Programming Exercise 1:
A First Program

Purpose: Introduction to Java programming, using the Eclipse Integrated Development Environment

Background readings from textbook: 1.7-1.10, 1.12, 2.2, 2.4-2.5

Due date for section 001: Monday, January 18 by 10 am
Due date for section 002: Wednesday, January 20 by 10 am

Overview:
In this assignment, you will familiarize yourself with the basic steps of creating a Java program using the *Eclipse* IDE. You will enter, compile, debug (if necessary), and run a simple program. You will then modify the program several times, compiling, debugging and running it as you modify it. For this assignment, you will need to understand the basic structure of a 1-method Java program, variable declarations and output statements using System.out. First, we look at using Eclipse. If you are using your home computer, you can download a free copy of Eclipse from the Eclipse website at http://www.eclipse.org/downloads/.

Part 1: Starting a Java Program in *Eclipse*
Start Eclipse. You will see a selection box called the "Workspace Launcher" (Figure 1) which will allow you to set the directory for your workspace files. You can set this location as the default so that this window does not appear in the future, otherwise it will appear every time you start Eclipse. Use the **`Browse`**... button to select an appropriate location. If you are using your home computer, set up a workspace directory that you will remember. If you are using a campus computer, use a flash drive. You should use the same workspace for all of your programs this semester both for convenience and to easily access them later in the semester if you need to see how you accomplished something.

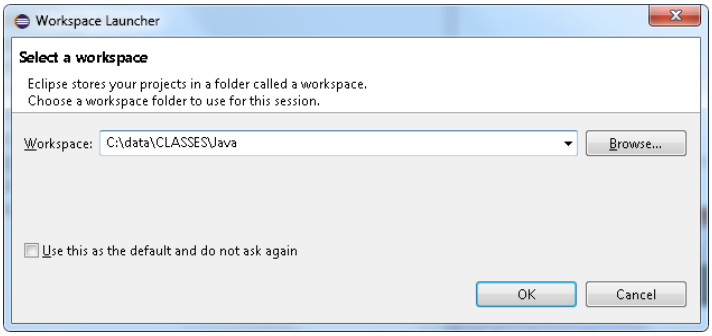

Figure 1 Select your workspace

After setting up your workspace, you will see a generic *Eclipse* Welcome window the first time you use Eclipse. The actual window may vary depending on which version you installed, but it will look something like what is shown in Figure 2. If you have already used Eclipse since installing it, this step is skipped and you will see a project interface window (see figure 3). Close the Welcome window

by clicking the X in the Welcome tab. You can return to this Welcome window by selecting **`Welcome`** from the **`Help`** menu.

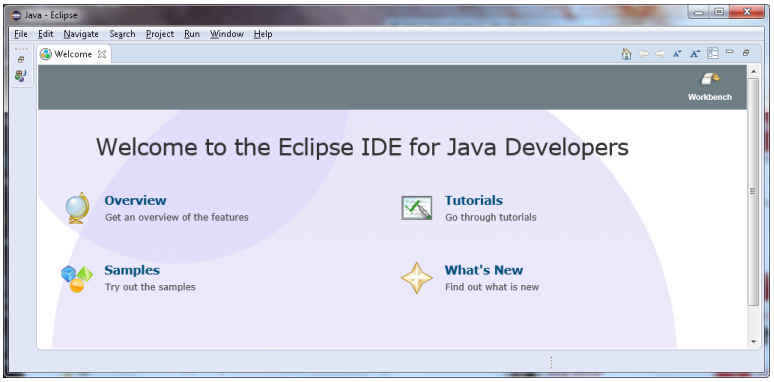

Figure 2: Eclipse welcome page.

