Objectives	Use the state machine approach to write a class for counting syllables in words. For this problem you don't need to use objects for states. Test your code by counting syllables in all words in a dictionary.
What to Submit	Commit your code to Bitbucket as Lab9. You should have these classes: WordCounter and Main (count words in a file or URL) Ask TA to check your State Machine diagram on paper before 18:00.

Assignment

1. Draw a State Machine Diagram of an algorithm for counting syllables in a word.

2. Write a class name WordCounter with a method named countSyllables( ) that implements the state machine and counts syllables in a word. If something is not a word, return 0.

4. Count the words and syllables in dictionary.txt located at http://se.cpe.ku.ac.th/dictionary.txt. The file has one "word" per line, but some of them may not be actual words according to our definition. If something is not a word according to the definition below, then don't count it.

How to count syllables?

This assignment uses the same rules as the *Flesch Readability Index* (PA4) to count syllables using vowel sequences.

Count syllables as the number of *vowels sequences* in a word. A *vowel sequence* is one or more vowels that occur together. A *vowel* is a, e, i, o, u, or (sometimes) y. Here are the cases with examples:

1. Sequences of consecutive vowels count as one syllable. vowels are: `a e i o u`. y counts only if it is the **first** vowel in a vowel sequence.

`banana` = 3 vowel sequences `b(a)n(a)n(a)`

`durian` = 2 vowel sequences `d(u)r(ia)n`

`beauty` = 2 vowel sequences `b(eau)t(y)`

`layout` = 2 vowel sequences `l(a)y(ou)t`

2. A final "e" by itself is **not** counted, **unless** it is the only vowel in the word.

`apple` = 1 vowel sequence. Don't count final "e".

`love` = 1 vowel sequence. Don't count final `"e".`

`The, me, he, she, we` = 1 vowel sequence. Count the final "e" because it is the only vowel.

`movie` = 2 vowel sequences. Final "e" is part of a multi-vowel sequence, so count the sequence.

`levee` = 2 vowel sequences. Same reason as above.

3. A dash '-' in the middle of word is like a consonant.

`anti-oxidant` = 5 vowel sequences `(a)nt(i)-(o)x(i)d(a)nt`
`-oxidant` = **not a word**. Dash cannot be at start of word.
`anti-` = **not a word**. Dash cannot be at end of word.

4. **Not a word**. Any string that contains a non-letter or doesn't contain any vowels is not a word. The only exception is "-" in between letters (case 3).

```
mrtg
Java5se
anti-
I.B.M.
```

```
7-Eleven
```

5. "y" is considered a vowel if is the first vowel in a sequence, a consonant otherwise.

```
beyond = 2 vowel sequences: b(e)y(o)nd
yesterday = 3 vowel sequences: (ye)st(e)rd(a)y
Yahoo = 2 vowel sequences (Ya)h(oo)
```

6. Ignore apostrophe (').

```
isn't = isnt
student's = students' = students
```

Problem 1. Identify States and Events, Draw a State Machine Diagram

Design a state machine for counting syllables in a sequence of characters without embedded spaces.

Draw a State Machine Diagram with States, Events, and Actions taken during transition or while in a state. *See document Programming a State Machine* in class week9 folder for UML examples.

Show all possible events and transitions, even transitions back to the same state.

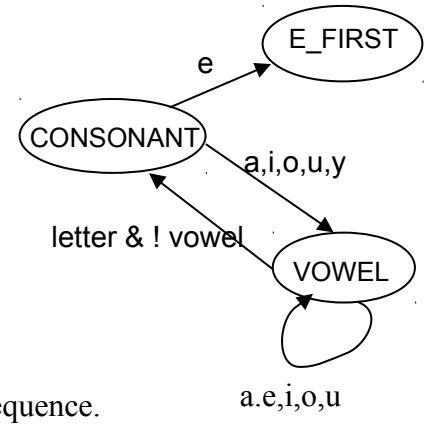
States: Some (**not all**) of the *States* are*:*

`START` = start of processing, no characters processed yet. You can use this state to skip leading white space characters.

`CONSONANT` = most recent character is a letter but not a vowel.

`E_FIRST` = most recent character was "e" and it is the first vowel after a non-vowel sequence. This is needed for the "final e" rule.

`NONWORD` = the character sequence is not a word.



Events: The *event* is reading (or processing) a character.

Actions: add 1 to the syllable count or abort processing the current char sequence.

Problem 2: Write a class to implement the State Machine

2.1 Write a WordCounter class has a method named **`countSyllables`** to count syllables in a String.

WordCounter
countSyllables(String) : int

`countSyllables` returns the number of syllables in the char sequence. If the parameter is *not a word* then return 0.

2.2 Use your state machine diagram to implement countSyllables.

Requirements:

- use the state machine approach
- only look at one character at a time: don't save the previous character and don't look-ahead at the next character.

