LAB 03 – Beginning Java

Introduction:

In today's lab you will be tasked with learning about some of the basic classes. You will also explore basic errors that can occur during the compilation of a Java program. Your first task will be to read though and understand a buggy program. Following this you will need to find the bugs based on the compiler's output and fix the program. Your final task will be to create a simple program using the knowledge and classes you learned about earlier in the lab.

Set Up:

Before beginning your lab you will first need to set up a directory to hold the files for this lab. The first step in this process will be to verify that you are in your home directory. Using the commands that you learned about in past two weeks navigate to your home directory.

```
$ cd ~
```

If you wish to verify that you are in the right directory you can use Unix's print working directory command (`pwd`) which will print the current directory which the terminal is "working" in. Running the command will look similar to the following.
NOTE: your_login will be replaced with your Purdue career account login

```
$ pwd
/homes/{your_login}
```

Now that you have reached your home directory, you need to move yourself into the `cs180` directory that created in your first lab. Using commands learned in the previous weeks, navigate to this directory now.
NOTE: If you do not have this directory, you will need to make it using commands learn in the previous weeks

```
$ cd cs180
```

Finally you will need to make a lab03 directory and navigate to that folder. Using the commands learned in the previous weeks, do this now.

```
$ mkdir lab03
```

```
$ cd lab03
```

Obtaining the files:

Your class TA will provide you with a location from which to download the files for this lab. These files downloaded will be `BuggyWelcome.java` and `CurrentTime.java`. To obtain copies of these files and store them on your computer you can use the `wget` Unix command that you were taught last week, or you can use your web browser of choice to navigate to the provided location, and download them into the `lab03` directory that you just created.

Introducing DrJava:

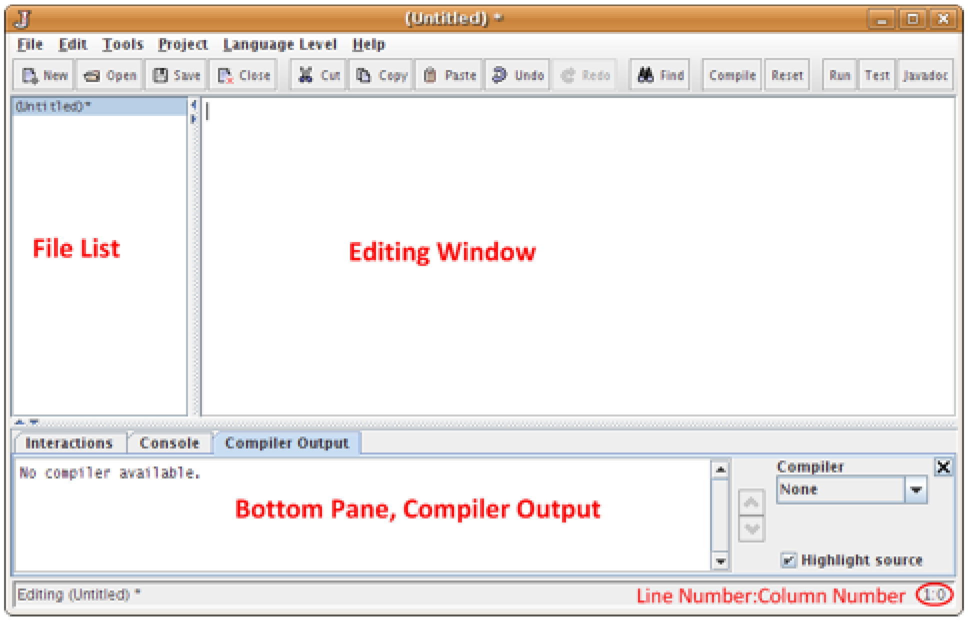
Some of you may not like writing and compiling all of your code through the terminal. If you are one such person, there are alternatives available. DrJava is a simple IDE (Integrated Development Environment) developed in Java for Java. Dr Java can be downloaded as a simple executable for OS X, Windows, and also Linux from the main site http://www.drjava.org/. If you set up Dr Java on your own computer, you will need to download the latest version of the Java JDK from http://java.sun.com/javase/downloads/index.jsp. Macintosh computers have the Java JDK pre-installed.

NOTE: IDEs are designed to be an all in one solution for development of programs. They often include ways to compile your code, include features such as Auto-indentation, and can offer a wide variety of other tools.

