

## **CMIS 141 Hands-on Lab Week 1**

### **Overview**

This document provides a series of exercises to assist the student becoming comfortable coding in Java. It is assumed the JDK 8 (or higher) programming environment is properly installed and the associated readings for this week have been completed.

### **Submission requirements**

Hands-on labs are designed for you to complete each week as a form of self-assessment. You do not need to submit your lab work for grading. However; you should post questions in the weekly questions area if you have any trouble or questions related to completing the lab. These labs will help prepare you for the graded assignments, therefore; it is highly recommended you successfully complete these exercises.

### **Objectives**

The following objectives will be covered by successfully completing each exercise:

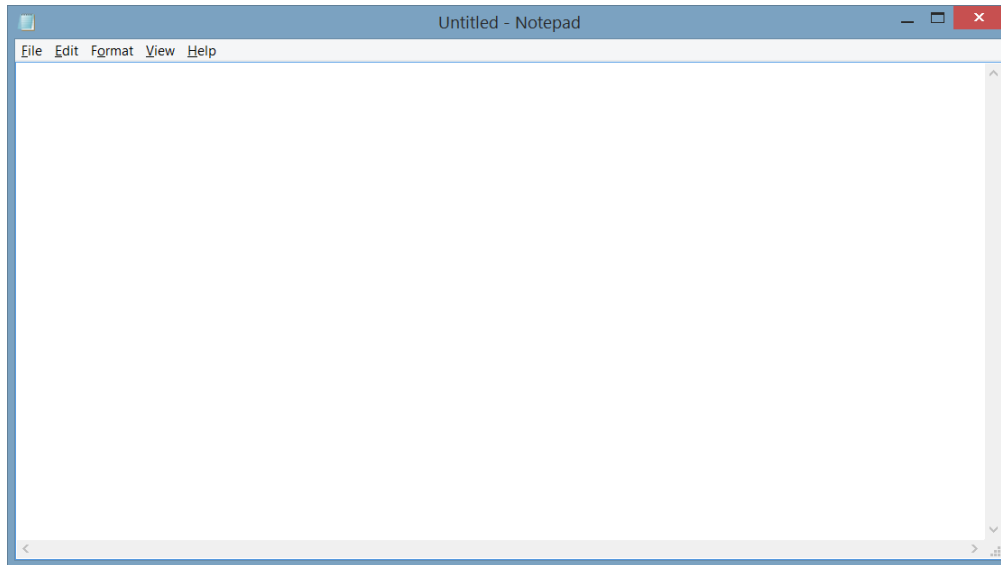
1. Create and edit simple Java programs in a text editor
2. Compile and run simple Java programs from the command prompt
3. Differentiate `System.out.print()` from `System.out.println()` functionality
4. Declare, initialize and use Java primitive variables
5. Use Java assignment, arithmetic, unary, equality, relational, conditional and bitwise operators

### **Exercise 1 – Creating and editing Java programs in a text editor**

During the first couple of weeks, you can use a text editor of your choice to create and edit Java programs. The steps shown are for Windows Notepad, but all text editors work similarly in that you open the application, start typing the code and save the code. We will introduce you to an Integrated Development Environment (IDE) (e.g. Netbeans) later in the semester. If you are comfortable using an IDE right now, you are welcome to start off using your IDE.

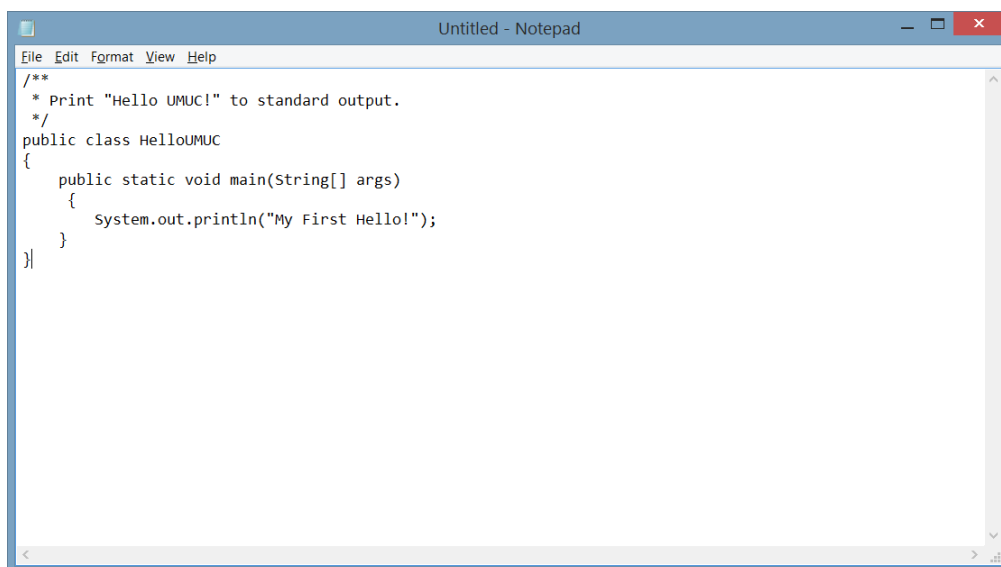
Word processors such as Microsoft word are **not** designed for text editing and coding as they generate additional characters which interfere with the compilation process.

- a. Open your favorite text editor.

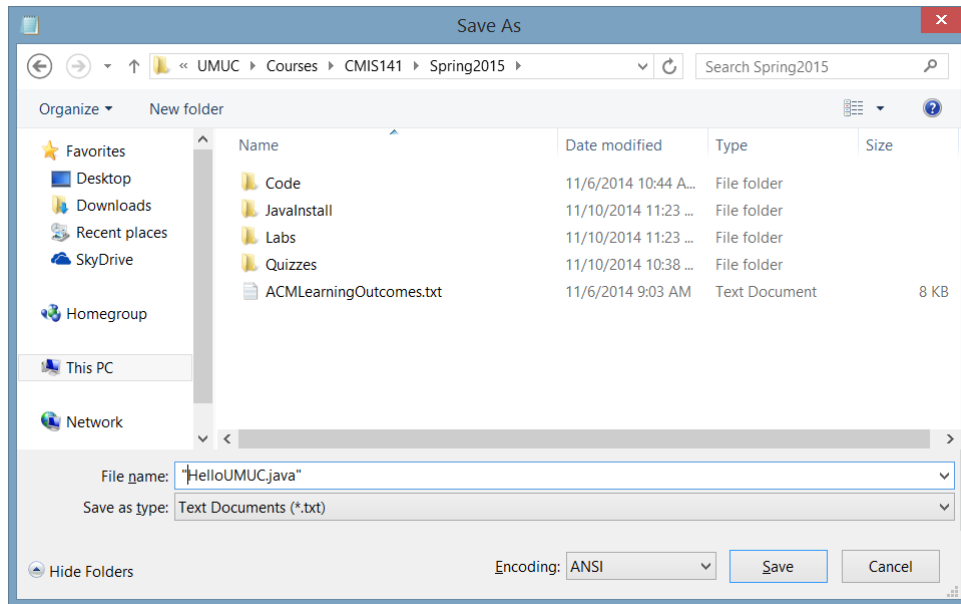


- b. Type (or copy and paste) the following Java code into your text editor

```
/**
 * Print "My First Hello!" to standard output.
 */
public class HelloUMUC
{
    public static void main(String[] args)
    {
        System.out.println("My First Hello!");
    }
}
```



- c. Save the file as “HelloUMUC.java”



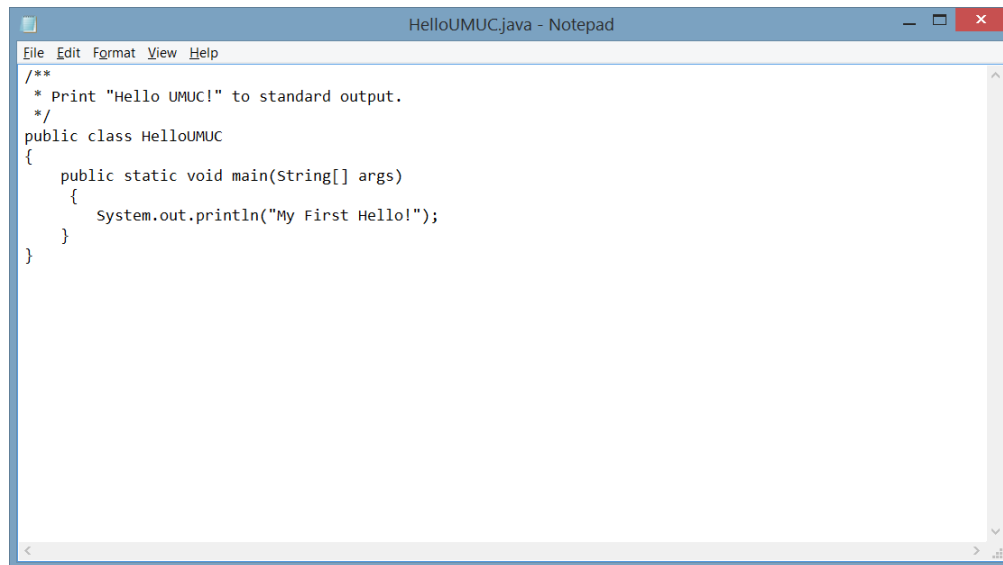
Note the following critical components of the save:

1. The filename is case sensitive. HelloUMUC.java is not the same as HelloUmuc.java
2. The filename must match the public class name listed in this line exactly:

```
public class HelloUMUC
```

Notice HelloUMUC is the name of the public class and HelloUMUC.java is the name of the file.

