Lab 1 – Basic I/O

Due: Tuesday, September 3rd, 10pm.

Lab Objectives:

- Continue to use the main method.
- Work with "**System.in**", "**System.out**", and "**System.err**".
- Learn about the "**String**" and "**Scanner**" classes.
- Create a program that accepts multiple lines of input and produces out from those lines and tracks information about each line.

Program Input/Output.

All programs, including Java programs, have three I/O streams, one input and two outputs, allowing the operating system to interact with the program. Java provides access to these streams using the static classes: **System.in**, **System.out**, and **System.err**. Each provides important functionality, with each having a specific use.

System.in – is the input into a program. Commonly this is from a command-line prompt, or a special text box provided by a programming environment like BlueJ. This will be discussed further in the section on the **Scanner** class.

System.out – is used to print output from the program using a method like "**System.out.println**". If you have worked with Java or Processing it is most likely that you have used **System.out.**

System.err – is probably something you have not used. It is a special output stream specifically for debugging information. It allows you to separate debugging information from the standard **System.out**. There are some other special features of this data stream that guarantee that this information is received, even if out from **System.out** is not provided.

BlueJ separates **System.out** from **System.err** allowing you to have two different streams of information. The following code and screenshot illustrate how BlueJ handles this, where the two streams are displayed in different panes.

```
public class Example1 {
    public static void main() {
        System.out.println("This is standard out.");
        System.err.println("This is error out.");
    }
}
```

In the above example main, the two different information streams are provided in the example Terminal Window.

- **System.out** allows a program to display normal operating information to a user.
- **System.err** allows a program to display error information to the user, or any information you do not want the user to miss.
- Another approach is to use **System.out** for all your output, but when you have a special debugging situation, then you use **System.err**.

String and Scanner.

Two commonly used classes are String and Scanner, one for storing character strings and the second to help manipulate strings. You probably have used String before this course. Scanner allows you to break Strings into pieces and read files as if they were a string. The String and Scanner classes are very useful and it is very important to learn as much as you can about them. (**Strings** are discussed in zyBook section 2.15 and the **Scanner** can be found in section 1.2)

The following pulls descriptions for each class from the Java API, and the titles in blue are links to those webpages. The Java API is the single most important programming language reference for Java, use it early and use it often.

String – The String class represents character strings. All string literals in Java programs, such as "abc", are implemented as instances of this class. Strings are constant; their values cannot be changed after they are created. String buffers support mutable strings. Because String objects are immutable, they can be shared.

Scanner – A simple text scanner which can parse primitive types and strings using regular expressions. A Scanner breaks its input into tokens using a delimiter pattern, which by default matches whitespace. The resulting tokens may then be converted into values of different types using the various next methods.

As indicated, the Scanner class is useful to read in information from System.in data string. The basic functionality is shown in the code below.

```
import java.util.Scanner;
public class Example2 {
    public static void main() {
        Scanner s = new Scanner(System.in);
        System.out.println("input >>" + s.nextLine() + "<<");
    }
}
```

The result of the code is to open the following Terminal Window, with the cursor in the bottom box. You will also see the inputted text “test this” is displayed twice, once for the input and then when the println displays the text. You will also notice that arrows are place on either side of the read in string from the method call “s.nextLine()”, this is to know exactly what information has been entered into the program.

Write to a File.

To write a string into a file, for the purpose of this lab, we will use Java classes **FileWriter** and **BufferedWriter** with both classes in the package **java.io.***.

FileWriter – The FileWriter class is used to write character files such as .txt files. Given a string representing the name and path of a file, we construct a FileWriter object using the constructor of the FileWriter class, taking the string as the parameter.

BufferedWriter – The BufferedWriter classes is used to write text to a character- output stream, buffering characters so as to provide for the efficient writing of single characters, arrays, and strings.

When the program finishes writing to the file, always remember to close the buffer using **close()**.

A tutorial for writing files using **BufferedWriter** can be found here, but there are some issues with it: https://www.javatpoint.com/java-bufferedwriter-class

Issues:

1. main() should not throw exceptions and when the throws keyword is used, the exception type should be better specified. The tutorial uses shortcuts here for simplicity. This will be ok for this lab assignment only. Once we learn more about exceptions this will be considered bad practice and lose points.

