## **COMBINING OBJECT-ORIENTED & FUNCTIONAL PROGRAMMING**

-------------------------------------------------------------------------------------------------------------------------------

**Notes:**

-> Common super classes (mostly interfaces) provide a reusable foundation for extensibility.

-> Subclasses extend the common classes to create various custom implementation strategies.

-> Java Functional Programming (FP) features are most effective when used to simplify computations within the context of and Object-Oriented (OO) software architecture.

-> Especially concurrent & parallel computations.

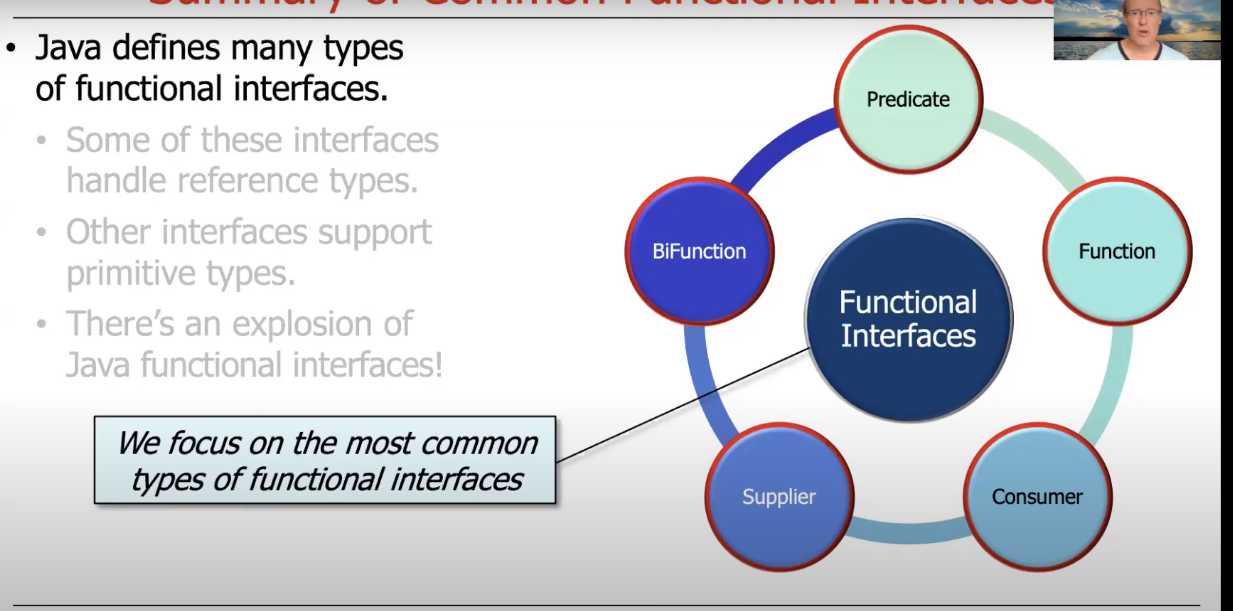
## **FUNCTIONAL INTERFACES**

-------------------------------------------------------------------------------------------------------------------------------

**Notes:**

-> A functional Interface is an interface that contains only one abstract method.

