Understanding the Java Streams Non-Concurrent Collector API

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

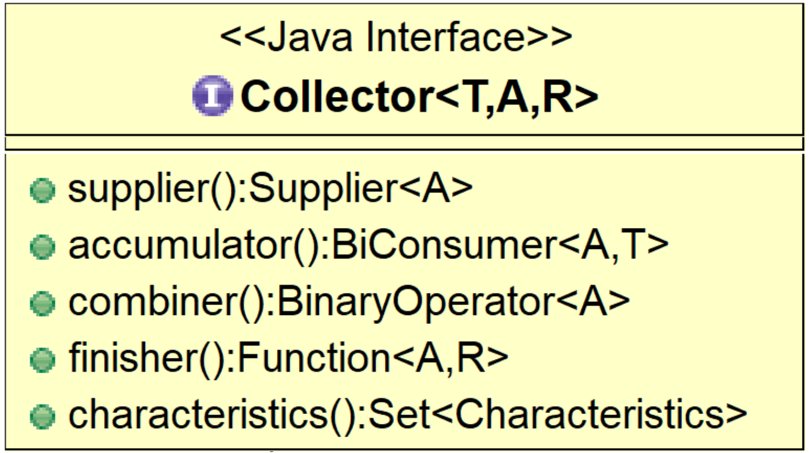
Professor of Computer Science

Institute for Software Integrated Systems

Vanderbilt University
Nashville, Tennessee, USA

Learning Objectives in this Part of the Lesson

- Understand the structure & functionality of non-concurrent collectors for sequential streams
- Know the API for non-concurrent collectors



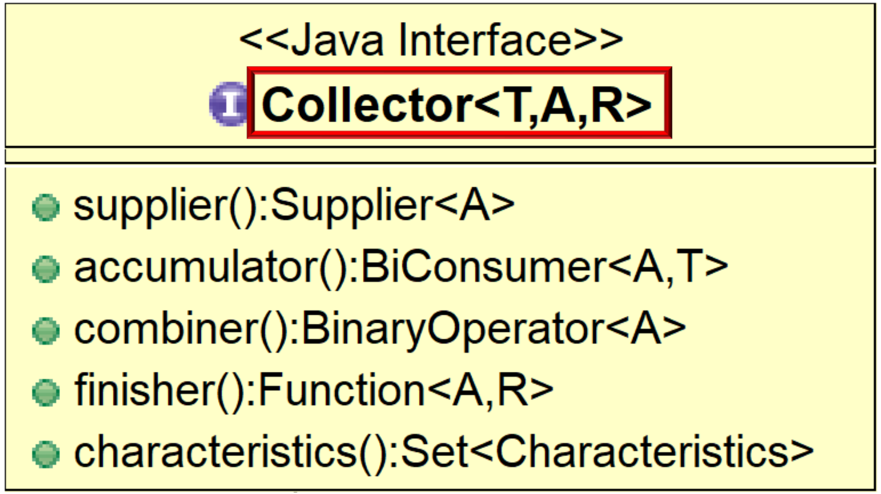
<<Java Interface>>
Collector<T,A,R>
supplier():Supplier<A>
accumulator():BiConsumer<A,T>
combiner():BinaryOperator<A>
finisher():Function<A,R>
characteristics():Set<Characteristics>

The same API is also used for concurrent collectors!

