Repetitive Program Execution

Quick Start

Compile step

once always
mkdir labs
javac Vowel3.java

Execute step

cd labs
mkdir 3
java Vowel3

cd 3

Submit step

cpy /samples/csc/156/labs/3/* .
emacs Vowel3.java &
submit csc156abc 3

Problem Statement and Specifications

This assignment will begin our study of nesting programming structures and the manipulation of String objects. In the previous exercise, we used several selection statements to help us determine if we were in one of several cases. The same will be true in this exercise, but we'll nest the selection statement inside of a repetition statement.

Assignment 3 Statement:
Write a program in a file named Vowel3.java that prompts the user for an input String and then determines and displays the number of characters as well as the number of each of the vowels (’a’, ’e’, ’i’, ’o’, ’u’) that are in the input String. Your program should display the input String as well.

△∇△ 1/2 lab extra credit △∇△

For designing a program stored in a file named Vowel3.java that allows the user to enter as many input Strings as they like. Simply prompt the user after execution of the program if they wish to enter another input String and process that String if they wish to continue. Modify your compilation, execution and submission steps to javac Vowel3.java, java Vowel3, and submit csc156abc 33, respectively.

Worth noting here is that the input String can contain embedded blanks that should be counted as character values in the String. In addition, we’ll need a separate counter for each of the vowels. Finally, upper case (’A’) and lower case (’a’) vowels should be counted as the same vowel.

Assignment 3 Specifications:
Input the String object, possibly with embedded blanks
Output the String that was input
the number of character values in the input String
the number of each vowel in the input String
We’ve already identified those objects in our program that should be String and it can certainly be the case that the counters for each of the vowels can be int. There are certain methods that are associated with String that can help us with our work. We’ll also need to determine a number of boolean expressions to understand if we are examining a vowel and which one it is. That will be part of our selection algorithm. Finally, we’ll need some form of repetition in order to visit the various char values in the input String. We’ll begin to develop our algorithm in the same manner that we always have.

1. Input
2. Process
3. Output

Once we’ve input the String, we’ll need to determine how many characters are in the object and how to examine each individual character. The first part is easy, since there is a method for that very purpose. Assume that we have the following String declaration.

```
String input_string;
```

Then the number of characters (including blanks) that are stored in the String is given by the expression `input_string.length()`. For now, we’ll only remind you that the characters in the String positions are numbered from 0 through `input_string.length()-1`. Thus far, our pseudo code looks like the following.

1. output a descriptive message about the program
2. output a prompt for the input String
   (a) process to determine the i’th character of the String
   (b) process to determine if the i’th character is a vowel
      i. process to update a counter for that vowel if it is a vowel
   (c) process to repeat for every character in the String
3. display the number of characters and the number of each vowel in the String

Steps (a), (b) and (c) are indented since they will be repeated for various values of the index variable i. Step (b).i is again indented as it represents a nesting of a selection structure within the repetition structure. All of our vowel counters will undergo what we refer to as a list reduction algorithm. This can be represented as follows.

I. initialize sum

II. loop to visit each list element

III. update sum as you visit each list element

In this instance, let’s assume that we’re talking about the value that will count the number of times that the character a occurs in the input_string. We’ll start by initializing the value to 0. Then, whenever we find that one of the characters in input_string has a value of a, we simply increase the counter by 1. Therefore, our final algorithm, with some renumbering, will look like the following. Note that the lines displayed in blue represent areas where you will need to modify or write code to complete the assignment.
CSC 156 - Assignment 3 Vowel

<table>
<thead>
<tr>
<th>Statement</th>
<th>Data objects</th>
</tr>
</thead>
<tbody>
<tr>
<td>1) display a descriptive message</td>
<td>String constants</td>
</tr>
<tr>
<td>2) initialize all vowel counters to 0</td>
<td>int variables &amp; constants</td>
</tr>
<tr>
<td>3) prompt for and input the String</td>
<td>String constant &amp; variable</td>
</tr>
<tr>
<td>4) visit String positions 0 through String.length()-1</td>
<td>int variable and constants</td>
</tr>
<tr>
<td>5) if position i of String is a vowel</td>
<td>int &amp; char variables, constants, &amp; expressions</td>
</tr>
<tr>
<td>6) increase that vowel’s counter by 1</td>
<td>int variable and constant</td>
</tr>
<tr>
<td>7) display the String</td>
<td>String variables &amp; constants</td>
</tr>
<tr>
<td>8) display the number of character in String</td>
<td>String &amp; int constants</td>
</tr>
<tr>
<td>9) display the number of each vowel in String</td>
<td>String constants, int variables &amp; constants</td>
</tr>
</tbody>
</table>

Just to clarify matters, **Step 5 and Step 6** are in the **for** loop determined at **Step 4** while **Step 5** is part of a selection statement that is within the body of the loop.