3. Your texteditor may need quotes around the filename to avoid appending .txt or some other default filename. If you are using Notepad, if you don't place quotes around the filename when saving, more than likely a .txt will be appended and when you to compile your filename will be named HelloUMUC.java.txt as opposed to HelloUMUC.java
4. Be sure to note where the folder where the file is saved. You will need this shortly to be able to compile and run it.



```
File Edit Format View Help
/**
 * Print "Hello UMUC!" to standard output.
 */
public class HelloUMUC
{
    public static void main(String[] args)
    {
        System.out.println("My First Hello!");
    }
}
```

Now it is your turn. Try the following exercise:

Create a Java class named HappyBirthday using your favorite text editor. Be sure you name the file “HappyBirthday.java”. Also, change the printed message to “Happy Birthday!”. Modify the file comments to reflect the new filename and functionality.

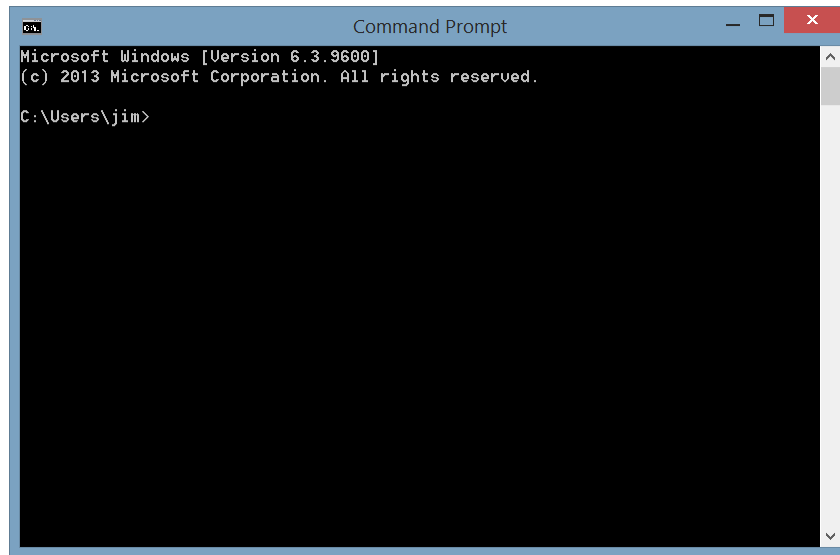
## Exercise 2 – Compile and running simple Java Programs from the command prompt

All operating systems have a command prompt allowing users to type commands and invoke programs and application. The process to compile and run a Java from the command prompt includes opening a command prompt, changing to the folder location where the .java file exists and then compiling and running the program.

The following steps show how to compile and run a Java program on a Windows machine. The process is identical for Mac OS/X and Linux. You just have to open a command prompt following your operating system specific instructions.

For this exercise, we will use the HelloUMUC.java we created in exercise 1.

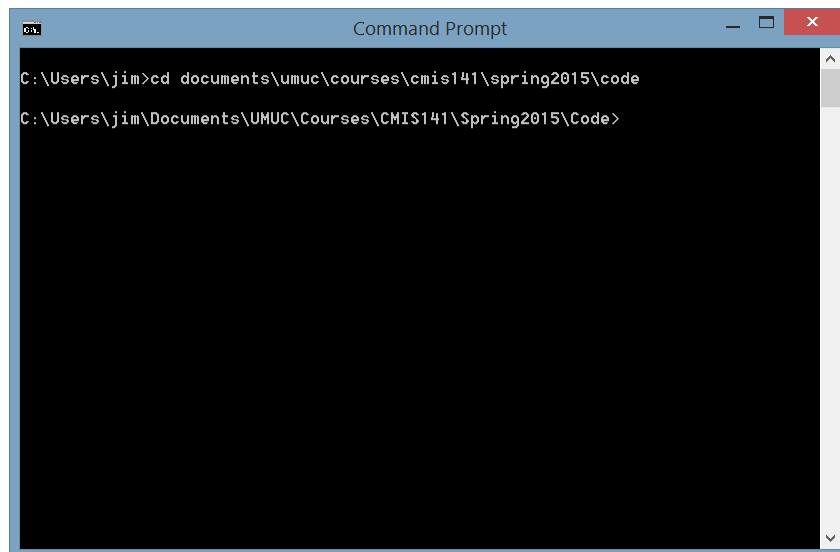
- a. Open a command prompt



```
Microsoft Windows [Version 6.3.9600]
(c) 2013 Microsoft Corporation. All rights reserved.

C:\Users\jim>
```

- b. Use the cd command to change to the folder (directory) location of your HelloUMUC.java file



```
C:\Users\jim>cd documents\umuc\courses\cmis141\spring2015\code
C:\Users\jim\Documents\UMUC\Courses\CMIS141\Spring2015\Code>
```

The cd (change directory) command has many different options and parameter you can use. Typically you can access online help by typing "help cd" at the command prompt. As a quick overview cd documents will change to the documents directory of the current directory. The "\" character is used to delimit the directories. You can enter very long pathnames such as:

```
cd documents\umuc\courses\cmis141\spring2015\code
```

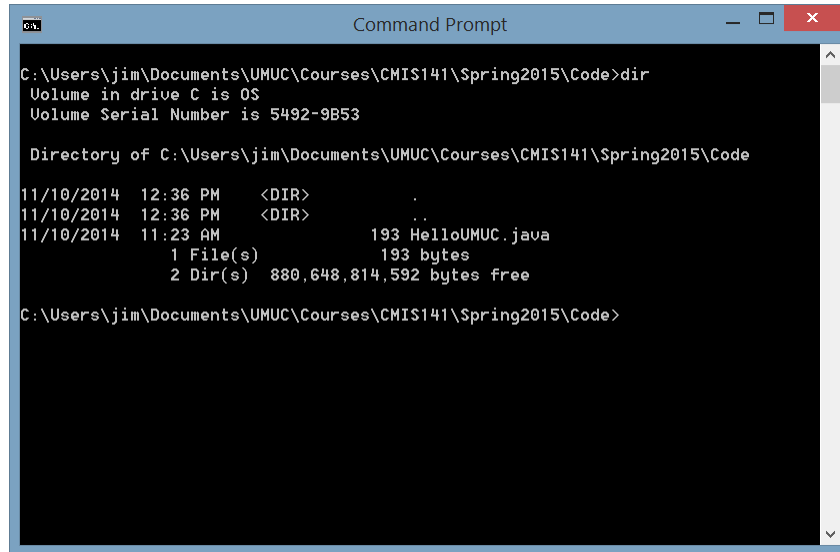
You can also make smaller directory change steps:

```
cd documents
```

```
cd umuc\courses\cmis141
```

`cd spring2015\code`

Be sure you are comfortable using the `cd` to traverse the directories and folders of your system. Once you are in the correct folder, you can type “`dir`” at the command prompt to show the directory contents.



