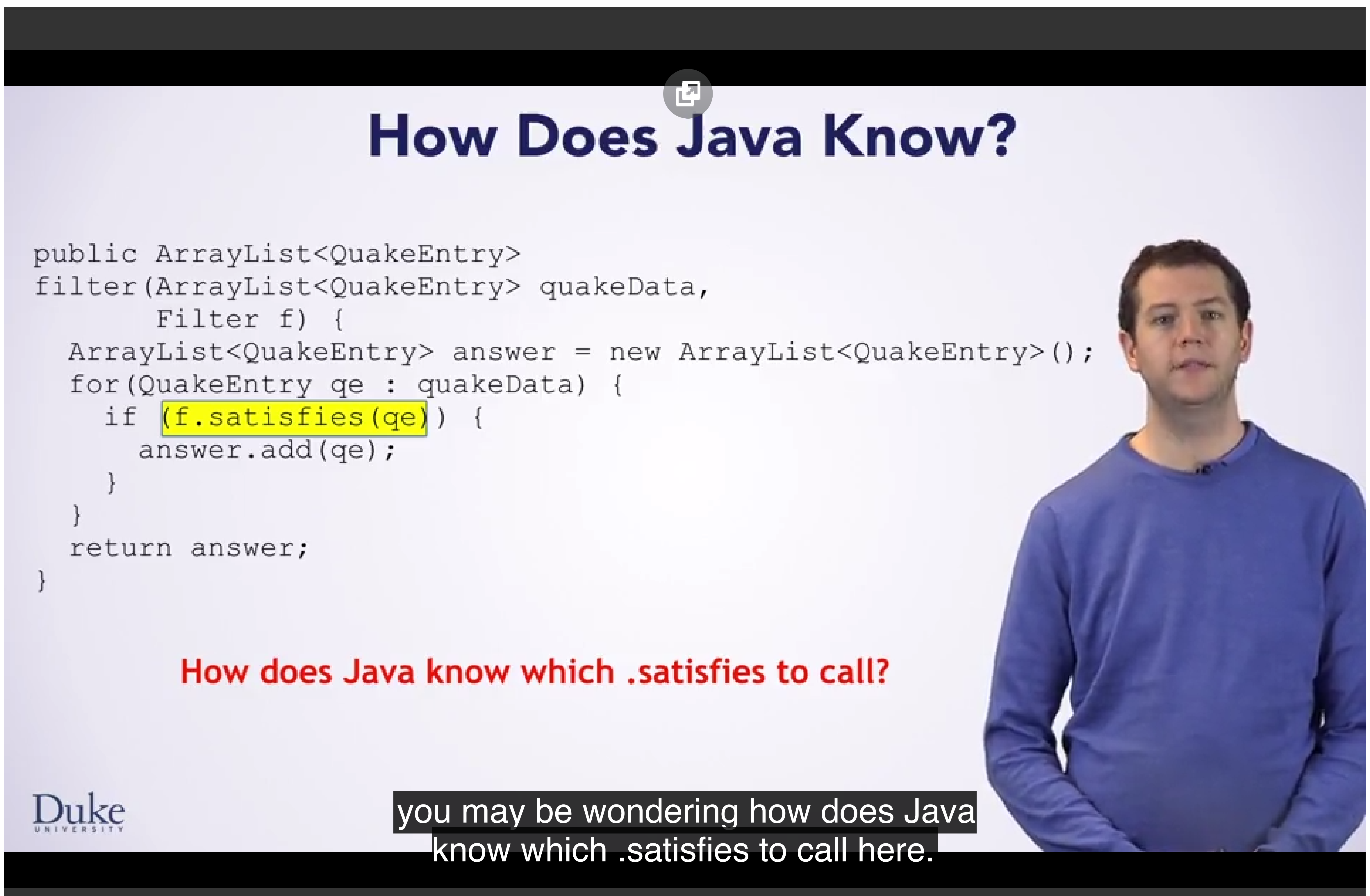
Introduction

Searching Earthquake Data

Filtering Data

Introduction	1 min
Interfaces to Avoid Duplication	6 min
Interfaces in More Depth	3 min
MatchAll	4 min
Summary	1 min
Programming Exercise: Filtering Data	10 min
Practice Quiz: Filtering Data	5 questions

Review



How Does Java Know?

```
public ArrayList<QuakeEntry>
filter(ArrayList<QuakeEntry> quakeData,
       Filter f) {
  ArrayList<QuakeEntry> answer = new ArrayList<QuakeEntry>();
  for(QuakeEntry qe : quakeData) {
    if (f.satisfies(qe)) {
      answer.add(qe);
    }
  }
  return answer;
}
```

How does Java know which .satisfies to call?

Duke UNIVERSITY

you may be wondering how does Java know which .satisfies to call here.

Interfaces in More Depth

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

Interactive Transcript

Search Transcript English

0:03
Now that you have seen interfaces in action, you may be wondering how does Java know which .satisfies to call here. The answer is that when you create an object with new, Java always notes the type of the object that was created inside the object. Then whenever you call a method on an object, Java looks into where it noted the actual type. And uses that to figure out what method to call. This process of using the actual type to figure out what method to call is known as dynamic dispatch. Let's see an example. Here you see some code that declares two variables of type filter, f1 and f2. These are each initialized with objects of two different types that implement filter MinMagFilterand DepthFilter. The dot dot dot indicates some other code that we don't care about. But what we do care about are the calls to f1.satisfies and f2.satisfies, and how Java knows which method to call each place.

1:04
When Java executes the first line of code, it makes f1 refer to a newly created MinMagFilter object with the value 4.0 stored inside of it. This object also contains some data saying that it is really a MinMagFilter object. No matter what type of variable is used to refer to it, Java always knows it's actual type is MinMagFilter. Likewise, when Java executes the second line of this code, it makes f2 refer to a newly created DepthFilter object. This object not only has some instance variables which store the minimum and maximum depths specified but also remember that it is actually a depth filter object.

1:47
Now we have reached the call to f1.satisfies. When Java goes to execute this call it looks at the actual object referenced by f!.

1:57
Here you can see as Java would that f1 is actually a MinMagFilter object. So this call, refers to the .satisfies method, inside of the MinMagFilter class. Java would then call this method, just as you're used to. This method would then execute exactly as you would expect it.

2:18
It would check the passed in quakes magnitude, against its MagMinFields value. The method would then return the appropriate value true or false back to the caller where execution continues.

2:34
When we reach the f2.satisfies call, a similar process happens. This time, when Java looks at the actual type of the f2 object, it finds that it is a DepthFilter object. So it calls the .satisfies method inside of the DepthFilter class. This method checks if the depth is between the min and max depths stored in the object's field and returns true or false to it's caller and then execution would continue as normal.

3:03
So now you have learned that Java remembers the actual type of each object you make at the time you do new. And this type is used to figure out which method to call which is known as 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?