Java Parallel Streams Internals: Non-Concurrent & Concurrent Collectors (Part 2)

Douglas C. Schmidt
d.schmidt@vanderbilt.edu
www.dre.vanderbilt.edu/~schmidt

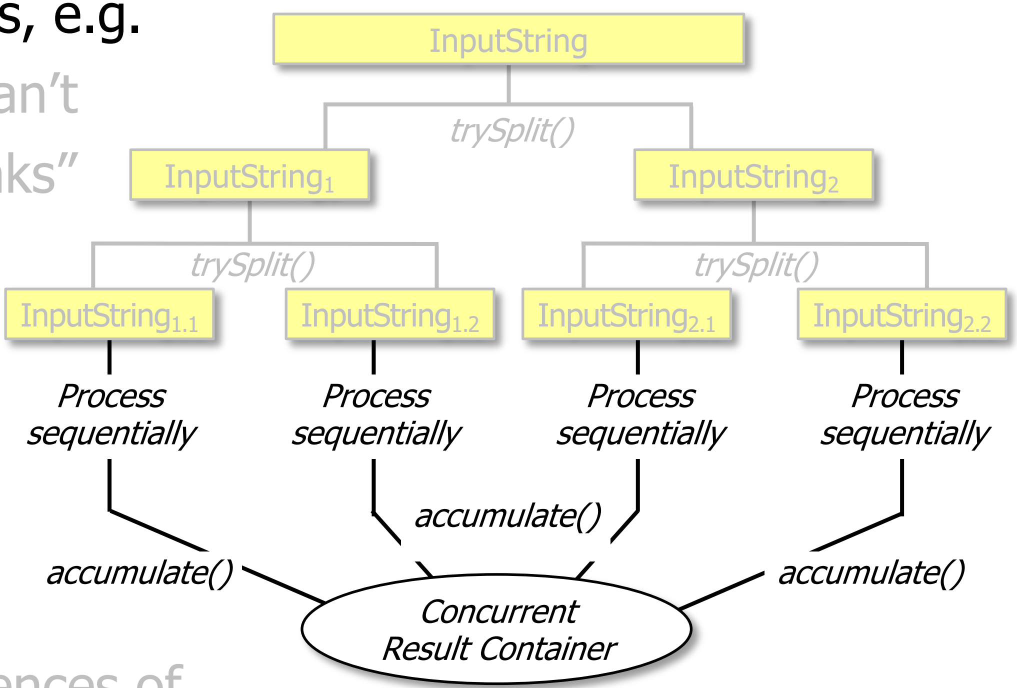
Professor of Computer Science

Institute for Software Integrated Systems

Vanderbilt University
Nashville, Tennessee, USA

Learning Objectives in this Part of the Lesson

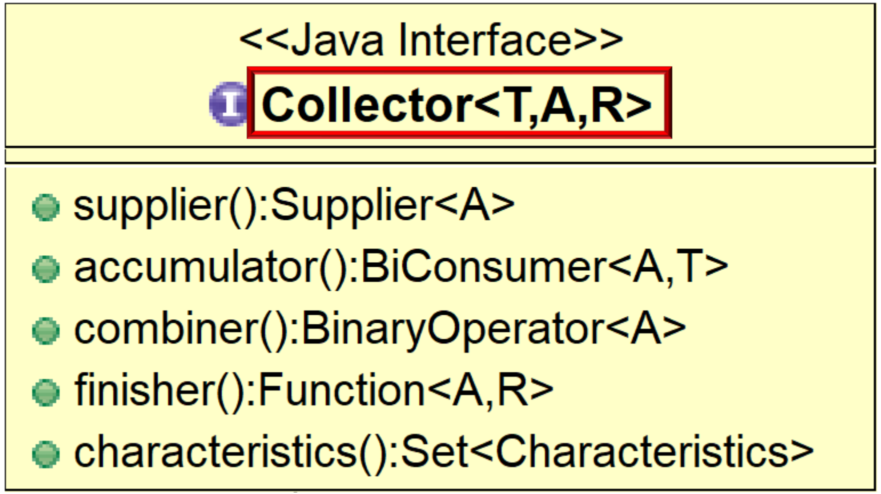
- Understand parallel stream internals, e.g.
  - Know what can change & what can't
  - Partition a data source into "chunks"
  - Process chunks in parallel via the common fork-join pool
  - Configure the Java parallel stream common fork-join pool
  - Perform a reduction to combine partial results into a single result
  - Recognize key behaviors & differences of non-concurrent & concurrent collectors
- Be aware of non-concurrent & concurrent collector APIs



Non-Concurrent & Concurrent Collector APIs

Non-Concurrent & Concurrent Collector APIs

- The Collector interface defines three generic types



<<Java Interface>>

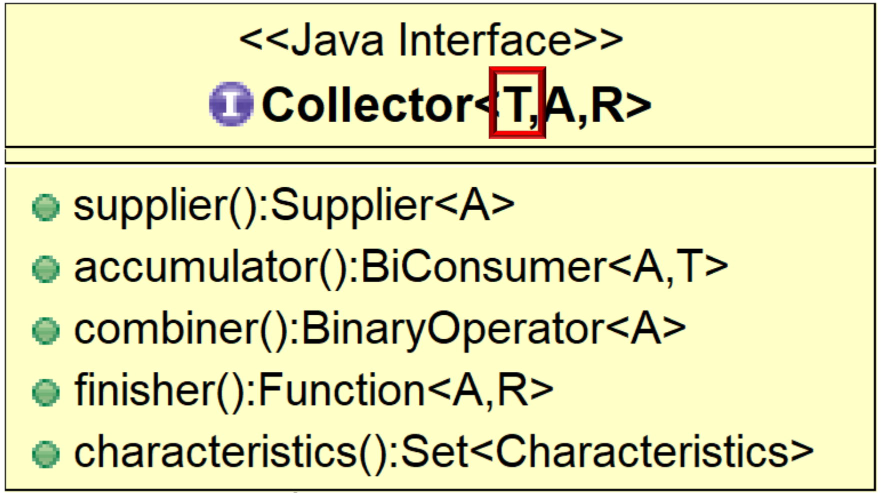
Collector<T,A,R>

- supplier():Supplier<A>
- accumulator():BiConsumer<A,T>
- combiner():BinaryOperator<A>
- finisher():Function<A,R>
- characteristics():Set<Characteristics>

See www.baeldung.com/java-8-collectors

Non-Concurrent & Concurrent Collector APIs

- The Collector interface defines three generic types
  - **T** – The type of objects available in the stream
    - e.g., Integer, String, Double, SearchResults, etc.