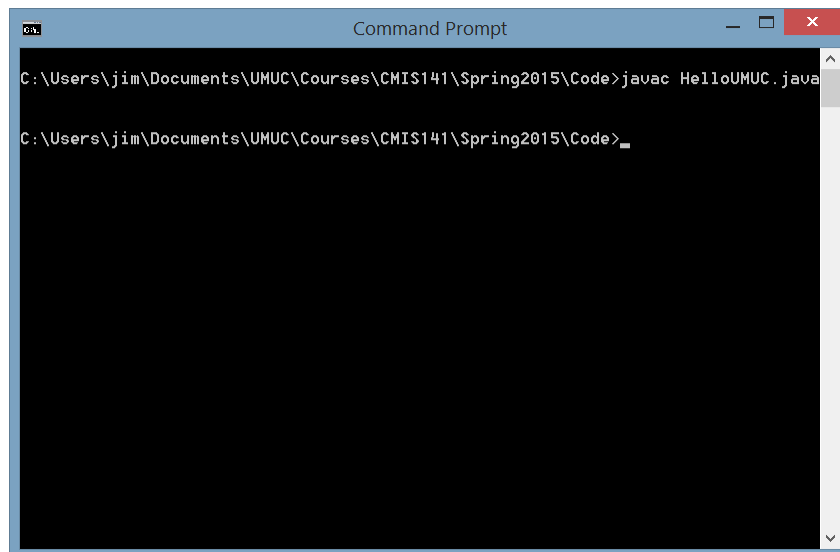
```
Command Prompt
C:\Users\jim\Documents\UMUC\Courses\CMIS141\Spring2015\Code>dir
Volume in drive C is OS
Volume Serial Number is 5492-9B53

Directory of C:\Users\jim\Documents\UMUC\Courses\CMIS141\Spring2015\Code

11/10/2014  12:36 PM    <DIR>          .
11/10/2014  12:36 PM    <DIR>          ..
11/10/2014  11:23 AM                193 HelloUMUC.java
               1 File(s)                193 bytes
               2 Dir(s)  880,648,814,592 bytes free

C:\Users\jim\Documents\UMUC\Courses\CMIS141\Spring2015\Code>
```

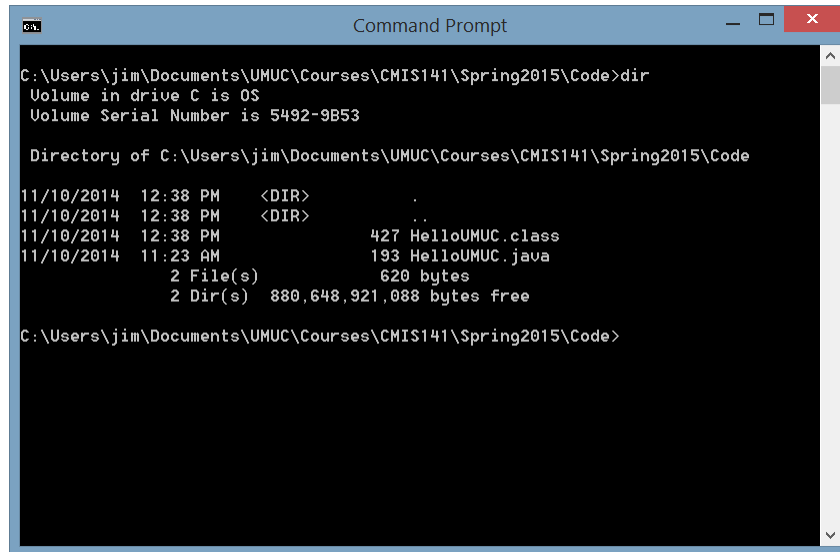
- c. To compile the `HelloUMUC.java` file, type `javac HelloUMUC.java` at the command prompt.



```
Command Prompt
C:\Users\jim\Documents\UMUC\Courses\CMIS141\Spring2015\Code>javac HelloUMUC.java
C:\Users\jim\Documents\UMUC\Courses\CMIS141\Spring2015\Code>_
```

Java code that doesn't contain any errors will not provide any feedback from your compile command. No news is definitely good news in this case.

If you run the `dir` command at the command prompt, you will notice the compile process generated a file named "HelloUMUC.class". These are the bytecodes that will be interpreted and run by the Java launcher in the next step.



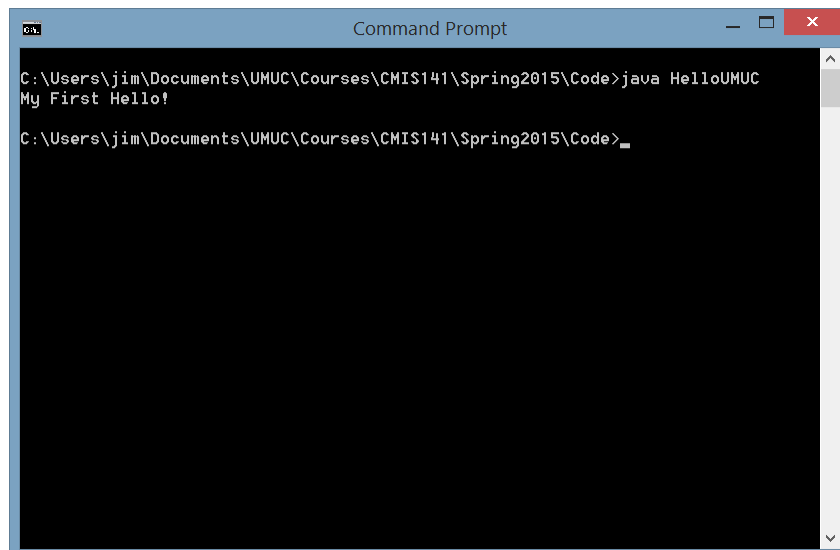
```
Command Prompt
C:\Users\jim\Documents\UMUC\Courses\CMIS141\Spring2015\Code>dir
Volume in drive C is OS
Volume Serial Number is 5492-9B53

Directory of C:\Users\jim\Documents\UMUC\Courses\CMIS141\Spring2015\Code

11/10/2014  12:38 PM    <DIR>          .
11/10/2014  12:38 PM    <DIR>          ..
11/10/2014  12:38 PM                427 HelloUMUC.class
11/10/2014  11:23 AM                193 HelloUMUC.java
                2 File(s)                620 bytes
                2 Dir(s)  880,648,921,088 bytes free

C:\Users\jim\Documents\UMUC\Courses\CMIS141\Spring2015\Code>
```

- d. To run the HelloUMUC.class file, type `java HelloUMUC` at the command prompt.



```
Command Prompt
C:\Users\jim\Documents\UMUC\Courses\CMIS141\Spring2015\Code>java HelloUMUC
My First Hello!
C:\Users\jim\Documents\UMUC\Courses\CMIS141\Spring2015\Code>
```

This results in the expected output of "My First Hello!"

Now it is your turn. Try the following exercises:

1. Open up your text editor and create a file named `Welcome.java` and change the output text of “My First Hello!” to “Welcome to CMIS 141 at UMUC!”. Save the file, recompile and run the bytecodes.
2. If successful, modify the file again to add three additional `System.out.println(“Your text here”)` ; messages. Replace “Your text here” with any text of your choice. Save the file, recompile and run the bytecodes. Be careful that each of the lines you add is identical to the original line except for the new text. If you experience compile errors, work to resolve them noting what was wrong and how you resolved the issue.

### Exercise 3 – Differentiate `System.out.println()` from `System.out.print()` functionality

Now that you are an expert at creating, compiling and running Java code, it is time to expand your knowledge of Java code to create somewhat more interesting applications. Although we haven’t covered classes and objects in any detail yet, the `println()` and `print()` are methods of the `System.out` class. We will talk more about classes later in the class. For now, realize a class is comprised of a group or related methods. Methods are small sequences of code providing certain types of functionality.

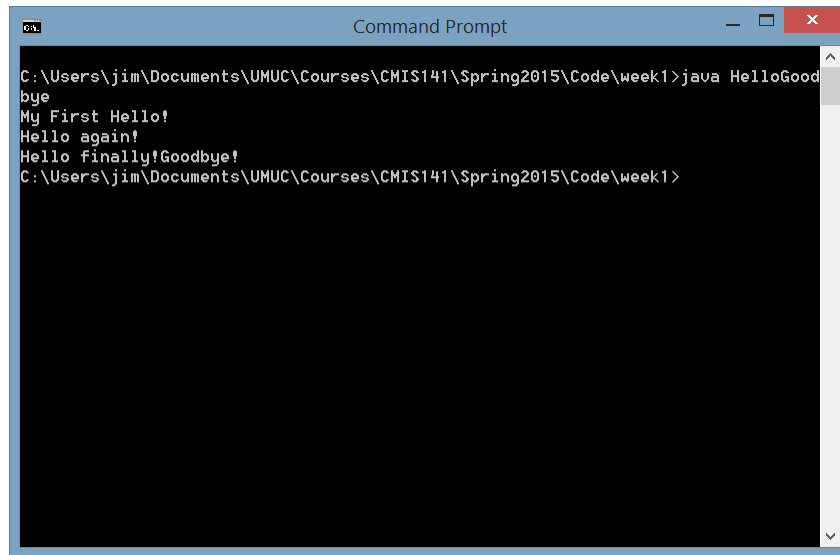
The fundamental difference between the `println()` and `print()` methods is `println()` provides a new line after the String is printed, while `print()` does not.