The Non-Concurrent Collector API

The Non-Concurrent Collector API

- The Collector interface defines three generic types



<<Java Interface>>

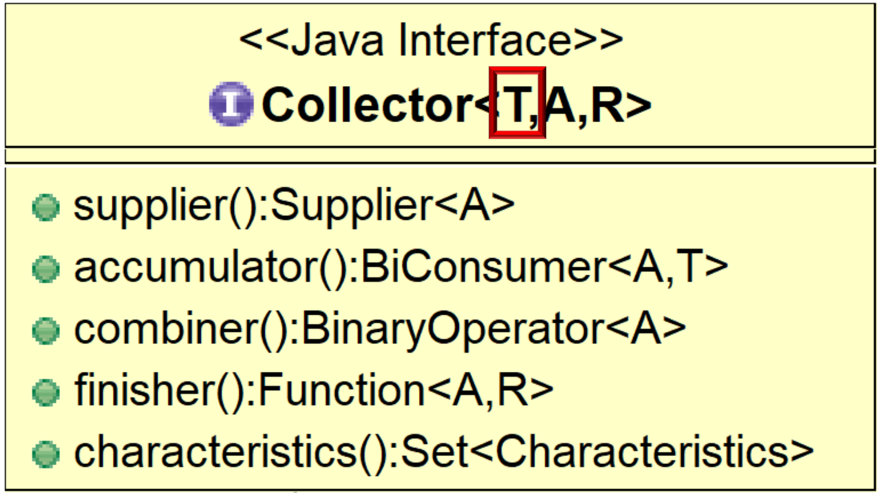
Collector<T,A,R>

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

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

The Non-Concurrent Collector API

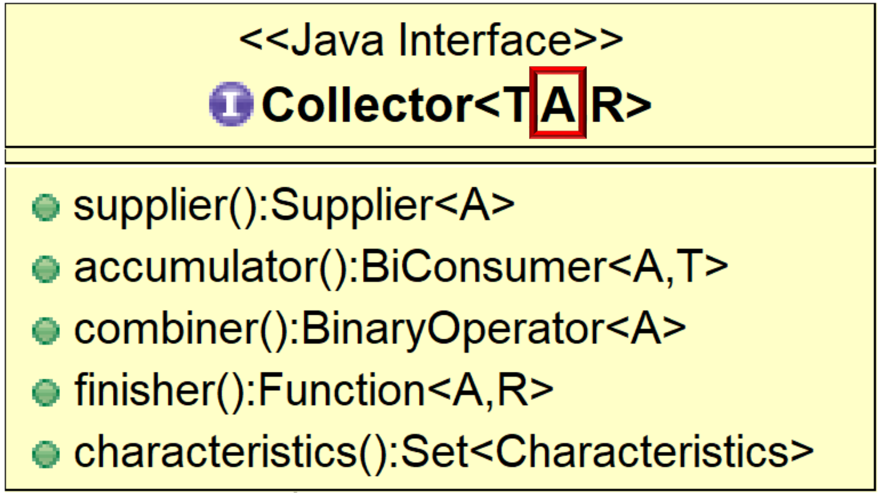
- The Collector interface defines three generic types
  - **T** – The type of elements available in the stream
    - e.g., Long, String, SearchResults, etc.



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

The Non-Concurrent Collector API

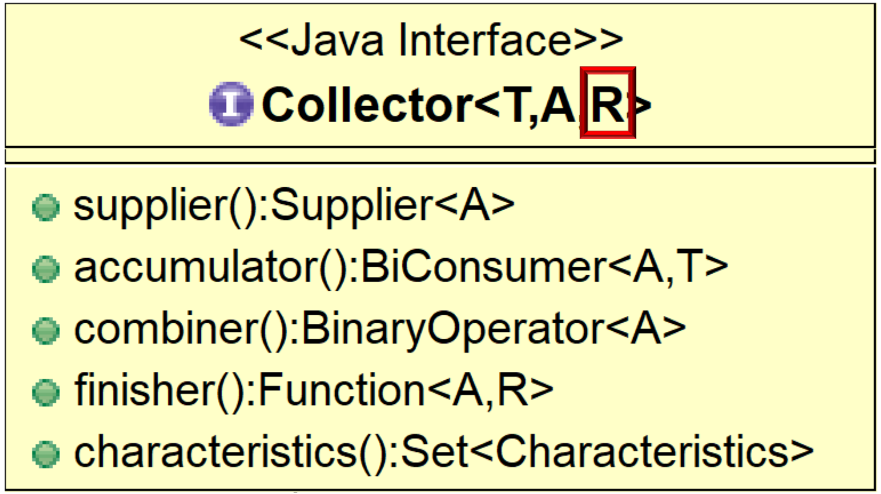
- The Collector interface defines three generic types
  - T
  - **A** – The type of mutable accumulator object to use for collecting elements
    - e.g., List or Map of T, which can be implemented via ArrayList, HashMap, etc.


