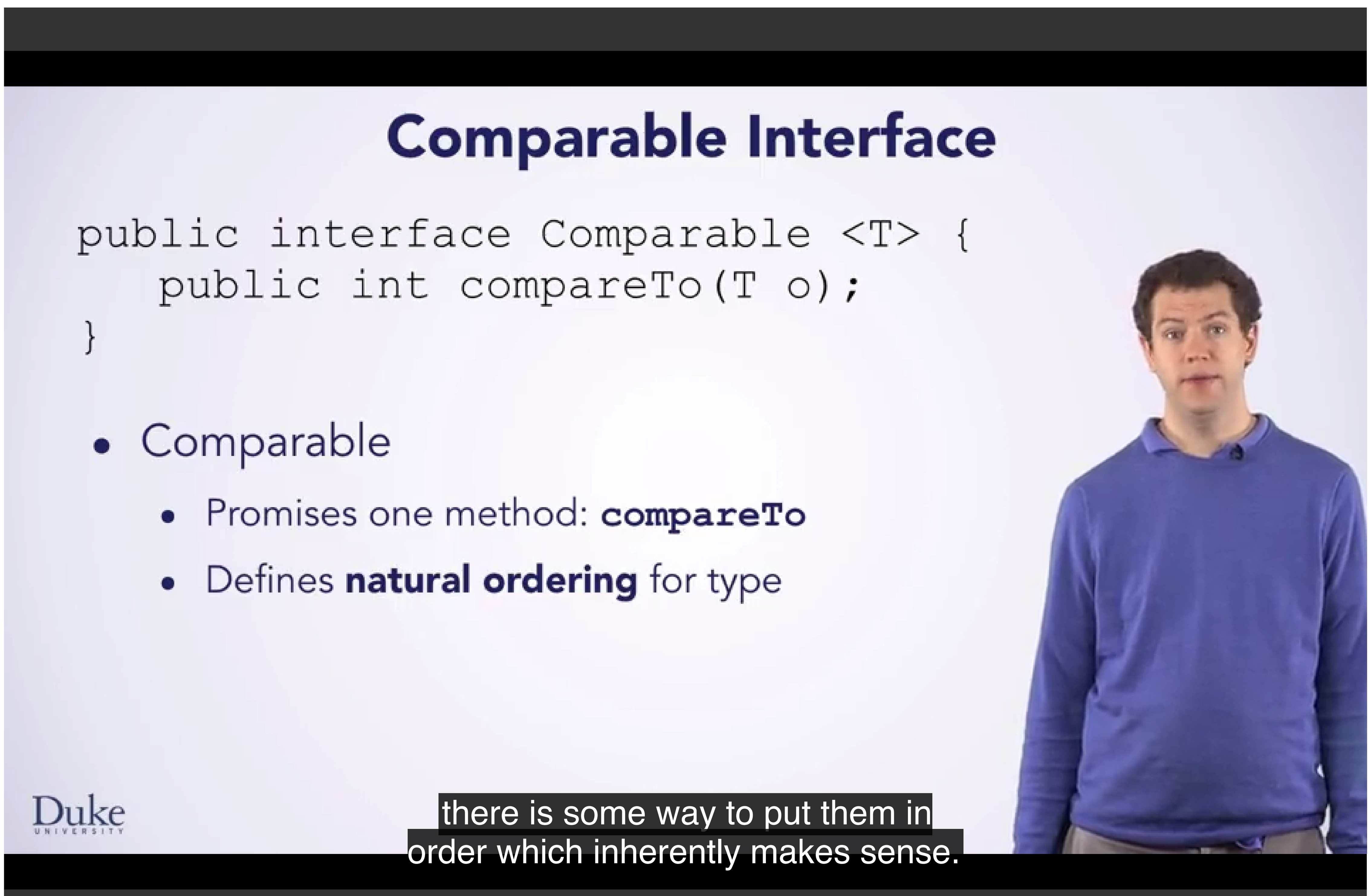

Comparable

Have a question? Discuss this lecture in the week forums.

Interactive Transcript

Search Transcript | English

0:03
Now that you have seen the compareTo method in Quake entry. It is time to learn about the comparable interface which promises this method. Classes which implement comparable have a natural ordering, there is some way to put them in order which inherently makes sense. The compareTo method which this interface promises, defines this ordering. You can see the interface definition here. It is a bit different than what you have seen before because of the angle brackets T. The angle bracketed T at the top specifies a type parameter for this interface. In this case, it specifies what type it can be compared against. QuakeEntry implements a Comparable of QuakeEntry, meaning that you can compare one QuakeEntry to another.

0:47
That type parameter can be used in the rest in the interface definition and will be whatever type is passed as a parameter. In the case of Comparable of QuakeEntry, the compareTo method would take a QuakeEntry object as its parameter.

1:03
So why does Comparable have the angle brackets T? Well, it's not just used for Quake entries, in fact, Comparable is a built-in part of Java, and QuakeEntry is not. When Java was created, its authors were not thinking about Quake entries, but instead making something generic enough to use with any totally ordered type. There are many types that implement comparable including several that you have seen before. Strings can be compared to strings, integers to integers, and so on. Let's take a minute to look at strings since their natural ordering is easy to understand. The string class implements comparable of string. You can compare any two strings to put them in order, what ordering is natural for strings? Alphabetical order. If you were alphabetize things, apple would come before bear. So apple is less than bear, which is less than cards, which is less than dino. As you may remember, strings can hold any character not just letters. So how do you alphabetize such strings, and what happens if they contain the same letter but different cases? Strings.compareTo method takes care of all of these. The ordering it uses is technically called lexicographical ordering.

2:16
That's a fancy way of saying that you look at each letter from start to end. As long as they're the same, you keep going. When you find a difference, that tells you how to order things. This is what alphabetical ordering does for letters, it is just the technical term for the more generalized algorithm for any characters. For example, if you wanted to compare "What!" to "What?", you would look at the capital W's, see that they're the same, the h's, a's, and t's are the same, but then the exclamation mark is different from the question mark. You then order these two strings based on which is greater or smaller, exclamation mark or question mark. Does that even make sense? Well remember, everything is a number. Java can just compare the two characters for you, and will do that comparison based on their numerical value. Should you remember the numerical values of these characters? Of course not. You generally don't need it, and if you ever do you can look it up or write a small test program just like we did.

3:17
To compare capital What with lowercase what, Java would find a difference in the first character. It turns out that capital letters have lower numerical values than lower case letters, so capital what is less. Comparing these last two strings proceeds much like comparing the first two. So how does compareTo return less than, equal to, or greater than? It returns an integer which is negative for less than. For example, apple.compareTo(bear) returns negative 1, of they are equal it returns 0. Such as when comparing bear to bear. Finally, it returns a positive number for greater than. Dino.compareTo ("cards") returns positive 1. Note that you should not rely on specific negative or positive values, like negative or positive 1. The method only promises to return a negative or positive number. For example, comparing apple to dino gives negative 3, and comparing lowercase what to uppercase What gives 32. These may seem weird but actually make a lot of sense. Often compared to the implemented using subtraction as is the case in strength. When it finds a difference in the characters it subtracts them from each other and returns the result. This is yet another instance of everything is a number in action.

Downloads

- **Lecture Video** mp4
- **Subtitles (English)** WebVTT
- **Transcript (English)** txt

Would you like to help us translate the transcript and subtitles into additional languages?