Consider the following `HelloGoodbye.java` file.

```
public class HelloGoodbye
{
    public static void main(String[] args)
    {
        System.out.println("My First Hello!");
        System.out.println("Hello again!");
        System.out.print("Hello finally!");
        System.out.print("Goodbye!");
    }
}
```

Notice, the first two output statements include a new line, whereas the last two do not. Running the bytecodes results in this output.





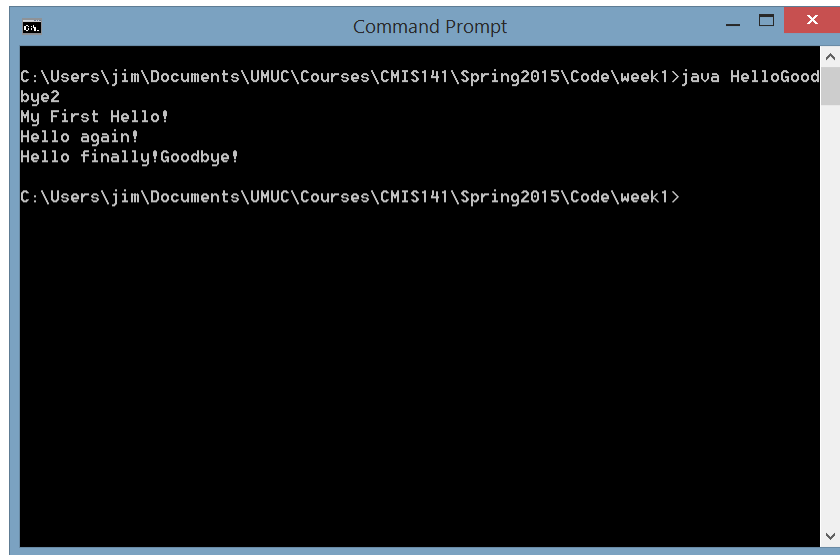
```
C:\Users\jim\Documents\UMUC\Courses\CMIS141\Spring2015\Code\week1>java HelloGood
bye
My First Hello!
Hello again!
Hello finally!Goodbye!
C:\Users\jim\Documents\UMUC\Courses\CMIS141\Spring2015\Code\week1>
```

Since there is no new line provided for “Hello finally!” The next output starts immediately after the previous statement resulting in “Hello finally!Goodbye!”

Changing the code to modify the last output statement to include the new line as shown in the following code:

```
public class HelloGoodbye2
{
    public static void main(String[] args)
    {
        System.out.println("My First Hello!");
        System.out.println("Hello again!");
        System.out.print("Hello finally!");
        System.out.println("Goodbye!");
    }
}
```

Will result in the following output. The change is subtle but notice the additional line after the “Hello finally!Goodbye!” output.



```
Command Prompt
C:\Users\jim\Documents\UMUC\Courses\CMIS141\Spring2015\Code\week1>java HelloGoodbye2
My First Hello!
Hello again!
Hello finally!Goodbye!
C:\Users\jim\Documents\UMUC\Courses\CMIS141\Spring2015\Code\week1>
```

Now it is your turn. Try the following exercises:

1. Create a new Java file and name it PrintDiff.java. Be sure you include the correct public class syntax. Since the file is named PrintDiff.java, the public class syntax should be “public class PrintDiff”. Also, be sure the content in the .java is correct and includes the main method. For example, your Java code should look similar to this:

```
/**
 * Print Experiment.
 */
public class PrintDiff
{
    public static void main(String[] args)
    {
        System.out.println("My First Hello!");
    }
}
```

2. Make sure your code compiles and runs as expected. (javac PrintDiff.java, java PrintDiff)
3. Modify your PrintDiff.java file by adding System.out.println() and System.out.print() methods as appropriate to create the following output:

```
a      a
bb     bb
ccc    ccc
dddd   dddd
ccc    ccc
bb     bb
a      a
```

#### Exercise 4- Declare, initialize and use Java primitive variables

Java provides primitive variables supporting the storage of numbers, characters and boolean variables. The readings for this week provides the specific data type names and their size and include byte, short, int, long, float, double, char, and boolean.

When declaring, initializing and using variables, the Java compiler will provide errors messages if you attempt to use or name the variable incorrectly. This exercise will declare, initialize and use several Java primitive data types. The code will then be modified to break some of the rules that the compiler enforces to demonstrate how error messages are generated by the compiler and an approach to debug and correct those errors.

- a. Open up a text editor and create a file named VariableDemo.java file. We will create a Java file that will let us declare and initialize variables for all available primitives. These declarations can be added to the main method.

```
// Declare and initialize primitive variables
byte myByte = 20;
short myShort = 2000;
int myInt = 20000;
long myLong = 2000000L;
float myFloat = 32.345f;
double myDouble = 32.932;
boolean myBoolean = true;
char myChar = 'C';
```

Some important reminders about these declarations:

1. Variables assigned to the type of Long need and L to value assignment denote the value is Long.
  2. Similarly, float values need to have f concatenated to the end of the value for the assignment.
  3. Knowing the upper and lower ranges of the primitive data types helps in selecting the best data type for your application. For example, if you are storing whole numbers and you know your values will always be less than -32,768 and 32,767, using the short data type is appropriate. If you don't know your data range, selecting the larger data type is better.
  4. Each declaration ends with a semi-colon (;).
- b. Add a final constant double variable declaration representing the speed of sound in m/s named SPEEDOFSOUND to the main method:  

```
final double SPEEDOFSOUND = 343.6;
```
  - c. Add println() statements to demonstrate the successful initialization of each variable. To do this, add a System.out.println() for each variable. Using the concatenation operator (+) provides for some formatting of results. These output statements are added after the initialization and declarations of the variables.

```
// Print each variable along with the value
    System.out.println("myByte: " + myByte);
    System.out.println("myShort: " + myShort);
    System.out.println("myInt: " + myInt);
    System.out.println("myLong: " + myLong);
    System.out.println("myFloat: " + myFloat);
    System.out.println("myDouble: " + myDouble);
    System.out.println("myBoolean: " + myBoolean);
    System.out.println("myChar: " + myChar);
    System.out.println("SPEEDOFSOUND: " + SPEEDOFSOUND);
```

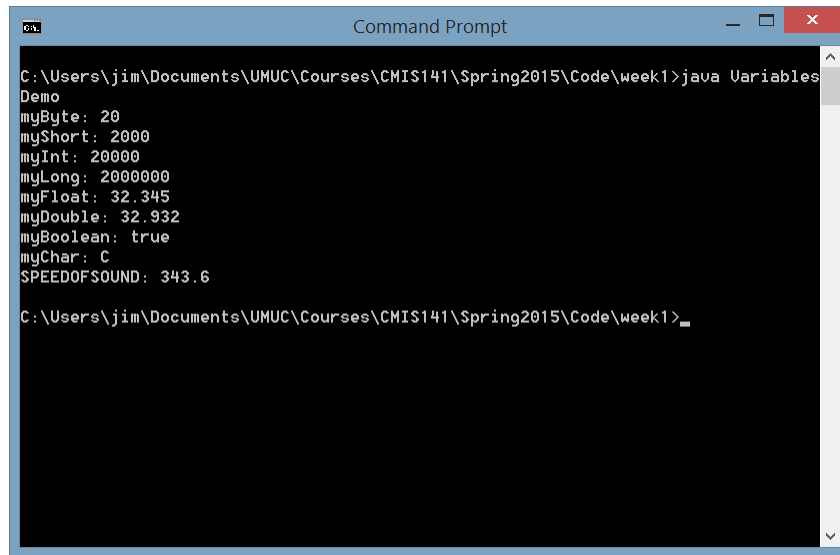
The complete code will look as like this:

```
/**
 * Variables Demonstration.
 */
public class VariablesDemo
{
    public static void main(String[] args)
    {
        // Declare and initialize primitive variables
        byte myByte = 20;
        short myShort = 2000;
        int myInt = 20000;
        long myLong = 2000000L;
        float myFloat = 32.345f;
        double myDouble = 32.932;
        boolean myBoolean = true;
        char myChar = 'C';

        // Declare a constant to hold Speed of Sound
        final double SPEEDOFSOUND = 343.6;

        // Print Variable name and value
        System.out.println("myByte: " + myByte);
        System.out.println("myShort: " + myShort);
        System.out.println("myInt: " + myInt);
        System.out.println("myLong: " + myLong);
        System.out.println("myFloat: " + myFloat);
        System.out.println("myDouble: " + myDouble);
        System.out.println("myBoolean: " + myBoolean);
        System.out.println("myChar: " + myChar);
        System.out.println("SPEEDOFSOUND: " + SPEEDOFSOUND);
    }
}
```

