Boolean Logic

By: Megan Avery
Updated March 2019

Before We Begin

- Make sure you have Java & Java JDK downloaded on your computer
  - can run **`java -version`** in terminal to check
- Make sure you have IntelliJ downloaded on your computer

- *Suggested:* Watch previous Java tutorials

Reminder

A **boolean** is a primitive that has a value of either true or false

Vocab Alert!

boolean expression: a statement that evaluates to a boolean

- ex: 2 < 3

bang: denoted by an !, indicates negation. turns true into false and false into true

Comparing Numbers

== ← equals, returns true if 2 numbers or expressions are equal

!= ← not equal, returns true if 2 numbers or expressions are not equal

< ← less than, returns true if left is less than right

<= ← less than or equal to, returns true if left is less than or equal to right

> ← greater than, returns true if left is greater than right

>= ← greater than or equal, returns true if left is greater than or equal to right

= vs. ==

= is used for assigning values to variables while **==** is used for comparing 2 values, variables, or expressions. If you try to use **=** in a boolean expression it will ALWAYS return true.

Mixing these two things up can have major consequences in your code.

Comparing Objects (like Strings)

Objects, like Strings, cannot be compared in the same way that numbers are. They must be compared using either objectOne.equals(objectTwo) or objectOne.compareTo(objectTwo). If you try to compare objects in the same way that numbers are you would actually be comparing the memory addresses of the objects which is not usually what you want.

The .compareTo Method

.compareTo returns a number, this number is basically reflective of taking the object the method is being called on and subtracting the object passed as a parameter from it. So, the return value is positive if the object being called on is bigger, negative if it is smaller, and 0 if it is equal to the object being passed as a parameter.

Expressions with Multiple Parts

Sometimes boolean expressions need multiple parts, like if you want to have a boolean for a number being negative AND even, or if you want a boolean for a number even OR a multiple of 5.

&& ← put between 2 (or more) boolean expressions to require both to be true in order for the whole expression to be true

|| ← put between 2 (or more) boolean expressions to require at least one to be true in order for the whole expression to be true

- This is not a lowercase L it is the symbol above the \ on your keyboard

Operator Precedence

&& has precedence over || so any AND parts of a boolean expression will be evaluated first. It is common practice to put a lot of parentheses in boolean expressions just to be extremely clear about what is evaluated first within the expression.

Evaluation of Multi Part Expressions

Multi part expressions are evaluated from left to right only as far as it takes to get the final answer. So, an expression with an **&&** in it will stop being evaluated as soon as something evaluates to false and an expression with an || in it will stop being evaluated as soon as something evaluates to true. This is helpful to keep in mind as you build up more complicated boolean expressions.

De Morgan’s Law

What is it? A set of rules used negate boolean expressions

!(a && b) == !a || !b

!(a || b) == !a && !b

The End