

1 Objectives

- Use Eclipse to debug Java code

2 Introduction

In this lab you will learn the important skill of using a debugger. You will be using Eclipse to debug a program.

3 Import Files to Debug

Begin by importing the files you need for today's lab.

- Start Eclipse and create a new project called lab03. (Select **File** → **New** → **Java Project**).
- You need to import these files. (Highlight the src folder and select **File** → **Import...**)

```
~csci204/2009-fall/student/labs/lab03/Point.java  
~csci204/2009-fall/student/labs/lab03/TestPoint.java  
~csci204/2009-fall/student/labs/lab03/TestWord.java  
~csci204/2009-fall/student/labs/lab03/Word.java
```

- Now add the files to your Subversion repository. Click on lab03 so that it is highlighted and then right click. Select **Team** → **Share Project...** Select SVN as the repository type and select your repository.

4 Read and Execute the Program

Open the file **Word.java** and read the comment at the top of the file. It explains how the class should work. The program will identify the number of syllables in a word. Open and read **TestWord.java** so you know which method in **Word** is used to do the job of counting syllables.

Keep the file **TestWord.java** open and click on the run button. Try using the input string `hello regal real`. The output will be

```
Syllables in hello: 1  
Syllables in regal: 1  
Syllables in real: 1
```

Clearly, something is wrong with the program!

5 Debug the Program

This section will explain how to use Eclipse to debug this program.

5.1 Set a Breakpoint

Eclipse will let you stop an executing program so that you can see what is going on. The place where the program stops is called a *breakpoint*.

The purpose of the class `Word` is to count syllables in a word but it is not doing that properly. A reasonable first guess is that there is a problem in the `countSyllables()` method. Open `Word.java` and locate the `countSyllables()` method. Eclipse has a convenient list of methods in a pane on the right of the screen. Clicking on a method name displays it in the editor.

The first line of the method declares a variable `count` and sets it to 0. Click anywhere on this line and it will be highlighted in light blue. Right click to the left of the blue area in the left margin and select **Toggle Breakpoint**. A small blue circle will appear in the margin. This means that execution will stop when the program reaches this line.

5.2 Run the Program

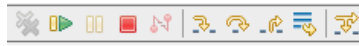
Run the program by selecting **Run** → **Debug...** Eclipse will once again prompt you for your input string. Enter the same string as before.

Next, Eclipse will ask if you want to switch perspectives. A *perspective* is just another way of looking at your project. When you are debugging, you need a different perspective. Click on the **Yes** button. Eclipse ran your program and stopped at your breakpoint. It has also opened some new windows for you.

- The window in the upper left is a debug window. It tells you that the program is stopped at your breakpoint in `countSyllables()`. It also tells you how you got there. In this case, it says that you called `countSyllables()` from `main()`.
- The window in the middle of the screen is your source window. It shows the source code for your program. The line with your breakpoint is highlighted in light green. In the left margin is a right arrow indicating that this is the next line to be executed.
- In the upper right is a window that shows your variables and their values.

5.3 Single Step

Above the debug window is a toolbar that you can use to control execution of the program.



If you hover the mouse over a symbol, descriptive text will appear telling you the purpose of the button. The **Step Over** button will let you step through the program one statement at a time. Click on it once and Eclipse will execute the statement to which the right arrow is pointing. Notice that executing this statement created a new variable which now appears in your variable window along with its value. Keep stepping over until the program reaches the following line.

```
if (ch == 'e')
```

As you step, notice that the variable window continues to update as you create new variables and their values change.

Note: When you are single stepping, you have the option of treating a method call as a single statement by using the **Step Over** button or entering the method with the **Step Into** button.

5.4 Examine Variables

`countSyllables()` thinks it's looking at the last character in a word, yet `ch` has a value of 1. Our word is `hello` and this is clearly wrong. The last character should be `o`.

The variable `end` should contain the index of the last character. It's value is 3. Since "hello" is a 5 letter word, the index of the last character should be 4!

In the **Variables** window, click on the disclosure triangle next to the word `this`. This is how you examine the instance variables of the current object. You can see that `text` has a value of `"hell"`. Now we're getting somewhere! There is a problem outside of `countSyllables()` that causes the instance variable `text` to get an incorrect value. The `Word` constructor seems like a good place to check next, since that is where the input word is first read.

Unfortunately, we have already passed the execution of the constructor and there is no way to go back. Select the **Terminate** button (it's a red rectangle) to terminate the program.

5.5 Run the Program Again

Let's try running the program again, stopping in the constructor this time. The previous breakpoint (the first line inside the method `countSyllables()`) should still

have a blue circle next to it. Right click on the circle and select **Toggle Breakpoint**. This deletes the breakpoint.