To access Dr Java on the linux machines, you will simply need to type `drjava` into the terminal. If you would like to open a file(s) in Dr Java, you can put the file name(s) after.

```
$ drjava Src.java
```

Below is an example of what DrJava looks like.


(Untitled) *
File Edit Tools Project Language Level Help
New Open Save Close Cut Copy Paste Undo Redo Find Compile Reset Run Test Javadoc
(Untitled)*
File List
Editing Window
Interactions Console Compiler Output
Bottom Pane, Compiler Output
Compiler
None
Highlight source
Editing (Untitled) *
Line Number:Column Number
1:0

No compiler available.

Exploring BuggyWelcome.java:

The first task to this lab will to look over and examine the provided Java source file `BuggyWelcome.java. BuggyWelcome.java,` if it was written correctly, would perform the simple task of greeting the user with the message: "`Welcome to Buggy Welcome.`" Following this, the user will be asked to enter a string through a JOptionPane, after submitting this string the user will be prompted to enter a string through the terminal. Finally the program will print out via the terminal the two messages the user provided.

Now that we know what `BuggyWelcome.java` is intended to do, open `BuggyWelcome.java` in your editor of choice (DrJava, vim, gedit, pico, etc.) so that we can begin exploring the source line by line.
NOTE: Some of the concepts discussed may be unfamiliar to you. The ideas will be explained later in this class; however it is good to familiarize yourself with the terminology now.

Line 1 – 2:
```
import java.util.*;
import javax.swing.*;
```
These two lines are what are known as **imports**. For many common tasks in programming, code has often already been created to perform those tasks. To access this pre-created code, you need to tell the Java compiler where to find that code. In this case, we are telling the compiler to search for the classes used within the code in the java.util library, and within the javax.swing library.

Line 3:
```
/*
```
This is the indicator for the beginning of what is known as a block comment. Comments are sections of the source code which are put in place by the programmer for future use of themselves or another programmer examining the code. It is ignored by the compiler A block comment is a comment that will span multiple lines

Line 4 – 6:
```
#name: your name here
#class: BuggyWelcome.java
#project: cs180_lab03
```
These next three lines are the body of the block comment. Since these lines appear after the opening indicator but before the closing indicator, these lines will be ignored by the compiler and are there for the programmer's information. **NOTE**: the # symbol at the beginning of each line is a formatting choice; they in no way affect how the complier sees this portion of the file.

Line 7:
```
*/
```
This is the closing indicator for the block comment. This indicates to the compiler that it should now begin paying attention to any code following this symbol.

Line 8:
```
public class BuggyWelcome
```
This is the class declaration. You will learn more about this later, one thing to notice however is that the class name (`BuggyWelcome`) is the same as the file name.

Line 9:
```
{
```
This is an opening bracket. In programming this is used to indicate the beginning of a section of code. In this case, the section of code is the class. Everything between this and the bracket on the same indentation is part of the class.

Line 10:
```
    public static void main(String [] args)
```
This is the declaration of the main method. A method is a portion of code which is run by the computer when called upon. The main method is special in that when a program is run, this is the method that will be called first. You will also notice that the line is made up of parts. Lets start at the end of the line first. `(String [] args)` this portion of the line specifies what the arguments to the method will be in this case it is something known as a String array (indicated by the `String []`) called `args`. Next is the word `main`, the method name, and is used when calling the method. Next is the word `void`, this indicates what the return type will be, in this case nothing. Finally there are two words `public static`, these specify specific attributes of the main method, and will be covered at a later date.

Line 11:
```
    {
```
This is another opening bracket. This time the bracket indicates that this is the start of the main methods portion of the code. Everything included in the after this bracket but before the closing bracket is part of the main method.

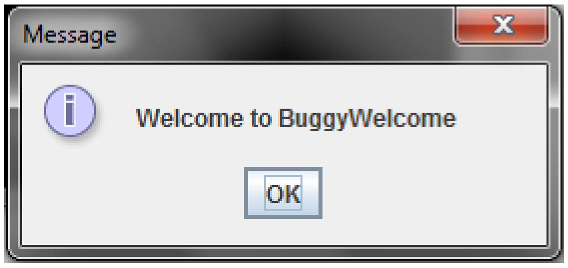
Line 12:
```
        //Create Scanner Object
```
This is what is known as an inline comment. The `//` indicates that anything following this symbol should be ignored by the compiler. Unlike the block comment an inline comment does not have an ending symbol. Anything following the // on the same line is considered to be the body of the comment

Line 13:
```
        Scanner s = new Scanner(System.in);
```
On this line we are creating a new instance of the `Scanner` class and calling it `s`. The `Scanner` class is a class used by programmers to read what is known as an input stream; in this case we are using the terminal. If we begin at the front of the line, the first thing we see is `Scanner`. This is specifying the type of object that we are creating, in this case a `Scanner` object. Next you will see `s,` This is the name of the Object that we are creating. Following this is `=`, which is known as the assignment operator. `Scanner s,` is not the actual object, instead it is better to think of it as a container in which you can put an object of type Scanner.

