Worked Example 6.3

A Sample Debugging Session

This Worked Example presents a realistic example for running a debugger by examining a Word class whose primary purpose is to count the number of syllables in a word. The class uses this rule for counting syllables:

Each group of adjacent vowels (a, e, i, o, u, y) counts as one syllable (for example, the "ea" in "peach" contributes one syllable, but the "e . . . o" in "yellow" counts as two syllables). However, an "e" at the end of a word doesn't count as a syllable. Each word has at least one syllable, even if the previous rules give a count of 0.

Also, when you construct a word from a string, any characters at the beginning or end of the string that aren't letters are stripped off. That is useful when you read the input using the next method of the Scanner class. Input strings can still contain quotation marks and punctuation marks, and we don't want them as part of the word.

Here is the source code. There are a couple of bugs in this class.

ch06/debugger/Word.java

```
/**
   This class describes words in a document. There are a couple
   of bugs in this class.
*/
public class Word
{
   private String text;

   /**
      Constructs a word by removing leading and trailing non-
      letter characters, such as punctuation marks.
      @param s the input string
   */
   public Word(String s)
   {
      int i = 0;
      while (i < s.length() && !Character.isLetter(s.charAt(i)))
         i++;
      int j = s.length() - 1;
      while (j > i && !Character.isLetter(s.charAt(j)))
         j--;
      text = s.substring(i, j);
   }

   /**
      Returns the text of the word, after removal of the
      leading and trailing non-letter characters.
      @return the text of the word
   */
   public String getText()
   {
      return text;
   }

   /**
      Counts the syllables in the word.
      @return the syllable count
   */
   public int countSyllables()
   {
      int count = 0;
      int end = text.length() - 1;
      if (end < 0) return 0; // The empty string has no syllables

      // An e at the end of the word doesn't count as a vowel
      char ch = Character.toLowerCase(text.charAt(end));
      if (ch == 'e') end--;

      boolean insideVowelGroup = false;
      for (int i = 0; i <= end; i++)
      {
         ch = Character.toLowerCase(text.charAt(i));
         String vowels = "aeiouy";
         if (vowels.indexOf(ch) >= 0)
         {
            // ch is a vowel
            if (!insideVowelGroup)
            {
               // Start of new vowel group
               count++;
               insideVowelGroup = true;
            }
         }
      }

      // Every word has at least one syllable
      if (count == 0)
         count = 1;

      return count;
   }
}
```

Here is a simple test class. Type in a sentence, and the syllable counts of all words are displayed.

ch06/debugger/SyllableCounter.java

```
import java.util.Scanner;

/**
   This program counts the syllables of all words in a sentence.
*/
public class SyllableCounter
{
   public static void main(String[] args)
   {
      Scanner in = new Scanner(System.in);

      System.out.println("Enter a sentence ending in a period.");

      String input;
      do
      {
         input = in.next();
         Word w = new Word(input);
         int syllables = w.countSyllables();
         System.out.println("Syllables in " + input + ": " + syllables);
```

```
            }
            while (!input.endsWith("."));
        }
    }
```

Supply this input:

```
hello yellow peach.
```

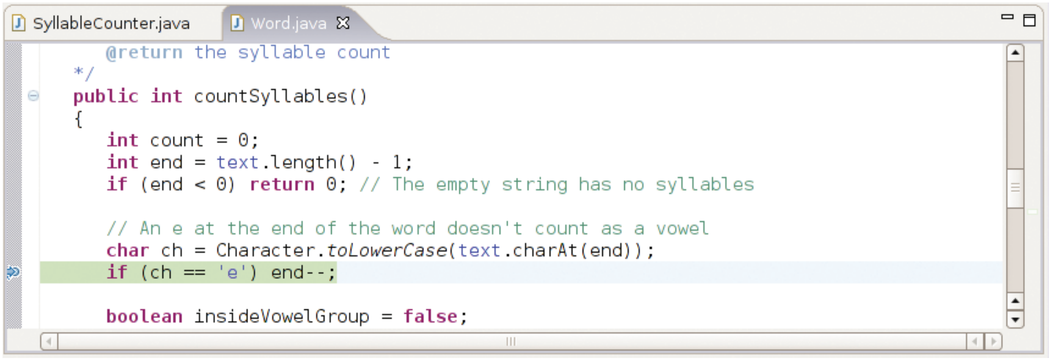
Then the output is

```
Syllables in hello: 1
Syllables in yellow: 1
Syllables in peach.: 1
```

That is not very promising.

First, set a breakpoint in the first line of the `countSyllables` method of the `Word` class, in line 39 of `Word.java`. Then start the program. The program will prompt you for the input. The program will stop at the breakpoint you just set.