## Coding for Compilation

Logon to your csc.oakton.edu account and create a new subdirectory of the labs directory called 3 to work on your assignment.

```bash
    cd labs
    mkdir 3
    cd 3
```

The documentation for this algorithm has already been included in the file `/samples/csc/156/labs/3/Vowel3.java` that you can copy over to your local directory by issuing the following `cp` command.

```bash
    cp /samples/csc/156/labs/3/Vowel3.java .
    emacs Vowel3.java &
```

### Declaration of Variables

We've already discussed the declaration of the String variable, `input_string`. We would also like to declare the 5 int variables that will count the number of occurrences of each vowel. Something like the following would do.

```java
    int a_count, e_count, i_count, o_count, u_count;
```

## Follow the Pseudo Code

### Step 1

// Display a descriptive message for the program

Write an appropriate output statement the displays the message shown in **Figure 1**

### Step 2

// initialize all vowel counters to 0

Something like the following would be appropriate.

```java
    a_count = e_count = i_count = o_count = u_count = 0;
```
Step 3

// prompt for and input the String

Code the appropriate statements for the prompt that is displayed in Figure 2 and the input that should read
the String based upon the user’s response at the keyboard.

Step 4

// Visit positions 0 through String.length()-1

Until we get past the Checkpoint, let’s limit the number of positions that we’ll look at. For now, let’s only look
at the first position of the String that the user has input. If that char value is not a vowel, all counters should
display 0 when the output is displayed. If that char value is a vowel, one of the counter variables should have a
value of 1. After we pass the Checkpoint, we’ll come back to this step and implement the steps necessary to do
what is being described in the pseudo code. For now, simply code the following statement.

int i = 0;

Steps 5-6

// If position i of String is a vowel
// increase that vowel’s counter by 1

This is a selection situation and we could compare the i’th String position against vowels using an if statement
along the following lines.

    if (input_string.charAt(i)== ‘a’ || input_string.charAt(i)== ’A’)
        a_count++;
    else ...
There is another means of providing support for selection in Java and that is with a statement that is called a **switch**. It is important to recognize that the **switch** statement should not be used with **double** values. It is often used for **int** or **char** data. Its general format is as follows.

\[
\text{switch}(\text{switch expression})
\begin{cases}
\text{case } \text{switch expression}_1 & \text{statement}_1,1 \text{statement}_1,2 \\
\vdots & \\
\text{case } \text{switch expression}_2 & \text{statement}_2,1 \text{statement}_2,2 \\
\vdots & \\
\text{default} : & \text{statement}_n,1 \text{statement}_n,2 \\
\vdots & \\
\end{cases}
\]

The **switch expression** is evaluated and compared to **switch expression**1, **switch expression**2, ... until it finds equality. If it can not find equality, it will execute the statements in the optional **default** case. If it finds equality with **switch expression**k, it will execute the statements in **case**k until either it finds a **break** statement or **case**k+1. If there is a **break** statement, execution transfers to next statement after the **switch** statement. If there is no **break** statement, execution will continue onto those statements in **case**k+1. For example, the appropriate **switch** statement that would provide the equivalent control of execution as the **if** statement example that we provided above would look as follows.

\[
\text{switch(input_string.charAt(i))}
\begin{cases}
\text{case } 'a' : \\
\text{case } 'A' : \\
\text{a_count}++; \\
\text{break} ; \\
\vdots \\
\end{cases}
\]

Code either the appropriate **if** or **switch** statement for these steps before proceeding.

**Steps 7-9**

// display the String
// display the number of characters in String
// display the number of each vowel in String

These output statements are relatively straight forward and should taken on the form as is shown in Figure 3.

It is worth remarking that the number of characters stored in **input_string** is **input_string.length**().

