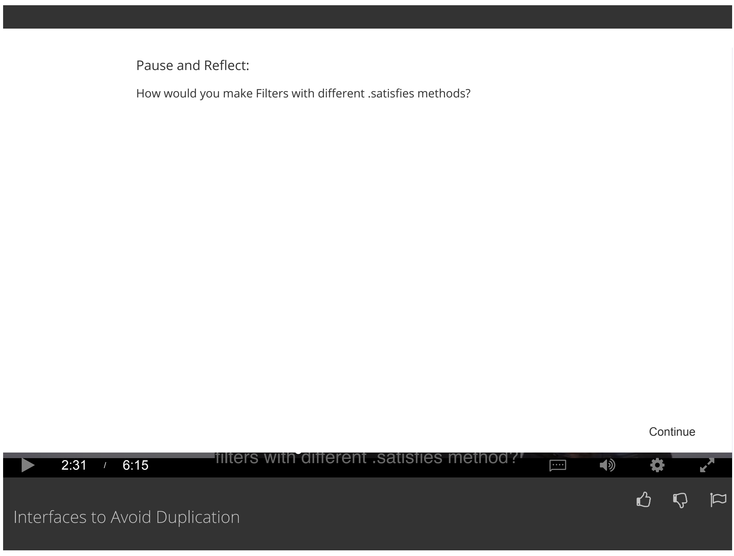

Interfaces to Avoid Duplication

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

Interactive Transcript

Search Transcript

English

0:03
Hi. To start, let's take a look at two filter methods you might write, and see the similarities between them. Here is a method that filters earthquakes by their magnitude, placing only those with at least a certain magnitude in the answer. There's nothing special about the fact that we wrote public ArrayList<QuakeEntry> on a separate line from the method name. It just makes it easier to fit on the screen. The line break is not important to Java, or to most other programming language.

0:35
This method proceeds in a straightforward way. It simply makes an empty array list for the answer, then has a for loop to examine each earthquake in the input parameter array list. And for each quake that matches its criteria, the magnitude being at least magMin, that quake is added to the answer array list. You might have written this exact code if you applied the seven steps to this problem.

1:00
Here is another method, which filters earthquake by their distances from a particular location. Notice how similar it is to the previous code. In fact, there are only three differences. The method name has been changed to reflect the task that this method does. The parameters are different. This one takes a maximum distance, and a location, whereas, the magnitude filter only took a minimum magnitude.

1:29
And the condition to decide whether to add the current earthquake being iterated over to the answer ArrayList is different. Whenever you have this much similarity in code, you might wanna find a way to avoid duplicating the code. In particular, whenever you find yourself wanting to copy and paste code, and then to make changes, you should generally think about whether there's a different approach.

1:53
Here you can see a different and better approach. This method is a generic filtering method, which takes a parameter Filter f that specifies how to determine if a particular earthquake should be included in the output list or not.

2:09
The code then calls f.satisifes on each of the earthquakes being iterated over and uses the return method of .satisfies to make a decision as to whether the earthquake is included. This approach looks great, but you are probably wondering how do you make filters with different .satisfies method? The answer is that filter is an interface, not a class. You can see its declaration here. Notice how the declaration says interface where you would normally write class.

2:45
Writing interface here instead of writing class tells Java that filter is not a class but is an interface. Instead of defining behavior of methods, an interface is just a type. A type that promises that certain methods will exist in all classes which implement the interface. Here, the filter interface promises one such method. Public boolean satisfies(QuakeEntry qe). Once you have declared an interface, you can write classes that implement it. When you write a class that implements an interface, you must define all of the methods promised in the interface specification. Objects of the class can then be treated as the interface type.

3:33
If you write a class that implements filter, objects that you make from that class can be assigned to variables of type filter or passed as parameters to methods that expect a filter. Let's look at an example.

3:46
Here's the MinMagFilter, a class that checks if an earthquake has a minimal magnitude. You can see the declaration of this class says implements Filter. When you write this Java will check and make sure the class has all of the methods promised by the filter interface. If one's missing, the class won't compile.

4:07
This allows you to treat MinMagFilter objects as filters in the code you write. Here you can see where this class has the promised .satisfies method. The method simply checks that the passed in earthquake's magnitude is greater than or equal to magMin, the minimum magnitude which is stored in an instance variable of the object. Notice how similar the body of this method is to the condition you saw earlier when we looked at the code that filtered by magnitude.

4:37
This class can also have other members beyond the specified methods. Here, there's an instance variable to hold the minimum magnitude, and a constructor which initializes that instance variable from its parameters. These aren't promised in the filter's specification. If needed, you could write other methods in the class too. You just must write satisfies.

5:01
At the top you can see the code we wrote earlier for the generic filtering method. At the bottom you can see how you can assign a MinMagFilter object to a filter variable, and pass it to this method. The right hand side of this first assignment statement makes a new MinMagFilter, passing in 4.0. It creates an object whose .satisfies method will test if an earthquake's magnitude is at least 4. The object has type MinMagFilter. The left hand side of this assignment statement is a variable of type filter. Even though the types are not the same, this assignment is legal because MinMagFilter implements the filter interface. The next line passes f, which is the object to filter by, which is a magnitude of at least 4, to the generic filtering method you saw earlier. If you had another class that implemented filter such as distance filter, you could assign an instance of distance filter to the variable f as well. And then, you would find out if your world is being rocked by a quake close to you.

6:09
I'm feeling my earth move right now, under my feet.

6:13
Happy programming.

Downloads

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

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