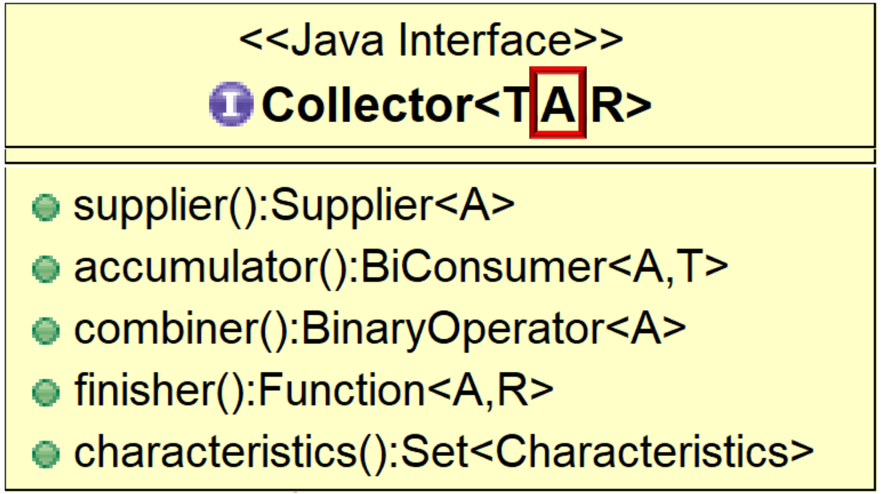


```
<<Java Interface>>
Collector<T,A,R>

supplier():Supplier<A>
accumulator():BiConsumer<A,T>
combiner():BinaryOperator<A>
finisher():Function<A,R>
characteristics():Set<Characteristics>
```

Non-Concurrent & Concurrent Collector APIs

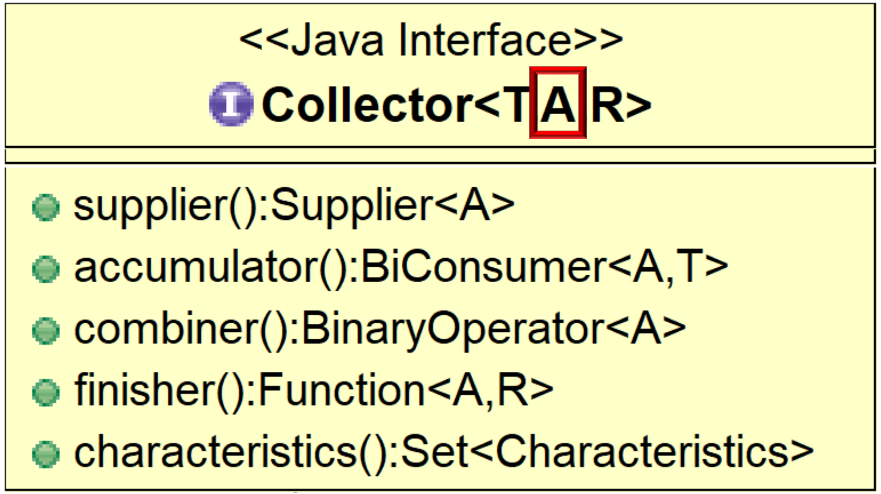
- The Collector interface defines three generic types
  - T
  - **A** – The type of a mutable result container for accumulation
    - e.g., List of T, Set of T, ConcurrentHashMap.KeySetView, etc.



<<Java Interface>>
Collector<T,A,R>
supplier():Supplier<A>
accumulator():BiConsumer<A,T>
combiner():BinaryOperator<A>
finisher():Function<A,R>
characteristics():Set<Characteristics>

Non-Concurrent & Concurrent Collector APIs

- The Collector interface defines three generic types
  - T
  - **A** – The type of a mutable result container for accumulation
    - e.g., List of T, Set of T, ConcurrentHashMap. KeySetView, etc.
      - Lists can be implemented by ArrayList, LinkedList, etc.

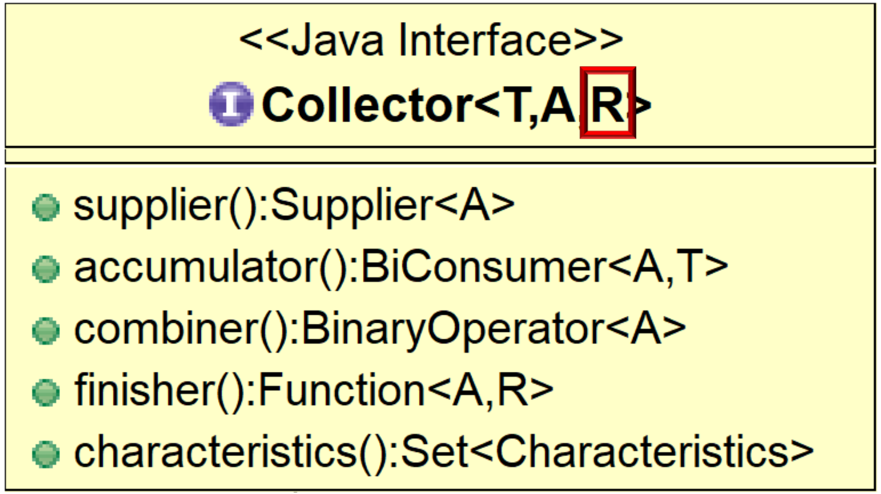


<<Java Interface>>
Collector<T,A,R>
supplier():Supplier<A>
accumulator():BiConsumer<A,T>
combiner():BinaryOperator<A>
finisher():Function<A,R>
characteristics():Set<Characteristics>

See docs.oracle.com/javase/tutorial/collections/implementations/list.html

Non-Concurrent & Concurrent Collector APIs

- The Collector interface defines three generic types
  - T
  - A
  - **R** – The type of a final result
    - e.g., List of T, CompletableFuture to List of T ConcurrentHashMap. KeyViewSet, etc.

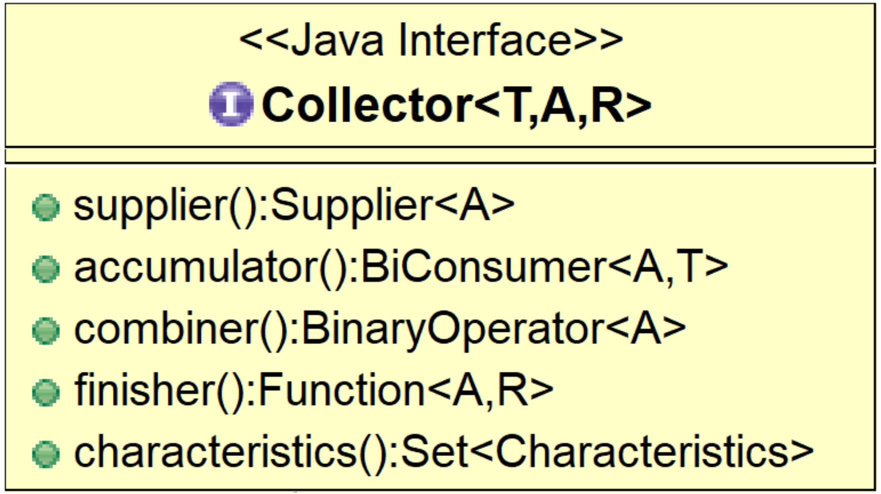


<<Java Interface>>
Collector<T,A,R>
supplier():Supplier<A>
accumulator():BiConsumer<A,T>
combiner():BinaryOperator<A>
finisher():Function<A,R>
characteristics():Set<Characteristics>