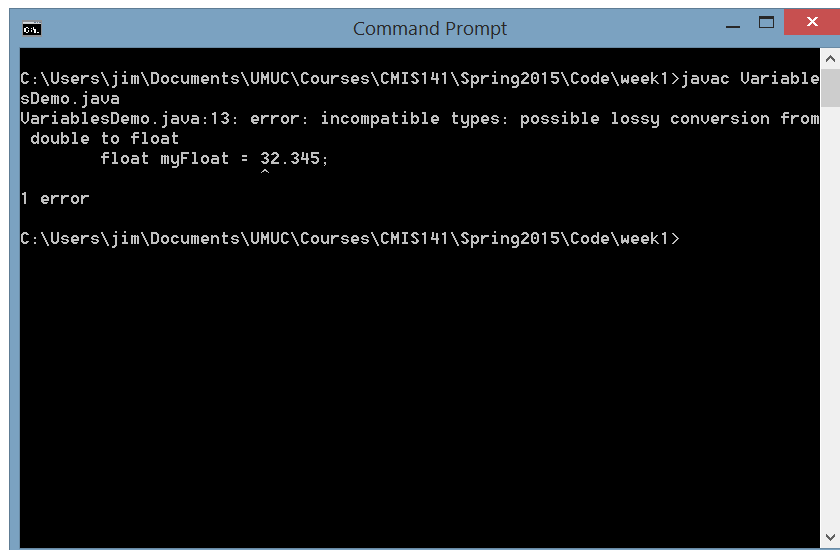
d. Compiling and running the application results in the following output.



```
C:\Users\jim\Documents\UMUC\Courses\CMIS141\Spring2015\Code\week1>java VariablesDemo
Demo
myByte: 20
myShort: 2000
myInt: 20000
myLong: 2000000
myFloat: 32.345
myDouble: 32.932
myBoolean: true
myChar: C
SPEEDOFSOUND: 343.6

C:\Users\jim\Documents\UMUC\Courses\CMIS141\Spring2015\Code\week1>
```

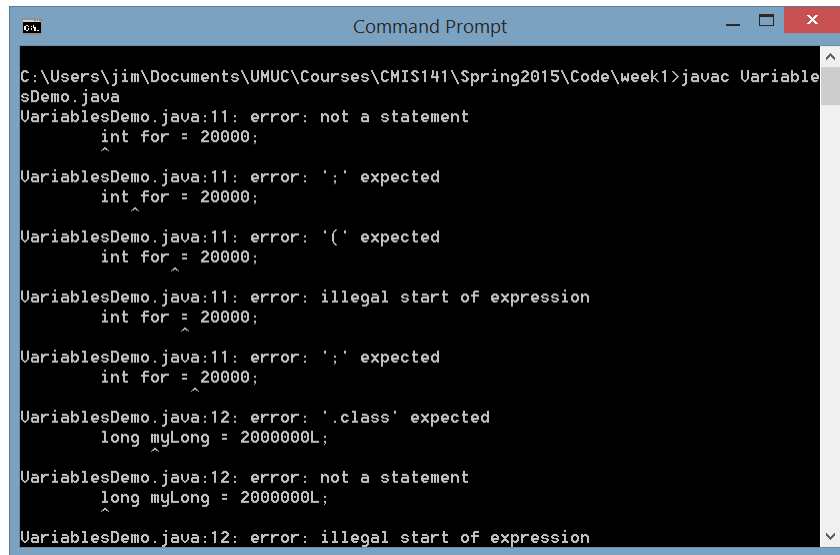
- e. To demonstrate how Java detects compile errors, we can modify the code to break some of the variable declaration rules and review the resultant output. First, if we remove the “f” in the float declaration (change float myFloat = 32.345f; to float myFloat = 32.345;) and recompile, we will receive an error message.



```
C:\Users\jim\Documents\UMUC\Courses\CMIS141\Spring2015\Code\week1>javac VariablesDemo.java
VariablesDemo.java:13: error: incompatible types: possible lossy conversion from
double to float
    float myFloat = 32.345;
                        ^
1 error

C:\Users\jim\Documents\UMUC\Courses\CMIS141\Spring2015\Code\week1>
```

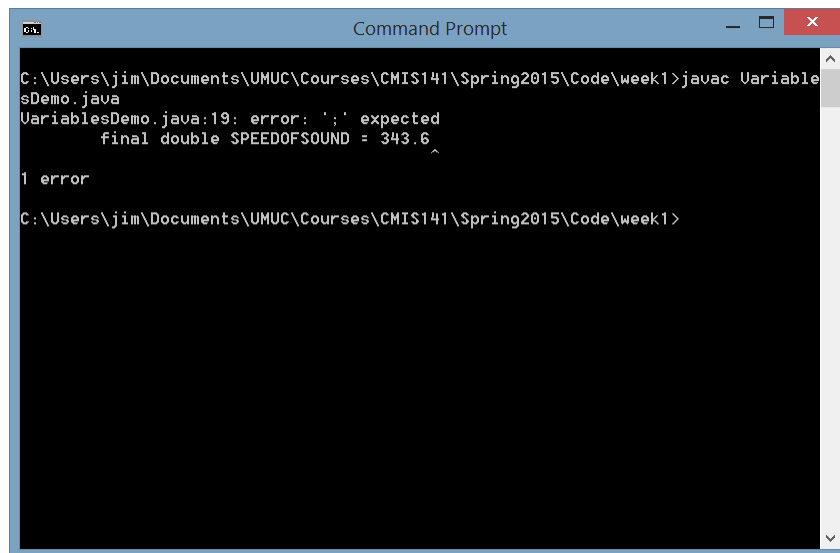
Second, if we change the myInt variable name to a reserved word (change int myInt = 20000; to int for = 20000; ) and recompile we will receive many error messages.



```
Command Prompt
C:\Users\jim\Documents\UMUC\Courses\CMIS141\Spring2015\Code\week1>javac VariablesDemo.java
VariablesDemo.java:11: error: not a statement
    int for = 20000;
    ^
VariablesDemo.java:11: error: ';' expected
    int for = 20000;
    ^
VariablesDemo.java:11: error: '(' expected
    int for = 20000;
    ^
VariablesDemo.java:11: error: illegal start of expression
    int for = 20000;
    ^
VariablesDemo.java:11: error: ';' expected
    int for = 20000;
    ^
VariablesDemo.java:12: error: '.class' expected
    long myLong = 2000000L;
    ^
VariablesDemo.java:12: error: not a statement
    long myLong = 2000000L;
    ^
VariablesDemo.java:12: error: illegal start of expression
    long myLong = 2000000L;
    ^
```

Clearly, error messages can be confusing because we introduced only one error yet many additional errors were reported. The best approach to work with these errors, is to find the very first error and correct that issue. This usually will fix the remaining errors – provided we didn't introduce any additional errors in the debugging activity.

Another common error is a missing semi-colon at the end of the statement. If we remove the semicolon in this line: `final double SPEEDOFSOUND = 343.6`, and recompile we receive this error message.



```
Command Prompt
C:\Users\jim\Documents\UMUC\Courses\CMIS141\Spring2015\Code\week1>javac VariablesDemo.java
VariablesDemo.java:19: error: ';' expected
    final double SPEEDOFSOUND = 343.6
                                ^
1 error
C:\Users\jim\Documents\UMUC\Courses\CMIS141\Spring2015\Code\week1>
```

These error messages are common, but fortunately easy to correct by adding the semi-colon.

