Word processing programs usually provide automatic spell-checking and spell-correcting facilities. Consider how such a tool might work. The system examines each word in a document. For each word, it attempts to find the word in a dictionary. If the word is found, the system considers the word to have been spelled correctly. If the word is not found, it may have been misspelled. In such a case, the system looks for alternative words in the dictionary that are “close”, in some sense, to the potentially misspelled word. The user is then asked to choose between (a) accepting his/her original word, (b) selecting one of the alternative words found in the dictionary, and (c) typing in a corrected version of the word.

In your lab and assignment for this week, you will write a Java program that checks and corrects the spelling of words in a text document. In the lab you will focus on the problem of finding words in a dictionary that are similar to a given, potentially misspelled word:

1. Open the SimilarWords NetBeans project provided in the files for this lab. You will begin by working with the Dictionary class. Locate and open the Dictionary.java file. This class has a constructor that reads the contents of the file: dictionary.txt (included with the files for this lab). The first line of this file indicates the number of words in the dictionary. The constructor reads this number as a string of characters, converts it into an integer (length) and uses the length initialize an array of strings (wordTable) that is large enough to hold the words in the dictionary. Read over the implementation of the Dictionary class constructor and be sure you know how it works.

2. In order to suggest alternatives to misspelled words, we need a way to determine when two words have nearly the same spelling. One way to do this is the following. Generate a second array (reducedWordTable) that will hold a simplified version of each word in the dictionary. Arrange that a misspelled word (e.g., “hersay”) and the correctly spelled word (“hearsay”) will have the same reduced form (“hrsy”). When the program identifies a misspelled word (e.g., “hersay”) it converts it to reduced form (“hrsy”) and then looks for other words with the same reduced form (e.g., “hearsay”). Now go ahead and define a private instance variable reducedWordTable to be an array of strings. Modify the Dictionary constructor to initialize reducedWordTable so that each entry in the reduced word table holds the reduced form of the corresponding entry in the original word table. Use the (as yet unwritten) String reduce(String word) to generate the reduced form of each word.

3. Define a method with signature: public static boolean isVowel(char c). This method takes a character c as its only input parameter. It returns true if c is a vowel (a, e, i, o or u) and returns false otherwise. The method should begin by defining a String constant called “VOWELS” with “aeiou” as its initial value. It should then use the String class method int

---

1 The procedure for identifying similar words used in this lab was taken from Java Elements, by Duane A. Bailey and Duane W. Bailey, McGraw-Hill 2000.
indexOf(char) to find the position at which \texttt{c} appears in \texttt{VOWELS}. If this method returns a non-negative integer, then \texttt{c} was found somewhere in \texttt{VOWELS}, and \texttt{c} must be a vowel. If the method returns the negative integer \texttt{-1}, then \texttt{c} was not found in \texttt{VOWELS}, and \texttt{c} must not be a vowel. The method \texttt{isVowel} should then return \texttt{true} or \texttt{false} accordingly.

4. Define a method with signature: \texttt{public static String reduce(String word)}. This method takes a \texttt{String} argument called “\texttt{word}” as its only input parameter. It simplifies \texttt{word} using the following rules:
   a. The input \texttt{word} is converted to lower case.
   b. If the \texttt{word} begins with a vowel, the vowel is replaced with “!”, an exclamation point.
   c. All other vowels are deleted from the \texttt{word}.
   d. Any sequence of two or more successive identical characters in the \texttt{word} is replaced with a single occurrence of the character.

The \texttt{reduce} method scans the input \texttt{word} from left to right, constructing a new, simpler version of the \texttt{word} along the way. When the scanning process is complete, \texttt{reduce} returns the simplified version of the \texttt{word}.

5. Define a method with signature: \texttt{public String similarWords(String word)}. This method should start by forming the reduced version of \texttt{word} storing it in the \texttt{String} variable \texttt{reducedWord}. It should also initialize a \texttt{String} variable (“\texttt{answer}”) to be the empty string. Next this method should iterate through the \texttt{reducedWordTable}. Each time it encounters \texttt{reducedWord} in \texttt{reducedWordTable}, the method should append the corresponding un reduced word to the growing \texttt{answer} string. After scanning the \texttt{reducedWordTable}, the method should return the \texttt{answer} string.

6. Now read through the \texttt{main} method for the \texttt{SimilarWords} class. The \texttt{main} method begins by constructing and storing an instance (“\texttt{dictionary}”) of the \texttt{Dictionary} class. It also initializes a \texttt{BufferedReader} that reads input from the keyboard. The \texttt{main} method then repeatedly prompts the user to type in a word of his/her choice. It removes white space from the beginning or end of the word. It also converts the word to lower case. Finally, it invokes \texttt{dictionary.similarWords} to obtain and display a \texttt{String} of similar words to be viewed by the user. The \texttt{main} method terminates when the user enters a word of length zero.

7. Run your program and verify that it can suggest corrections for some misspelled words.