See www.baeldung.com/java-8-collectors

Non-Concurrent & Concurrent Collector APIs

- Five methods are defined in the Collector interface



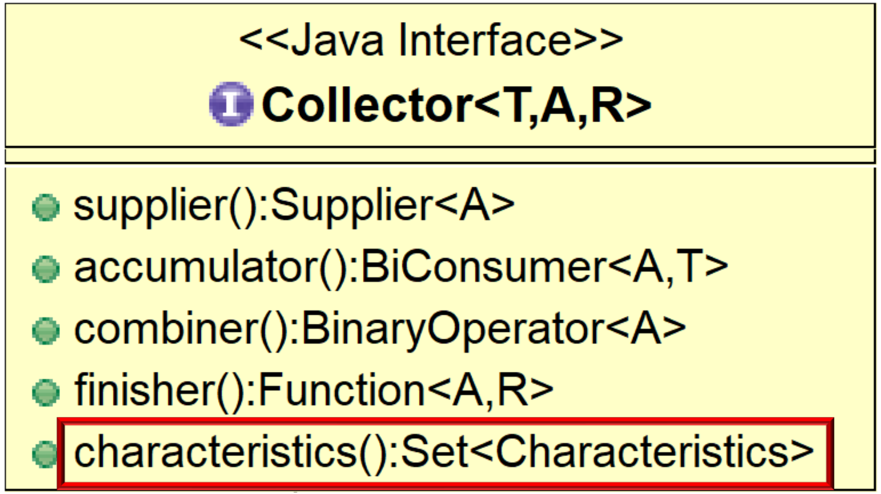
<<Java Interface>>

Collector<T,A,R>

- supplier():Supplier<A>
- accumulator():BiConsumer<A,T>
- combiner():BinaryOperator<A>
- finisher():Function<A,R>
- characteristics():Set<Characteristics>

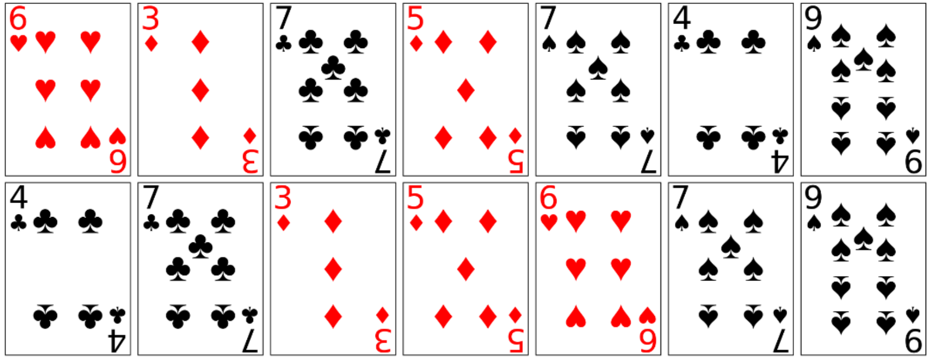
Non-Concurrent & Concurrent Collector APIs

- Five methods are defined in the Collector interface
  - **characteristics()** – provides a stream with additional information used for internal optimizations, e.g.
    - UNORDERED
      - The collector need not preserve the encounter order


<<Java Interface>>

Collector<T,A,R>

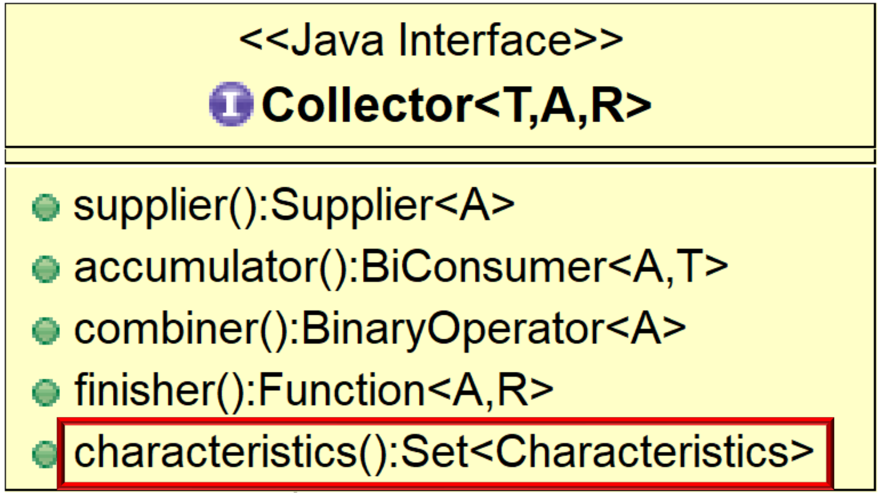
- supplier():Supplier<A>
- accumulator():BiConsumer<A,T>
- combiner():BinaryOperator<A>
- finisher():Function<A,R>
- characteristics():Set<Characteristics>



A concurrent collector *should* be UNORDERED, but a non-concurrent collector *can* be ORDERED

Non-Concurrent & Concurrent Collector APIs

- Five methods are defined in the Collector interface
  - **characteristics()** – provides a stream with additional information used for internal optimizations, e.g.
    - UNORDERED
    - IDENTITY_FINISH
      - The finisher() is the identity function so it can be a no-op
        - e.g. finisher() just returns null

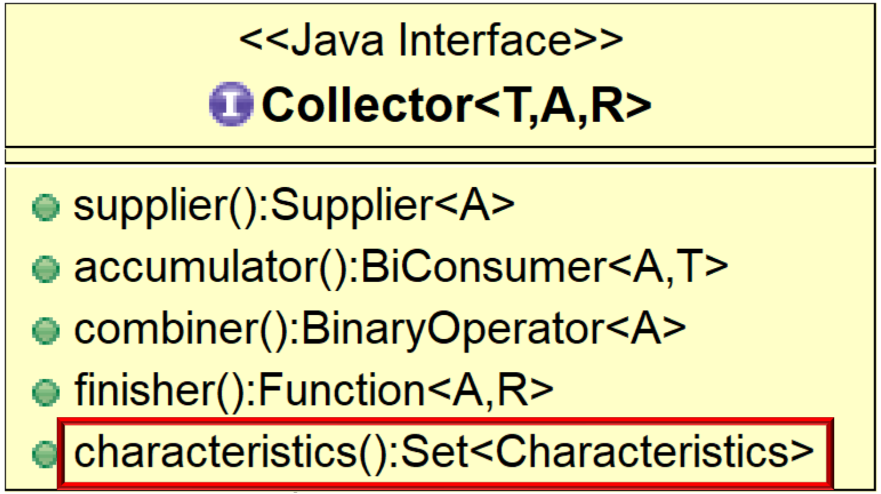


```
<<Java Interface>>
Collector<T,A,R>

supplier():Supplier<A>
accumulator():BiConsumer<A,T>
combiner():BinaryOperator<A>
finisher():Function<A,R>
characteristics():Set<Characteristics>
```