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

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

The Non-Concurrent Collector API

- The Collector interface defines three generic types
  - T
  - A
  - **R** – The type of the final result
    - e.g., List or Map of T



<<Java Interface>>

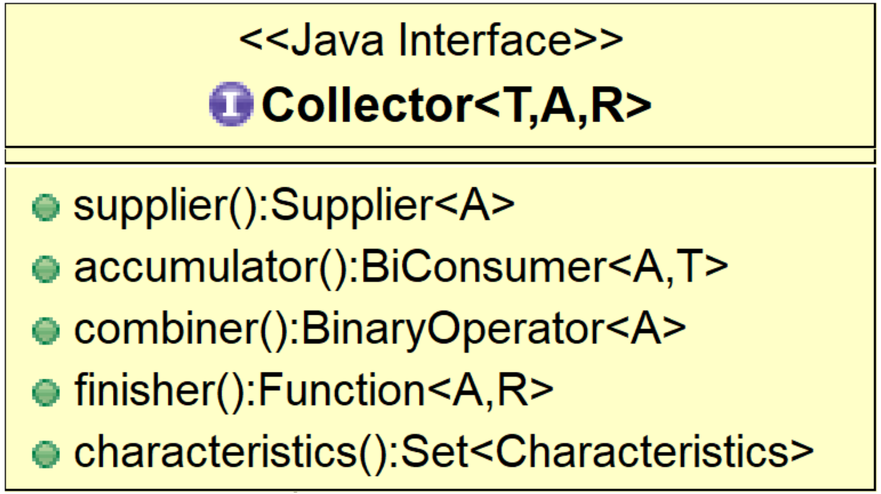
Collector<T,A,R>

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

The type of R & A may or may not be different (& are often the same)!