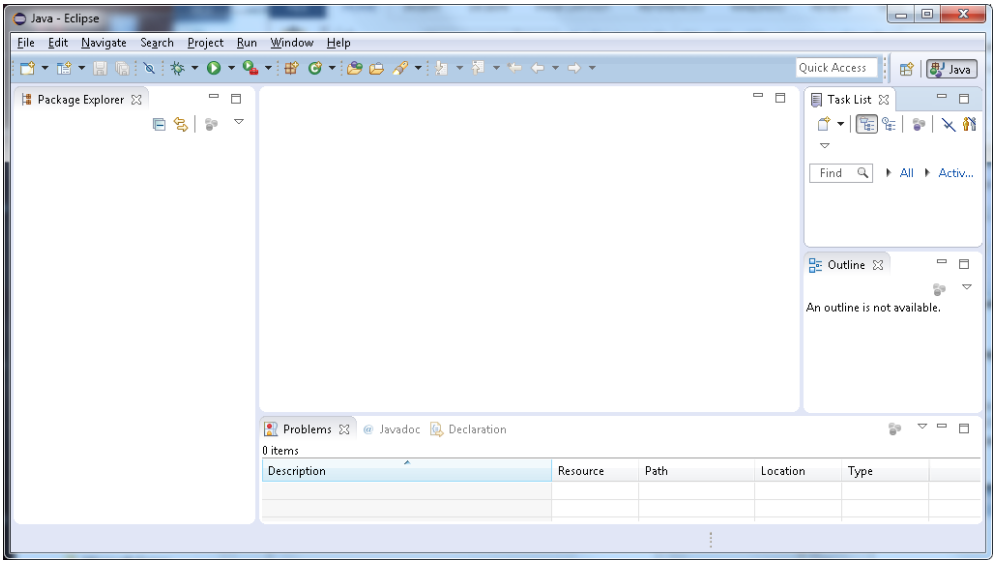

Figure 3: Eclipse Project Interface Window

Before creating Java programs in Eclipse, you need to first create a project to hold the files that are associated with that program. For most of this semester, we will only create a single file, but it still must be placed inside a project. To do this, we create the project first.

From the Eclipse Project Interface Window, create a new project by selecting **`New`**... from the **`File`**... menu and then **`Java Project`** from the submenu. This brings up a New Java Project pop-up window, as shown in figure 4. From here, you will see the default location (workspace). You can change this by de-selecting the **`Use default location`** checkbox but since we just set this up, we will keep it as is. Type in your project name. The name can be anything but it should be meaningful. Use **`HelloWorld`** for this assignment (spell it exactly as shown here). Under **`JRE`** (Java Run-time Environment), you should not have to adjust anything but if you need to, you would select the appropriate button (e.g., Use default JRE) and adjust the drop down box or click **`Configure JREs`**... under **`Project`** layout, leave it as it defaults. There are no working sets to add so you would leave that blank. Finally, click on **`Finish`**.