Non-Concurrent & Concurrent Collector APIs

- Five methods are defined in the Collector interface
  - **characteristics()** – provides a stream with additional information used for internal optimizations, e.g.
    - UNORDERED
    - IDENTITY_FINISH
    - CONCURRENT
      - accumulator() is called concurrently on result container


<<Java Interface>>

Collector<T,A,R>

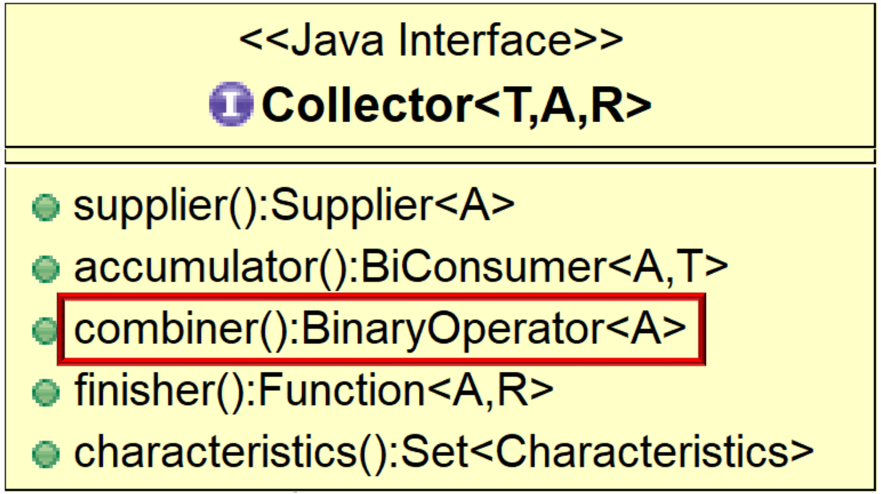
- supplier():Supplier<A>
- accumulator():BiConsumer<A,T>
- combiner():BinaryOperator<A>
- finisher():Function<A,R>
- characteristics():Set<Characteristics>

The mutable result container must be synchronized!!

A concurrent collector *should* be CONCURRENT, but a non-concurrent collector should *not* be!

Non-Concurrent & Concurrent Collector APIs

- Five methods are defined in the Collector interface
  - **characteristics()** – provides a stream with additional information used for internal optimizations, e.g.
    - UNORDERED
    - IDENTITY_FINISH
    - CONCURRENT
      - accumulator() is called concurrently on result container
      - The combiner() method is a no-op



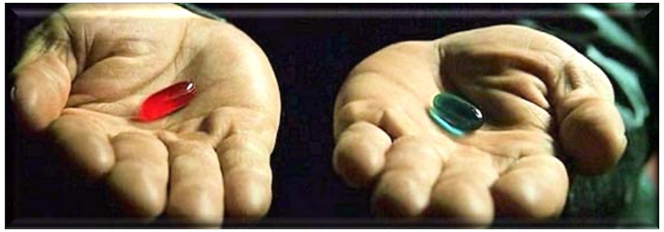
<<Java Interface>>
Collector<T,A,R>
supplier():Supplier<A>
accumulator():BiConsumer<A,T>
combiner():BinaryOperator<A>
finisher():Function<A,R>
characteristics():Set<Characteristics>

Non-Concurrent & Concurrent Collector APIs

- Five methods are defined in the Collector interface
  - **characteristics()** – provides a stream with additional information used for internal optimizations, e.g.
    - UNORDERED
    - IDENTITY_FINISH
    - CONCURRENT
      - accumulator() is called concurrently on result container
      - The combiner() method is a no-op
      - A non-concurrent collector can be used with either sequential or parallel streams

<<Java Interface>>
Collector<T,A,R>

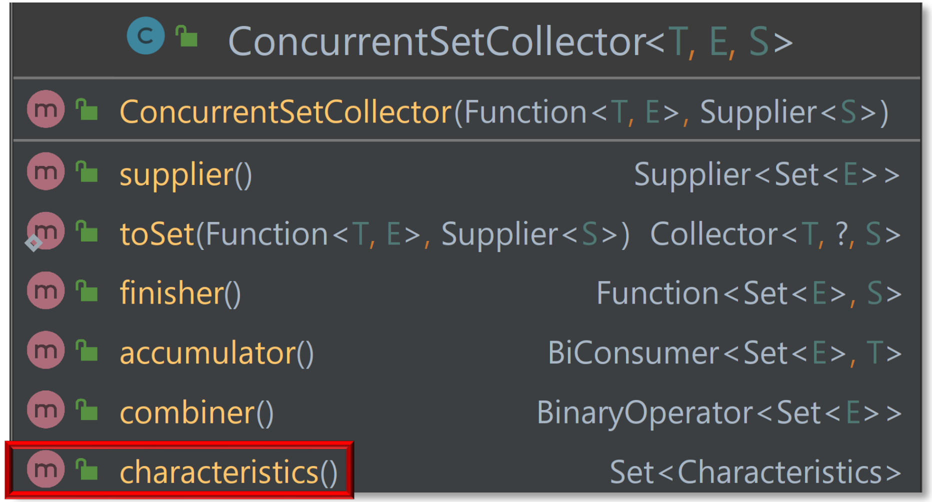
- supplier():Supplier<A>
- accumulator():BiConsumer<A,T>
- combiner():BinaryOperator<A>
- finisher():Function<A,R>
- characteristics():Set<Characteristics>



Internally, the streams framework decides how to ensure correct behavior

Non-Concurrent & Concurrent Collector APIs

- Five methods are defined in the Collector interface
  - **characteristics()** – provides a stream with additional information used for internal optimizations, e.g.