When we use the = operator we are filling that container with whatever is coming after. Next on the line is the word `new`. This indicates that we wish to create a new instance of an object. Finally we have `Scanner(System.in);` This portion of the code is indicating what type of instance we are creating. In this case an Instance of the `Scanner` class where we are passing the argument `System.in` to the Scanner's constructor.

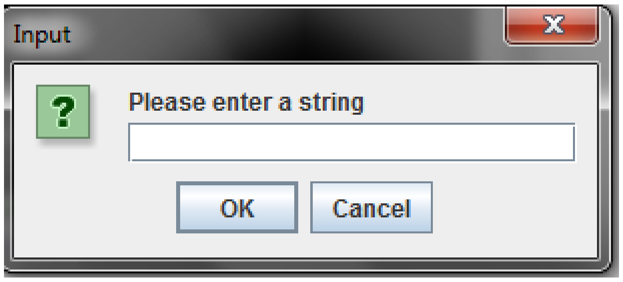
Line 14, 16, 18, 20, 22, 24:
These lines contain more inline comments

Line 15:
```
        JOptionPane.showMessageDialog(null, "Welcome to
BuggyWelcome")
```
This line is calling the `showMessageDialog` method of the `JOptionPane` class, and is sending the parameters of `null` and the literal string `"Welcome to BuggyWelcome".` This function is used to create a Graphical User Interface (GUI) which displays the provided string to the user. It should look similar to the image below. For now do not worry about the `null` parameter.



Welcome to BuggyWelcome

Line 17:
```
        string guiResult = JOptionPane.showInputDialog(null,
"Please enter a string");
```
This line will create a container of type `String` called `guiResult` and will store within that container, using the assignment operator, the return of `JOptionPane`'s function `showInputDialog`. `showInputDialog` will display a GUI with the provided string, and also with an area to input text. It should look similar to below.



Please enter a string

Line 19:
```
        System.out.println("Please enter a string: ");
```
This line is calling the `println` method with the string literal parameter "`Please enter a string: `". When this line of code is run, provided string will be printed to the terminal.

Line 21:

```
        String terminalResult = s.nextLine();
```

This line is calling the method `nextLine` on the instance of the `Scanner` which we created earlier and called s, and storing that result of that call into the String container `terminalResult.`

Line 23 & 25:

```
System.out.println("The following string was entered in the
JOptionPane: \n" + guiResult);
System.out.println("The following string was entered in the
terminal: \n" + terminalresult);
```

These two lines are printing the results of the program to the terminal. This is similar to the string we printed in Line 19 in that the method `println` is called, however there are a few items of note in these lines. First, within the string literals, the sequence of characters "`\n`" appears. This sequence is one of a group of similar sequences called escape characters. Each escape character has a special function; in this case, \n causes the terminal to go to the next line of output. The next item of notice is that after the string literal there is a + followed by 2 one of the String container names we created earlier. The + symbol is the addition operator and will add the second string onto the end of the first. By supplying the name of a String container, we are able to access the string stored inside of the container.

Debugging BuggyWelcome.java:

Now that we have examined BuggyWelcome.java, it is time to try and compile it. If you are editing the file in DrJava, press the Compile button. If you are editing the program in another editor such as vim, compile the Java program using the following command.

```
$ javac BuggyWelcome.java
```

You should get and output with lines that contain information similar to the following.

```
BuggyWelcome.java:15: ';' expected
```

By looking at this output, we are able to tell what the error is that the Java compiler has found. In the example above, we can see first that there is an issue in the file BuggyWelcome.java. Next we see a number, In this case 15. This number indicates the line number on which the error was found. Finally we are given a description of what the error is thought to be. The above error says that a semicolon is expected. Examine line 15 in the code, should there be a semicolon? If so, fix it.