The Non-Concurrent Collector API

- Five factory methods are defined in the Collector interface



<<Java Interface>>

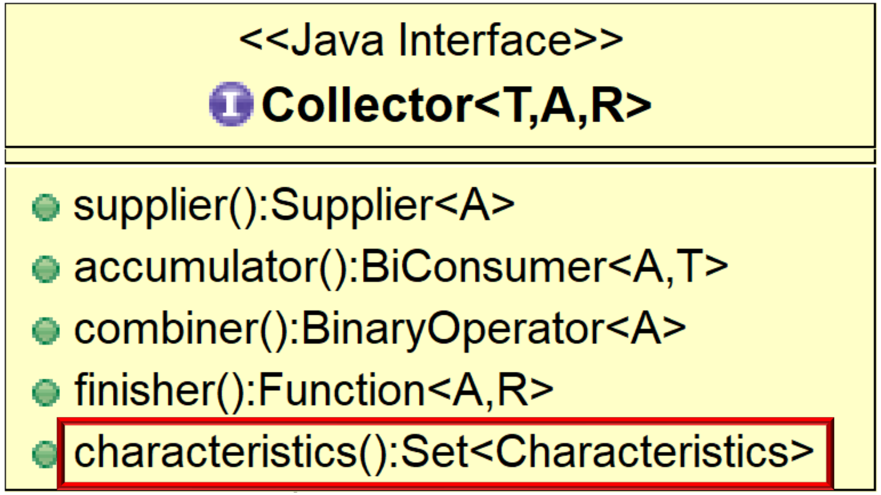
Collector<T,A,R>

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

Again, this discussion assumes we're implementing a *non-concurrent* collector

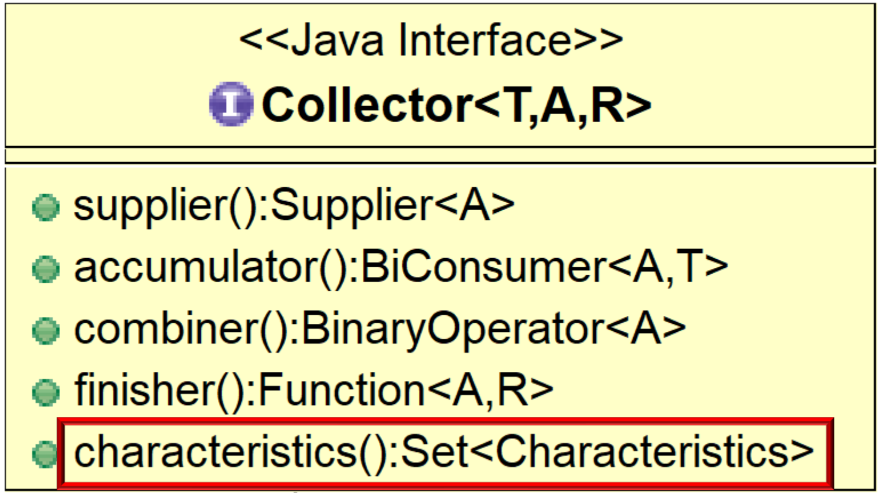
The Non-Concurrent Collector API

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


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

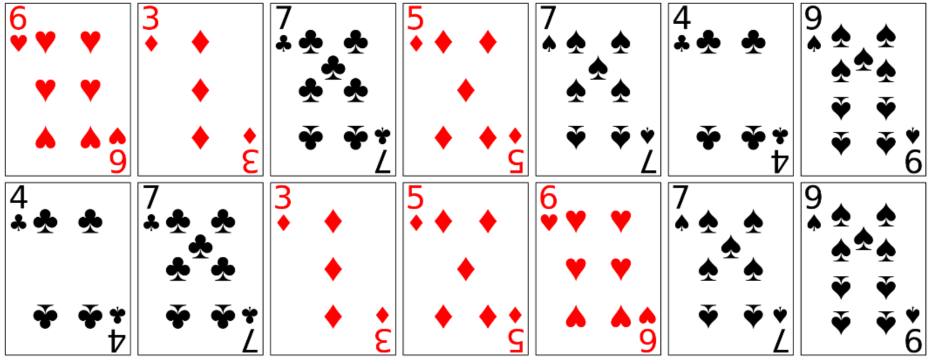
The Non-Concurrent Collector API

- Five factory 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