Now it is your turn. Try the following exercises:

1. Create a new Java file and name it MyVariables.java. Be sure you include the correct public class syntax. Since the file is named MyVariables.java, the public class syntax should be “public class MyVariables”. Also, be sure the content in the .java is correct and includes the main method.
2. Declare and initialize two boolean, two double, two int and two char variables. When creating the variables be sure to use recommended naming conventions and do not use any reserved words.
3. Add two additional int variables with values of hexadecimal and binary assignment notation hint: 0x0022, 0b1100)
4. Add one additional int variable that uses underscores to make the value more readable (hint: 2\_000\_000\_000)
5. Print the names and values for each of the variables, one variable per line to standard output.
6. Experiment with debugging by modifying the code by removing semi-colons, changing variable names or other techniques to introduce compile errors. Resolve the errors as needed to successfully compile and execute the Java application.

### **Exercise 5 - Use Java assignment, arithmetic, unary, equality, relational, conditional and bitwise operators**

Assignment statements and the use of operators are critical for developing useful and complex applications in Java. Understanding the precedence rules for evaluating components of a complex Java statement composed of multiple operators assists in ensuring the application logic is correct.

This exercise will use multiple operators and provide show the expected output for simple and complex Java statements.

- a. Open up a text editor and create a file named OperatorsDemo.java file. We will create a Java file that will let use multiple Java operators and statements. First add some simple arithmetic and assignment operators to the main method. In the following main method of the OperatorsDemo.java file, two variables are declared to hold double and int value results. Then the arithmetic operators of +, -, \*, / and % are applied to several different values. The results are printed in each case.

```
public static void main(String[] args)
{
    System.out.println("Math Operators Demo");

    // Place holders to store calculation
    double results = 0.0;
    int intresults = 0;

    // Addition (+)
    results = 4.1 + 8.1;
    System.out.println("4.1 + 8.1 = " + results);
```

```

results = 4.1 + 10.5;
System.out.println("4.1 + 10.5 = " + results);

// Subtraction (-)
results = 4.1 - 8.1;
System.out.println("4.1 - 8.1 = " + results);
results = 4.1 - 10.5;
System.out.println("4.1 - 10.5 = " + results);

// Multiplication (*)
results = 4.1 * 8.1;
System.out.println("4.1 * 8.1 = " + results);
results = 4.1 * 10.5;
System.out.println("4.1 * 10.5 = " + results);

// Division (/)
results = 4.1 / 8.1;
System.out.println("4.1 / 8.1 = " + results);
results = 4.1 / 10.5;
System.out.println("4.1 / 10.5 = " + results);

// Modulus (%)
intresults = 20 % 3;
System.out.println("20 % 3 = " + intresults);
intresults = 20 % 5;
System.out.println("20 % 5 = " + intresults);

```

Running the bytecodes generated after successfully compiling the files results in the following output:

```

C:\Users\jim\Documents\UMUC\Courses\CMIS141\Spring2015\Code\week1>java Operators
Demo
Math Operators Demo
4.1 + 8.1 = 12.2
4.1 + 10.5 = 14.6
4.1 - 8.1 = -4.0
4.1 - 10.5 = -6.4
4.1 * 8.1 = 33.209999999999994
4.1 * 10.5 = 43.05
4.1 / 8.1 = 0.5061728395061729
4.1 / 10.5 = 0.3904761904761904
20 % 3 = 2
20 % 5 = 0
C:\Users\jim\Documents\UMUC\Courses\CMIS141\Spring2015\Code\week1>_

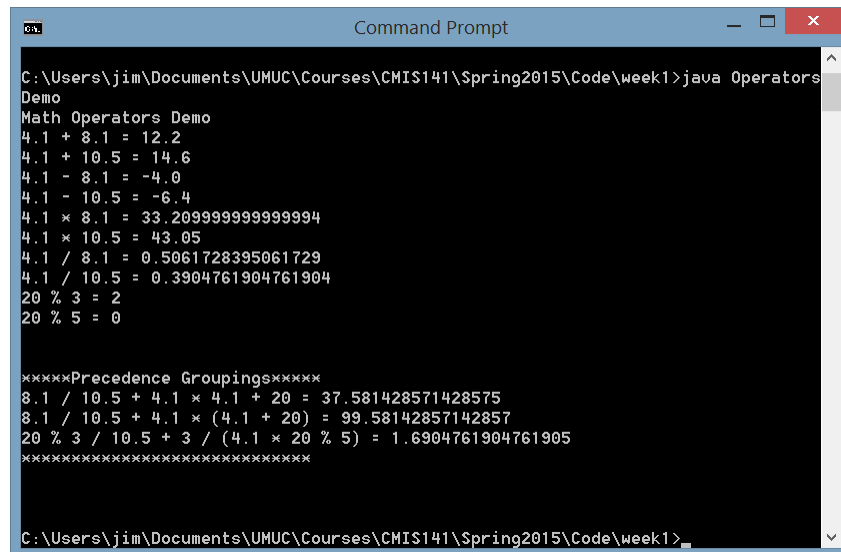
```



- b. Continuing to add code to the same OperatorsDemo.java file, we can add code that will provide some simple examples of precedence grouping. Add the following code to the main method of the OperatorsDemo.java file.

```
// Precedence groupings
System.out.println("\n");    // New Line to separate Output
System.out.println("*****Precedence Groupings*****");
results = 8.1 / 10.5 + 4.1 * 4.1 + 20;
System.out.println("8.1 / 10.5 + 4.1 * 4.1 + 20 = " + results);
results = 8.1 / 10.5 + 4.1 * (4.1 + 20);
System.out.println("8.1 / 10.5 + 4.1 * (4.1 + 20) = " +
results);
results = 20 % 3 / 10.5 + 3 / (4.1 * 20 % 5);
System.out.println("20 % 3 / 10.5 + 3 / (4.1 * 20 % 5) = " +
results);
System.out.println("*****");
System.out.println("\n");
```

Running the compiled code will result in this output:



```
Command Prompt
C:\Users\jim\Documents\UMUC\Courses\CMIS141\Spring2015\Code\week1>java Operators
Demo
Math Operators Demo
4.1 + 8.1 = 12.2
4.1 + 10.5 = 14.6
4.1 - 8.1 = -4.0
4.1 - 10.5 = -6.4
4.1 * 8.1 = 33.209999999999994
4.1 * 10.5 = 43.05
4.1 / 8.1 = 0.5061728395061729
4.1 / 10.5 = 0.3904761904761904
20 % 3 = 2
20 % 5 = 0

*****Precedence Groupings*****
8.1 / 10.5 + 4.1 * 4.1 + 20 = 37.581428571428575
8.1 / 10.5 + 4.1 * (4.1 + 20) = 99.58142857142857
20 % 3 / 10.5 + 3 / (4.1 * 20 % 5) = 1.6904761904761905
*****
C:\Users\jim\Documents\UMUC\Courses\CMIS141\Spring2015\Code\week1>
```

As you experiment with Operator precedence, create many different example to try. Be sure to calculate the expected results prior to coding to see if you are confident with this foundational concept of programming. Note the use of the escape character (\) to denote a new line ("\\n").

- c. Add Unary operators (++ , -- , !) by copying and pasting the following code to your main method.

```

// Unary Operators
// Variables for demonstrating Unary Operators
System.out.println("*****Unary Operators*****");
int varA = 20;
int varB = 3;
int varC = 5;
int varD = 10;

// Post Increment (++)
intresults = varA++;
System.out.println("varA++ = " + intresults);

// Pre increment (++)
intresults = ++varB;
System.out.println("++varB = " + intresults);

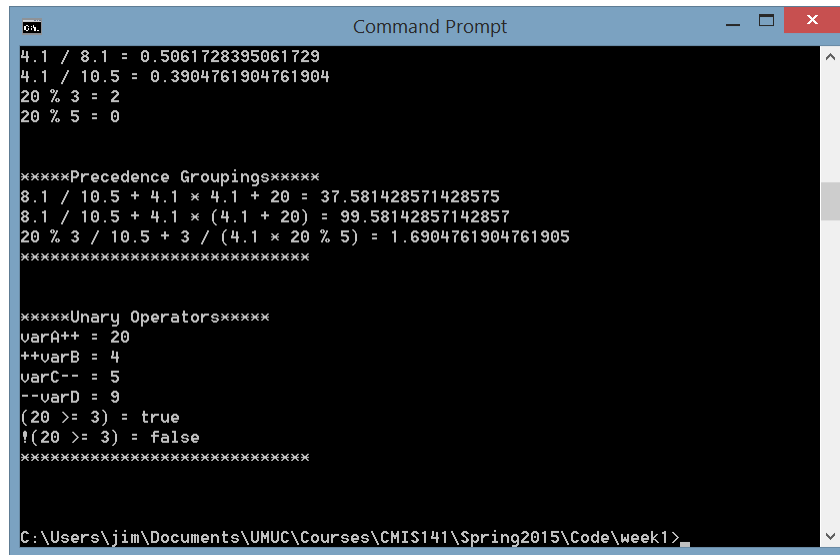
// Post decrement (--)
intresults = varC--;
System.out.println("varC-- = " + intresults);

// Pre decrement (--)
intresults = --varD;
System.out.println("--varD = " + intresults);

// Not Operator (!)
// This should be true since 20 >= 3
boolean flag = (20 >= 3);
System.out.println("(20 >= 3) = " + flag);
// This should be false since ! inverts the boolean value
flag = !(20 >= 3);
System.out.println("!(20 >= 3) = " + flag);
System.out.println("*****");
System.out.println("\n");