When you first compiled the program, there should have been a total of 4 errors. The next three errors all say that they are on line 20, and they all say that a semicolon is expected. Examine Line 20, does it appear that 3 semicolons are needed? Or perhaps is there something else that should be changed/added to this line. Fix this line.
NOTE: Although the Java compiler can give a good idea of what is wrong and where the issue is

occurring, it cannot always tell you the exact issue. Using the output however we able to examine the code around the error for bugs that the compiler cannot detect on it's own.

Now that these two lines are fixed, let's try and compile the program again. Oh No, we get a couple more errors. It's almost like someone designed it this way. This time, the two errors should display something similar to the following 4 lines.

```
BuggyWelcome.java:17: cannot find symbol
symbol  : class string
BuggyWelcome.java:25: cannot find symbol
symbol  : variable terminalresult
```

Examining these errors, we immediately can tell that the errors are occurring on lines 17 and 25. Let's examine the error on line 17 first. The error provided to us is cannot find symbol. Now on it's own, this error message tells us very little. However when we examine the second line we get more information. First we see `symbol  :` This indicates that what follows will be the symbol from the first line. Next we see `class string`. This indicates that the compiler could not find a class of type "`string`". Why would the compiler have thrown this error? Is there something wrong with "`string`"?

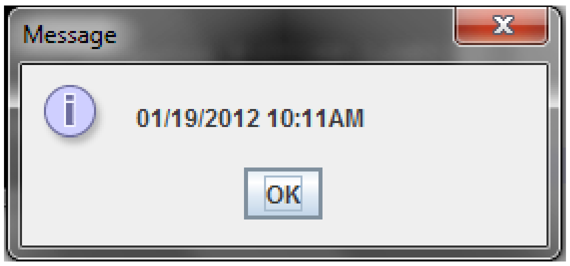
Finally let's look at the error on line number 25. This error again cannot find symbol. When we look at the second line, we see similar output to the error on line 17 except now the second portion of the line states `variable terminalresult`. Using the same pattern that we used on the error on line 17, we see that the compiler was looking for a variable (variable is the commonly used name for what I have been calling containers) called `terminalresult`. Is there an issue with the name terminal result? Is there a variable (container) created earlier in the program which is similar? Fix these two errors, and then compile the program. The program should compile, and you should be able to run the program using the run button in DrJava, or by running the following command in the terminal.

```
$ java BuggyWelcome
```

Writing A Java Program:

Now that you've looked through and fixed the `BuggyWelcome.java`, it's time to write a simple program. The program that you will construct will be a simple application to print the current time using a `JOptionPane`. From class, you should remember the `Date` object. When you create a new `Date` object, the object will be initialized with the current time. Once you have created your `Date` object, you will need to output your `Date` to the `JOptionPane`. Remember, these tasks can be performed with the `JOptionPane`'s `showMessageDialog` and `Date's toString` methods. Write the required lines of code into the main method of the provided `CurrentTime.java`.

After you get the initial code to compile, we now want to format the Date into the format of our choice. This can be done with the SimpleDateFormat (Also introduced in class). Format the date object now using the format string “MM/dd/yyyy hh:mmaa” The final result should look close to the following.



```
01/19/2012 10:11AM
```

Turn In:

Before turning in your labs, please remove all class files from your lab03 directory. To do, first verify that you are in the lab03 directory, if you are not in this directory, navigate there now.

```
$ pwd
/homes/your_login/cs180/lab03
```

Now remove all class files.

```
$ rm *.class
```

Finally move to lab03’s parent directory, and submit your code via turnin.

```
$ cd ..
$ turnin -v -c cs180=XXX -p lab03 lab03
```

Check the output to make sure BuggyWelcome.java and CurrentTime.java were both submitted

Note: the XXX stands for your section id. Use the following table to determine it:

Lab Day	Lab Time	Section-id to use in the turnin command
T	9:30	L01
W	9:30	L03
W	11:30	LM3
W	1:30	L02

F	11:30	LM2
F	1:30	LM1
F	3:30	L04

Grading:

You will be graded on the following items.

- Fix BuggyWelcome.java 50%
    - Each Bug is worth 10% (4 bugs)
    - Program successfully compiles 10%
- Code CurrentTime.java 30%
    - Create Date Object 10%
    - Output current time using JOptionPane 10%
    - Date is formated 10%
- Successful turnin 20%