The Lambda Expression

The functional interface is the framework that lets a lambda expression be used.

Lambda expressions are also called lambdas for short.

Many of Java's classes, use functional interfaces in their method signatures, which allows us to pass lambdas as arguments to them.

You'll soon discover that lambdas will reduce the amount of code you write.

In an upcoming video, I'm going to cover many of Java's built-in functional interfaces.

The Consumer Interface

The Consumer interface is in the java.util.function package.

It has one abstract method, that takes a single argument, and doesn't return anything.

```
void accept(T t)
```

This doesn't seem like a very interesting interface at first, but let's see what this means in practice, as far as the lambda expressions we can use with it.

lambda expression variations, for a single parameter

This slide shows the different ways to declare a single parameter in a lambda expression.

Lambda Expression	Description
`element -> System.out.println(element);`	A single parameter without a type can omit the parentheses.
`(element) -> System.out.println(element);`	Parentheses are optional.
`(String element) -> System.out.println(element);`	Parentheses required if a reference type is specified.
`(var element) -> System.out.println(element);`	A reference type can be var.

lambda expression variations, the lambda body

The lambda body can be a single expression, or can contain a lambda body in opening and closing curly brackets.

Lambda Expression	Description
`(element) -> System.out.println(element);`	An expression can be a single statement. Like a switch expression, that does not require yield for a single statement result, the use of return is not needed and would result in a compiler error.

`(var element) -> {`
`  char first = element.charAt(0);`
`  System.out.println(element + " means " + first);`
`};`
	An expression can be a code block.
	
	Like a switch expression, that requires yield, a lambda that returns a value, would require a final return statement.
	
	All statements in the block must end with semi-colons.

Functional Programming

Lambdas are Java's first step into functional programming.

This is a programming paradigm that focuses on computing and returning results.

And just for more information about functional programming, there's a good wikipedia article here.

https://en.wikipedia.org/wiki/Functional_programming

Check that out to find out a bit more about functional programming.

Streams

Another feature of Java, makes extensive use of lambda expressions, and that's streams.

Streams are exciting, because they create a pipeline of work that can be formed into a chain.

Many of the stream operations take functional interfaces as parameters, meaning we can code them with lambda expressions.