2. The directory is hardcoded in the example. We should be using relative directories instead to promote portability of our code. For example, by putting only a file name with no directory information, the default will be that the current project folder is used. This helps ensure that the folder being used actually exists, for example.

The Lab Assignment.

Step 1 -- Prepare a new BlueJ project for this lab:

1. Open BlueJ and create a new project and class.
2. Remove all code from the class.
3. Update the JavaDoc with your information.
4. Create a main method.

This step might seem like something that is not important, but in actuality it is how you should start all your work.

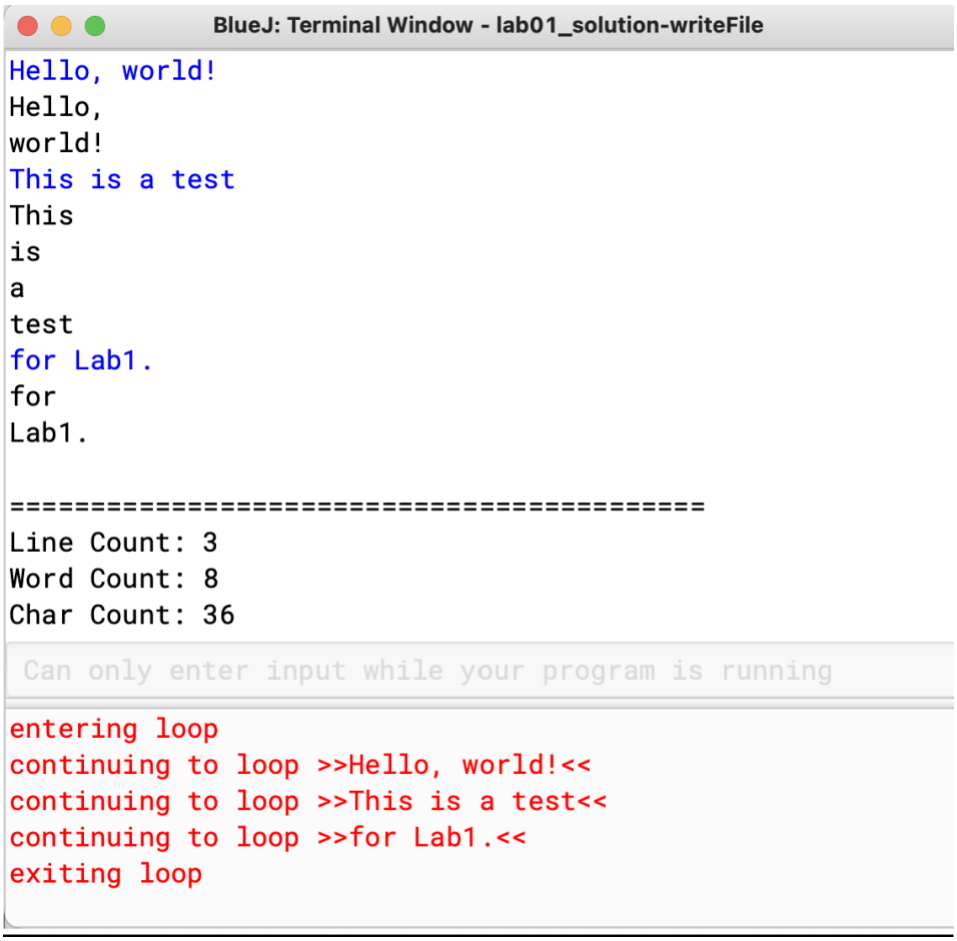
Step 2 -- Create a loop in the main method that will read in multiple lines from **System.in** using the scanner class. This will continue until a line of no characters is entered, this is done by hitting the return key without typing any characters.

1. A good loop to use is either the while loop or do-while loop.
2. Send the following information to "**System.err**":
    a. when the program enters the loop;
    b. when the program leaves the loop;
    c. the string that is entered using "**System.in**".
3. Using "**System.out**" display each word in the entered string to a separate line in the terminal.
4. You need to use two Scanner variables to do this work.