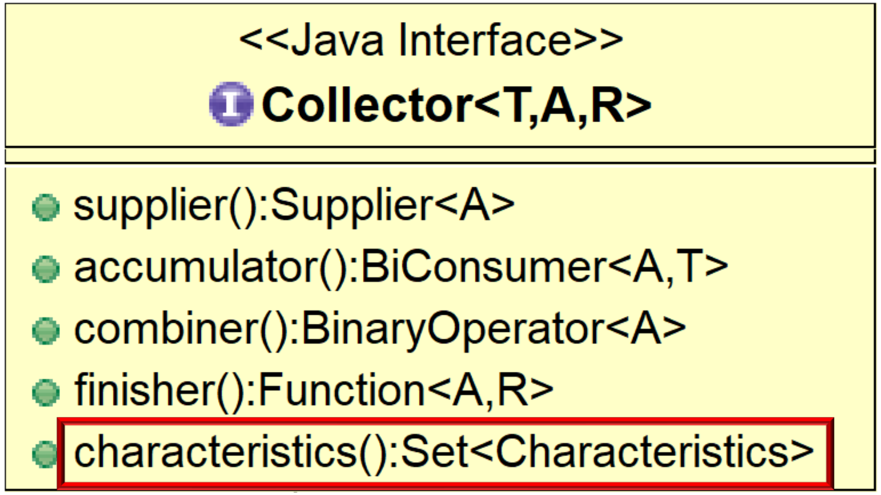
```
<<Java Interface>>
Collector<T,A,R>

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

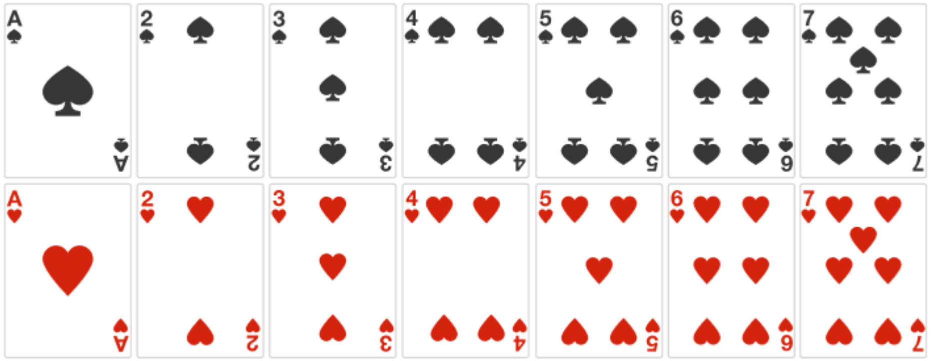


The Non-Concurrent Collector API

- Five factory 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>

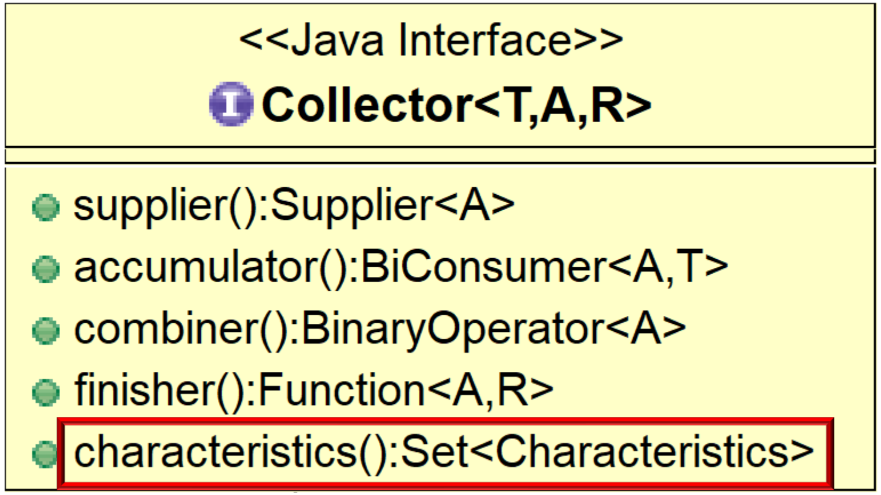
- supplier():Supplier<A>
- accumulator():BiConsumer<A,T>
- combiner():BinaryOperator<A>
- finisher():Function<A,R>
- characteristics():Set<Characteristics>



A collector may preserve encounter order if it incurs no additional overhead

The Non-Concurrent Collector API