First, the `countSyllables` method checks the last character of the word to see if it is a letter `'e'`. Let's just verify that this works correctly. Run the program to line 47 (see Figure 9).



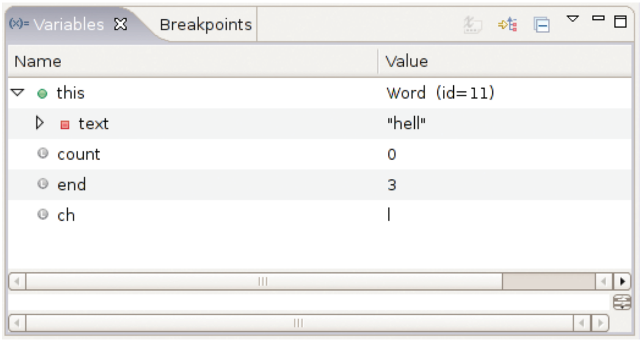
```
       @return the syllable count
    */
   public int countSyllables()
   {
      int count = 0;
      int end = text.length() - 1;
      if (end < 0) return 0; // The empty string has no syllables

      // An e at the end of the word doesn't count as a vowel
      char ch = Character.toLowerCase(text.charAt(end));
      if (ch == 'e') end--;

      boolean insideVowelGroup = false;
```

Figure 9 Debugging the `countSyllables` Method

Now inspect the variable `ch`. This particular debugger has a handy display of all current local and instance variables—see Figure 10. If yours doesn't, you may need to inspect `ch` manually. You can see that `ch` contains the value `'l'`. That is strange. Look at the source code. The `end` variable was set to `text.length() - 1`, the last position in the `text` string, and `ch` is the character at that position.

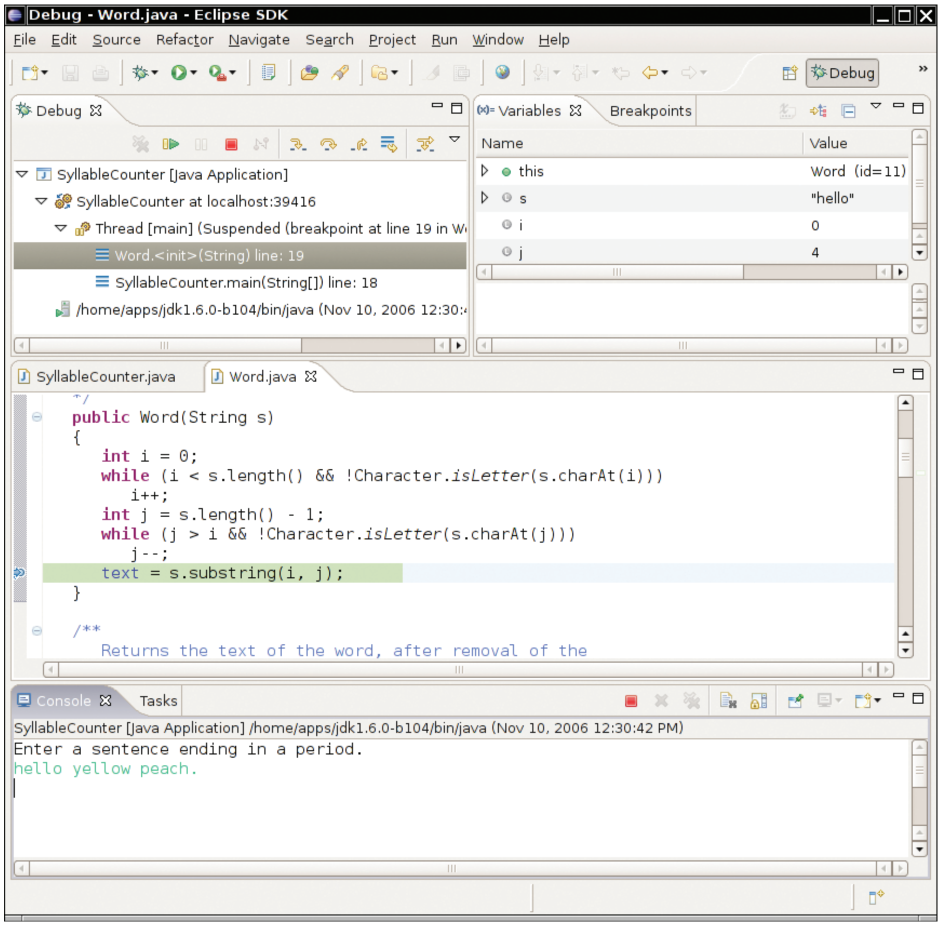


Name	Value
this	Word (id=11)
text	"hell"
count	0
end	3
ch	l

Figure 10 The Current Values of the Local and Instance Variables

Looking further, you will find that `end` is set to 3, not 4, as you would expect. And `text` contains the string `"hell"`, not `"hello"`. Thus, it is no wonder that `countSyllables` returns the answer 1. We'll need to look elsewhere for the culprit. Apparently, the `Word` constructor contains an error.