ConcurrentSetCollector<T, E, S>

ConcurrentSetCollector(Function<T, E>, Supplier<S>)	
supplier()	Supplier<Set<E>>
toSet(Function<T, E>, Supplier<S>)	Collector<T, ?, S>
finisher()	Function<Set<E>, S>
accumulator()	BiConsumer<Set<E>, T>
combiner()	BinaryOperator<Set<E>>
characteristics()	Set<Characteristics>

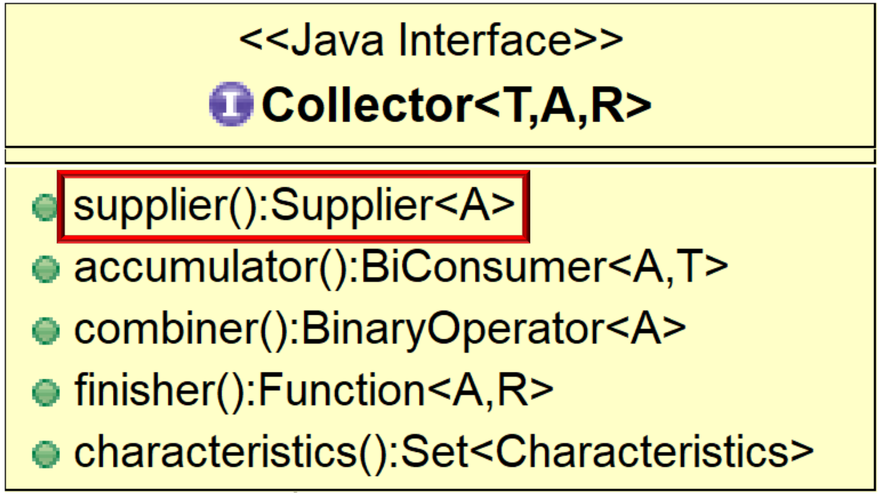
Any/all characteristics can be set using EnumSet.of()

```
Set<Characteristics> characteristics() {
  return Collections.unmodifiableSet
    (EnumSet.of(Collector.Characteristics.CONCURRENT,
                Collector.Characteristics.UNORDERED));
}
```

See docs.oracle.com/javase/8/docs/api/java/util/EnumSet.html

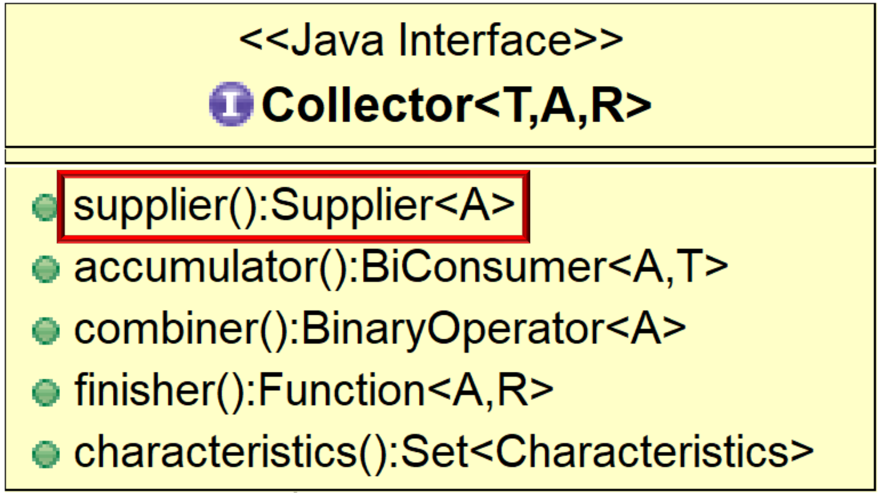
Non-Concurrent & Concurrent Collector APIs

- Five methods are defined in the Collector interface
  - characteristics()
  - **supplier()** – returns a supplier that acts as a factory to generate an empty result container


<<Java Interface>>
Collector<T,A,R>
supplier():Supplier<A>
accumulator():BiConsumer<A,T>
combiner():BinaryOperator<A>
finisher():Function<A,R>
characteristics():Set<Characteristics>

Non-Concurrent & Concurrent Collector APIs

- Five methods are defined in the Collector interface
  - characteristics()
  - **supplier()** – returns a supplier that acts as a factory to generate an empty result container, e.g.
    - `return ArrayList::new`

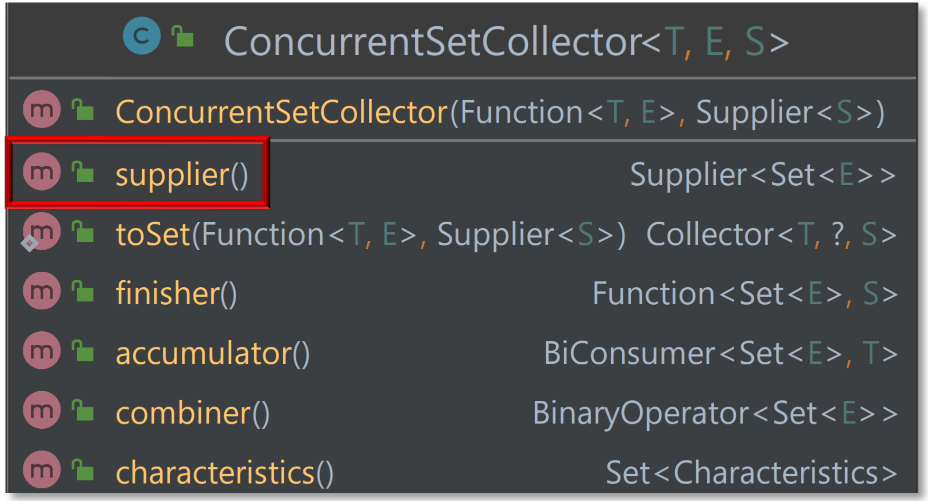


<<Java Interface>>
Collector<T,A,R>
supplier():Supplier<A>
accumulator():BiConsumer<A,T>
combiner():BinaryOperator<A>
finisher():Function<A,R>
characteristics():Set<Characteristics>

A non-concurrent collector provides a result container for each thread in a parallel stream

Non-Concurrent & Concurrent Collector APIs

- Five methods are defined in the Collector interface
  - characteristics()
  - **supplier()** – returns a supplier that acts as a factory to generate an empty result container, e.g.
    - `return ArrayList::new`
    - `return ConcurrentHashMap::newKeySet`

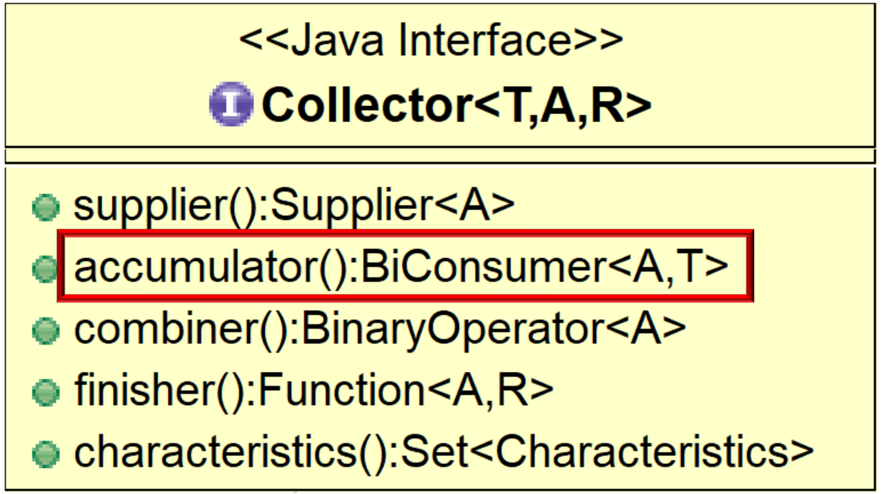


```
ConcurrentSetCollector<T, E, S>
ConcurrentSetCollector(Function<T, E>, Supplier<S>)
supplier()                                   Supplier<Set<E>>
toSet(Function<T, E>, Supplier<S>)   Collector<T, ?, S>
finisher()                                   Function<Set<E>, S>
accumulator()                                BiConsumer<Set<E>, T>
combiner()                                   BinaryOperator<Set<E>>
characteristics()                            Set<Characteristics>
```