- Five factory 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

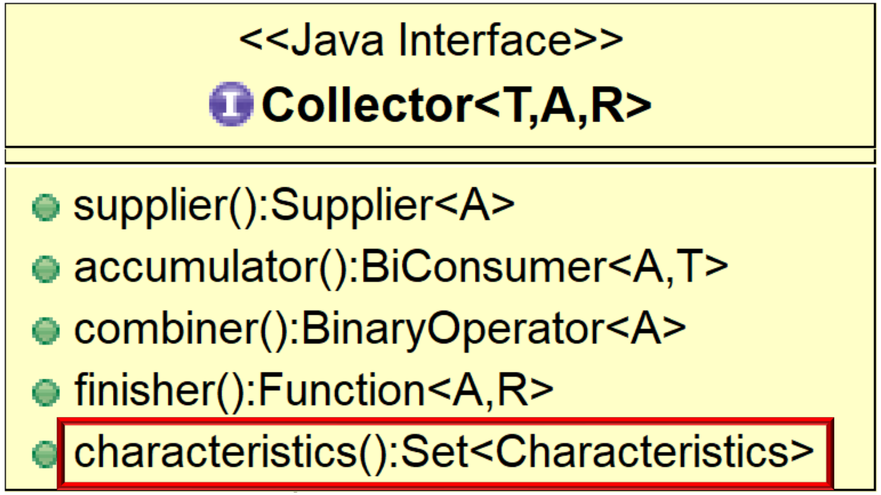


<<Java Interface>>
Collector<T,A,R>

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

The Non-Concurrent Collector API

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

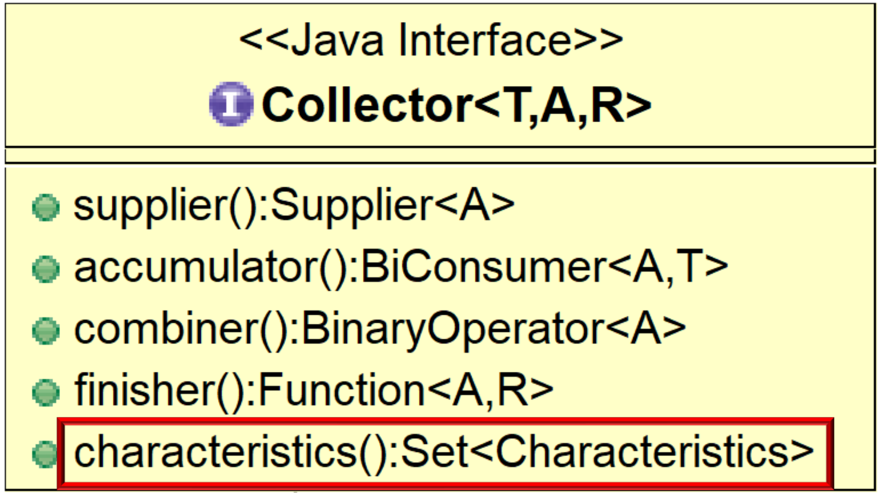

<<Java Interface>>
Collector<T,A,R>

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

The mutable result container must be synchronized!!

The Non-Concurrent Collector API

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



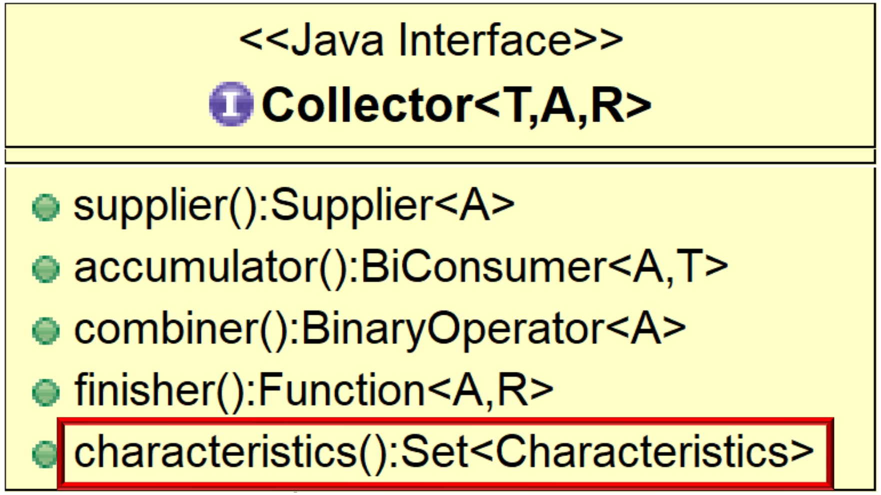
```
<<Java Interface>>
Collector<T,A,R>

