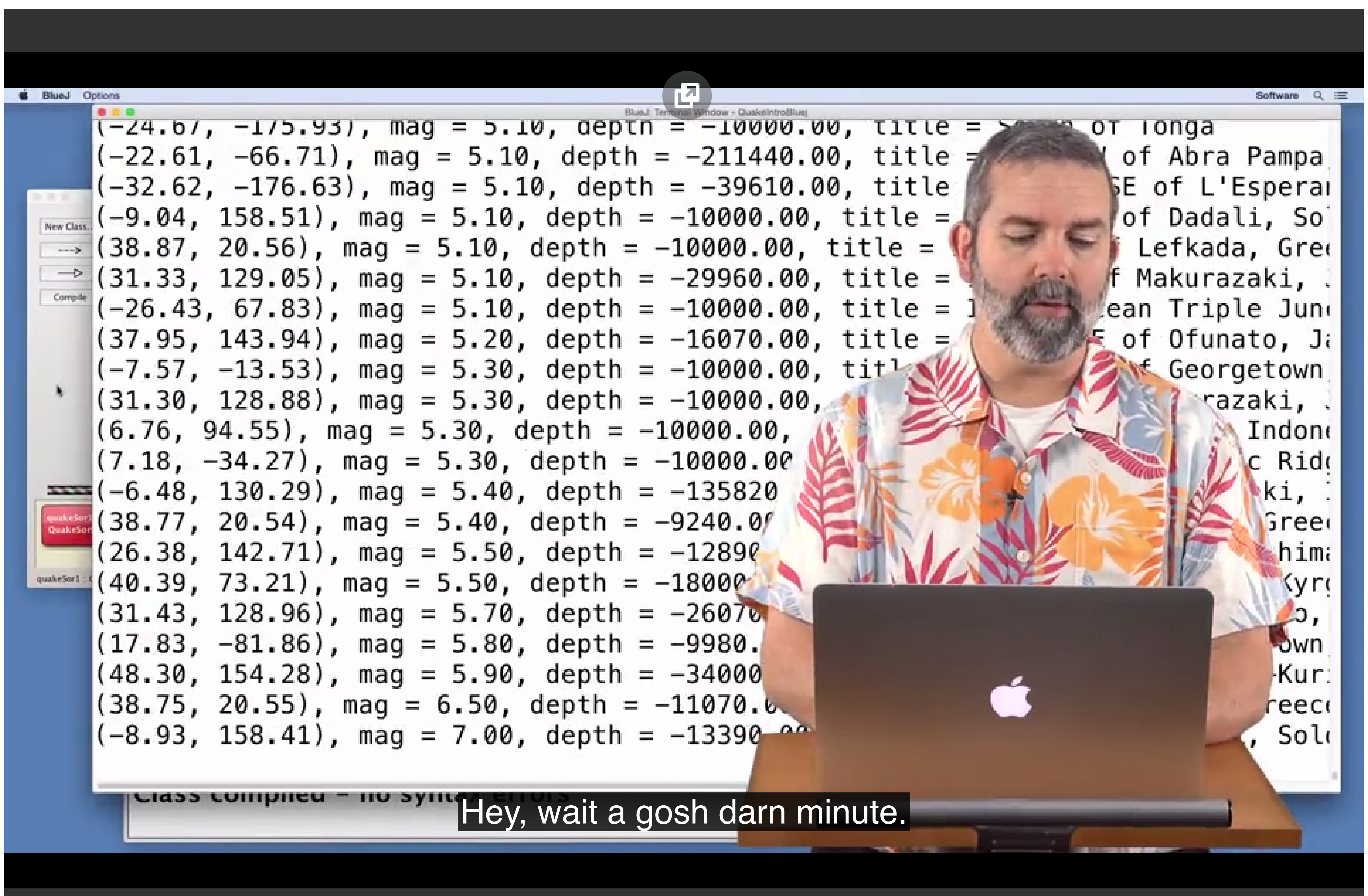

```
(-24.67, -175.93), mag = 5.10, depth = -10000.00, title = S    of Tonga
(-22.61, -66.71), mag = 5.10, depth = -211440.00, title =     of Abra Pampa
(-32.62, -176.63), mag = 5.10, depth = -39610.00, title      SE of L'Esperai
(-9.04, 158.51), mag = 5.10, depth = -10000.00, title =      of Dadali, So
(38.87, 20.56), mag = 5.10, depth = -10000.00, title =       Lefkada, Gre
(31.33, 129.05), mag = 5.10, depth = -29960.00, title =      f Makurazaki,
(-26.43, 67.83), mag = 5.10, depth = -10000.00, title = I    ean Triple Jun
(37.95, 143.94), mag = 5.20, depth = -16070.00, title =      E of Ofunato, J
(-7.57, -13.53), mag = 5.30, depth = -10000.00, titl         f Georgetown
(31.30, 128.88), mag = 5.30, depth = -10000.00,              razaki,
(6.76, 94.55), mag = 5.30, depth = -10000.00,                 Indon
(7.18, -34.27), mag = 5.30, depth = -10000.00                c Rid
(-6.48, 130.29), mag = 5.40, depth = -135820                   ki,
(38.77, 20.54), mag = 5.40, depth = -9240.0                   Gree
(26.38, 142.71), mag = 5.50, depth = -1289                     hima
(40.39, 73.21), mag = 5.50, depth = -18000                    Kyr
(31.43, 128.96), mag = 5.70, depth = -2607                     o,
(17.83, -81.86), mag = 5.80, depth = -9980.                   own
(48.30, 154.28), mag = 5.90, depth = -34000                   -Kur
(38.75, 20.55), mag = 6.50, depth = -11070.0                  reec
(-8.93, 158.41), mag = 7.00, depth = -13390                 , Sol
Class compiled - no syntax errors
```

