**Primitive Types: Post- Activity**

You can assign a value to a variable in one of several ways including:

* Type a literal ***value*** after the equals sing (x = ***12*,** isGood= ***true***, etc.)
* Assign the value of one variable to another (x = y)
* Use an expression combining the two (x = y + 43)

In the examples below, the literal values are in bold italics. Next to each example, write a sentence describing what each line of code means.

|  |  |
| --- | --- |
| int size = **32**; | Declares an int named size, assigns it a value 32 |
| char initial = “**j**”; |  |
| double d = **456.709**; |  |
| boolean isCrazy; |  |
| isCrazy = **true**; |  |
| int y = **x + 456**; |  |

From the following list, circle the statements that would be legal if these lines were in a single method:

1. int x = 34.5;
2. boolean boo = x;
3. int g = 17;
4. int y = g;
5. y= y + 10;
6. short s;
7. s = y;
8. byte b = 3;
9. byte v = b;
10. short n = 12;
11. v = n;
12. byte k = 128;

In the computer activity, you saw that you can’t put a large value into a “small cup.” For example, you can’t pour an int-full of stuff into a byte-sized container, as follows:

int x = 24;

byte b = x;

//won’t work!!

Why doesn’t it work, you ask? After all, the value of x is 24, and 24 is definitely small enough to fit into a byte. You know that, and I know that, but all the compiler cares about is that you’re trying to put a big thing into a small thing and there is a possibility of spilling. Don’t expect the compiler to know what the value of x is, even if you happen to see it literally in your code.

|  |
| --- |
| **Primitive Types** |
| Type | Bit Depth | Value Range |
| **boolean and char** |
| boolean | (JVM-specific) | *true* or *false* |
| char | 16 bits | 0 to 65535 |
| **numeric (all are assigned)** |
| ***integer*** |
| byte | 8 bits | -128 to 127 |
| short | 16 bits | -32768 to 32767 |
| int  | 32 bits | -2147483648 to 2147483647 |
| long | 64 bits | -huge to huge |
| ***floating point*** |
| float | 32 bits | varies |
| double | 64 bits | varies |

We’ve talked about the sizes of various primitives and how you can’t shove a big thing into a small thing:

long y = 42;

int x = y; //won’t compile

A long is bigger than an int and the compiler can’t be sure what value has been assigned to the long. To force the compiler to jam the value of a bigger primitive variable into a smaller one, you can use the cast operator. It looks like this:

long y = 42; //so far so good

int x = (int) y; // x = 42 cool!

Putting in the cast tells the computer to take the value of y, chop it down to int size, and set x equal to whatever is left. If the value of y was bigger than the maximum value of x, then what left will be a weird (but calculable\*) number:

long y = 40002;

 //40002 exceeds the 16-bit limit of a short

short x = (short) y; //x now equals -25534!

Still, the point is that the compiler lets you do it. And let’s say you have a floating point number , and you just want to get at the whole number (int) part of it.

float f = 3.14f;

int x = (int) f; // x will equal 3

*\*It involve sign bits, binary, two’s complement and other geekery.*

\*Adapted from *Head First Java* by Sierra and Bates