```

Running the compiled code will result in this output:



```
Command Prompt
4.1 / 8.1 = 0.5061728395061729
4.1 / 10.5 = 0.3904761904761904
20 % 3 = 2
20 % 5 = 0

*****Precedence Groupings*****
8.1 / 10.5 + 4.1 * 4.1 + 20 = 37.581428571428575
8.1 / 10.5 + 4.1 * (4.1 + 20) = 99.58142857142857
20 % 3 / 10.5 + 3 / (4.1 * 20 % 5) = 1.6904761904761905
*****

*****Unary Operators*****
varA++ = 20
++varB = 4
varC-- = 5
--varD = 9
(20 >= 3) = true
!(20 >= 3) = false
*****

C:\Users\jim\Documents\UMUC\Courses\CMIS141\Spring2015\Code\week1>
```

As before, experiment with many variables of the pre-increment, post-increment and Not Unary operators. These operators can be tricky and may take several iterations for you to become comfortable with their behavior. The key is the pre-increment will increment the value before assigning that new value to the variable whereas the post-increment will assign the value before incrementing it. The Not operator switches the boolean value to the opposite value. For example, if var1 was true, then !var1 would be false.

- d. The equality operators provide the foundation for selection statements and other functionality in Java and other programming languages. The equality operators include ==, !=, >, >=, <, and <= . When used in an expression, they will return true or false. To experiment with equality operators, copy and paste the following code into the main method of the OperatorsDemo.java file.

```
// Equality Operators
// Equality (==)
// 20 is not equal to 3 so flag should be false.
System.out.println("*****Equality Operators*****");
flag = (20 == 3);
System.out.println("(20 == 3) = " + flag);
flag = (20 / 4 == 5);
System.out.println("(20 / 4 == 5) = " + flag);

// Not Equal (!=)
// 20 is not equal to 3 so flag should be true.
flag = (20 != 3);
System.out.println("(20 != 3) = " + flag);
flag = (20 / 4 != 5);
System.out.println("(20 / 4 != 5) = " + flag);

// Greater than (>)
// 20 is greater than 3 so flag should be true.
flag = (20 > 3);
```

```

System.out.println("(20 > 3) = " + flag);
    flag = (20 / 4 > 5);
System.out.println("(20 / 4 > 5) = " + flag);

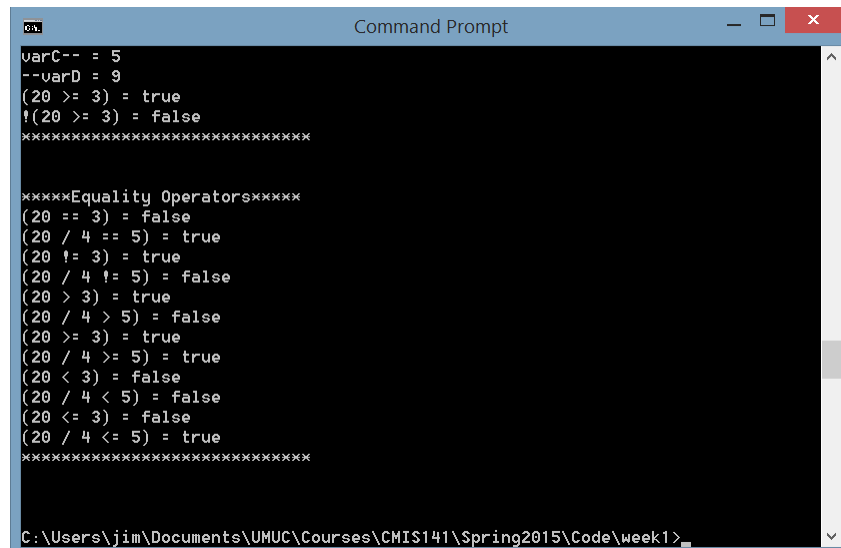
    // Greater than or equal to (>=)
    // 20 is not equal to 3 so flag should be false.
flag = (20 >= 3);
System.out.println("(20 >= 3) = " + flag);
flag = (20 / 4 >= 5);
System.out.println("(20 / 4 >= 5) = " + flag);

// Less than (<)
// 20 is not less than 3 so flag should be false.
flag = (20 < 3);
System.out.println("(20 < 3) = " + flag);
flag = (20 / 4 < 5);
System.out.println("(20 / 4 < 5) = " + flag);

// Less than or equal (<=)
// 20 is not less than or equal so 3 so flag should be false.
flag = (20 <= 3);
System.out.println("(20 <= 3) = " + flag);
flag = (20 / 4 <= 5);
System.out.println("(20 / 4 <= 5) = " + flag);
System.out.println("*****");
System.out.println("\n");

```

Running the compiled code will result in this output:



```

varC-- = 5
--varD = 9
(20 >= 3) = true
!(20 >= 3) = false
*****

*****Equality Operators*****
(20 == 3) = false
(20 / 4 == 5) = true
(20 != 3) = true
(20 / 4 != 5) = false
(20 > 3) = true
(20 / 4 > 5) = false
(20 >= 3) = true
(20 / 4 >= 5) = true
(20 < 3) = false
(20 / 4 < 5) = false
(20 <= 3) = false
(20 / 4 <= 5) = true
*****

C:\Users\jim\Documents\UMUC\Courses\CMIS141\Spring2015\Code\week1>

```

You should experiment with equality operators, apply different values and observe the results.

- e. Conditional operators (&&, ||) allow programmers to build larger and more complex expressions using various types of operators. When evaluating the results, a truth table can be

useful. A truth table is a table representing the possible results for conditional operators for various boolean values. For example, true && true results in true, true && false results in false, and false and false results in false. However; true || true results in true, true || false results in true, and false || false results in false. Table 1 illustrates a typical truth table.

Table 1. Truth Table for && and ||

Condition1	Condition2	Condition1 && Condition 2	Condition1    Condition 2
true	true	true	true
true	false	false	true
false	true	false	true
false	false	false	false

To experiment with conditional operators, add the following code to the OperatorsDemo.java main method.

```
// Conditional Operators
    // And (&&)
    System.out.println("*****Conditional Operators*****");
    flag = (20 == 20 ) && (15 >= 1) ;
    System.out.println ("(20 == 20 ) && (15 >= 1) = " + flag);
    flag = (20 == 20 ) || (15 >= 1) ;
    System.out.println ("(20 == 20 ) || (15 >= 1) = " + flag);

    // Conditional Operators can become complex
    // this can be tricky and precedence rules are critical
    // Recall: && is evaluated before ||.
    flag = (20 == 20 ) && (15 >= 1) || (true == false) && (1 == 4);
    System.out.println ("(20 == 20 ) && (15 >= 1) || (true ==
false) && (1 == 4) = " + flag);
    System.out.println("*****");
    System.out.println("\n");
```

Running the compiled code will result in this output:

```
Command Prompt
*****Equality Operators*****
(20 == 3) = false
(20 / 4 == 5) = true
(20 != 3) = true
(20 / 4 != 5) = false
(20 > 3) = true
(20 / 4 > 5) = false
(20 >= 3) = true
(20 / 4 >= 5) = true
(20 < 3) = false
(20 / 4 < 5) = false
(20 <= 3) = false
(20 / 4 <= 5) = true
*****
*****Conditional Operators*****
(20 == 20 ) && (15 >= 1) = true
(20 == 20 ) || (15 >= 1) = true
(20 == 20 ) && (15 >= 1) || (true == false) && (1 == 4) = true
*****
C:\Users\jim\Documents\UMUC\Courses\CMIS141\Spring2015\Code\week1>
```

I recommend you take your time as you experiment with the conditional operators. Be sure to apply truth tables and precedence rules as needed. Notice in the last conditional operator example, the results return true. You have to apply the conditional operator logic but also apply the precedence rules where && operators take place prior to evaluating || operators. For example, for the following expression, we evaluate the expressions in the parenthesis first, then evaluate the && operators and then finally the || operators. The following steps represent the sequence of steps the computer would use to evaluate this complex statement.

```
flag = (20 == 20 ) && (15 >= 1) || (true == false) && (1 == 4);

// Evaluate each parenthesis first

flag = (true ) && (true) || (false) && (false);

// Then evaluate the &&

flag = true  || false;

// Finally, evaluate the ||

flag = true;
```

- f. Bitwise operators apply the same conditional logic as shown in the truth table; however; the logic is applied to each bit of the number to determine the results. Also, in binary, true is represented by 1 false is represented by 0. Table 2, is a the truth table with true replaced with 1 and false replaced with 0.