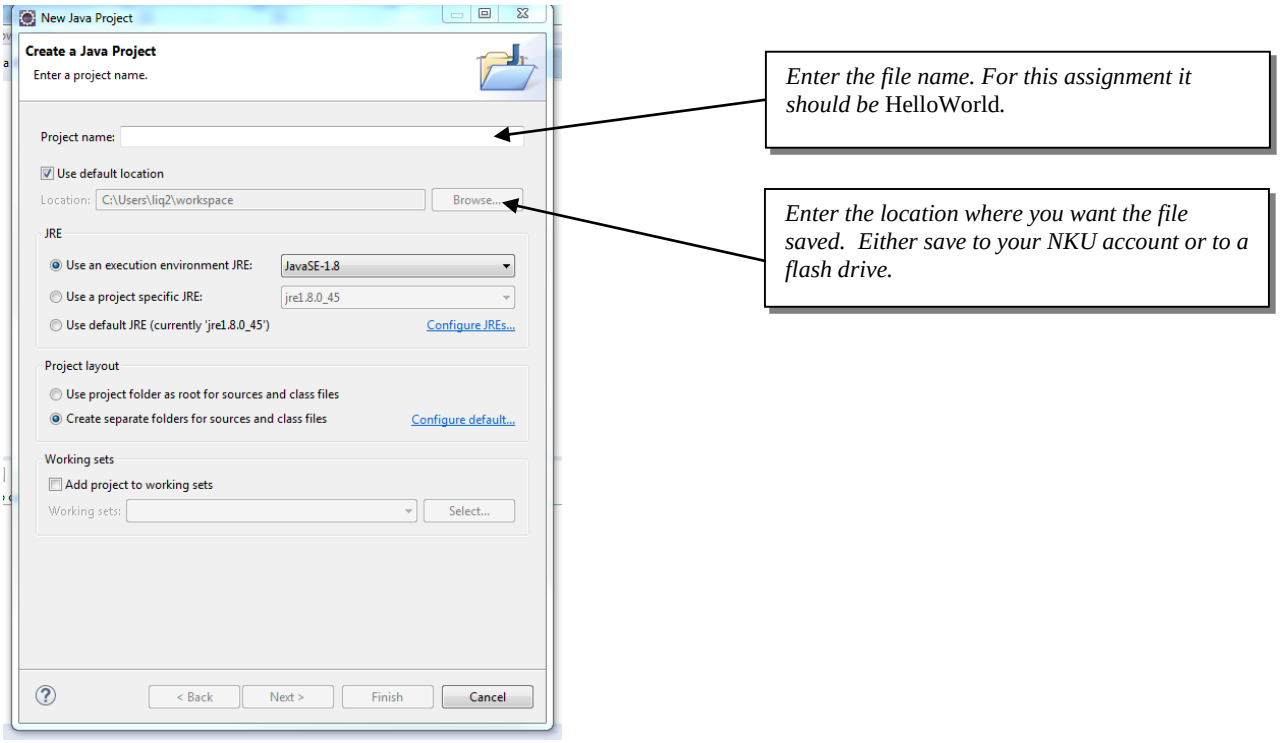

Figure 4: New Java Project Window

Now let's explore the entire Eclipse Project Interface window. You should see in the leftmost pane the Package Explorer. There is one package, named HelloWorld. If you click on the triangle to expand this, you will see two items, src and JRE System Library. The src item is where your Java source code is placed. The JRE System Library lists all of the jar files that are available. Figure 5 shows this list expanded. The upper right pane is where you will edit your source code. The lower right pane is where output (and any error messages) will appear.

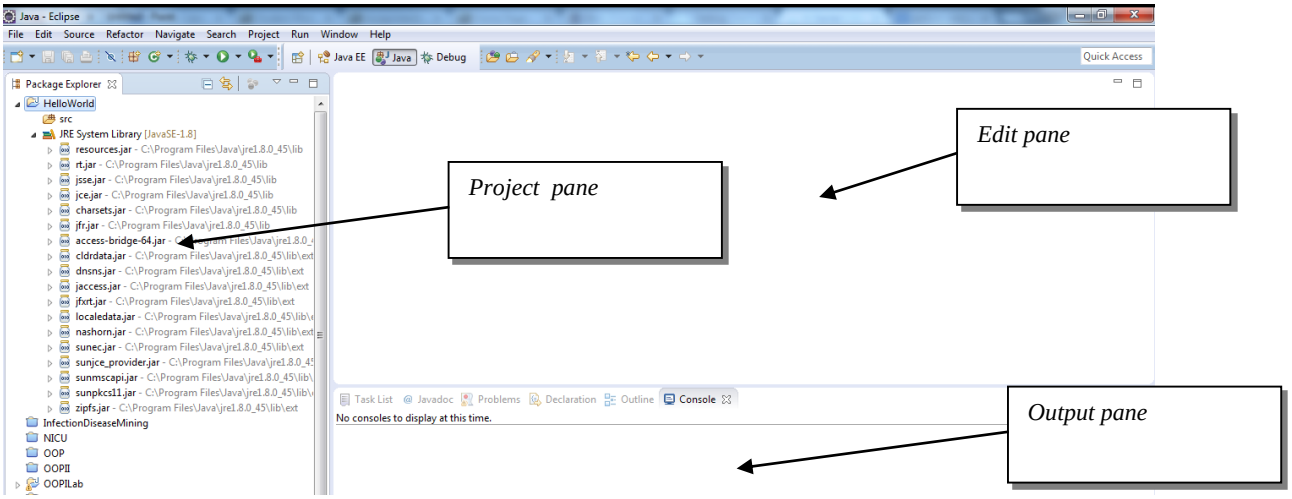

Figure 5: Java Project Interface Panes for HelloWord Project

At this point, we will begin to write our first program.

Part 2: Writing, Compiling and Running a Program

We have a project but now we need at least one file. Through most of this class, our programs will consist of a single file which itself will store code which includes a main method. The main method is the first set of code run whenever you run any Java program. For several weeks, we will only use main methods. Later in the semester we will write additional methods.