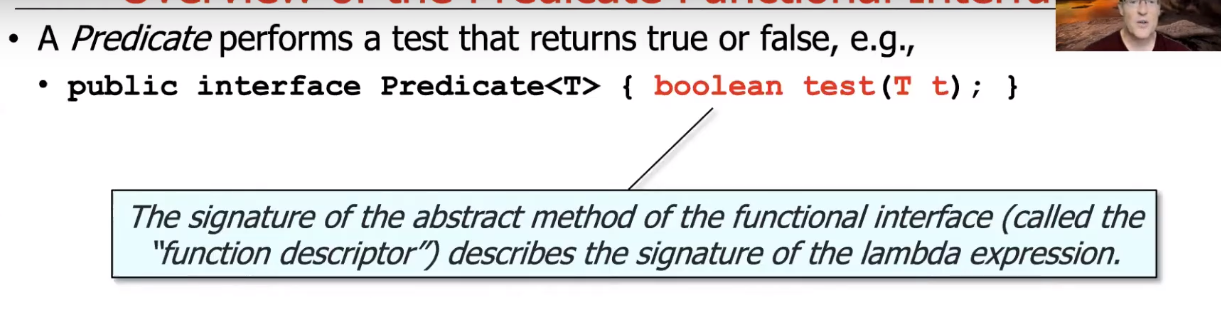
-> Examples: Runnable

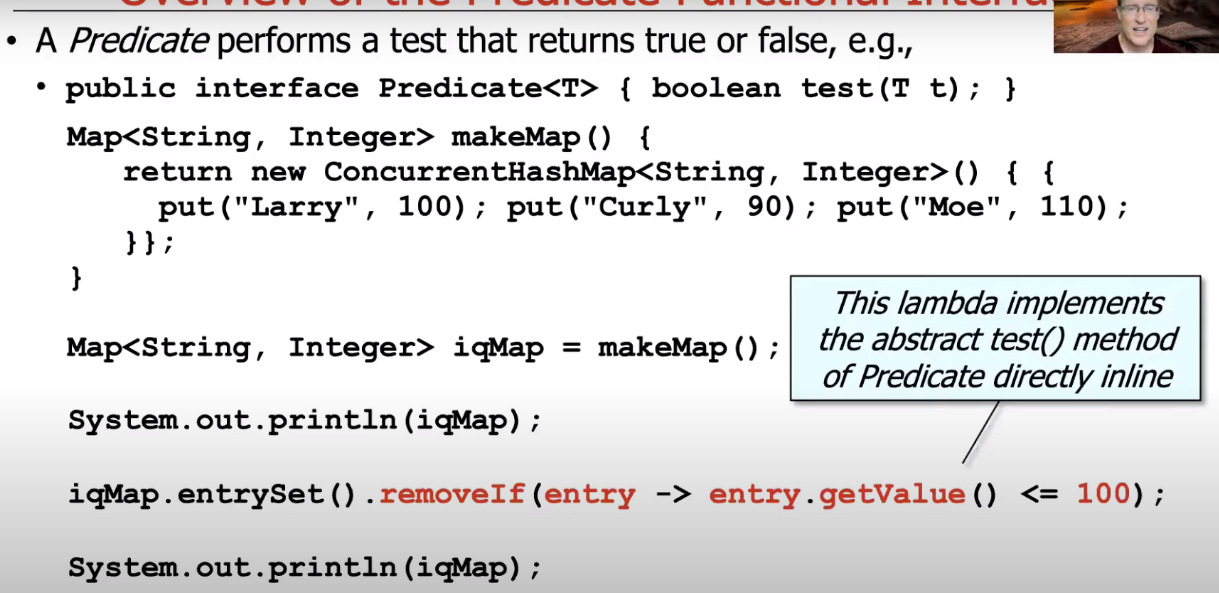


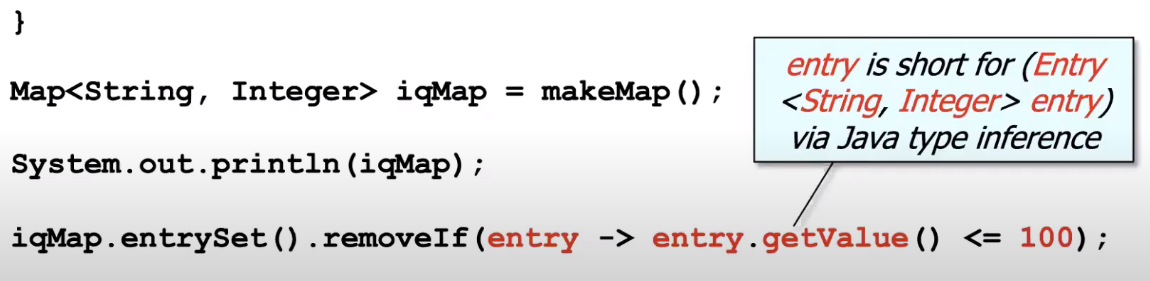