Unfortunately, a debugger cannot go back in time. Thus, you must stop the debugger, set a breakpoint in the `Word` constructor, and restart the debugger. Supply the input once again. The debugger will stop at the beginning of the `Word` constructor. The constructor sets two variables `i` and `j`, skipping past any nonletters at the beginning and the end of the input string. Set a breakpoint past the end of the second loop (see Figure 11) so that you can inspect the values of `i` and `j`.


Debug - Word.java - Eclipse SDK
File Edit Source Refactor Navigate Search Project Run Window Help
Debug
Debug
Variables
Breakpoints
SyllableCounter.java
Word.java
Console
Tasks

```
SyllableCounter [Java Application]
  SyllableCounter at localhost:39416
    Thread [main] (Suspended (breakpoint at line 19 in W
      Word.<init>(String) line: 19
      SyllableCounter.main(String[]) line: 18
  /home/apps/jdk1.6.0-b104/bin/java (Nov 10, 2006 12:30:
```

Name	Value
this	Word (id=11)
s	"hello"
i	0
j	4

```
   */
   public Word(String s)
   {
      int i = 0;
      while (i < s.length() && !Character.isLetter(s.charAt(i)))
         i++;
      int j = s.length() - 1;
      while (j > i && !Character.isLetter(s.charAt(j)))
         j--;
      text = s.substring(i, j);
   }

   /**
      Returns the text of the word, after removal of the
```

```
SyllableCounter [Java Application] /home/apps/jdk1.6.0-b104/bin/java (Nov 10, 2006 12:30:42 PM)
Enter a sentence ending in a period.
hello yellow peach.
```

Figure 11 Debugging the `Word` Constructor

At this point, inspecting `i` and `j` shows that `i` is 0 and `j` is 4. That makes sense—there were no punctuation marks to skip. So why is `text` being set to `"hell"`? Recall that the `substring` method counts positions up to, but not including, the second parameter. Thus, the correct call should be

```
text = s.substring(i, j + 1);
```

This is a very typical off-by-one error.

Fix this error, recompile the program, and try the three test cases again. You will now get the output

```
Syllables in hello: 1
Syllables in yellow: 1
Syllables in peach.: 1
```

As you can see, there still is a problem. Erase all breakpoints and set a breakpoint in the `countSyllables` method. Start the debugger and supply the input `"hello."`. When the debugger stops at the breakpoint, start single stepping through the lines of the method. Here is the code of the loop that counts the syllables:

```
boolean insideVowelGroup = false;
for (int i = 0; i <= end; i++)
{
   ch = Character.toLowerCase(text.charAt(i));
   String vowels = "aeiouy";
   if (vowels.indexOf(ch) >= 0)
   {
      // ch is a vowel
      if (!insideVowelGroup)
      {
         // Start of new vowel group
         count++;
         insideVowelGroup = true;
      }
   }
}
```

In the first iteration through the loop, the debugger skips the `if` statement. That makes sense, because the first letter, `'h'`, isn't a vowel. In the second iteration, the debugger enters the `if` statement, as it should, because the second letter, `'e'`, is a vowel. The `insideVowelGroup` variable is set to `true`, and the vowel counter is incremented. In the third iteration, the `if` statement is again skipped, because the letter `'l'` is not a vowel. But in the fifth iteration, something weird happens. The letter `'o'` is a vowel, and the `if` statement is entered. But the second `if` statement is skipped, and `count` is not incremented again.

Why? The `insideVowelGroup` variable is still true, even though the first vowel group was finished when the consonant `'l'` was encountered. Reading a consonant should set `insideVowelGroup` back to `false`. This is a more subtle logic error, but not an uncommon one when designing a loop that keeps track of the processing state. To fix it, stop the debugger and add the following clause:

```
if (vowels.indexOf(ch) >= 0)
{
   . . .
}
else insideVowelGroup = false;
```

Now recompile and run the test once again. The output is:

```
Syllables in hello: 2
Syllables in yellow: 2
Syllables in peach.: 1
```

Is the program now free from bugs? That is not a question the debugger can answer. Remember: Testing can show only the presence of bugs, not their absence.