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

We're focusing on a non-concurrent collector, which doesn't enable CONCURRENT

The Non-Concurrent Collector API

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



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

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

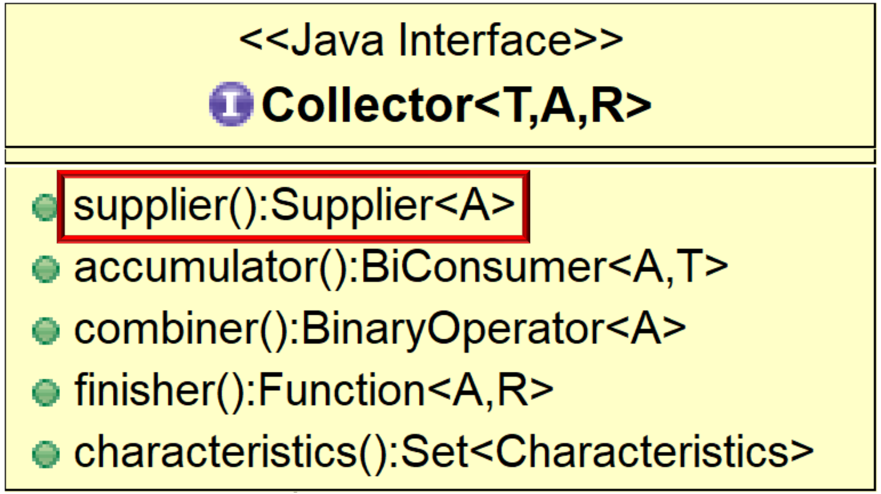
Any/all characteristics can be set using EnumSet.of()

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

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

The Non-Concurrent Collector API

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

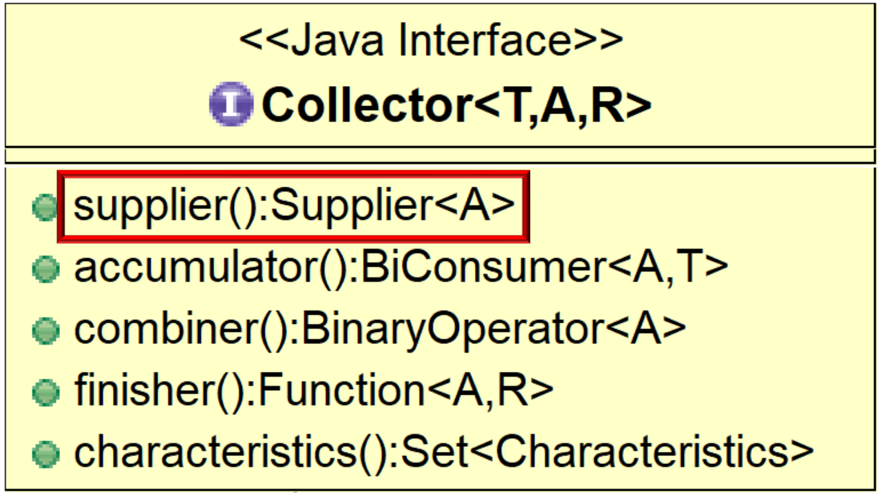


<<Java Interface>>
Collector<T,A,R>
supplier():Supplier<A>
accumulator():BiConsumer<A,T>
combiner():BinaryOperator<A>
finisher():Function<A,R>
characteristics():Set<Characteristics>

The Non-Concurrent Collector API

- Five factory 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.

```
Supplier<List> supplier() {
  return ArrayList::new;
}
```


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

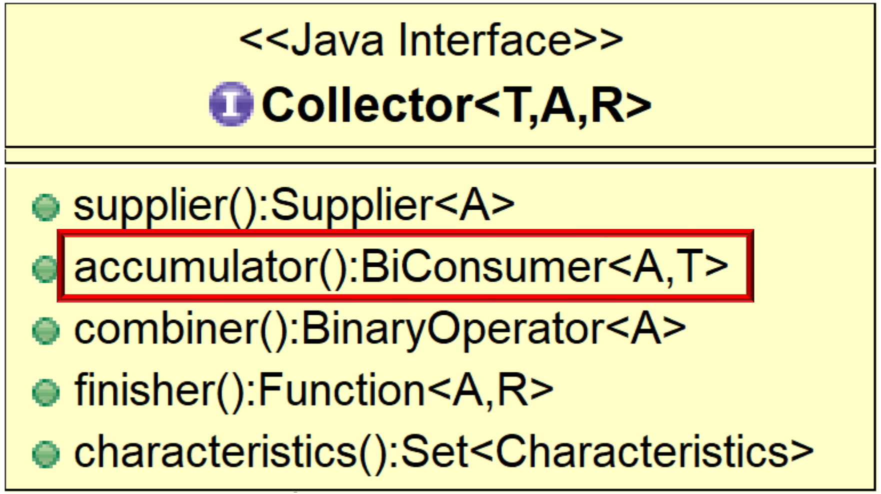
- supplier():Supplier<A>
- accumulator():BiConsumer<A,T>
- combiner():BinaryOperator<A>
- finisher():Function<A,R>
- characteristics():Set<Characteristics>