## **PREDICATE FUNCTION**

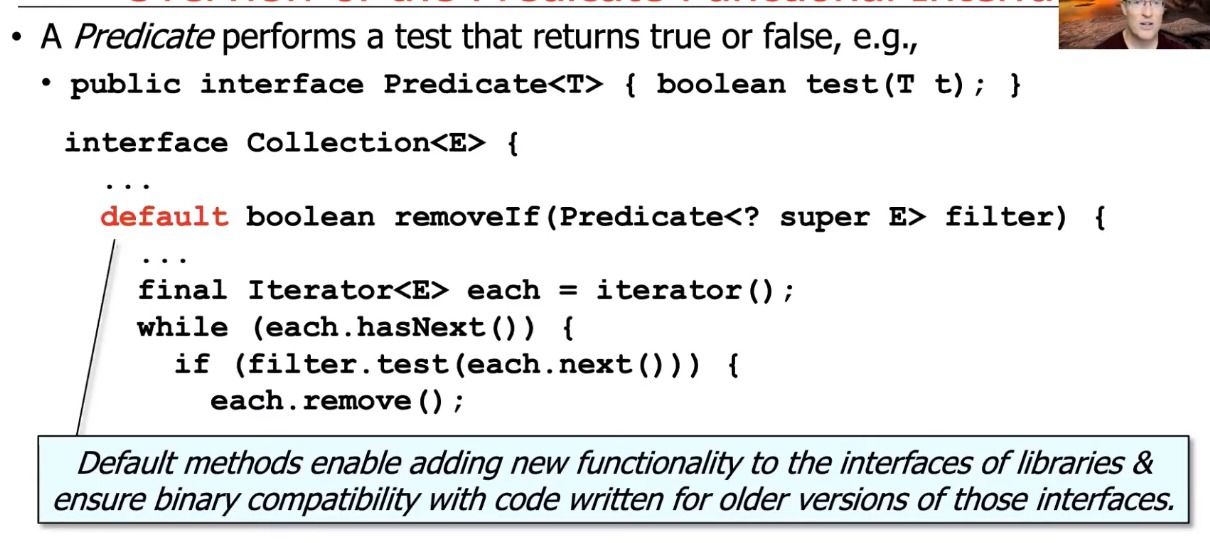
-------------------------------------------------------------------------------------------------------------------------------