To create our file, select **`New`**... from the **`File`** menu and **`Class`** in the submenu. This brings up a New Java Class pop-up window, as shown in figure 6. In this window, you see the folder which stores this file, and any package that we want to place this file in (we won't be doing that in this class). Beneath this, we find several selections. First, we have to enter a name for this class. Use **`HelloWorld`**. Do we want to make this class public, package, private or protected? We want to use public. We can also make the class abstract, final or static. We will not use any of these. We can enter this class' parent class (superclass) if we want to use inheritance. We will cover this at the end of the semester. Without specifying a parent, the class automatically uses the Object class as the parent; we don't need to change that setting. We can also select a method stub to create. In our case, we want **`public static void main(String[ ] args)`**. We can click this checkbox or we can write it ourselves. In this case, we will use this automated feature, so select the checkbox for **`main`**. Finally, click on **`Finish`**.

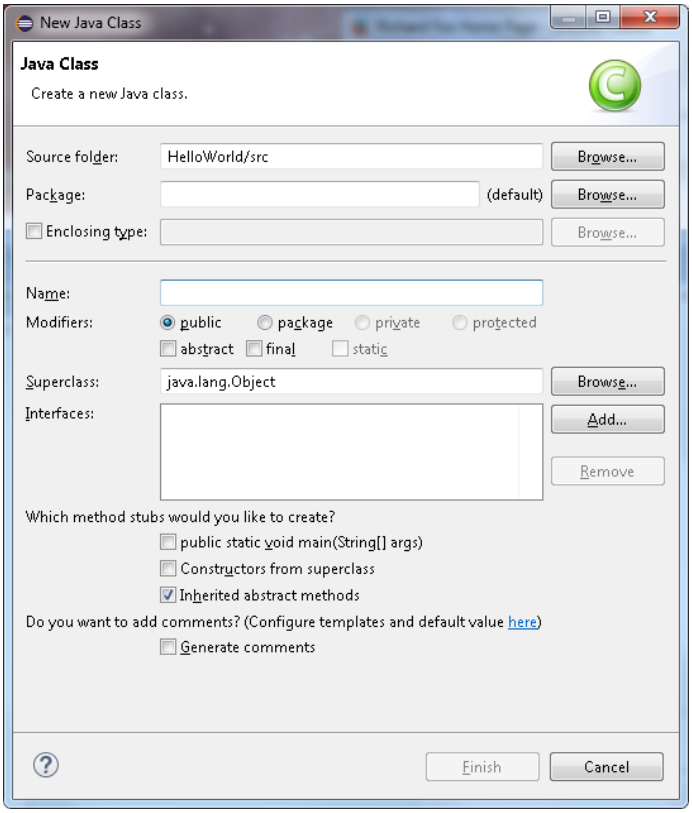

Figure 6: Creating a Class

Creating the class causes Eclipse to automatically generate some of the code for us. This appears in the Edit pane in a window labeled with the name of our file that we just created, HelloWorld.java

(this is the file containing our class, the class is named HelloWorld). See figure 7 which shows the edit pane.

Figure 7: Automatically Generated Code for New Class

Where you see **`// TODO Auto-generated method stub`**, you will add your own code. The notation **`//`** means that what follows is a comment. Comments are ignored by the compiler and are there for us humans to read. Delete that comment. Enter what you see below exactly as shown except that you need to fill in what appears in italics (such as your name, the section number and the date). The items that appear between /* and */ notation are also comments. The rest is the code. Notice that you do not have to type **`public class HelloWorld`**, **`{`**, **`public static void main(String[] args)`**, **`}`** or **`}`** yourself, that is already there. Fill in the rest.

```
/*
   Author: your name
   Course: CSC260L lab section
   Date: today's date
   Assignment: #1
   Instructor: your instructor's name
*/

public class HelloWorld
{
    public static void main( String[] args )
    {
          System.out.println( "Hello World!" );
          System.out.println( "Welcome to the fun world of Java programming." );
          System.exit(0);
    }
`
```

Notes:

- The class name **must** match the file name (excluding the .java part) otherwise you will receive an error.
- The last instruction has a zero in parens.
- Spell all the words exactly as shown, and use the same capitalization. Java, is a *case sensitive* programming language.
- Use the same punctuation including the parentheses, quote marks, semicolons, and brackets exactly as shown.
- Spacing does not have to be as shown above, but *Eclipse* automatically indents for you, so you should stick as closely to the above as possible.

What does this program do? Not much. The executable code appears between the inner { } and consists of two System.out.println statements and a System.exit(0) statement. The println statements output anything found inside of the ( ) while the exit statement causes the program to terminate. This program, like many that we will write early in this semester, has a single class (HelloWorld) with a single method (main).

