Transcript: Data Conversion

>> Now, let's take a look at data conversion. Sometimes it's necessary to convert data from one type to another. Recall, Java is strongly typed, so every variable has a type. But sometimes we need to convert it from one type to another. We may wanna treat an integer as a floating point value.

And when we do this sort of thing, conversions need to be handled carefully to avoid losing information. We have this idea of narrowing conversions and widening. Let's look at narrowing first. That's going from a large data type to a smaller one, or from a floating point type to an integer type, which has less detail.

Even though a floating point type may be the same size, for instance 32 bits for a float and an int, could, is 32 bits. The floating point type can hold much smaller and much larger values than the int type. So we think of that as narrowing when you go from a floating point type to an integer type.

So for example here, if we had an int value of 700, and we were trying to store that on a byte, which can only go to 127, we would lose information. So that's a narrowing thing. Think about your grade of 89.8, a double. If that's converted to an int type, the new guy, you would be 89.

We would lose that fraction there when we went on int, and that wouldn't be a good thing. On the other hand, widening conversions go from a smaller to a larger data type, or from an integer type to a floating point type, again, which has more detail. If a byte with a value of 95 is converted to an int, no problem.

The new value's still 95, we didn't lose a thing. And you know, then your grade could go to up to two meg there, or two gig, I should say. So let's talk about data conversion and how it happens. This narrowing conversion think of our suitcase analogy here. This is gonna be not good if the big suitcase is full.

Not so bad if it's small, but you do need to take care there when you convert. Now, Java does data conversion in three ways. There's assignment conversion, and promotion, and casting. The first two, more or less, take place by implication. The last one is, is more explicit. Casting requires you to do something explicit, and we'll see how that works there.

Transcript: Data Conversion

So let's look at assignment conversion first. Here, if we've got a value of one type, and it's assigned to a variable of another type, as long as they're compatible, if the assignment is compatible, it's not gonna be an issue. Or if it's a widening, conversion. That's gonna work as well.

So here we've got variable money as type double. And then we've got variable dollars as type int. So here we've got an assignment, essentially an int into a double. And that's gonna be okay. There's a value inside dollars in int, and and that same value now is in money, but it's been converted to a double.

And that's a different underlying representation, but we didn't lose anything in the process, any accuracy, if you will. Now, when this takes place, the type and value of dollars, didn't change. In other words, dollars, we just took the value that was in dollars, and during the assignment, it got converted, but dollars is unaffected here.

So Java allows only widening inversions, when when we do assignment conversion, so so only widening ones are permitted. If you try narrowing one, it would actually not compile. So, let's look at, at data conversion by promotion. Promotion happens when operators in expressions convert their operands. We've talked about this a little bit previously.

Here we've got an example where sum is a double, as is result. So we've got sum in result, and count is an int. And here we wanna divide sum by count. Since count is a double and int is rather, sum is a double, and count is an int, in order to do the division, the value from count here has to get converted to a double.

And then the division takes place, and the result is a double. And it's assigned in the variable result, in this case. But notice, this promotion takes place in expressions. And it's essentially done automatically in order to do whatever the operator is, when you've got two different types there.

The third way here is casting. Casting, allows narrowing conversions and widening, but you do have to be careful here, and know what you're doing. It's also easy to detect in code, because it's a specific, thing that you enter. And you can do a search and find that pretty easily.

So to cast, the type in parentheses is simply placed in front of the value being converted. For example, if total and count are integers, and the value total, could be converted to a double with a cast to avoid integer

division. So here's the example. Total and encounter integers, let's say, and the result, this looks like we want an average, maybe.

Let's assume we want an average. We're dividing some, some total value by a count. Well, if we don't, if integer division is done here, we lose the fraction, and so it's not gonna be much of a an average, so to speak. But on the other hand, if we promote or cast, total to a double, then we've got a double involved, and then count will automatically get promoted, since it's an integer.

And then the division will place, and we won't lose our fraction there. So that's kind of what we've got in the way of casting, and you could do any cast. You could, try this in interactions. You could key in, open parenthesis int close parenthesis, and then follow that by something with a decimal point, say 12.999.

And, and the result will be 12 because that double is being cast to an int, in that case. Now let's consider constants. A constant is similar to a variable, but as its name suggests, it's gonna have a value that we don't expect to change. Therefore, it's called a constant.

We place these at the class level. And meaning, if we've got just a main method, this would be up above main. And and we also write the variable name in all CAPS, and we use underscores to sort of divide the words since we're not using camel case. And once it's assigned its initial value, it can't be changed.

If anybody tries to change it in the code elsewhere, it won't compile. Now, we also use the static modifier, so they can be shared among all methods in the class. And and then the final modifier is used to prevent the initial, value from changing. So, if we had the, the constant, min height, we would write it in all CAPS and underscore to separate the words there.

And then we would modify it with static final int as the type, and could be double, whatever we want whatever we need it to be, but we need it to be static and final. Static says it could be shared. Final says it can't change. And then we give that, initial value there.

If you do try to change the values I indicated, the the compiler is going to, issue an error. Now, constants are useful for three important reasons. They improve code readability. For instance, suppose we had the constant MAX_LOAD, or 250. If you're reading coding you see the literal integer, 250.

We call that a magic number because after some time nobody knows what 250 meant. But if you use the variable MAX_LOAD, it gives it meaning. We can actually see that means max load. Second, constants facilitate program maintenance. If a constant is used in lots of different places, its value only needs to be updated in one place, and that's where it was initialized.

And so that's gonna save some time. And then third, as we mentioned, they prevent a value from changing to avoid inadvertent errors by other programmers. If a programmer actually tries to assign a value to a constant then then it won't compile. Now, again, we use all CAPS, to indicate this constant, simply because that's very recognizable.

So as a programmer's reading, that's a convention. That's not a Java thing, but a style checker will check for that. So, if you're reading a program and you see a variable all CAPS, you immediately think this is a constant, and I can't change it.