Table 2. Truth Table for && and || (1,0 version)

Condition1	Condition2	Condition1 && Condition 2	Condition1    Condition 2
1	1	1	1
1	0	0	1
0	1	0	1
0	0	0	0

The following represents applying the truth table shown in Table 2 to two different binary numbers.

```
int Num1 = 0b0001;
int Num2 = 0b0011;
```

Num1 & Num2 is equivalent to this operation for each bit:

0	0	0	1
&			
0	0	1	1

Looking at each bit from left to right, we have 0 && 0, 0 && 0, 0 && 1, 1 && 1. Applying the conditional operator will result in 0b0001 = 1 in decimal form. These can be tedious to conduct by hand as you have to perform the conditional operation on each bit.

Similarly, if we performed a bitwise || on these same two numbers, the results would be:

```
int Num1 = 0b0001;
int Num2 = 0b0011;
```

Num1 || Num2 is equivalent to this operation for each bit:

```
0 0 0 1
|
0 0 1 1
```

Looking at each bit from left to right, we have 0 || 0, 0 || 0, 0 || 1, 1 || 1. Applying the conditional operator will result in 0b0011 = 3 in decimal form.

Hexadecimal integers can also be used in bitwise conditional operators. One approach to understanding how to perform conditional operators to hexadecimal values is to first convert the value to binary, then follow the approach discussed above to perform the operation.

Consider to hexadecimal values:

```
int Num1 = 0x0004;
int Num2 = 0x000A;
```

Converting to binary (using any online calculator, or by hand) will yield

```
Num1 = 0b0100;
Num2 = 0b1010;
```

Num1 && Num2 is equivalent to this operation for each bit:

0	1	0	0
&			
1	0	1	0

Applying the conditional operator will result in 0b0000 = 0 in decimal form.

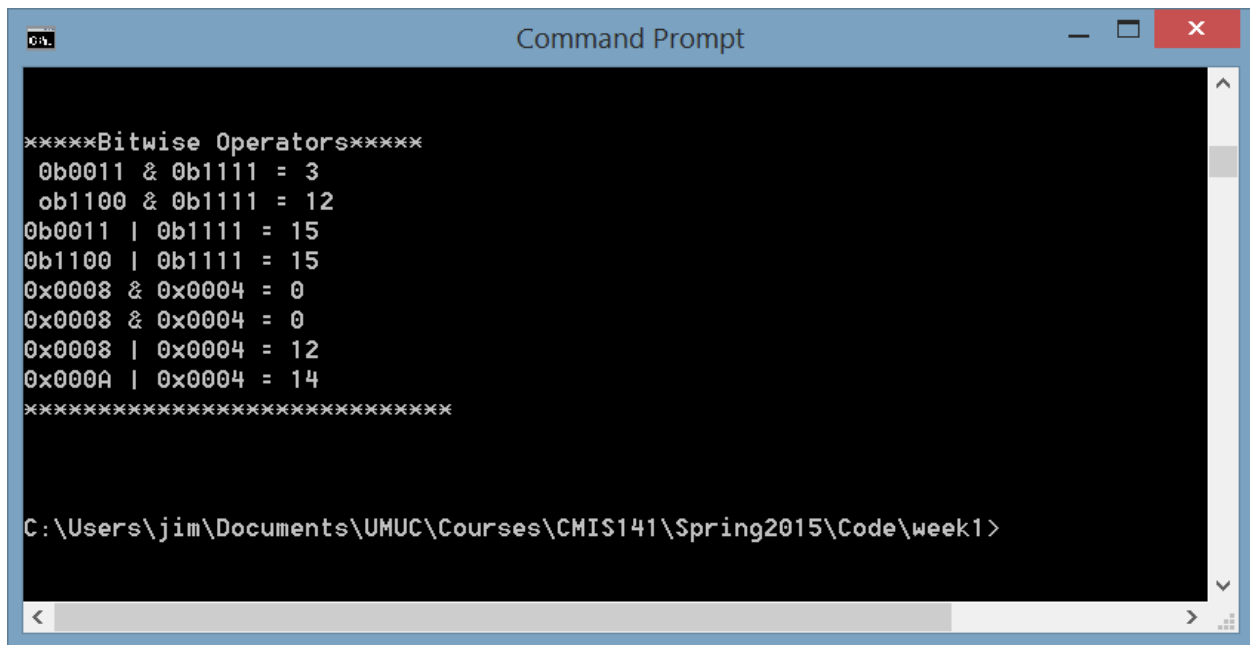
Similarly, Num1 | Num 2 becomes 0b1110 = 14.

To experiment with bitwise conditional operators, add the following code to the OperatorsDemo.java main method.

```
// Bitwise Operators
    System.out.println("*****Bitwise Operators*****");
    int mask = 0b1111;
    int val = 0b0011;
    int val2 = 0b1100;
    int hmask = 0x0004;
    int hval = 0x0008;
    int hval2 = 0x000A;
    // Apply the & to each bit
    System.out.println (" 0b0011 & 0b1111 = " + (val & mask) );
    System.out.println (" 0b1100 & 0b1111 = " + (val2 & mask) );
    // Apply the | to each bit
    System.out.println ("0b0011 | 0b1111 = " + (val | mask) );
    System.out.println ("0b1100 | 0b1111 = " + (val2 | mask) );
    // Apply the & to each bit
    System.out.println ("0x0008 & 0x0004 = " + (hval & hmask) );
    System.out.println ("0x0008 & 0x0004 = " + (hval2 & hmask) );
    // Apply the | to each bit
    System.out.println ("0x008A | 0x0004 = " + (hval | hmask) );
    System.out.println ("0x000A | 0x0004 = " + (hval2 | hmask) );
    System.out.println("*****");
    System.out.println("\n");
```

Running the compiled code will result in this output:





```
****Bitwise Operators****
0b0011 & 0b1111 = 3
0b1100 & 0b1111 = 12
0b0011 | 0b1111 = 15
0b1100 | 0b1111 = 15
0x0003 & 0x0004 = 0
0x0003 & 0x0004 = 0
0x0003 | 0x0004 = 12
0x000A | 0x0004 = 14
*****

C:\Users\jim\Documents\UMUC\Courses\CMIS141\Spring2015\Code\week1>
```

Starting with the sample code will allow you to quickly experiment with bitwise conditional operators. I recommend you open up your favorite online hexadecimal to decimal to binary convertor. This will help you double check the values coming from the application as well as experiment and try different values or binary and hexadecimal values.

Now it is your turn. Try the following exercises: (Hint: Start with the OperatorsDemo.java and make changes as needed.

1. Create a new Java file and name it MyOperators.java. Be sure you include the correct public class syntax. Since the file is named MyOperators.java, the public class syntax should be "public class MyOperators". Also, be sure the content in the .java is correct and includes the main method.
2. Declare and initialize a variety of int and double variables. Initialize the variables to int and double values of your choice. Use arithmetic operators of your choice and print the results to the screen.
3. Using the existing variables, create your own complex arithmetic statement that includes, parenthesis, and at least 4 arithmetic operators.
4. Declare two new additional int variables and assign initial values of 20 and 100. Add Java code to perform and print pre-increment and post-increment results for both values. In addition, explore the Not (!) Unary operator to demonstrate the flipping of an expression from false to true.
5. Using the existing variables, use the equality operators to compare variables and print the results of the comparisons.
6. Using the existing variables, create your own complex Java statement that uses && and || and print the results.

7. Create two binary and two hexadecimal integers and perform bitwise & and | operators on each. Print the results and be sure to verify the results by hand.