Convenience Methods on functional interfaces in java.util.function package

Category of Interface	**Convenience method example**	**Notes**
Function	function1.**andThen**(function2)	Not implemented on IntFunction, DoubleFunction, LongFunction
Function	function2.**compose**(function1)	Only implemented on Function & UnaryOperator
Consumer	consumer1.**andThen**(consumer2)	
Predicate	predicate1.**and**(predicate2)	
Predicate	predicate1.**or**(predicate2)	
Predicate	predicate1.**negate**()	

For andThen, and compose, on the function category of interfaces, any Interim Functions are not required to have the same type arguments.

Instead, one function's output becomes the next function's input, and the next function's output is not constrained to any specific type, except the last function executed.

Convenience Methods on functional interfaces in java.util.function package

Category of Interface	**Convenience method example**	**Notes**
Function	function1.**andThen**(function2)	Not implemented on IntFunction, DoubleFunction, LongFunction
Function	function2.**compose**(function1)	Only implemented on Function & UnaryOperator
Consumer	consumer1.**andThen**(consumer2)	
Predicate	predicate1.**and**(predicate2)	
Predicate	predicate1.**or**(predicate2)	
Predicate	predicate1.**negate**()	

The Consumer's andThen method is different, because it never returns a result, so you use this when you're chaining methods independent of one another.

The Predicate methods always return a boolean, which will combine the output of the two expressions, to obtain a final boolean result.

Comparator's additional helper methods

Now, I want to cover the additional convenience methods, since as you can see from this table, many take a functional interface instance, as an argument.

Type of Method	**Method Signature**
static	Comparator **comparing**(Function keyExtractor)
static	Comparator naturalOrder()
static	Comparator reverseOrder()
default	Comparator **thenComparing**(Comparator other)
default	Comparator **thenComparing**(Function keyExtractor)
default	Comparator **reversed**()

There is a comparing static method, and an overloaded default method named, thenComparing, and finally a default reversed method.