Transcript: Finer Details

>> In the second part of the class notes on data expressions, we're gonna go back and pick up some details that we omitted on our first pass of these features. So we're gonna take a look at primitives again and the details there. Character sets. Operator precedence. Increment and decrement operator, we looked at this briefly but now we'll go through the idea of the prefix and postfix form.

Data conversion and reading input. So let's take a look at numeric primitive types, recall, we we have a number of these. Multiple types for integer and floating point. They're all different sizes, in memory, as in the variable location and that dictates the range of possible values. So, this this, table here essentially gives all those ranges.

A byte is gonna be eight bits and it can take on the values -128 to +127. Short is 16 bits and we can go plus or minus 32k. An int 32 bits, plus or minus 2 gig. A long 64 bits is pretty large, we go from -9, to 10 to the 18th, up to 9 x 10 to the 18th.

And then we have our two floating point values or types or float in double. The float is 32 bits and there you see we got, we can do a plus or minus 3.4 times 10 to the 38th with 7 significant digits and then the double gives us 64 bits, just +/- 1.7 x 10 to the 308, much, much larger, with 15 significant digits.

Now the important thing here about these, floating point numbers, is the number of significant digits. A float only has 7 and a double has 15 but notice we can have, 308 places here as in zeros we're multiplying by 10 to the 308. So essentially what you have to imagine here, in this digital representation, is that the further out you go on the number line, zero is in the middle there, so the further positive you go or the further negative you go, the further apart the numbers get because there's only 15 significant digits.

We'll see this in some examples a little later, in the course. So, looking at numeric primitive types, suppose we need a variable that's gonna hold an integer. If we're using this for the number of scheduled courses, say in in a student schedule, you know, we're not gonna need a big number there.

So, this minus 127, to 127 would be plenty, 8 bits, we could do that. If we're doing inventory, we we might using int and there we can go all the way up to two billion and change, and so that's 32 bits. You can think of this as, sort of choosing a suitcase for how much space you need, how much you'll be able to hold and, and and there we have those.

Now, in our course here, your computer or your phone or or, whatever really has plenty of space for our purposes, so we're gonna mostly use int and double values. So let's look at an example here, the types example. This is a, a fairly straight forward example, make this a little bit bigger for us.

Notice we're going to declare a byte b equal 15, a short, an int, and a long are all equal to 15. And then a, float x and y, a float x and a double y, I should say, both also equal 15. And then a char c equal to a capital A, and a boolean, equal to true here.

So, I've got a break point set, let's just step through this, I'll run it in debug mode. We'll see what it looks like over in the variables tab. I'm gonna go all the away to the end to get all on there. And so, we got bite and if we, pull this out, we could put it on the canvas or just pull it out into a separate viewer.

Here's what a byte looks like, it's 15, but we can, look at it with different views. This is the detailed view you're seeing now, if we change it to numeric view, we actually get to see its decimal value, hex value, octal value, and binary value. So there's the eight bits if you will and if we pull out, I'm going to go and pull out an int here.

It's, it's essentially the same, same values. Let's look at the numeric view. The, the difference is it's got 32 bits, instead of, just 8. And, if we pull out a double, let's pull out the float first here. Here's a float. We look at the details of this, this is a bit different.

So this is, where we see our scientific notation, if you will, with the sign, exponent and mantissa, we see the, those in the, in the, in the binary, octal and decimal. I think this actually hex here. And looking at the arithmetic, we got the sign, it's positive, since it's zero, zero means positive, the exponent is, is 30 minus a bias of 127 equals 3.

The mantissa is an assumed 1 plus a long number there divided by 2 to the 23, which gives us 1.875 and when we do the arithmetic we end up with 15. And, you can look at the mantissa, the mantissa is where we think of the significant digits and then there you see what we've got.

Now if we pull out a double. And we'll look at its detail view or numeric view rather. Notice it's got a few more digits, it's 32 bits for one thing, but notice the uh,the mantissa here, and let's look at it right here. Get a little more room. Comparing the two, see if we compare there, both of these are 15, both of these are the number 15 but notice the mantissa is quite a bit different.

This has the 7 digits of accuracy and this has more, and so on, so the the underlying representation is actually quite different even though they're they're the same value. You got the same value, 15, but underlying representation's quite a bit different. And I guess more to the point.

If we're looking at an int, this is our 32 bit int, this is our 32 bit double. Both of these, are the same value 15, but if we wanted to add these together, we would have to sort of do some conversion here and what happens is, the int gets promoted, to a double so this 15 gets put into this representation and then the arithmetic can can take place.

We do floating point arithmetic then and the results in this case would be a double. So looking at how they're represented underneath the hood, sorta shows why, when you're adding an int and a double that the int has to get promoted before we can do the arithmetic and, and if the result is actually going to be a double.

Now we could of, actually put these on the canvas. On a canvas, we saw that earlier. And if we created a canvas, I'm a go and create a blank one here. Again, you can drag these on here, and, and, put in whatever, in whatever representation you'd like to have them.

Here's here's that char, just to put a couple of on here. And, to change its viewer once it's on the canvas, you click the little menu button there and, choose a viewer. So it's numeric viewer for char looks like this And, and so on. You see a char here is 16 bits, to represent an A, and so on.

So canvas is useful, you can save it, but more importantly, I wanted to show you the underlying representations of, of all these items. We go back to the nodes here, and you'll see, these are the separate viewers. Now, the disadvantage of a viewer, they're, they're quick to open, but once the program ends, they disappear.

But whereas the canvas, if you created a canvas and saved it when you run the program in the canvas, it'll actually open that canvas and you'll have it to go again.