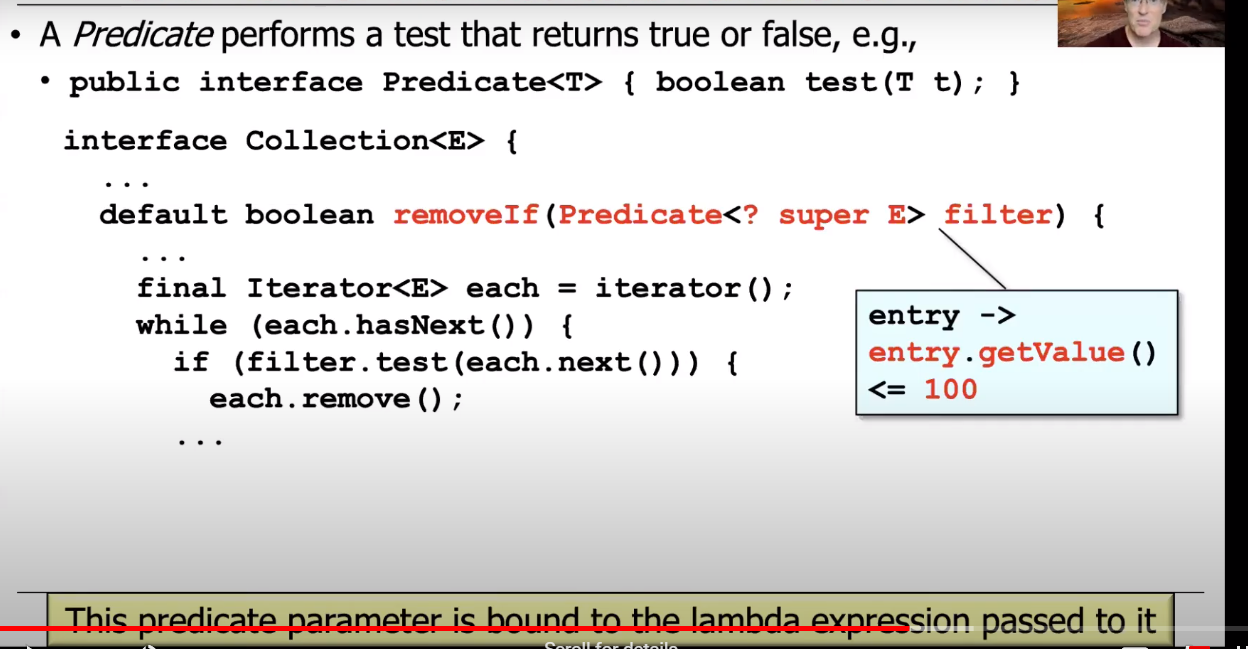
**Notes:**

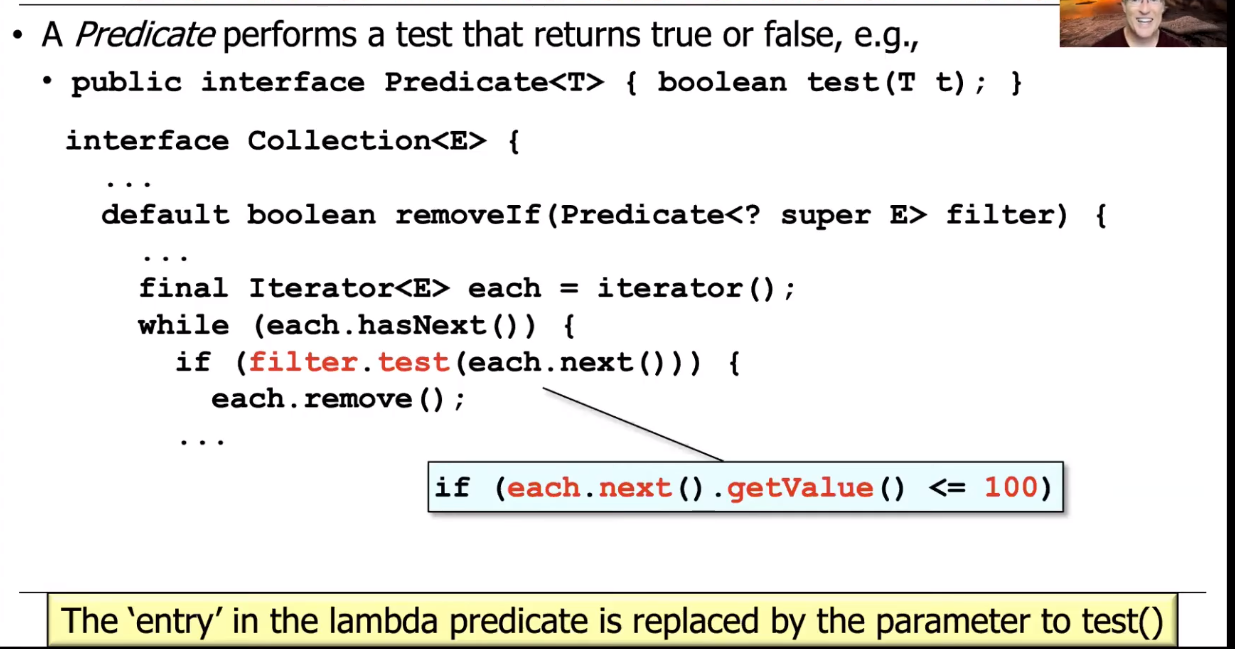


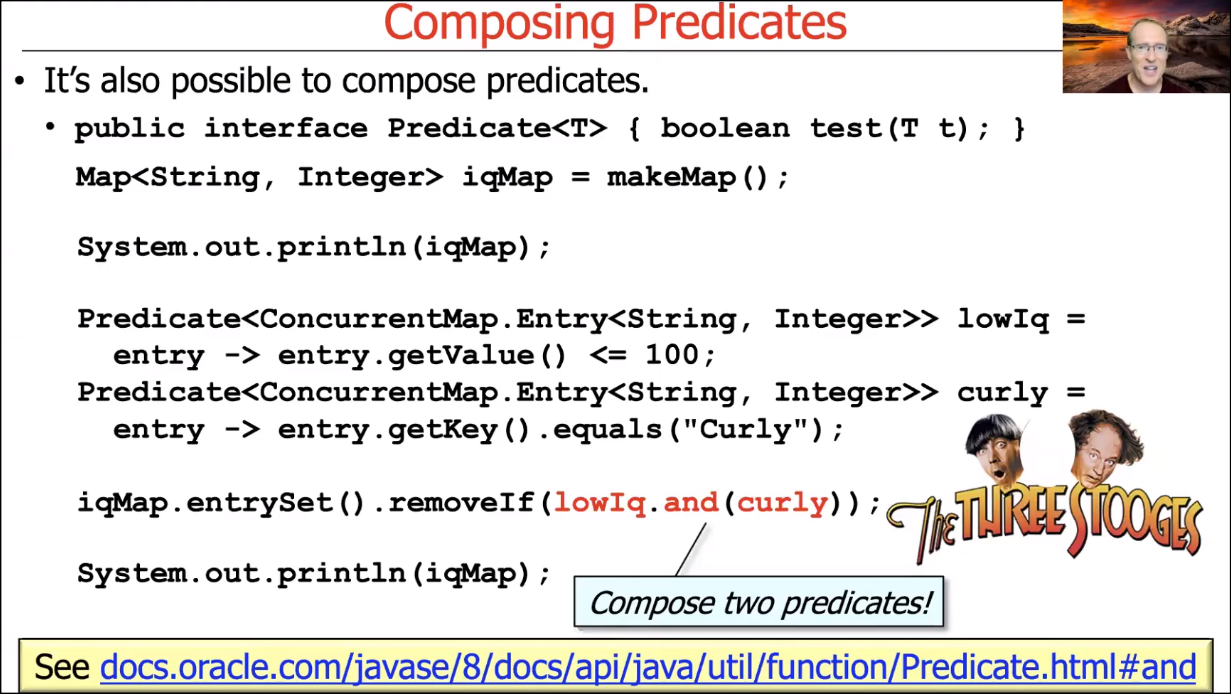






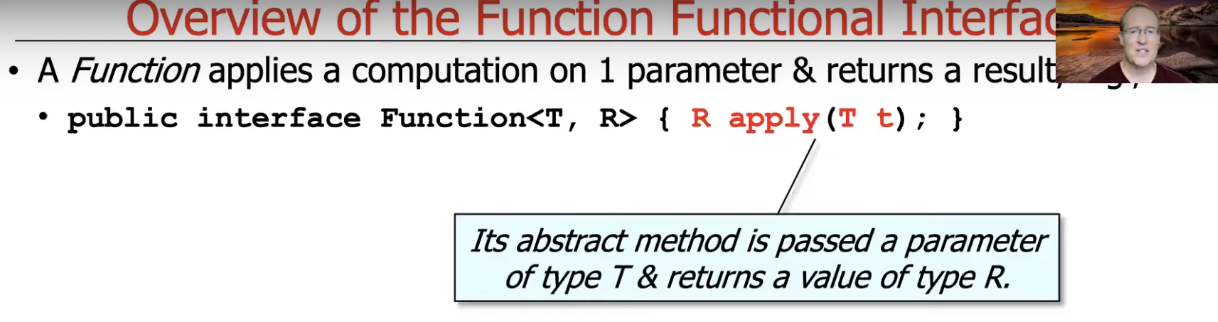






FUNCTION INTERFACE

-------------------------------------------------------------------------------------------------------------------------------



## **LAMDAS**

-------------------------------------------------------------------------------------------------------------------------------

Notes:

-> Java's Lambda expression supports concise "behavior parameterization".