A concurrent collector has one result container shared by all threads in a parallel stream

Non-Concurrent & Concurrent Collector APIs

- Five methods are defined in the Collector interface
  - characteristics()
  - supplier()
  - **accumulator()** – returns a bi-consumer that adds a new element to an existing result container

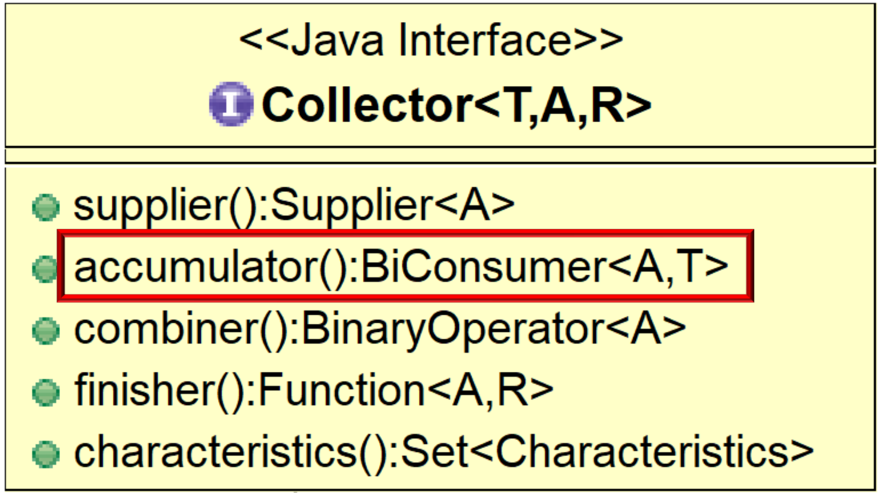


<<Java Interface>>
Collector<T,A,R>

- supplier():Supplier<A>
- accumulator():BiConsumer<A,T>
- combiner():BinaryOperator<A>
- finisher():Function<A,R>
- characteristics():Set<Characteristics>

Non-Concurrent & Concurrent Collector APIs

- Five methods are defined in the Collector interface
  - characteristics()
  - supplier()
  - **accumulator()** – returns a bi-consumer that adds a new element to an existing result container, e.g.
    - `return List::add`



<<Java Interface>>
Collector<T,A,R>

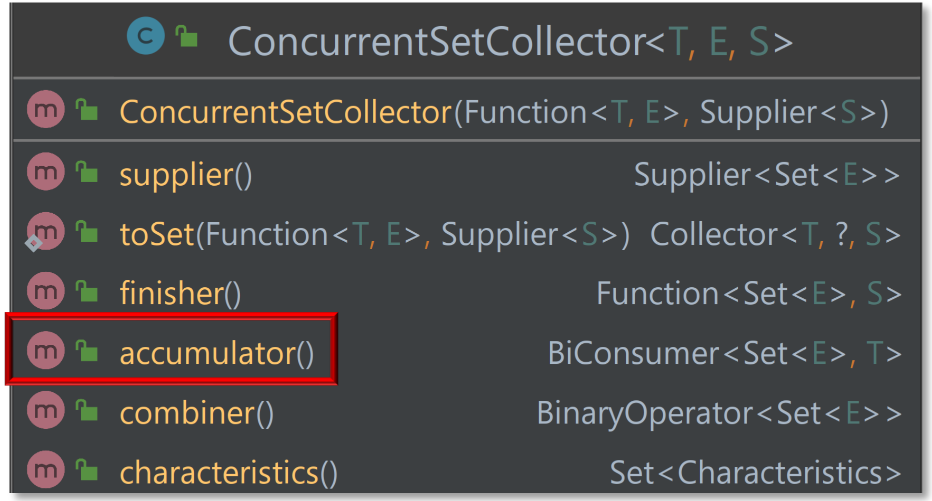
- supplier():Supplier<A>
- accumulator():BiConsumer<A,T>
- combiner():BinaryOperator<A>
- finisher():Function<A,R>
- characteristics():Set<Characteristics>

A non-concurrent collector needs no synchronization

See docs.oracle.com/javase/8/docs/api/java/util/List.html#add

Non-Concurrent & Concurrent Collector APIs

- Five methods are defined in the Collector interface
  - characteristics()
  - supplier()
  - **accumulator()** – returns a bi-consumer that adds a new element to an existing result container, e.g.
    - `return List::add`
    - `return ConcurrentHashMap.KeySetView::add`



ConcurrentSetCollector<T, E, S>

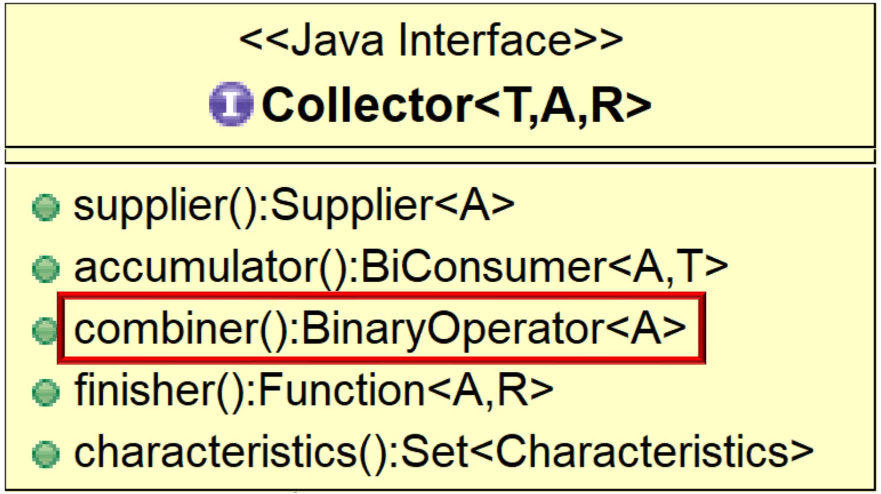
ConcurrentSetCollector(Function<T, E>, Supplier<S>)	
supplier()	Supplier<Set<E>>
toSet(Function<T, E>, Supplier<S>)	Collector<T, ?, S>
finisher()	Function<Set<E>, S>
accumulator()	BiConsumer<Set<E>, T>
combiner()	BinaryOperator<Set<E>>
characteristics()	Set<Characteristics>

