CS 240: Using Generic Interfaces Example Revisited Transcript

[00:00:00] **INSTRUCTOR:** Now that you know how lambda expressions work, I want to go back and show you again the example that I used when I introduced generics and show you how you can use lambdas to improve that example.

[00:00:10] So, this is the interface example that I use with generics.

[00:00:15] Remember we created an interface; we've created a function interface; we created some subclasses on the slide.

Start slide description. Following content is shown:

```
public interface Function<T, R> {

R apply(T param);

}

public class Capitalizer implements Function<String, String> {

    @Override

    public String apply(String param) {

    return param == null ? null : param.toUpperCase();

    }

}

public class StringManipulator {

    public String manipulateString(

        String str,
```

Function<String, String> manipulationFunction) {

return manipulationFunction.apply(str);

}

}

End slide description.

[00:00:21] I only have one, but in my code, I showed you that I created a capitalizer and a space remover.

[00:00:26] So, I created not subclasses, but implementing classes of this function.

[00:00:29] And then, I had my string manipulator class that needs to receive the function.

[00:00:34] Now that you know about lambdas, you know that most of this is redundant.

[00:00:37] So, I don't need the function interface because that's one of the 43 that already exists.

[00:00:42] So, there's no reason to create it.

[00:00:44] There's already a function interface.

[00:00:46] And I don't need to create a separate class that implements the interface because I can do that with a lambda.

[00:00:52] So, this is what that example becomes.

[00:00:55] So here, the example on the right is still the same.

Start slide description.

manipulateString("my string",

```
  str -> str == null ? null : str.toUpperCase())

manipulateString("my string with spaces that will be removed",

  str -> str == null ? null : str.replaceAll(" ", ""))

public class StringManipulator {

    public String manipulateString(

      String str,

      Function<String, String> manipulationFunction) {

      return manipulationFunction.apply(str);

    }

}
```

End slide description.

[00:00:59] It still receives a function, but now it's the function that has already declared.

[00:01:03] It's already part of Java.

[00:01:06] And it will still call the apply method of whatever function it gets.

[00:01:11] But now, I can just call that method and pass two parameters without creating an interface, without creating any subclasses of some interface.

[00:01:19] I just create the lambdas.

[00:01:21] So, here, I have my pass string.

[00:01:26] And my second parameter is this lambda expression.

[00:01:31] So, it’s the parameter list is on the left.

[00:01:34] I’m only passing one parameter.

[00:01:36] So, I only need...

[00:01:36] I don’t need parentheses.

[00:01:38] So, it’s passing STR as the parameter, and the code just checks to see if the string is null, and if not, it uppercases it.

[00:01:48] And then for the string, the space remover, same thing, I don’t need a space remover subclass.

[00:01:53] I just call this method, and I pass the lambda as the second parameter.

[00:02:00] And then remember in the background, the JVM is doing all the work that we did in the generic example.

[00:02:05] It’s creating a subclass of the interface, creating an instance of it, and passing it in.

[00:02:09] But you don’t have to worry about any of that because all that is done for you.