**Operators and Decision Structures**

**Normal Operators**

Operators in programming are (mostly) the same as you use in math:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| + | - | \* | / | %\* |
| Addition | Subtraction | Multiplication | Division | Modulus (Remainder) |
| > | < | >= | <= | =\*\* |
| Greater Than | Less Than | Greater Than or Equal to | Less Than or Equal to | Equals |

\*Modulus is used to find the remainder. For example, 6 % 3 would equal 0, because 3 divides evenly into 6. However, 6 % 4 would equal 2, because 4 divides into 6 one time with 2 as a remainder.

\*\*The equals sign is used differently in java. It’s used to assign a value. You can say that int x = 2. That means that you are assigning a value of 2 to x.

**And, Or, and Not**

In Programming, there are three other operators that are commonly used:

|  |  |  |
| --- | --- | --- |
| && | || | ! |
| And | Or | Not |

The best way to demonstrate what they do is to use a truth table:

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| True | && | True | = | True |  | True | || | True | = | True |
| True | && | False | = | False |  | True | || | False | = | True |
| False | && | True | = | False |  | False | || | True | = | True |
| False | && | False | = | False |  | False | || | False | = | False |

And (&&) returns true whenever both values are true. Or (||) returns true if at least one of the values are true. Not(!) returns the opposite of the value. If it is true, using ! would return false. For example:

boolean bool = false;

boolean bool2 = !bool; //the value of bool2 is true

**Precedence**

When you have many operators in one line, the computer has to decide which to do first. This is similar to Order of Operations from math. Here is the order of precedence:

For example,

5 + 6 % 4 – 8 / 2 = 3

AND

4 > 5 || 3 < 7 && 8 > 5 = true

|  |  |  |  |
| --- | --- | --- | --- |
| \*, / | % | +, - |  |
| >, <, >=, <= | ==, != | && | || |

When in doubt, use parenthesis. Expressions separated by parenthesis always come first.

**== and .equals**

The = sign is used to assign value to an object. However, == is used to check value. For example, saying 6 == 6 would return true, and 3 == 4 would return false (assuming they are ints) This works for most primitive types. However, the .equals method is used to compare objects and strings. For example, (“Hello”).equals(“Hello”) would return true while (“Hello”).equals(“hello”) would return false.

String s = new String(“Hello”);

String y = new String(“Hello”);

return s == y; //returns false;

This is because the == is comparing the reference and not the value. It still works for primitive types, but not for objects.

**if-else Statements**

In programming, it’s useful to have a way to tell the computer to make decisions based on a certain condition. If-statements are a good way to do this:

int x = 0;

if(x == 0)

{

System.out.println(“x equals zero”);

}

else if(x == 1)

{

System.out.println(“x equals one”);

}

else

{

System.out.println(“x doesn’t equal one or zero”);

}

Basically, this is saying that if a certain condition is true, do something. If a different condition is true, do something else. Otherwise, do something different. You can write an if-else ladder slightly differently as well:

int x = 7;

if(x > 0)

{

System.out.println(“x is greater than zero”);

}

if(x > 5)

{

System.out.println(“x is greater than five”);

}

if(x > 10)

{

System.out.println(“x is greater than ten”);

}

else

{

System.out.println(“x is less than zero”);

}

In this case, it would print out “x is greater than zero” and “x is greater than five”. If we had written it with else-if instead of just if, it would only print out “x is greater than zero” because it would see that the condition was true, execute the line, and jump to the end. In this case, it sees that the condition is true, executes the line, and moves on to check the next condition.

If statements are a place where operators are commonly used. For example:

int z = 7;

int p = 4;

int h = 9;

if(z \* p > z \* h && h \* p > h \* z)

{

System.out.println(“p is the greatest”);

}

else if(p \* z > p \* h && h \* z > h \* p)

{

System.out.println(“z is the greatest”);

}

else if(z \* h > z \* p && p \* h > p \* z)

{

System.out.println(“h is the greatest”);

}

else

{

System.out.println(“IDK”);

}

**Switch Statements**

If-statements aren’t the only way to check a condition. Switch statements are another variation. We could turn the first if-statement above into a switch statement:

int x = 0;

switch(x)

{

case 0:

System.out.println(“x equals zero”);

break;

case 1:

System.out.println(“x equals one”);

break;

default:

System.out.println(“x doesn’t equal one or zero”);

break;

}

This is basically doing the same thing. Where it says case 0, that is basically saying if(x == 0). It is necessary to include a break after every line; otherwise, it would continue to execute every command after that, even if x didn’t have the right value. Because we have the proper break statements, it will only print “x equals zero”. If we got rid of the breaks, it would print “x equals zero”, “x equals one”, and “x doesn’t equal one or zero”, which really doesn’t make sense.

**Projects**

Here are some projects that will make sure that you know everything you need to from this lesson:

Don’t forget, you can always ask and answer questions on the website, and if that doesn’t help, bring your questions to our next meeting.