A concurrent collector's result container must be synchronized

See docs.oracle.com/javase/8/docs/api/java/util/concurrent/ConcurrentHashMap.KeySetView.html

Non-Concurrent & Concurrent Collector APIs

- Five methods are defined in the Collector interface
  - **characteristics()**
  - **supplier()**
  - **accumulator()**
  - **combiner()** – returns a binary operator that merges two result containers together

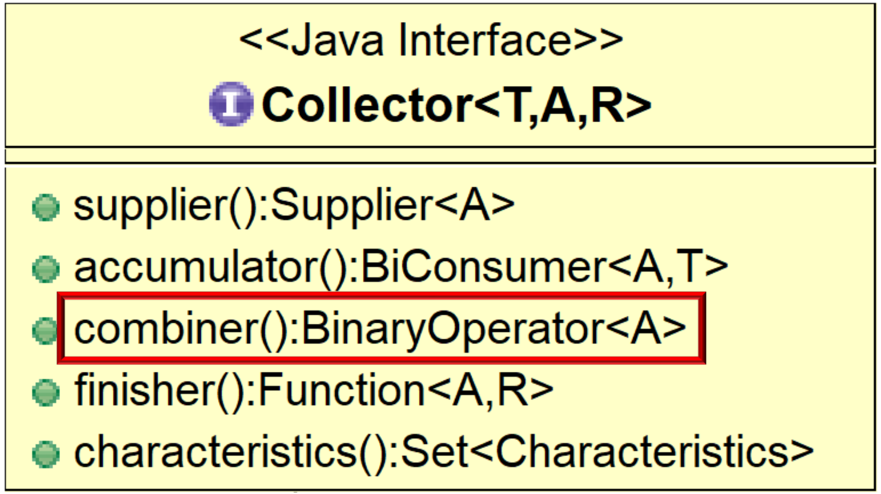


<<Java Interface>>
Collector<T,A,R>
supplier():Supplier<A>
accumulator():BiConsumer<A,T>
combiner():BinaryOperator<A>
finisher():Function<A,R>
characteristics():Set<Characteristics>

Non-Concurrent & Concurrent Collector APIs

- Five methods are defined in the Collector interface
  - characteristics()
  - supplier()
  - accumulator()
  - **combiner()** – returns a binary operator that merges two result containers together, e.g.

    -
      ```
      return (one, another) -> {
                    one.addAll(another); return one;
                 }
      ```

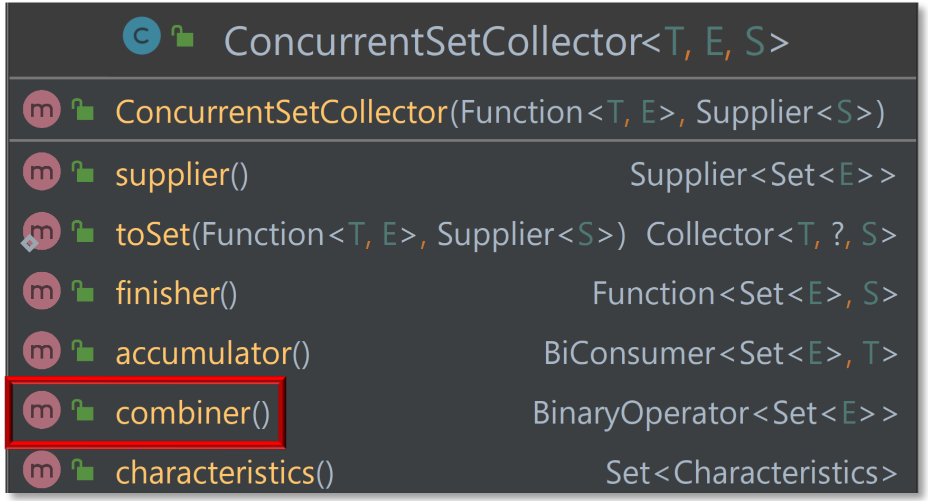


<<Java Interface>>
Collector<T,A,R>
supplier():Supplier<A>
accumulator():BiConsumer<A,T>
combiner():BinaryOperator<A>
finisher():Function<A,R>
characteristics():Set<Characteristics>

A combiner() is only used for a non-concurrent collector

Non-Concurrent & Concurrent Collector APIs

- Five methods are defined in the Collector interface
  - **characteristics()**
  - **supplier()**
  - **accumulator()**
  - **combiner()** – returns a binary operator that merges two result containers together, e.g.
    - `return (one, another) -> {`
      `one.addAll(another); return one;`
      `}`

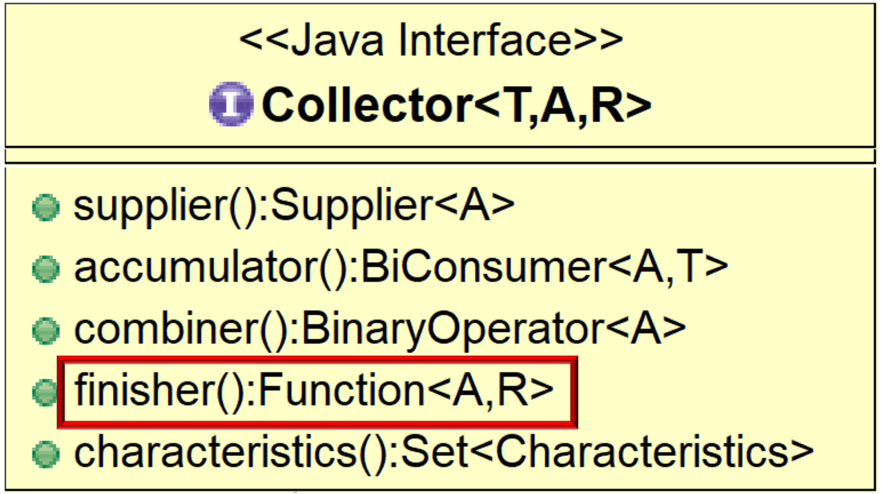


```
ConcurrentSetCollector<T, E, S>
ConcurrentSetCollector(Function<T, E>, Supplier<S>)
supplier()                                  Supplier<Set<E>>
toSet(Function<T, E>, Supplier<S>)          Collector<T, ?, S>
finisher()                                  Function<Set<E>, S>
accumulator()                               BiConsumer<Set<E>, T>
combiner()                                  BinaryOperator<Set<E>>
characteristics()                           Set<Characteristics>
```

The combiner() method is not called when CONCURRENT is set

Non-Concurrent & Concurrent Collector APIs

- Five methods are defined in the Collector interface
  - **characteristics()**
  - **supplier()**
  - **accumulator()**
  - **combiner()**
  - **finisher()** – returns a function that converts the result container to final result type

