Java Data Types and Variables - LAB

The following instructions are to follow the lecture on types and variables. These should be done in a tool that allows for immediate compilation as some of the instructions will be to enter something that will break. This is in order to see how the compiler enforces some of the language rules. There will also be heavy use of the the `System.out.println()` method to see the results once the code compiles.

Keep in mind the instruction *can* be written in a plan text editor and then compiled on the command line using `javac` but the process will be much slower.

As discussed in the lecture, primitive types are those that exist directly in memory. As such we will not be talking about class variables directly here - that is for a completely different lecture. However, since primitive types most commonly exist in two places, class variables and method variables, it may be necessary to use certain aspects of classes during this lab. Don't worry however, the details of classes will be fully explained in the upcoming lecture.

First, there has to be a basic program to begin experimenting with variables. Start with a very basic class with `main()` method and `Hello World!` so that it can be run easily.

```
public class Main
{
  public static void main(String[] args)
  {
    // TODO Auto-generated method stub
    System.out.println("Hello World!");
  }
}
```

Declare two variables in the class:

- an integer named `methodCount`
- a string varible named `className`

Think about:

- Should these varibles have any modifiers on them?
- Should these variables have an initial value?
- If not, do they have an initial value? How could you tell?
- Where are these varibles visible?

Once these variables have been properly declared they can be referenced in the `main()` method. Write two statements to print out the value of each variable.

Next, declare an integer variable inside of the `main()` method named `mainInt`. Using the add-assignment operator, add the value of `mainInt` to `methodCount` and print the value of `methodCount` to ensure the value is correct. What happens if `mainInt` is not initialized?

Declare a floating point varible inside of the main method and initialize its value to `845.4f`. Now declare an integer and try to initilaize it with the value of the floating point varible. What happens?

To assign the value of a float to an int the cast operator must be used. Use this operataor and then print the value of both the float and the integer. What do you notice?

At this point your program and the output should be similar to this:

```
public class Main
{
  static int methodCount;
  static String className;

  public static void main(String[] args)
  {
    int mainInt = 1;
    // TODO Auto-generated method stub
    System.out.println("Hello World!");
    System.out.println("Method count: " + methodCount);
    System.out.println("Class Name: " + className);

    methodCount += mainInt;
    System.out.println("Method count: " + methodCount);

    float f1 = 856.2f;
    int castFloat = (int)f1;

    System.out.println("Float: " + f1 + ", Cast float: " + castFloat);

  }
}
```

```
Hello World!
Method count: 0
Class Name: null
Method count: 1
Float: 856.2, Cast float: 856
```

Next, declare a couple of integer variables with initial values. Add them together and place the result in a third variable. Then print that value.
Declare another variable with an initial value. Construct three different statements

- x + y * z
- (x + y) * z
- x + (y * z)

Print the values of each operation. Which are the same? Which are different? Why?

Using the add-assignment operator, increment the value of x by 15 and print that value.
Next, increment the value of x by the value of y and print that value.
Finally, to prove that a compound statement can be on the right side of a shortcut operator, type in the following and then print the value.

```
z += (x*2)-6;
```

Now lets move on to boolean operators. Reuse the previous int values and declare them as follows (I've used x, y, and z. Yours may be different)

```
x = y = 13;
z = 15;
```

After that print the value of the following expressions:

- x == y
- x <= y
- x != y
- x > y
- z < (x+y)
- z > (x+2)

For binary operators, write the statements that will verify the first 6 lines of the table on binary operations. The statements should be in the form (using slightly different varible names):

```
b3 = b1 & b2;
System.out.println("50 AND 19 is: " + b3);
```

For the right shift carry and non-carry operators, type in the following and note the output:

```
b3 = b2>>>2;
System.out.println("19 SHIFT RIGHT (NON CARRY) 2 is: " + b3);

b3 = (~b1)>>2;
System.out.println("50 COMP SHIFT RIGHT (CARRY) 2 is: " + b3);

b3 = (~b1)>>>2;
System.out.println("50 COMP SHIFT RIGHT (NON CARRY) 2 is: " + b3);
```

Strings

Declare three string variables, `firstName`, `lastName`, `middle` and initialize them with the appropriate values. Use different methods; one directly with a literal string, one with the `new` operator, and one with assigning a value after initialization. Print out each variable to ensure that the values are correct.

Create a string called `fullName` that will be made from concatenating the previous three values. Will you use the `concat()` method or the `+` operator? Be sure the spacing is correct.

Create three strings with the values of "James", "James" and james. Are the first two equal if you use the "==" operator? How about the equals() method? What about the first and third if you use equals()? How can you compare them and ignore the case?

Create a string with a full sentence of a least ten words. Using that string and any resultant variables carry out the following operations:

- Check to see if the sentence contains the word 'and'.
- Declare a char variable and put the 16th letter in it.
- Declare an int variable and find the first instance of the letter 'u'.
- Declare an int variable and find the last instance of the letter 'c'.
- Declare in int and find the total length of the string.
- Does the sentence start with the word "The"?

If there are any questions about the above methods reference the documentation for a String here

Create a string variable that represents a template to print an integer. Now use that template and an integer to print the templete with the values of 22, 89, and 93.

Wrappers

Declare an integer wrapper object and initialize it with the value of 42. Declare another wrapper object and initialize it from the literal "88". Will you use a constructor or a Parse() function? Look up the details of each in the online documentation here.

Finally, to wrap things up and combine a wrapper class with a string substitution (and to prove out some of the earlier statements), enter the following three lines and examaine the output:

```
b3 = b2>>2;
System.out.println("19 SHIFT RIGHT (NON CARRY) 2 is: " + b3);
System.out.println(String.format("In binary %s", Integer.toBinaryString(b3)));

b3 = (~b1)>>2;
System.out.println("50 COMP SHIFT RIGHT (CARRY) 2 is: " + b3);
System.out.println(String.format("In binary %s", Integer.toBinaryString(b3)));

b3 = (~b1)>>>2;
System.out.println("50 COMP SHIFT RIGHT (NON CARRY) 2 is: " + b3);
System.out.println(String.format("In binary %s", Integer.toBinaryString(b3)));
```

Note that in the output, leading zeros aren't included. So the fact that the last string is 2 characters shorter than the previous one indicates that there are two leading zeros - hence zeros were inserted and the sign bit not kept.