Save your program (select **Save** from the **File** menu or select the **Save button** on the button bar, it looks like a floppy disk, it should be the third from the left. Before you can run your program, it must be compiled. We can compile it and then run it, or it will automatically compile when you try to run it. So we will just run it. You can run it by selecting the **Run button** (it's a white triangle pointing to the right in a green button), or by selectin **Run** from the **Run** menu, or by **right clicking** on **HelloWorld.java** in the **Package Explorer pane** and selecting **Run As** and from there **Java Application**. If you had not saved your file, you will be asked if you want to do so now. Eclipse will first attempt to compile your code. Assuming there are no errors, the compiled program will then run. If there are errors, you will receive syntax error messages. We explore syntax error messages and how to find and fix them in the next section. For now, if you have any errors you should see the lines that contain erroneous instructions selected with a red circle and an **x** off to the left of the instructions in the edit pane. Fix any errors you can by referencing the program above and save and compile it again. Once all errors are fixed, run it.

NOTE: if you prefer to compile the program first before it runs, from the **Project** menu select **Build Automatically** (which is checked) to shut that feature off. Now, you can build (compile) when you want by selecting **Build Project** from the **Build** menu. Don't forget to save your file first after making changes and before compiling. If you leave **Build Automatically** on, then you don't have to worry about compiling as your program will be compiled automatically every time you try to run it.

Part 3. Debugging

There are three sources of errors for any program. The first is known as a *syntax* error. This error arises because there is something syntactically invalid with your program code. There are many reasons for having syntax errors such as misspelling words, forgetting to declare variables, having incorrect punctuation marks, or a common error in Java is not naming the file the same as the class. The second source of error is called a *run-time* error. This error arises after successfully compiling the program but while it is executing something went wrong. For instance, if the program expects the user to input a number and the user types in a name instead, this results in a run-time error. Because we do not know what the user will enter when we compile the program, this error is not caught until running the program, thus it is a run-time error. Another source of run-time errors is asking the computer to do something it cannot such as divide by 0. The final source of error is the hardest to catch and fix, a *logical* error. Here, your logic is incorrect. With very complex programs (consisting of hundreds of thousands or millions of instructions), finding logical errors is one of the biggest challenges that programmers will ever encounter. A logical error might be as simple as adding when you meant to subtract but it can also be an infinite loop so that your program never terminates. Here, we will explore syntax errors only but you will come across run-time and logical errors throughout the semester and quite likely your entire career.

Return to your Edit Pane and change the first word from **`public`** to **`Public`** as shown below.

```
Public class HelloWorld
```

Compile your program (try to run it). You will receive an error and your environment will look something like that of figure 8. Notice in the figure three things. First, next to HelloWorld.java is an X indicating an error arose. The error is in line 9. We see an X next to 9 and we see the word Public underlined in red. Finally, in the Output pane we see a message indicating an Unresolved compilation problem in HelloWorld.main(HelloWorld.java:11). The "Unresolved compilation problem" is the syntax error message. In this case, the error message is not helpful. Some messages are very helpful others are not. We are told that the error exists in the main method of HelloWorld at (or near) line 11. In fact the error is in line 9 (at least in this figure). The error is that we misspelled the word public as Public. Change it back and in a few moments you will see the red underline disappear and the X disappear. You can then run the program and upon compilation, you will see the X next to HelloWorld.java disappear and the error message in the Output pane be replaced with the program's output.

