(48024) Applications Programming Lab 3 guide

Below is a plan similar to what most groups will have arrived at based on guidance from their tutor (although of course other plans are possible). If you do not yet have a workable plan for lab 3, you may use the one described below.

Instructions:

1. “Break it down”. This bit has been done for you - see the section below called “Method breakdown. This is where you break down your program into different levels of processing with a separate procedure or function for each level. There are 5 levels: top, sentence, word, part, character.
2. “Build it up”. You should now write code for each method from bottom to top. That is, the first method you should write is the level 5 isVowel() function. NOTE: The solution to the isVowel() function is in the lecture notes! Next you should write the level 4 function which is either vowelCount or hasTwoVowels. You have a choice for level 4 which you can read about below.

NEWS: PLATE will now give you partial marks for each level you got working, but only if you start from the bottom up. That is, implement isVowel() first. Given that the solution to isVowel() is in the lecture slides, all students should be able to at least get 20% of the marks for free - *IF* you build it up, from bottom to top.

Method breakdown

Complete goals, method headers and plans are below. To translate the plans into code, it will help you to look at the lecture examples as well as the ZCount and the ZWordCount programs which are included in your skeleton code download for this lab.

```
public class StretchWith2Vowels {
    // GOAL: read sentences until *.
    //       show number of matching words in sentence
    // PLAN: read loop, output
    public static void main(String[] args) {
        // HINT: See the lecture slides for an example
        // of a read loop for strings. Don’t use !=
    }

    // GOAL: read a sentence
    // PLAN: read
    public static String readSentence() {
    }
    // GOAL: count how many matching words in sentence
    // PLAN: count, sentence.split(" +"), for-each loop
    public static int matchCount(String sentence) {
    }

    // GOAL: check if a word matches
    // PLAN: any, word.split("z"), for-each loop
    // i.e. split a word into parts separated by z's
    //      and return true if "any" part has two vowels.
    // e.g. The word "aoezotazo" has 3 parts separated by z:
    // part 1: aoe
    // part 2: ota
    // part 3: o
    // Your goal here is to split the word into these parts,
    // and if ANY part has two vowels, return true to indicate
    // that yes, this word matches. Solution: 4 lines of code.
    // Just a direct application of the ANY pattern.
    public static boolean matches(String word) {
    }

    // GOAL: count how many vowels in a part
    // PLAN: count, string loop
    public static int vowelCount(String part) {
        // See below for an alternate level 4 goal.
    }

    // GOAL: check if a character is a vowel
    // PLAN: return "aeiou".contains("" + c);
    public static boolean isVowel(char c) {
    }
}
```

An alternative breakdown is to replace the vowelCount function by a hasTwoVowels function which simply returns a boolean:

```
    // GOAL: check if this part has exactly 2 vowels
    // PLAN: count, string loop, return count == 2
    public static boolean hasTwoVowels(String part) {
    }
```

If you make a vowelCount function, then the matches method will need to check if the result of vowelCount == 2. But if you make a hasTwoVowels function, then it is the hasTwoVowels function that is responsible to check if the vowel count == 2.

After you break it down, you build it up. Write each method starting from bottom to top.

NOTE!! Beware of copying and pasting code from documents such as this, or even the lecture slides. The editing software used to produce these documents likes to insert “slanting” double quotes from unicode which are not the same as the "ASCII" double quotes used in programming. Other symbols may similarly be replaced by unicode variants in these documents. To avoid this, type in the ASCII symbols manually.

NOTE!! Even if you copied and pasted all of this skeleton code with the correct ASCII symbols, it will not compile because functions only compile if they return something. Therefore, you should only add a method to your program when you’re ready to write the full code for it, including the return statement. Don’t add the next method to your program until after you’ve finished coding the previous method. Then at each stage, your program will compile. (An alternative development strategy is shown at the end of this document.)

Testing

If you use BlueJ, you can test each function by right-clicking on your StretchWith2Vowels class and selecting the function you want to test. Examples:

1. Right-click on the StretchWith2Vowels class and select the isVowel function. A window will pop up asking you to supply a test parameter value. Type in ‘a’ with single quotes as the parameter value and press OK. The function will (or should if correct) return true. Try again with ‘b’ as the parameter value, and the function should return false.
2. Right-click on StretchWith2Vowels and select either the countVowels function or the hasTwoVowels function - whichever one you chose to do for level 4. Try putting in a string such as “ota” with double quotes, then press OK. If you did the countVowels version, it should return 2. Or if you did the hasTwoVowels version, it should return true.

If your function does not produce the expected results, check your patterns against your patterns book and the lecture notes and make sure you have applied them exactly. It may also help to step through your code using the debugger as shown in the week 2 lecture.

Alternative development strategy

It is recommended for this week that you add methods one by one when you’re ready to code them. Starting with just one method:

```
public class StretchWith2Vowels {
    public static boolean isVowel(char c) {
        return ...insert code from the lecture....;
    }
}
```

Get this to compile first, submit to plate, get marks, move onto the next level up.

However, an alternative strategy is to put empty methods for all methods into your program. As mentioned, this won't compile because functions require a return statement to compile. But if you really want to put all methods into your program at the outset, just put dummy return statements into each function like this:

```
public class StretchWith2Vowels {
    public static void main(String[] args) {
    }

    public static String readSentence() {
        return "*";
    }

    public static int matchCount(String sentence) {
        return 0;
    }

    public static boolean matches(String word) {
        return false;
    }

    public static int vowelCount(String part) {
        return 0;
    }

    public static boolean isVowel(char c) {
        return false;
    }
}
```

This will compile, and then you can proceed to fill in the code for each method.