Now set a breakpoint at the first line of the `Word` constructor and run the program by selecting **Run** → **Debug**. Enter the string `hello` when prompted. The debugger should stop at the first line of the constructor.

Look in the **Variables** window and check the value of the parameter `s` and verify that it is correct. It should be `"hello"`.

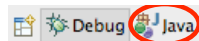
The constructor contains two loops. The first skips over any leading blanks in the word, and the second skips over any trailing blanks. After the second loop, `i` and `j` should indicate the beginning and end of the word. Place another breakpoint on the last line of the method and press the toolbar's **Resume** button. The program will stop at your new breakpoint.

You will see that `i` and `j` have the values 0 and 4 respectively. These are the indexes of the first and last characters and they are correct. Click the **Step Over** button to execute the last line in the method. Examine the value of `text` in the **Variables** window. So why is `text` being set to `"hell"` instead of `"hello"`? The problem is with the second parameter of `substring()`. The substring is taken up to, but not including this character! So, the correct call should be

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

This is a classic *off by one* error.

Quit running the debugger, and change back to the Java perspective by clicking on the **Java** icon in the upper right of your screen. (If it's not present, click on the **Open Perspective** icon and then the **Java** icon.) Make the above change and save the file.



This is a good time to commit your changes to your Subversion repository, documenting the fact that you fixed a bug. Do that now by selecting your `lab03` project, right clicking, and selecting **Team** → **Commit...**

5.6 Run the Program a Third Time

Compile and run the `TestWord` program again. Use the input string `"hello regal real"`. You should get the following output.

```
Syllables in hello: 1  
Syllables in regal: 1  
Syllables in real: 1
```

Unfortunately, there's still a problem with the program.

5.7 Clear All Breakpoints

Switch back to the debug perspective by clicking on the **Open Perspective** icon and then **Debug**.

If you need to remove all the breakpoints, select **Run** → **Remove All Breakpoints**. Set a new breakpoint at the first line of `countSyllables()`.

5.8 Debug Syllable Recognition

With the breakpoint that you set above, select **Run** → **Debug** and run the program again.

This time you will watch the code that does the syllable recognition more closely. Use "regal" as your input string. When the program stops, step over until you reach the following syllable recognition code.

```
boolean insideVowelGroup = false;
for (int i = 0; i <= end; i++) {
    ch = Character.toLowerCase(text.charAt(i));
    if ("aeiouy".indexOf(ch) >= 0) {
        if (!insideVowelGroup) {
            count++;
            insideVowelGroup = true;
        }
    }
}
```

Use the **Step Over** button to step through the `for` loop to verify the following behavior. The first time through the `for` loop the program skips the `if` statement because 'r' is not a vowel. This is correct. The second time through it enters the `if` because 'e' is a vowel. The third time through the loop, the program skips the `if` statement because 'g' is not a vowel. This is correct and so far everything is OK. When the loop processes the 'a', something odd happens. It enters the first `if` statement as it should, but it does not enter the second `if` and so the counter is not incremented.

The problem is that the `boolean` variable `insideVowelGroup` is still true even though we are finished with the first vowel group. Each time the program sees a consonant, it should be setting `insideVowelGroup` to false to indicate that it is no longer in a vowel group. The above code should become

```
boolean insideVowelGroup = false;
for (int i = 0; i <= end; i++) {
    ch = Character.toLowerCase(text.charAt(i));
    if ("aeiouy".indexOf(ch) >= 0) {
        if (!insideVowelGroup) {
            count++;
            insideVowelGroup = true;
        }
    }
}
```

```
    } else {  
        insideVowelGroup = false;  
    }  
}
```

Make the change and save the program. Now compile and run the program again. The output now should be correct.

```
Syllables in hello: 2  
Syllables in regal: 2  
Syllables in real: 1
```

Once you have verified that this last change corrected the problem, commit your changes to the repository. Be sure to provide a helpful comment that explains what you have done.

6 Conclusions

The program now appears to be working properly. Is it bug free? It's hard to say. The testing process can only show the presence of bugs. It cannot show their absence.

The Eclipse debugger is a very useful tool. The bugs that were in this program were subtle. They would have been difficult to uncover without the debugger.

7 Another Exercise

Change back to the Java perspective in Eclipse, and close the files `Word.java` and `TestWord.java`. Double click on `TestPoint.java` and click on the **Run** button to run the program. You should see a run-time error. Use the debugger to help find and fix the error. Each time you fix a bug, be sure to commit your changes to the repository. (There may only be one error in this exercise).

8 Upon Completion

Be sure that you have committed your lab03 project to the repository. You should **not** see a brown and white star next to your file names in Eclipse.