This document shows how to use BlueJ to debug a Java program. We will use the SyllableCounter program in Chapter 6 as example.

Let’s first execute the program to see that there are bugs in the program. In the project window, right-click on the SyllableCounter class and select “void main (String [] args)” to execute the program.
You will be presented with a dialog box asking for command line arguments. Click OK to accept it.

All input/output are performed in a terminal window.
Enter the sentence “hello yellow peach.” and press return.

The program continues and produces the following (incorrect) output.
Therefore, we need to debug the program. To see the debugger, select “View | Show Debugger”.

The debugger window looks like the following:
Now, let us set a breakpoint at the first line of the countSyllables method in the Word class. We must first double click on the Word class in the project window to open the class in the editor. When the class is opened in the editor, we move the cursor to and highlight the line that we want to set the break point. In our case, it is line 38.

```java
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--;
    
    // ... (remaining code)
```

Then select “Tools | Set/Clear Breakpoint”.
A stop sign will appear on top of the line number.

Now, go back to the project window, right-click on the SyllableCounter class and select “void main (String [] args)” to execute the program. Enter the sentence “hello yellow peach.” and press return.
The program will run to the breakpoint we have set. An arrow will appear on the line of the breakpoint in the class editor:

And the debugger will look like the following:
It shows that the main thread (don’t worry if you don’t understand what a thread is) is at breakpoint. It reaches the breakpoint by the following call sequence (left middle pane): Word.countSyllables, which is invoked by SyllableCounter.main. The right middle panes show that we do not have any static variables or local variables. We have an instance variable, its name is “text”; it is a String, and its value is “hell”. There is a problem here. However, let’s follow the textbook and execute the program until the line “if (ch == ‘e’) end--;” (line 44). We can do this by using the Step button in the debugger four times (once per line, not counting the blank and comment lines):
Now the program stops at line 44 (note the arrow):

```java
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 = text.charAt (end); if (ch == 'e') end--;
    //
    if (end >= 0) count += 1;
```

And the thread changes to “stopped”. The debugger also shows that we now have three local variables: count and end are integers, having the values of 0 and 3, respectively; ch is a character, having the value of “l”.

```java
    // An e at the end of the word doesn't count as a vowel
    char ch = text.charAt (end); if (ch == 'e') end--;
    //
    if (end >= 0) count += 1;
```
As mentioned in the textbook, at this point, we know that something is wrong in the constructor (since text contains “hell” instead of “hello”). But we cannot go back in time. So we have to terminate (reset) the execution and start again. We can terminate the execution by using the “Terminate” button in the debugger:

When you press the “Terminate button”, the debugger window will be reset.
Unlike other debugger, BlueJ will remove all breakpoint when you terminate a debug session. This may be a good or bad feature, depending on how you view it. In our case, we don’t need the breakpoint we have set, so it actually saves us some work.

Now set breakpoints in the constructor of the Word class, at lines 11 and 17:

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

Rerun the program. It will stop after you provide the input. The debugger shows that the local String variable s contains “hello”; and the instance variable “text” contains nothing.
Now use the continue button to continue executing the program until the next breakpoint:

![BlueJ Debugger screenshot](image)

Now the debug stops at the second breakpoint and shows that the value of \( i \) is 0 and the value of \( j \) is 4. As explained in the textbook (page 233), it is clear at this point that line 17 should read “text = s.substring \((i, j + 1)\);” instead of “text = s.substring \((i, j)\);”.

So let’s stop the execution, fix the error, recompile the program, and try the three test cases again. If you don’t stop the execution, BlueJ will allow you to fix the error, but it won’t let you recompile the program. If you try, you will get the following error dialog box:

![BlueJ Message screenshot](image)

Now try the program again, as shown in the textbook, we still have problem(s). We leave it as an exercise for you to debug and fix this problem. You can refer to the textbook on Page 234.