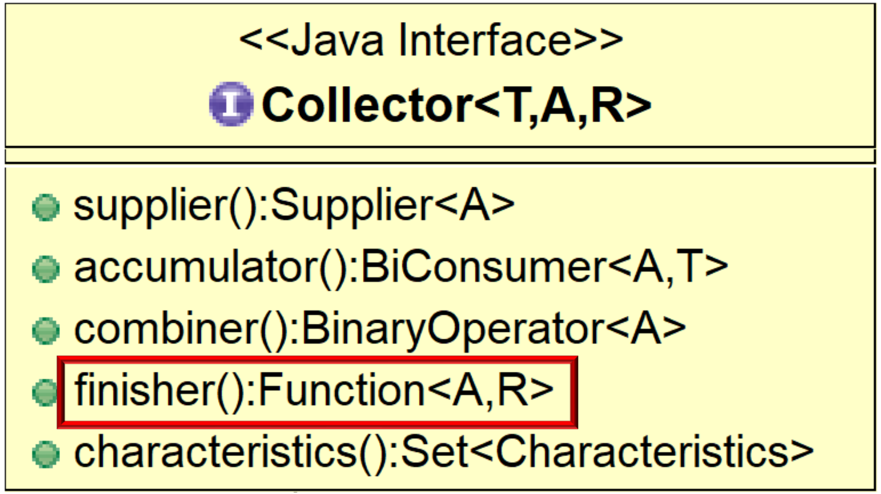


<<Java Interface>>
Collector<T,A,R>

- supplier():Supplier<A>
- accumulator():BiConsumer<A,T>
- combiner():BinaryOperator<A>
- finisher():Function<A,R>
- characteristics():Set<Characteristics>

Non-Concurrent & Concurrent Collector APIs

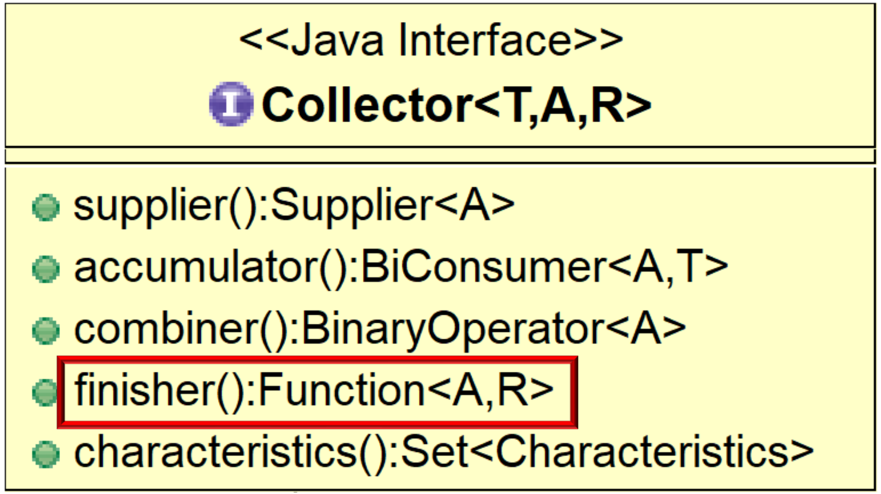
- Five methods are defined in the Collector interface
  - **characteristics()**
  - **supplier()**
  - **accumulator()**
  - **combiner()**
  - **finisher()** – returns a function that converts the result container to final result type, e.g.
    - `Function.identity()`



<<Java Interface>>
Collector<T,A,R>
supplier():Supplier<A>
accumulator():BiConsumer<A,T>
combiner():BinaryOperator<A>
finisher():Function<A,R>
characteristics():Set<Characteristics>

Non-Concurrent & Concurrent Collector APIs

- Five methods are defined in the Collector interface
  - characteristics()
  - supplier()
  - accumulator()
  - combiner()
  - **finisher()** – returns a function that converts the result container to final result type, e.g.
    - `Function.identity()`
    - **`return null`**



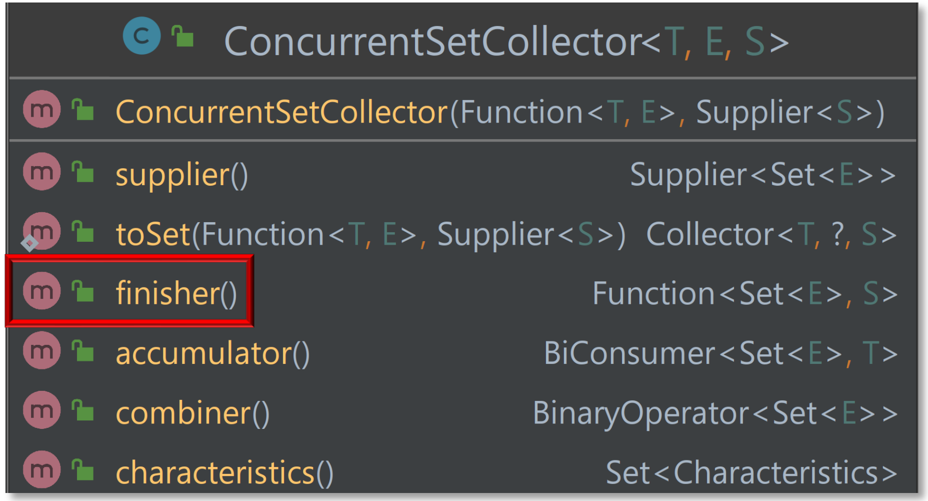
<<Java Interface>>
Collector<T,A,R>

- supplier():Supplier<A>
- accumulator():BiConsumer<A,T>
- combiner():BinaryOperator<A>
- finisher():Function<A,R>
- characteristics():Set<Characteristics>

Should be a no-op if IDENTITY_FINISH characteristic is set

Non-Concurrent & Concurrent Collector APIs

- Five methods are defined in the Collector interface
  - characteristics()
  - supplier()
  - accumulator()
  - combiner()
  - **finisher()** – returns a function that converts the result container to final result type, e.g.
    - `Function.identity()`
    - `return null`



```
ConcurrentSetCollector<T, E, S>
ConcurrentSetCollector(Function<T, E>, Supplier<S>)
supplier()                              Supplier<Set<E>>
toSet(Function<T, E>, Supplier<S>)      Collector<T, ?, S>
finisher()                              Function<Set<E>, S>
accumulator()                           BiConsumer<Set<E>, T>
combiner()                              BinaryOperator<Set<E>>
characteristics()                       Set<Characteristics>
```

```
return set -> {
  S ns = mSetSupplier.get();
  if (ns instanceof ConcurrentHashMap
                        .KeySetView)
    return (S) set;
  else { ns.addAll(set); return ns; }
};
```

finisher() can also be more interesting!

End of Java Parallel Streams Internals: Non-Concurrent & Concurrent Collectors (Part 2)