**Checkpoint**

Although we’re almost done, at this point, it might be a good idea to compile this much before we build more. Save your program to disk by choosing the **Save** command from the **Files** menu of your **emacs** session. Then,
compile your program by choosing the **Compile**... command from the **Tools** menu and change the `make -k` that is displayed to `javac Vowel3.java`. Compiler errors can be parsed with the keystroke C-x ‘ ‘ and need to be repaired before your program can execute. When your program has compiled, click on your **xterm** window to access your command line prompt, and issue the command `java Vowel3`. If you type input in response of the prompt that is the same as Figure 2, you should get output that looks like Figure 3. Similarly, for any vowel that you type in response to the prompt, there should be an update only on the counter of that vowel. You might try to enter upper and lower cases vowels, consonants or other symbols to completely test your program out. Once you’re satisfied of its correctness, we’ll return to Step 4 and implement the code necessary to process **String** input that has more than one character.

**Step 4 (revisited)**

// Loop over every character in the String

The style of loop that we’re interested in using at this point in our algorithm is referred to as an indexed for statement and it has some features that are particularly attractive for this exercise. It’s general format is the following.

```java
for (initial_expression; boolean_expression; update_expression)
{
    statement_1;
    statement_2;
    ...
    statement_m;
}
```

Upon entering the statement, we a) evaluate the **initial_expression**. Then b) evaluate the **boolean_expression**. Whenever the outcome of this is **false**, we leave the loop and execute any statements after the closing brace `}`. If the outcome of the **boolean_expression** is **true**, we execute the body of the loop `statement_1, statement_2, ..., statement_m`. Then, c) execute the **update_expression** and go back and repeat step b).

For example, in our current situation, we’d like to generate all of the integers 0, 1, 2, ..., `input_string.length()-1`. So, we’ll use the **int** variable `i` to loop with, start it at 0 and end it at `input_string.length()-1`. The following should be coded for Step 4 instead of the initialized declaration of the variable `i`. 
for (int i=0; i < input_string.length(); i++)
switch(input_string.charAt(i))
{
  
}

Note that the switch statement is viewed syntactically as a single statement, so the braces {} for the for statement are optional. They are only necessary when more than one statement is to be repeated. However, the braces {} for the switch statement are not optional and must be present, regardless of the number of case statements that occur. Code this for statement before continuing.

Testing for errors

You need to save your program, compile and execute it as you did up above at the Checkpoint. In general, you should expect that longer single lines of input, even with embedded blanks, should be handled correctly. For instance, if you use the output that is shown in Figure 4 you should see the output displayed in Figure 5. When you are writing programs that use repetition, it will sometimes happen that the condition(s) upon which the repetition is based do not occur as expected. In these cases, we say that the program is in an infinite loop since the repetition is in control of the execution rather than the programmer. If output is occurring, you might see one line of output fill up the monitor in these instances. When this happens, you will need a means of halting execution from the command line. If you’ve started your execution with the following command:
java Vowel3

the command line can be returned to your control with the Ctrl-z keysequence. Once that you have regained control of the command line prompt, issue the report process status command.

ps

This will display every process that your account currently has active in the system. You are seeking the Process ID (PID) of the process that has java as the front part of the command. For example, the following displays the active processes when running java. We wish to eliminate PID 27110 which is the process associated with the java command.

$ java Vowel3
(issue the Ctrl-z keysequence)
Stopped java

$ ps
PID TTY TIME CMD
27076 pts/12 00:00:00 bash
27110 pts/12 00:00:00 java
27130 pts/12 00:00:00 ps

$ kill -9 27110

$ ps
PID TTY TIME CMD
27076 pts/12 00:00:00 bash
27131 pts/12 00:00:00 ps

This is a means that can be used to gracefully shut down your program when execution has taken control. There is a correctly functioning version of Vowel3.java at [this link] and a correctly functioning version of Vowel33.java at [this link].

Printing and submitting

Once you are satisfied with the correctness of your program, print it as you did with previous assignments by using the following command that assumes that you are working in the room 1234 at Oakton. Retrieve your copy from the printer.

printer 1234 Vowel3.java

Finally, submit your program with the following command that assumes that you are registered in section abc of CSC 156.

submit csc156abc 3

If you pursue the extra credit option, store your solution in a file named Vowel33.java and issue the following commands to print and submit your assignment.

printer 1234 Vowel33.java
submit csc156abc 33