Objectives	Design a state machine to count syllables in words. Draw a state machine diagram and then implement it using a) a simple non-OO approach (enum for states) and b) the OO approach (interface for states). Verify your code by counting syllables for all words in a dictionary.
What to Submit	Commit your code to Github as project named syllable-counter. Ask TA to check your State Machine diagram on paper before 16:00.

Assignment

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

2. Write two classes, each with a method named countSyllables(String word) that implements a state machine and counts syllables in a word. It returns the number of syllables. If something is not a word, return 0.

3. **Use the state machine appraoch**. A state machine should **never** need to look at the previous character or the next char in deciding what to do. Your code should depend only on the current state and current character (or event). Avoid "if" as much as possible (some "if" are necessary).

4. Count the words and syllables in dictionary.txt located at http://se.cpe.ku.ac.th/dictionary.txt. The file has one string per line, but some of them are not actual words according to our definition. Write a separate Main class to read words, call the syllable counter, and print results.

How to count syllables?

This assignment uses the same rules as the *Flesch Readability Index* (PA4) to count syllables. The number of syllables in a word is equal to the number of vowel sequences -- groups of vowels.

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

   `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.` `"y"` is not counted as a vowel because it comes after another vowel.

2. A final "e" as a single vowel is **not** counted, **unless** it is the only vowel in the word.

   `apple` = 1 vowel sequence `(a)pple` Don't count final "e".

   `louvre` = 1 vowel sequence `l(ou)vre` 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 `m(o)v(ie)` Final "e" is part of a multi-vowel group, so count it!

   `levee` = 2 vowel sequences `l(e)v(ee).` Same reason as "movie".

3. "y" counts as a vowel if is the first vowel in a vowel group; it is a consonant otherwise.

   `try` = 1 vowel sequence, "y" acts like vowel: `tr(y)`

   `beyond` = 2 vowel sequences, "y" acts like consonant: `b(e)y(o)nd`

   `yesterday` = 3 vowel sequences: `(ye)st(e)rd(a)y`

   `Yahoo` = 2 vowel sequences `(Ya)h(oo)`

4. A dash '-' in the middle of word acts like a consonant and divides vowel sequences.

`anti-oxidant` = 5 vowel sequences `(a)nt(i)-(o)x(i)d(a)nt`
`next-door` = 2 vowel sequences in one word `n(e)xt-d(oo)r`
`-oxidant` = **not a word**. Dash cannot be at start of a word.
`anti-` = Ignore dash at end of word. It sometimes occurs in writing.

5. Ignore apostrophe (') anywhere in the word, including beginning and end.

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

6. **Not a word**. Any string that contains non-letters or doesn't contain any vowels is not a word. The only exceptions are "-" and apostophe (') in case 4 and 5.

```
mrtg
Java5se
I.B.M.      ("." between letters)
7-Eleven  (contains "7")
```

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: *Define your own states. Here is one possible set of states.*

`START` = start of the string, no characters processed yet.

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

`SINGLE_VOWEL` = the most recent character is a vowel, including 'y', that is the first vowel in a vowel group.

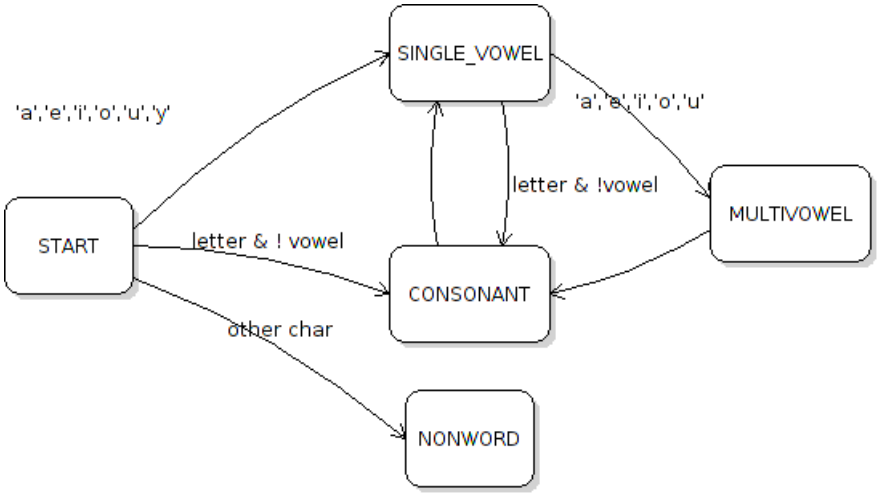
`MULTIVOWEL` = most recent char is a vowel that follows another vowel (2 or more vowels together).

`HYPHEN` = most recent character is a hyphen. In the middle of a word, hyphen behaves like a consonant, but you cannot have two hyphen together.

`NONWORD` = the character sequence is not a word. Enter this state if you see any character other than letter or hyphen.

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

Actions: add 1 to the syllable count. Show this action at correct places on state diagram.



Problem 2: Write a class to implement a Simple State Machine

2.1 Write a SimpleSyllableCounter class with a method named **`countSyllables`** to count syllables in a String.

SimpleSyllableCounter
countSyllables(String) : int

`countSyllables` returns the number of syllables in the String. If the string parameter is not a word then return 0.

