CSC 204 Lab 12: Boolean Expressions and Strings

This lab gives you the opportunity to try out a variety Boolean expressions. It points out different precedences and shows a dangerous Java problem in dealing with numbers of type `double`. It also will give you a chance to write some code using String methods.

Goals

After doing this lab, you should be able to:

- describe how Boolean values are represented in a program.
- predict the outcome of Boolean expressions that use <, <=, >, >=, ==, and `!=`
- predict the outcome of Boolean expressions that use `&&`, `!`, and `||`
- determine the precedence of Boolean operators
- write code with String methods

Lab Preparation

To prepare for this lab, read the material on Boolean expressions in Chapter 5 of our textbook. Read through this lab carefully. Make a copy of, and then read through your Lab12 Worksheet. Look at the Boolean expressions on the worksheet and fill in the "Expected Value" fields with what you expect on your copy of the worksheet.

Fill in your expected answers on the worksheet before you do this lab.

Materials Needed

Be sure you have the following on hand during the lab.

- This sheet and your Lab 12 worksheet.
- Your course notebook and textbook.

Method

Open Eclipse and create a new Java Project named Lab12 on Orion. Copy the code from our blackhawk folder named Lab12 into your Lab12 project (remember to 'copy' it into your 'src' folder).

Working with `boolean` Values

Compile and run the program `MyBools.java`. Right now it looks at just one possible boolean expression (`true`) and prints its value. We're going to change it to look at numerous boolean expressions.

Set 1: Boolean expressions

1. Let's look at the relational operators which give us a way to compare values. Edit the program `MyBools.java`.

- Copy the lines from "`boolean B1 = true;`" through "`printBool (B1);`".
- Paste these lines further down in your program.
- In the new lines, change references of `B1` to `B2`.
- In the new lines, change the assignment statement so it reads `B2 = (3 == 3);`
- Now the body of your program should read:

```
boolean B1 = true;
System.out.print("B1 (true)");
printBool (B1);

boolean B2 = (3 == 3);
System.out.print("B2 (3 == 3)");
printBool (B2);
```

- Compile and run your program. Write down the actual value on your copy of the Lab 12 Worksheet, and compare it to your expected value.
- Try the other expressions listed in your worksheet, using the same steps as above. You should have 7 lines of output after these steps.

Set 2: A problem with numbers of type double

- Multiply 3.3 by 2.4 and put the result on the worksheet.

- Now, write the Java code to do the comparison `3.3 * 2.4 == 7.92` and print out the results. It should return true, but due to the way floating point numbers are stored, it doesn't! This is not an easy problem to solve, so for now, the best you can do is to be aware it happens. Feel free to discuss possible solutions with me if you're interested!

Set 3: Combining operators (the wrong way!)

You might think the way to combine relational operators is just by combining symbols as you do in math:
`3 < 4 < 5`. But it doesn't work in Java and here's an example of how it doesn't!

- Add Java code and try to print the results of the boolean expression (`3 < 4 < 5`) to see if it will compile. Write the compiler message you see on the worksheet.

  Now, modify this statement using the && operator so that it does compile.

- Now try to print the results of (`4 = 4`) to see if Java will allow this to be assigned to a boolean variable. Write the compiler message you see here.

Be careful. This is a very common mistake to make! The = operator is only for assignment, and == is for testing equality. Modify your operator now so that it does compile.

Set 4: && and ||

To test that all conditions are true, we will use `&&`. To test that at least one is true, use `||`. Work through the examples in set 4 of the worksheet to see that this is the case.

Set 5: ! and deMorgan

1. To negate a truth value, use !. The first two examples in set 5 show how this works in simple cases. Do them and put the results on the worksheet.
2. Negating an `&&` gives the "or" of the negated parts. We'll discuss this in class, but try the next three examples in set 5 to see that this really works! This one may need some extra work on your own! Can you find an example that shows the negation of an "or" gives the same results as the "and" of the negated parts?

Set 6: Operator precedence

Which operator has higher precedence `&&` or `||` ? Of course you can look it up, but try it out here. Consider

```
false && false || true
```

If `&&` has higher precedence, the expression simplifies to `false || true`, which is true. If `||` has higher precedence, the expression becomes false && true, which is false.

Confusing? YES! Some languages don't even allow you to mix these operators without parentheses for fear of making a mistake. Please use parentheses in your code!

Part 2: Using the String Class

The `String` class has a wide variety of methods available to manipulate it. Take a look at a few of these on page 110 of the text. We're going to try some of these out with a program to remove "cute comments" from names. Consider the name:

```
Henry "Hammerin' Hank" Aaron
```

which, without the quotes would just be:

```
Henry Aaron
```

The process to do this with `String` methods is:

- Find the location of the first double quote (`"`) in the String, using `indexOf`. In the example above, that would be 6.
- Take a substring from the first character (in position 0) to the location you found in the previous step. You'll need to use the version of `substring` with two parameters. Call the substring `firstName`. In the example above, that would give you `"Henry "` (without the quotes).
- Trim the blank spaces off the ends of `firstName`, using the `trim` method. In the example above, that would give you `"Henry"` (still without the quotes).
- Find the location of the last double quote (`"`) in the String, using `lastIndexOf`. For the example above, the index would be 21.
- Take a substring from the space after the last double quote (whose position you found in the previous step) to the end of the string. You'll can use the version of `substring` with one parameter. Call this substring `lastName`. For the example above, this would be `" Aaron"` (again, without the quotes).
- Trim the blank spaces off the ends of `lastName`, using the `trim` method. For the example above, this would be `"Aaron"` (you guessed it, without the quotes).
- Create and print a new string consisting of the first name, a single space, and the last name.

Each of these steps is a single method in the `String` class.

In the file `ConvertName.java`, add the code necessary to convert a name. There are comments to describe each line. Get the program to compile and run. If you want top practice with if statements, be sure the program will work if given a name without quotes as well as one with quotes.

Deliverables

When finished, copy your Eclipse src folder into our shared Google folder, and change its name to Lab12. Also, put your copy of the Lab12 Worksheet into your shared Lab12 folder.