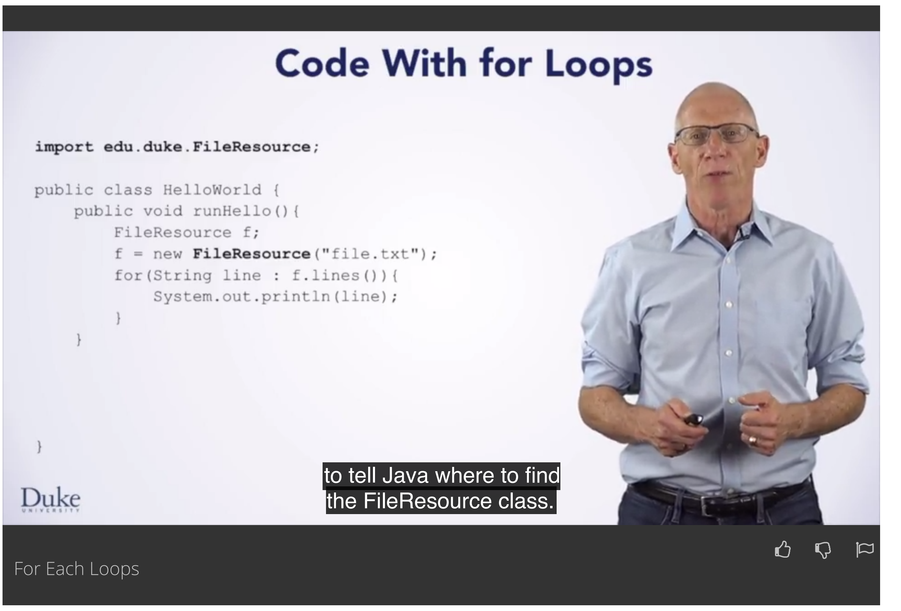


Have a question? Discuss this lecture in the week forums.

## Interactive Transcript

Search Transcript

English

0:03
Hi, now we're going to learn how a for-each loop works. This piece of code looks similar to the hello around the world example that you started with earlier. To make this work, we need to add an import statement to the top to tell Java where to find the FileResource class. FileResource is in a package that we provide to you to let you manipulate data in files before you learn the more advanced techniques and concepts that would let you do this in Java directly without the classes we've created. Accordingly, this is found in the edu.duke package. If you want to run this code in BlueJ, you could do so by creating a HelloWorld object. And then invoking its runHello method. We could also add a main method, which does the same thing if we want to be able to run the code directly, but outside of BlueJ, in another environment. Let's start executing the code by hand. We start in main with a frame that contains args. We're not going to use this args argument, so we're not going to worry about its value. The first line declares hw, and initializes it using new. You learned about new in a previous lesson, but there's a slight difference here. The HelloWorld class does not have a constructor, so what do you do? In this case, Java provides a default constructor, one with no arguments, that does nothing. So we execute this line as you're used to, but we don't call a constructor. We make an object. Class HelloWorld has no fields, so the object itself doesn't have any state or anything else we can see in it. Then we finish the assignment statement. Next, we call runHello, passing in hw as the implicit this argument. Inside of runHello, our first statement declares the variable, f.

1:53
The next statement initializes f to a new FileResource object. This raises two questions. First, what does a FileResource object look like? Meaning, what fields are there in the FileResource object? It turns out we don't actually need to know this precisely. The details of how a FileResource object works can remain hidden from us as long as we know what it does. This is an instance of the important programming principle known as abstraction. Second, what does the constructor for a FileResource object do? To find the answer to this question, we would consult the documentation for FileResource. Consulting documentation for libraries, sometimes called APIs, that you use in programming is an important task. Since FileResource comes from the Duke Learn to Program libraries, we would look at the documentation website. And we would read about the constructors for a FileResource class. There are three constructors, and the one we want is the one in the middle. Since we're passing a string literal in, the string, "file.txt". This documentation says the constructor will find that file with that name on our computer. Given this information, we're just going to represent the FileResource object as knowing what file we asked for, the file who's name is "file.txt". We don't know a precisely what fields are stored in this object, but that's okay. We'l represented as precisely as we can because we know what the object does. We finish the assignment statement, so now f references this FileResource object we just created by calling the constructor. The next line is new to us, It's a for loop. In particular, this for loop is often called a for-each loop because it does something for each value in an iterable. What's an iterable? The short and simple answer is that's an object which gives you a sequence of values. Let's dig a little more closely into the details of this loop. The first part declares a variable. The type is String and the name of the variable is line. This declaration will behave like any other, will create a box for line. Next is a :, this is the syntax of a for-each loop. We don't use an equal or a assignment operator since we aren't assigning one value to line. Instead, we're going to each value in the iterable sequence. Many people read the colon as in when they read the code out loud or to themselves, for String line in f.lines. Next, we have an expression which evaluates to the iterable whose values we want variable line to refer to one after the other. In this case, that expression is a call to f.lines. To understand what this does, we'll need to understand what f.lines returns. So we'll, again, need to consult our documentation. From this API documentation, we see that f.lines gives us an iterable whose values are each line in the file in the order that they appear. This means that we'll need to look at the file whose name is file.txt to see what its contents are in order to understand and simulate the code behavior. On my computer, I've made file.txt, and put these two lines in it. Hello on one line and World on the next line. It isn't a very exciting file, but we want this example to be relatively short.

5:29
Returning to our code, we now create this sequence of strings, Hello and World. We'll create the box for line. And we'll make it refer to the first element of this iterable sequence. Now we go into the body of the for loop and begin executing statements there. The next line is a print statement. So we print out the value of line, which is Hello.

5:53
Now our execution arrow is just before the closed curly brace of the for-each loop. We've reached its end. When you reach the end of a for loop, you move the execution arrow back to the start of the loop to do the next iteration, the next time through the loop, with the new value for the loop variable. In this case, our loop variable is the String line. So we need to update it to have the next value in the iterable. We update line's arrow to point at the next value in this sequence. And again go into the body of the loop.

6:28
We encounter our print statement, but this time, line has a different value, so we print World. Now we've once again reached the end of the body of the for loop. So we go back to the beginning, the start of the loop. We need to update line to refer to the next value in the iterable sequence. However, if we try to do that, we'll find that there are no more elements in the iterable sequence. We already used the last one, so instead, we exit the loop. We move the execution arrow past the body of this for each loop and begin executing code there. When we do this, the loop variable, in this case the string line, goes out of scope. That means it no longer exists, so we remove its box. Now we're ready to return from the method runHello, destroying its frame. Since the call to that method, runHello, was the last line in main, we also return from main, exiting the program.

## Downloads

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

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