BlueJ
Options
BlueJ: Terminal Window - QuakeIntroBlueJ
Software
New Class...
Compile
Hey, wait a gosh darn minute.

Comparator

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

Interactive Transcript

Search Transcript | English

0:03
Finally I've carefully designed and implemented a compareTo method that will compare my quakes from south to north pole order.

0:14
We can use that to sort the quakes for my important project. Hey, wait a gosh darn minute. This isn't right. Somebody changed my compareTo method. >> Yeah, I needed to sort the quakes by magnitude for my problem, so I changed the code. I'm sure magnitude is a better way to order quakes anyways. It's way more important how strong they are. >> But you just broke my code. I need to solve problems where they are ordered by latitude. >> Wait! I don't like either of those ways. I want to order the earthquakes by how far away they are from me. This is about me. That way I can solve useful problems like finding the closest earthquake or how many quakes are within a given distance of my house. >> No, no, no, no, no. I don't like any of these orderings. We should order quakes by time.

1:05
I wanna analyze earthquake activity over time, so I need to know which are recent and which happened a long time ago. >> Okay, clearly we cannot keep changing the compareTo to satisfy all the different ways people want to order their earthquakes. Instead we need another approach, a way to specify an alternate ordering.

1:26
Fortunately, Java has an interface for exactly this purpose, Comparator, which sounds a lot like Comparable but is different. The Comparator interface allows a class to define orderings for some other class, while the Comparable interface allows a class to define its own natural ordering. The Comparator interface promises one method, compare, which takes two parameters, the things to be compared. Before we seen an example of a comparator, it's helpful to understand how it's different from comparable. They seem to do pretty much the same thing. Here you can see two earthquakes, quake1 and quake2. If you do quake1.compareTo(quake2) using the .compareTo method promised by Comparable, then you're asking quake1 to compare itself to quake2. The compareTo method lives inside of one of the objects being compared. The method inside of quake1 will then look at the information it needs from itself and from quake2, for example, their magnitudes, and then make the comparison.

2:29
Now, let's look at Comparator. Here, you can see the same two earthquakes. But now we make another object to compare them, comparatorA. When you do comparatorA.compare(quake1, quake2), you are asking this object, comparatorA, to compare these two quakes. The code for the .compare method lives inside of this third object, not either of the ones being compared. This code inside of comparatorA gets the information it needs from the two quakes. This Comparator looks at their location and compares them based on their distance to its location. The benefits of comparators are that you can create some other Comparator such as comparatorB here and ask it to .compare the quakes. comparatorB could be a different type with different code in its .compare method from comparatorA. For example, it could look at their dates and order them based on which happened more recently. This is how we can solve the problem of everyone wanting to sort their quakes differently. Everyone can make a different class which implements Comparator and use it. So what would such a class look like? Here, you can see an example of a comparator, which orders earthquakes by their magnitude. Notice that the class implements Comparator of QuakeEntry. This class can be used to compare two QuakeEntries to each other. Here, you can see the compare method promised by the interface. As the interface promises, it is public and returns an int. Since this class implements comparator of QuakeEntry, the interface promises that the parameters will be QuakeEntries. The body of this method compares the two quakes by their magnitudes. This code is very similar to the compareTo method that we changed to compare by magnitude. The difference is that this method gets the magnitude out of both of its parameters, while the compareTo method got the magnitude out of its one parameter and out of the object that it lives inside of. Now that you have seen an example of how to write a Comparator, let's take a look at how to use a Comparator to sort. The Collections.sort method can take a second parameter which is a comparator. When you pass this second parameter to collections.sort, it will use that comparator to determine the ordering of the objects in the list.

4:49
Here we are passing in a new MagnitudeComparator.

4:53
Remember that when a method's parameter type is an interface, you can pass in an instance of any class that implements that interface. Here, MagnitudeComparator implements Comparator. So it is fine to pass it in for this parameter. As you learned earlier, the code inside of Collections.sort will call the right .compare method based on what type of Comparator is passed in by using dynamic dispatch.

Downloads

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

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