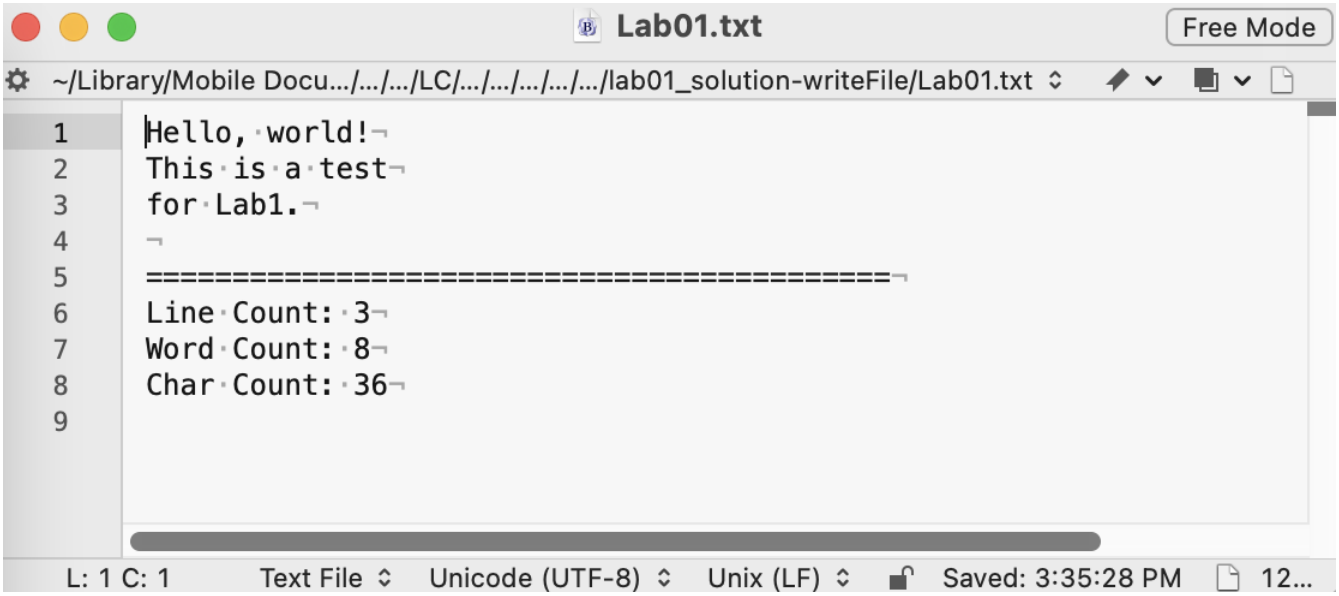
Step 3 – Add counters to track the number of lines, words and characters entered. Then print this information at the end of the program.

Step 4 – Create a **FileWriter** object and a **BufferedWriter** object that will write all the user input as well as the program output, but not the errors, into a file named "Lab01.txt".

A successful run of the program should look something linked the screen shot below.


BlueJ: Terminal Window - lab01_solution-writeFile
Hello, world!
Hello,
world!
This is a test
This
is
a
test
for Lab1.
for
Lab1.
==========
Line Count: 3
Word Count: 8
Char Count: 36
Can only enter input while your program is running
entering loop
continuing to loop >>Hello, world!<<
continuing to loop >>This is a test<<
continuing to loop >>for Lab1.<<
exiting loop

The content of the file created by your program should look like the screen shot shown below. Note that if you run your program multiple times, the file content might change based on your input. (The program used for the screenshot here includes grey icons for spaces and line breaks, which you may ignore.)



Submitting the lab and grade breakdown.

To submit the lab: zip the directory that contains your BlueJ project and upload it to the submission item on Moodle for Lab 1.

Grade breakdown:

Program complies	required for a grade
Well organized code	2pts
Good comments	2pts
Correct use of the Scanner class	2pts
Correctly looping and string information	2pts
Correctly counting lines, words, and characters	2pts