An example of using a non-OO state machine is:

```
int countSyllables( String word ) {
    int syllables = 0;
    state = START;
    for( char c = ??? ) {  // process each character in word
        switch(state) {
            process character c using state machine
            case CONSONANT:
                if (c=='e'||c=='E') state = E_FIRST;
```

```
                else if (isVowel(c)) state = VOWEL;
                else ...
                break;
            case VOWEL:
                if (isVowel(c)) /* stay in VOWEL state */;
                else if (Character.isLetter(c)) state = CONSONANT;
                ...
    }
    // End of word. Correct syllable count for the "final e" rule.
```

The Character class has some useful methods for testing characters:

`Character.isLetter( c )` - true if c is a letter
`Character.isWhitespace( c )` - true if c is whitespace (space, tab, newline)

2.3 Ignore accidental *whitespace* before the beginning of the word. *Whitespace* means a space, tab, or newline character.

2.4 Please **don't write BAD CODE**. Don't do this:

```
int countSyllables(String word) {
    int syllables = 0;
    state = START;
    for( int k=0; k<word.length(); k++ ) {
        switch(state) {
            case VOWEL:
                if (word.charAt(k)=='y' &&
                 (word.charAt(k-1) == 'a' || word.charAt(k-1)=='e' ...
```

1. **Don't** repeatedly call `charAt(k)`. It's *inefficient* and makes the code hard to read.

2. **Don't** look ahead (next char) or look back (previous char). The state should contain all the information you need to decide what action and/or transition to perform for every possible input.

In a state machine, you don't need look-ahead or look-back.

If it appears you *do need* to look-ahead or look-back, then redefine your states or add more states to differentiate the cases.

Alternative Approach: Design an O-O style State Machine

You can use the object-oriented approach to a state machine if you want. Since reading a character is an event, each State object needs a method like `handleChar(char)`.

Problem 3: Write countWords( ) in the WordCounter class

Add methods to the WordCounter class to count all the words and syllables in an input source. We're interested in counting words/syllables in both (a) an InputStream, (b) a URL.

3.1 Write a countWords( ) method that accepts an `InputStream` as parameter and counts all the words read from the `InputStream`. It also counts syllables.

```
/**
 * Count all words read from an input stream an return the total count.
 * @param instream is the input stream to read from.
 * @return number of words in the InputStream.  -1 if the stream
 *         cannot be read.
 */
public int countWords(InputStream instream) {
```

If the caller wants to read from a File, he can open the File as a `FileInputStream` object.

3.2 Write a method that counts all words and syllables from a URL. Avoid duplicate code! Just open the URL and invoke the `countWords(InputStream)` method.

```
/**
 * TODO Write this javadoc
 */
public int countWords(URL url) {
```

Here is an example of how to create a URL and open an InputStream from a URL. The code may throw MalformedURLException and IOException, which you must handle.

```
final String DICT_URL = "http://se.cpe.ku.ac.th/dictionary.txt";
URL url = new URL( DICT_URL );
InputStream in = url.openStream( );
```

3.3 Write a separate getSyllableCount( ) method to get the syllable count from the most recent call to countWords. countWords should reset the syllable counter each time it is called.

```
final String DICT_URL = "http://se.cpe.ku.ac.th/dictionary.txt";
URL url = new URL( DICT_URL );
WordCounter counter = new WordCounter();
int wordcount = counter.countWords( url );
int syllables = counter.getSyllableCount( );
```

Problem 4: Write a Main class to count a Dictionary. And calculate elapsed time

Write a Main class that reads all the words from a URL or File.
Compute the elapsed time (use your StopWatch), and print the results on the console.
Use this URL: `http://se.cpe.ku.ac.th/dictionary.txt`

Output From Your Main Class

Use a timer to measure how long it takes your code to count the words in the dictionary.

```
Reading words from http://se.cpe.ku.ac.th/dictionary
Counted 102,000 syllables in 38,600 words
Elapsed time: 0.020 sec
```

Programming Hints

2. java.util.Scanner is slow. A faster way to read *lines* of input is BufferedReader.

2. An InputStream contains only bytes. We need characters or Strings. A Reader reads input as *characters*. InputStreamReader and BufferedReader are subclasses of Reader.

```
Reader reader = new InputStreamReader( inputStream ); // read InputStream
Reader reader = new StringReader( string );           // read String
```

You *could* write countSyllables() to accept a Reader as parameter, and just read characters until you get to the end of a "word". That would be fast and efficient. But for this lab, we'll use a String a parameter. How to get Strings from a Reader?

3. BufferedReader is a decorator (wrapper) for Reader that can read an entire line at once.

```
// You can create a BufferedReader object from any Reader.
BufferedReader breader = new BufferedReader( new InputStreamReader(in) );
// read the input one line at a time.
```

```
// readLine() returns null when there is nothing to read (end of stream)
while( true ) {
    String line = breader.readLine( );
    if (line == null) break;
    // process the line
```

4. If the input line contains more than one word you can split it using a **`StringTokenizer`**. StringTokenizer.nextToken( ) is much faster than scanner.next() for reading words. Scanner is more flexible, as it lets you use a *regular expression* for the word delimiters. You don't need this for the lab, it is useful for the *Flesch Readability* assignment.

Reference

- *Programming a State Machine* in class week9 folder.
- Wikipedia, *Finite State Machines*.