-> Lambda expressions are most effective when they are "stateless".

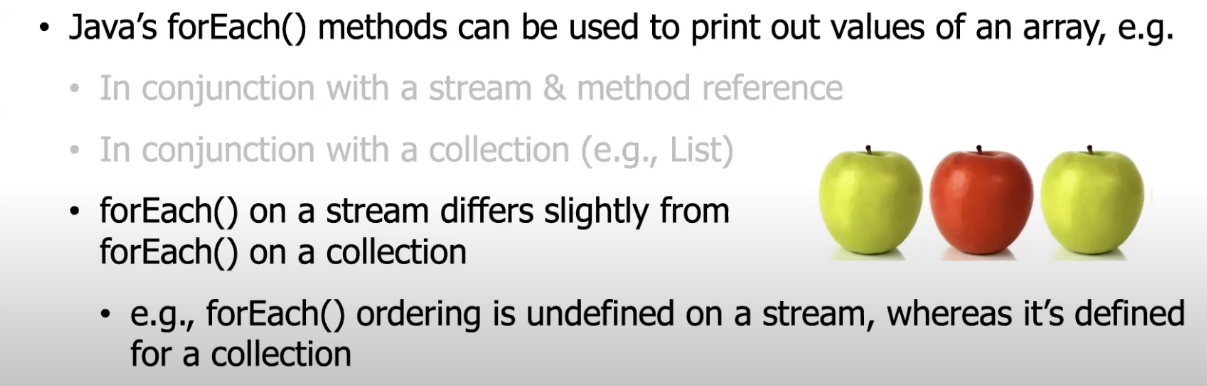
-> Stateless Lambda expressions are particularly useful when applied to Java Parallel streams.

**For-Each method:**

-> For-Each method differs for Streams and Collections.

-> For Streams, the order is not maintained.

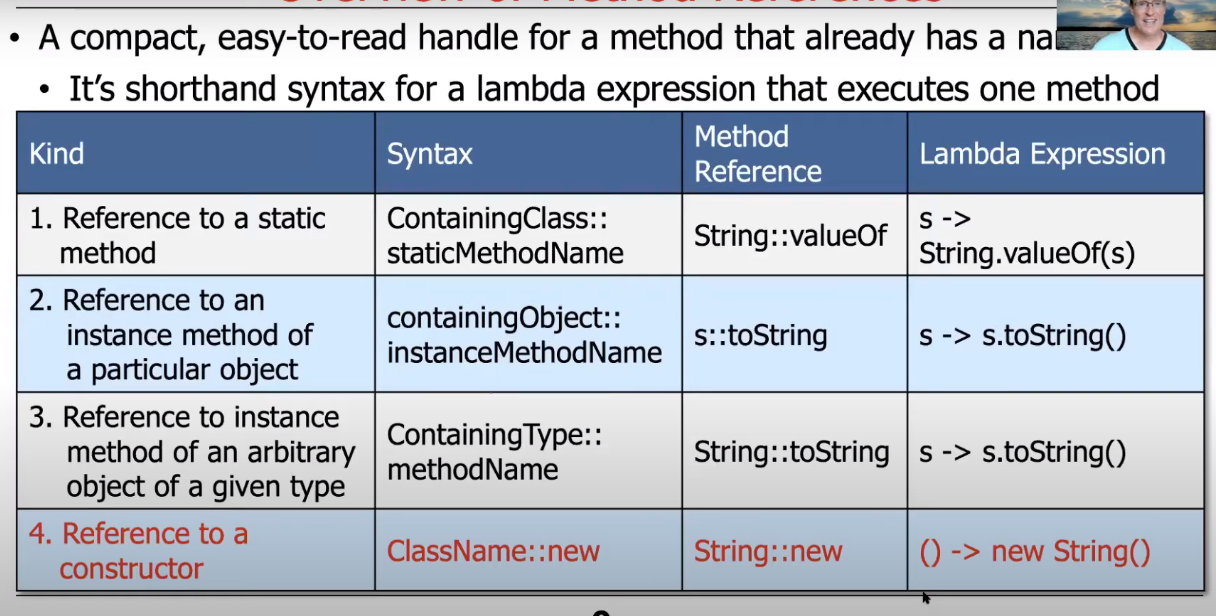
-> For Collections, the order is maintained.