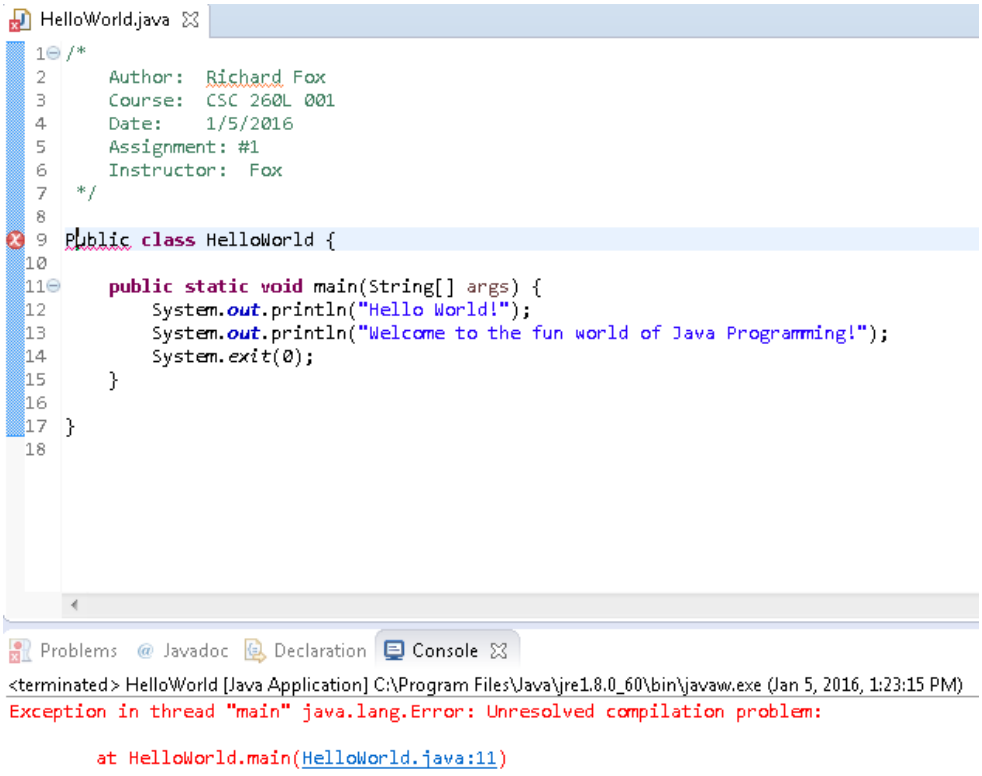

Figure 8: A Syntax Error

Let's explore some additional sources of errors. In the same line (public class HelloWorld), change the name of the class from **`HelloWorld`** to **`Hello`**. Try to run (or compile) your program. What error message did you get? It is asking for the class HelloWorld because that is the name of the file. The class it found was not named what was expected. This is an error fairly unique to Java but a common error. Correct the name. Now, remove the ; at the end of the first println statement. Compile your program. In this case, you are told that you need to insert a ; in the line where the error arose. Although it does not tell you where to insert the ; as a Java programmer you should know to end each instruction with one. Fix this error. Now remove the last } at the bottom of your program. Again, the error message tells you how to fix it, insert a } to complete the ClassBody which means the class code. For another syntax error, remove the close quote mark from the first println statement (the

quote after **world!**). Again, compile/run your program. What error message do you get? The message given is less instructive than the last couple but you should still be able to make some sense out of it. Replace the quote mark before continuing. We will try one more syntax error. Remove the word void (between static and main). Again, compile/run your program. The error message this time may make no sense at all but it outputs the line that it expected to see. What is missing is the word void. Replace it.

Let's explore a minor logical error. Your program contains two **System.out.println** statements. Change the first statement to **System.out.print** (remove the **ln** from println). Compile and run your program. No syntax error. Why is there a logical error? If you look at the output, you will see that the two output lines are now on one line. That in itself is not an error, but there is no space between them. You can fix this error by adding some blank spaces inside the quote marks either after the ! in World! or before Welcome. For instance, our first statement could become:

```
System.out.print("Hello World!  ");
```

What is the difference between print and println? With println, after the output a new line character is placed so that the next output statement will output on a new line. The print statement does not output the new line character so the next output statement will appear on the same line. By using a print instead of a println, while the output of the program is not incorrect, it does not look nice. Change print back to println before you continue.

Part 4: Program Enhancements

We will enhance your program now to make it slightly more interesting. We will do this by adding variables and assignment statements. A variable represents the name of a place in memory to store information. In Java, every variable must be *typed*. To provide a type, you declare the variable before you use it (this differs from some other languages like Python). A declaration is simply the type followed by the variable you are declaring. For instance, an int variable (an integer) is declared as **int *name;*** where *name* is the name you are giving to the variable. Types in Java include int (integer numbers), float (floating point or real numbers), double (floating point numbers with more precision), char (single characters like a letter, a digit or a punctuation mark), String (note the capitalization, to store a group of 0 or more characters) and boolean (to store either true or false). There are other types but for now, we will be dealing with these.

You can declare more than one variable at a time if they are the same type by separating each name by a comma as in **int x, y, z;** and **String firstName, lastName;** Notice the spelling we used for the two Strings. This is called *camel notation*. We can either start each new word with a capital letter or insert an underscore (_) between names like income_tax or income_Tax. We typically use camel notation instead.

