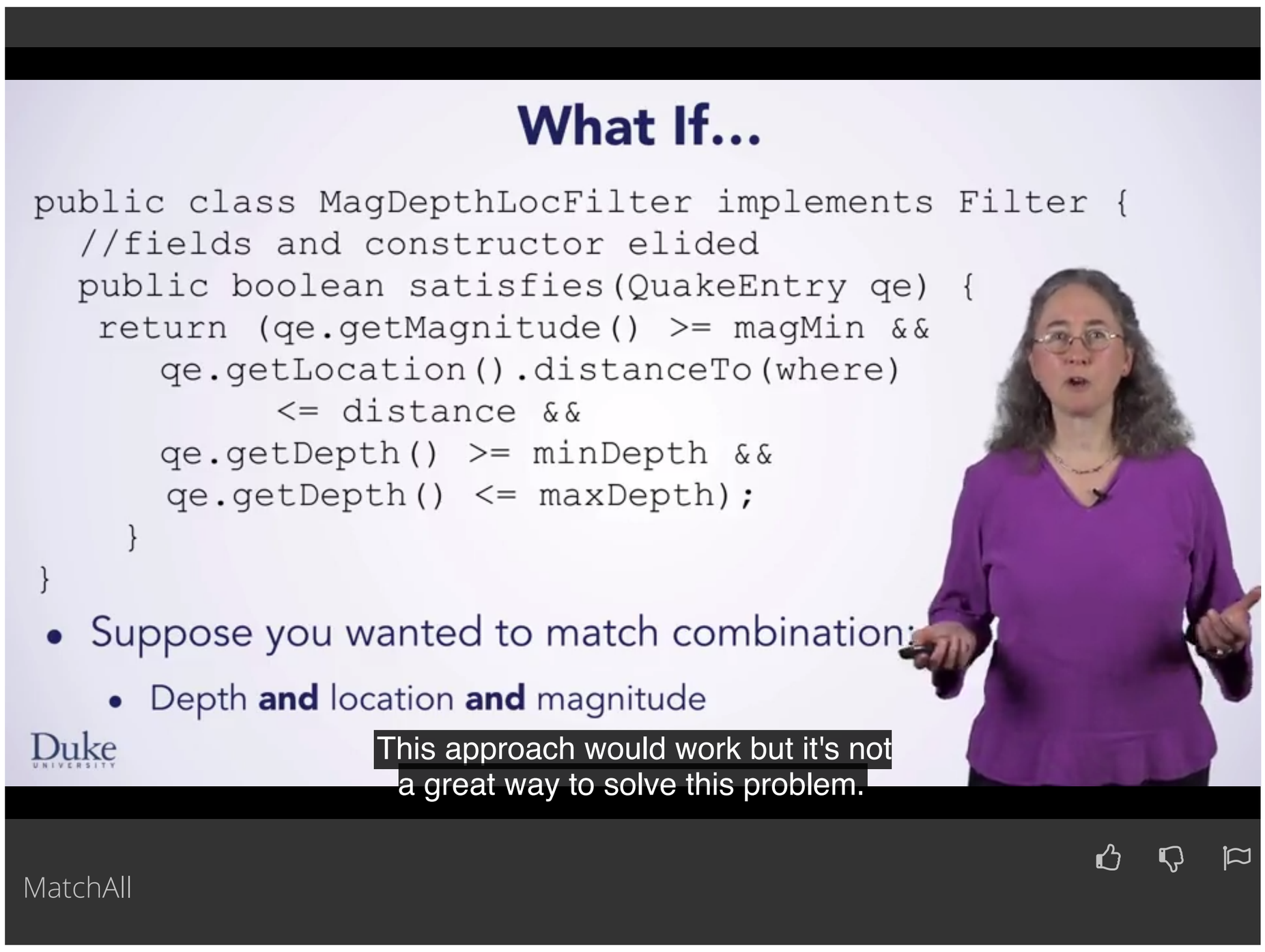

Have a question? Discuss this lecture in the week forums.

Interactive Transcript

Search Transcript

English

0:03
Suppose that you wanted to make a filter for your earthquake data that combined several other criteria. For example, you want to find the earthquakes that are in a certain range of depths, and close to a particular location. And at least some minimum magnitude. You could just go and write one big complicated filter, which would look like this. It would have a big condition that ands together all this criteria. It would also have a lot of instance variables. And of a constructor with a lot of parameters, which we don't show here. This approach would work but it's not a great way to solve this problem. Why not?

0:43
It duplicates code. You already wrote and tested filters for each of these criteria. It would be much better to make use of those. So if you already have filters that check each of these criteria, could you write another filter that combines them together making use of their existing code? To make the result even better, could you make this combined filter generic so that it could combine any set of filters, and check if an earthquake matches a criteria for all of them? If you could do this, you could write some code that looks like this. You could make a MatchAllFilter the class we want to write then add a filter for the minimum magnitude to it and then add a filter for the range of depths to it, and finally add a filter for the distance from a location to it. Then you can use this MatchAllFilter to filter your earthquakes, this approach looks great. You can reuse the filters you already made and if you want to combine other filters, you can use MatchAllFilter to do that, but how do you write MatchAllFilter? Here is half of the code for MatchAllFilter. We'll see the method satisfies in just a second, it just doesn't all fit on this screen at once.

1:57
Here you can see that we declared the MatchAllFilter class and said that it implements filter. This is just like you learned for any of the other filters you wrote. This class has a field for an array list of filters. Can you make an array list of filters? You can. Which is another great example of composability. Filter is a type. And you can make an ArrayList of any type. So this works just as you expect.

2:23
Next, there's a constructor that initializes the ArrayList to a new ArrayList. And a method that takes a filter and adds it to this ArrayList. Now lets look at the code for the satisfies method in this class like all the filters we have seen this satisfies method takes a QuakeEntry and returns a boolean. However, this filter makes it's decision differently then the ones you have seen so far. Rather than examining the earthquake itself it iterates over each of the filters in it's ArrayList. For each of these filters, it sees if this QuakeEntry satisfies that filter's conditions. Remember that this call will be dynamically dispatched, using the .satisfies method, inside whatever type of filter was actually created. If that filter return false, remember explanation mark means not, then this filter returns false.

3:17
After iterating through all the filters, if none of them rejected this quake, then this filter returns true. So now you have seen how to make the MatchAllFilter, which can combine any set of filters and see if an earthquake matches all of their conditions. It's.satisfies method is implemented by using a for each loop to check all the filters that it has stored in an ArrayList. You could also write other combinations. For example, if you wanted to write a MatchAnyFilter that tested whether an earthquake matched the conditions of any of a set of filters, you could do that with the same principles. Have fun filtering the earthquake data in search of that ultimate quake.

Downloads

Lecture Video mp4

Subtitles (English) WebVTT

Transcript (English) txt

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