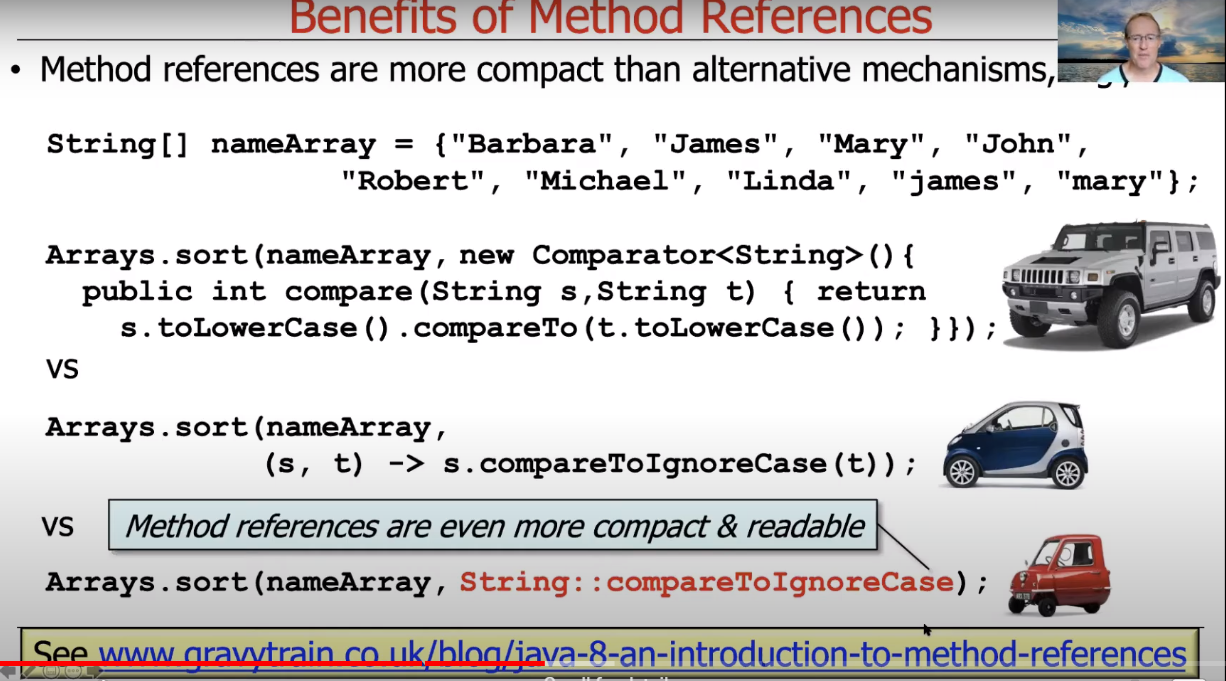


## **METHOD REFERENCES**

-------------------------------------------------------------------------------------------------------------------------------

**Notes:**





## **AGGREGATE OPERATIONS IN JAVA STREAMS:**

-------------------------------------------------------------------------------------------------------------------------------

**Notes:**

-> An aggregate operation performs a behavior on elements in a stream.

-> Some aggregate operations perform behavior on all elements in a stream. (Run-to-Completion Operations)

-> Other aggregate operations perform behavior on some elements in a stream. (Short-Circuited Operations)

-> The output of one aggregate operation can be input into the next aggregate operation in the stream.

**Intermediate Operations:**

**Terminal Operations:**

**Notes:**

-> Java Streams supports pipelining of aggregate operations via "fluent interfaces".

-> Java Streams iterate internally (& invisibly) between aggregate operations.

-> Internal iterations enhances opportunities for transparent optimization & incurs fewer accidental complexities.

->

Links to Check:

-> www.github.com/douglascraigschmidt/LiveLessons/tree/master/Java8/ex16

-> www.github.com/douglascraigschmidt/CS253