Once we have a variable declared, we can use it. We can assign it a value through an assignment statement, or input a value into it. If it has a value, we can use it to assign another variable or to output it, among other things. For now, we will use very simple assignment statements of the form ***variable = value;*** The *value* must match the proper type. For instance, from above, we could do **x = 1; y = 5;** and **z = 10000;** but we could not do **x = 1.123;** or **y = "Frank";** Note that spaces are not needed in our assignment statements. We could just as easily do `x=1;` instead of `x = 1;`

To output a value, use a System.out.print or System.out.println statement. Our previous two print/println statements output only *literally* characters (items placed in quote marks). If we want to output a variable, we do not use the quote marks, as in **`System.out.println(firstName);`** or **`System.out.print(x);`** What if we want to print multiple things such as **`Hello`** and **`firstName`**? For such an output, we must *concatenate* the items together. In Java, Strings are concatenated together by using the + before each String item. For instance, we can have our output appears as the word Hello followed by the contents of the variable firstName as **`"Hello " + firstName`**, giving us the instruction **`System.out.println("Hello " + firstName);`** Notice the blank space after the 'o' in Hello before the close quote mark. Why is it there? You will explore this below.

Add to your program two variable declarations, the **`String name`** and the **`int age`**. Assign them both your values (for name, use your first name and for age, use your age such as **`age=18;`**) Place each variable declaration prior to the assignment statement that gives that variable its value. Now, modify your first System.out.println statement so that it outputs Hello and your name. Add another System.out.println statement after this and before the Welcome message that says "You are ___ years old" where ____ is your age. Note: this statement will require two + signs, one before age and one after age. See if you can figure out how to do this. If you can't, ask your instructor for help.

Compile and run your program. Once successful, move on to the last part.

Part 5: Additional Enhancements

With a print statement, as we explored above, no new line character is output. You can output a new line character "by hand" by outputting the character **`\n`**. For instance, **`System.out.print("Hello " + name + "\n");`** is the same as **`System.out.println("Hello "+name);`** because the first statement, while being a print, also outputs \n to end the line, causing a new line to appear. Modify your program by including another String variable called message. Create whatever message you like. Make sure your message is no longer than 80 total character. Replace the **`Welcome to the`**… output statement with the word **`Done`**. Now, add a System.out.println statement that outputs literally **`The message of the day is`** followed by the value stored in your variable message. In this println statement, add two **`\n`**'s before "The" and add two **`\n`**'s after message is output. The result will be two blank lines before and after the message. Your output might look like this:

Hello Richard
You are 51 years old

The message of the day is Information is not knowledge, knowledge is not wisdom!

Done

Once done modifying your program, compile and run it. If it does not compile, fix any errors you find. If it does run but the output does not look right, fix it and try again. When done, run your program and copy the output from the Output pane.

At the very bottom of your program in the Edit pane, after your last } press the **enter** key add **/***, press the **enter** key and **paste your output there**, and then add one more ***/**. This will put your output in a comment at the bottom of the program.

To submit your program, do one of the following:

- **Print** your source code including your output as inserted in comments (**Print** from the **File** menu) and hand the printout in to your instructor *or*
- **Email** your source code file to your instructor (foxr@nku.edu). You will have to locate the source file to attach it to the email. It will be beneath the directory you specified as your working directory, under the subdirectory of the name of the project (HelloWorld) in a subdirectory called **src**. In this case, the program is called **HelloWorld.java**.

Part 6: Common Pitfalls

In each lab, we will see some common sources of syntax, logical or run-time errors to help you better learn what not to do. Here are some pitfalls for this lab.

1. The class name does not match the file name (HelloWorld and HelloWorld.java in our case).

2.

```
public class HelloWorld;
{
       ...
}
```

Syntax error!

Do not *place semicolons at the end of class or method declarations. This may give more than one error about "illegal start of expression" or "unclosed literal."*

3.

```
public static void main( String[] args )
{
       System.out.println( "Hello World!" );
       System.exit(0);
```

Syntax error!

All opening quote marks, opening parentheses, and opening braces must have a closing partner. In this case, there is no } at the end

4.

```
public static void main( String[] args )
{
       name = ''Frank'';
       System.out.println( "Hello " + name );
       System.exit(0);
}
```

Syntax error!

The variable name is not declared. It should be declared as a String prior to the assignment statement.