Transcript: Comparing Data, Values & Objects

>> Let's consider the following. Numeric types (including their corresponding wrapper classes) and type char can be compared using the equality and relational operators. Since numbers and char have an underlying, numeric value, it's meaningful to think in terms of equality and less than and greater than, and so on.

One caveat, though, numeric types double and float should use == or != with care due to possible rounding. It's usually best to check that the absolute value of the difference between two doubles or two floats is less than a specified tolerance. Object types, other than numeric wrapper classes, can be compared using equality operators.

But in most cases, the object's equals or compareTo method should be used. Let's talk about char. Recall that a char, type char, is represented by a 16-bit numeric value, it's Unicode. And the letters A through Z, capital A through Z, have numeric values 65 to 90. The lowercase letters, a through z, and numeric values 97 to 122.

Notice the difference between a capital A and a lower case a is 32. So what happens if you add 32 to an uppercase char value? For example, we could add 32 to the char G, and see what the char value is. We would assume it would be a lower case g.

Let's take a look, in interactions. So first let's just key in a char G, char uppercase G. And that should just evaluate to the g char, just as you expected there. And now let's see the underlying, numeric value is. I'm gonna cast g Do an int, and we see it's a 71.

Now, we're gonna add 32 to, g, so let's, Just add 32 and this is an integer literal of course. And, we get 103. And now let's see what, what 103 is, that, that, sum there is as a character. So we can do that. I'm just going to, cast, That sum, if you will.

To the char. And we see it's a lower case g. And you can do all this, with char values that you can use them in arithmetic and add them and so on. And you can also compare them using the, these relational operators, in fact let's see what a capital A, let's see if it's less than a, capital C.

We would expect it to be. And it is. What about a lowercase a? Is a lowercase a less than a capital C? That's false, and the reason that's false, of course, is because a lowercase a is a 97 and an uppercase A is, a 65, so a lowercase, and uppercase C rather, is gonna be a 66, 67 I suppose.

And so it's false that a lowercase a is less than an uppercase C, if you will. So we can use this information, for instance, in an if statement, if we got a, a char, say letter value, we could check to see. We wanted to determine whether it's a capital letter or not, we could just make sure it's greater, than, or equal to 65 and less than or equal to 90.

That's the range up here for A through Z, and we would end up with a capital letter. So a lot of things can be done with, with char types. They can, they're actually numeric types underneath so just keep that in mind. Now, about comparing, a double values, as we've said, when calculations are done on a double or a float, there can be rounding.

And this is due to the underlying representation, know, recall, we've got a sign, exponent, and mantissa, and this is 64-bit floating-point, standard. So rather than using ==, if you're trying to check to see if 2 doubles that have, especially if they've undergone calculations, it's best to check to see if, if, if they're closer than some tolerance so to speak.

And the way you would do that, is, is actually take the absolute value of the difference and see if it's less than, say a constant tolerance. In this case, we've got tolerance set to a 10 to the minus 6 and that's pretty close. And so, in that case, we would consider these numbers equal.

Now, comparing objects, you can also use == and != on objects. But remember that reference variables, these object variables, old memory addresses. So the results may not be what you expect. You're not actually comparing the value of the objects, so to speak, you're comparing its, its address in memory.

So you could try the following interactions. We could say string s1 is a new string Red Sox. And then we could say string s2 is a new string Red Sox. So essentially s1 and s2 have the same string value. However, if you say s1 == s2, it will be false because these are two different objects, if you will.

So rather than using ==, it would be better to use the equals method if a class provides one. And, equals method returns boolean indicating whether objects are equal as defined in the class. You can find out how that works looking, in in the Java API. So, for a string, string considers them equals, equal, if they have the same character string.

So, here, if we use, we've got the same two strings there. If we say s1.equals(s2), it's true, rather than saying == like we did previously. So keep that in mind. If you're ever, ever comparing objects for equality, if you wanna know if it's the same identical object, in other words an alias, you can use ==.

But if you wanna know if the two objects are, have the same value, as in this example here with, Red Sox for s1 and s2, then you would wanna use the equals method. In addition, some classes provide a compareTo method, that returns an int, especially if, if there's an ordering.

In fact, this gives the default ordering, if you will, for, for objects of a particular class. And the way it works is like this, you're gonna get an int back, here we're assigning it to int comparison. So here I've got obj1.compareTo(obj2). And these objects would presumably blee, be of the same, type.

Two strings, for example, or two triangles, are, assuming that triangles add a compare to, and so on. So the way this, the, the integer is interpreted, this will be, seem obvious after you get into this a bit but it does take a little bit of study. When you do this compare, obj1.compareTo(obj2), we end up with an int and if the int is less than 0.

That indicates that obj1 comes before obj2 or it's less than, if you will. If you get back a 0, that indicates that the two objects are equal. And if you get back, a value greater than 0, it turns out that object 1 is greater than object 2. So that's what we have got there.

Just keep in mind that the compareTo method implemented by the class, is gonna be, sort of specific for their objects. And they may not all work the same. So you need to probably check the Java, API for that. So here's an example here using compareTo with two string objects.

And, what happens when you do that on two strings, the individual characters are compared left to right until the method can determine if, if one is less than, equal to, or greater than. So it starts with the first character, of course. And if the first characters are different, well, it can make the determination then but if its got the same two first characters that we go to the second character and see if, if they're, they're different and so on.

So here's an example, we've got string food1 = "Apple", and then we've got food2 also a string = "Banana". And, notice we've got capital A for Apple, and capital B for Banana. And then we do a compareTo. So we say food1.compareTo(food2), and if it comes back less than 0, then we're gonna say food1 comes before food2.

If it comes back greater than 0, this comparedTo, then we're gonna say food2 comes first. Otherwise, we'll say they're the same. So what's gonna happen here? Well, Apple gets printed Apple before Banana is the result here because the capital A comes before a capital B, just like we would think.

So let's look at another example. This is essentially the same example, but now we're comparing lowercase Apple to, uppercase Carrot, if you will. And in this case, we're gonna end up printing out Carrot comes before apple, and the reason for that is that lowercase a comes after an uppercase C.

Just like showed we in, in interactions there. So fortunately string, has a compare-ignore case, and equalsIgnoreCase. Otherwise, you can always, convert both of them to uppercase, or both of them to lowercase, but here we're using the, compare-ignore case. And so when we use this, in our, when we do the comparison, we end up with apple before Carrot.

So lots of details here, but as you use, these building expressions and compare items, hopefully this will become, fairly obvious and commonplace for you.