2.2 Use your state machine diagram to implement countSyllables.

- use the state machine approach with a simple **`enum`** for states
- only look at one character at a time: don't use the previous character and don't look-ahead at the next character.

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

```
int countSyllables( String word ) {
    int syllables = 0;
    char c = ' ';
    State state = State.START;   // State is an enum of the states
    for(int k=0; k<word.length(); k++) {
        c = word.charAt(k);
        if (c == '\'') continue; // ignore apostrophe
        switch(state) {
        // process character c using state machine
        case CONSONANT:
            if (isVowelOrY(c)) { state = SINGLE_VOWEL; syllables++; }
            else if (isLetter(c)) /* stay in consonant state */;
            else if (c == '-') state = State.HYPHEN ;
            else state = State.NONWORD;
            break;
        case VOWEL:
            if (isVowel(c)) state = State.MULTIVOWEL;
            else //TODO
            break;
        //TODO other cases
        }
    } // end of loop for chars in word
    // End of word: Correct syllable count for the "final e" rule.
    // You only need the current state and current letter to do this
```

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* at the beginning of the word. *Whitespace* means a space, tab, or newline character.

2.4 **Don't** look ahead (next char) or look back (previous char). The state and current character should contain all the information you need to decide what action to take. 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.

Problem 3: Design and Write an O-O style State Machine

Write another class named OOSyllableCounter with a countSyllables(String word) method. In this class use the O-O approach to state machine. Define an interface or abstract class for states, and a concrete class for each of the states.

You can write the State interface and State classes an inner classes or external classes. These instructions are for *inner classes* (inside the WordCounter class) to simplify the code.

3.1 Since reading a character is an event, each State needs a method like `handleChar(char)`. You should also define `enterState()` and use it to increment the syllable count when entering the initial vowel (or single-vowel) state.

```
abstract class State {
     public abstract void handleChar(char c);
     public void enterState( ) { /* default is to do nothing */ }
}
```

3.2 Write an implementation for each of the states.

```
class SingleVowelState extends State {
     public void handleChar( char c ) {
          if ( isVowel(c) ) setState( MULTIVOWEL );
          else if ( isLetter(c) ) setState( CONSONANT );
          //TODO handle other cases
     }
     public void enterState( ) {
          syllableCount++;
     }
}
```

3.3 In the WordCounter class, you need to provide a setState( ) method.

```
class WordCounter {
     private final State START = new StartState( );
     private final State SINGLEVOWEL = new SingleVowelState( );
     //TODO add other states
     private State state;  // the current state
     private int syllableCount = 0;

     /** change to a new state */
     public void setState( State newstate ) {
          // this "if" may not be necessary.
          if (newstate != state) newstate.enterState( );
          state = newstate;
     }
```

3.4 Write the `countSyllables` method using states. It *delegates* the work of handling each character to the current state. The "for" loop just calls `state.handleChar(c)`. It should be a lot simpler than in Problem 2.

3.5 What about special cases at the end of word? Can you *delegate* that to the state classes?

Problem 4: Test the syllableCounter

Create a test class to test syllableCounter using some words that you know syllable count. There is a **`WordCounterTest.java`** class in the same folder as this lab assignment. You should add more tests to this file.

You should design a test for each case, such as words with final e (the) and with another vowel (move, tire), and multi-vowel (movie), an "e" by istelf (means "and" in Spanish), no vowels, and so forth.

Points will be deducted from this lab for lazy or incomplete set of tests. Testing is an essential part of software development.

Problem 5: Write a Main class to count a dictionary and calculate elapsed time

5.1 Write a Main class with a method (not the "main" method) that reads all the words from a URL or File and calls countSyllables. Output the total number of words, syllables, and the elapsed time in seconds. For example:

```
Reading words from http://se.cpe.ku.ac.th/dictionary.txt
Counted 102,000 syllables in 38,600 words
Elapsed time: 1.220 sec
```

5.2 Use this URL for the dictionary file: `http://se.cpe.ku.ac.th/dictionary.txt`
There is one word for line, but the file may contain blank lines and whitespace chars (check for them).
Example code for opening a URL as input stream is:

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

For fast reading of input as Strings, use a BufferedReader (its faster than Scanner). Since the dictionary file contains only one word per line, parsing it is easy.

```
BufferedReader reader =
                    new BufferedReader( new InputStreamReader( input ) );
while( true ) {
    String word = reader.readLine();
    // BufferedReader.readLine() returns null at end of the input
    if (word == null) break;
```

There is a short, C-style idiom for this (but harder to write try-catch):

```
while( (word = reader.readLine()) != null ) {
    // process this word
}
```

Optional: Use Recursion and/or Regular Expression

For a challenge, try using recursion to count syllables and a *regular expression* to match vowel groups. A partial regular expression match the first syllable group in a word (as match group 1) and then match everything else as match group 2 is: "[b-z'&&[^aeiouy]]*([aeiouy][aeiou]*+)([a-z'\\-]*)".

Regular expressions are defined in Javadoc for the Pattern class and in the *Java Tutorial*.

Programming Hints

Don't try to write everything at once. Write code to handle some kinds of words first and test that it works. For example: ignore the "final e" or "-" special cases. Let those test cases fail while you concentrate on handling the basic cases. When you get the basic cases to work, then add more states to make the specials test cases pass.

Reference

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