "vee".

   (b) The second set of rules are for short substrings that could occur at any position within a syllable:
      i. If you find "cc", replace it with "k".
      ii. If you find "ca", replace it with "ka".
      iii. If you find "co", replace it with "ko".
      iv. If you find "au", replace it with "aw".
      v. If you find "ea", replace it with "ee".
      vi. If you find "qu", replace it with "kw".

   You are to apply these rules in the order given above. That is, first check the character(s) at the end of the syllable and use any rules that apply, and then check the resulting modified syllable to see if any of the rules from the second group apply. Within each group, always check the rules in the order shown.

   For example, consider the syllable "cau". We can't apply any of the 'end of syllable' rules to it, because none of them handle a syllable that ends with 'u' or "au". When we move on to the second group, we see that two of those rules apply (specifically, "ca" to "ka", and "au" to "aw"). By applying those two rules in the order listed, we change "cau" into "kau", and then change "kau" into "kaw".

(Continued...)

Data: We are giving you a sample program with which you can *begin* testing your `WordHelper` class. You can find it linked to the class web page, just below the link to this assignment. The SLs will be creating a much more comprehensive program that they will use to test your class. So should you! Use the sample as a starting point, and add tests to it to check that all of the requirements of the assignment are met by your class. Name your testing program `Prog3.java`.

The sample program `T01n19.java`, with the sample class `Fraction.java`, demonstrates that the class containing `main()` can easily be in a different file than the other classes it relies on. So long as they are stored in the same directory, when you compile the file containing `main()`, Java will find the other file(s) automatically. (And now you know why the file names and class names have to match!)

Output: Because the `WordHelper` class does not produce any output, you don't need to worry about meeting any output specifications.

Turn In: Use the 'turnin' page to electronically submit both of your files (`Prog3.java` and `WordHelper.java`) to the `cs227p03` directory at any time before the stated due date and time.

Want to Learn More?

- This site has a lot of information about *phonics*, the approach to learning language based on word components. It includes a page devoted to rules for identifying syllables:
  `http://www.phonicsontheweb.com/`
- `readspeaker.com` has a nice text-to-speech demo:
  `http://www.readspeaker.com/voice-demo/`

Hints and Reminders:

- As mentioned above, this program is asking you to do just fragments of the work that something like a text-to-speech system would have to do. Your output will often be incorrect by the norms of the English language. You'll be graded against the specifications found in this document, not against English itself.
- We have mentioned only a few of the available `String` and `StringBuilder` methods in class; you are free to use any of the methods of those classes to complete this project.
- Feel free to add additional *private* instance methods to your class, but do not add any more *public* instance methods. Do not create any `static` methods.
- It may seem that finding the "vccv" and "vcv" patterns could be quite hard. You can make it easier by first converting words to a new representation. For example, you could convert "hello" into "cvccv" before looking for the "vccv" pattern.
- As usual, we'll be looking for good documentation, indentation, variable names, etc. Please allocate enough time to do a good job with these. Because this assignment includes writing a new class with methods, you'll need to include block comments with those elements, as well as at the top of your program. Here's a web page that will help you understand what we're expecting to see:
  `http://www.cs.arizona.edu/people/mccann/styletemplates227.html`
- This project contains a lot of detail. We recommend that you (surprise!) start early.

  Related: Resist the temptation to try to do everything at once. Try to create for yourself an ordering of the methods that you have to write. For example, `pronounce` depends on `translateSyllable`, making it reasonable to code (and test!) `translateSyllable` before worrying about coding `pronounce`.