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

The Non-Concurrent Collector API

- Five factory 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.

```
BiConsumer<List, Integer> accumulator() {
  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>

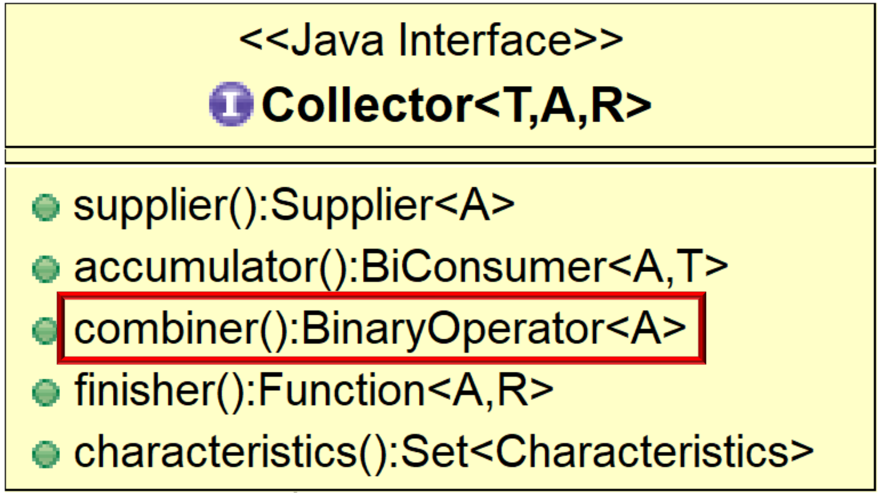
A non-concurrent collector needs no synchronization

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

The Non-Concurrent Collector API

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

```
BinaryOperator<List> combiner() {
  return (one, another) -> {
             one.addAll(another);
             return one;
  }};
```

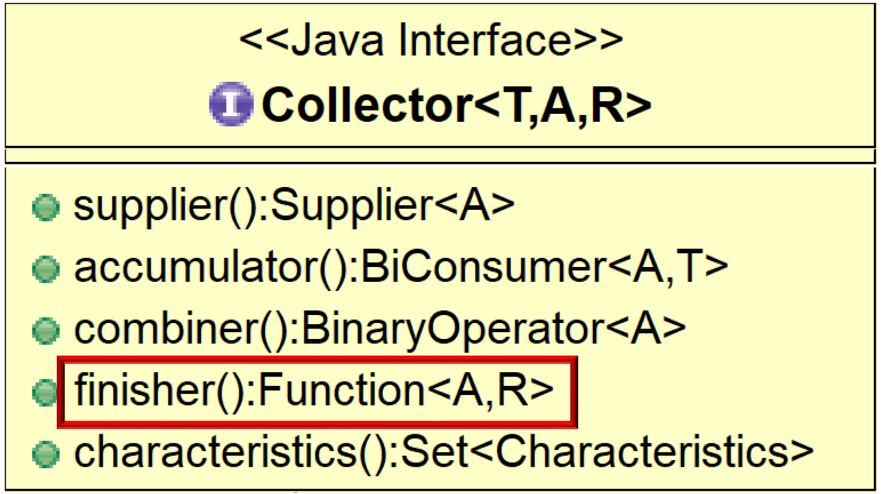


<<Java Interface>>
Collector<T,A,R>
supplier():Supplier<A>
accumulator():BiConsumer<A,T>
combiner():BinaryOperator<A>
finisher():Function<A,R>
characteristics():Set<Characteristics>

This combiner() will not be called for a sequential stream..

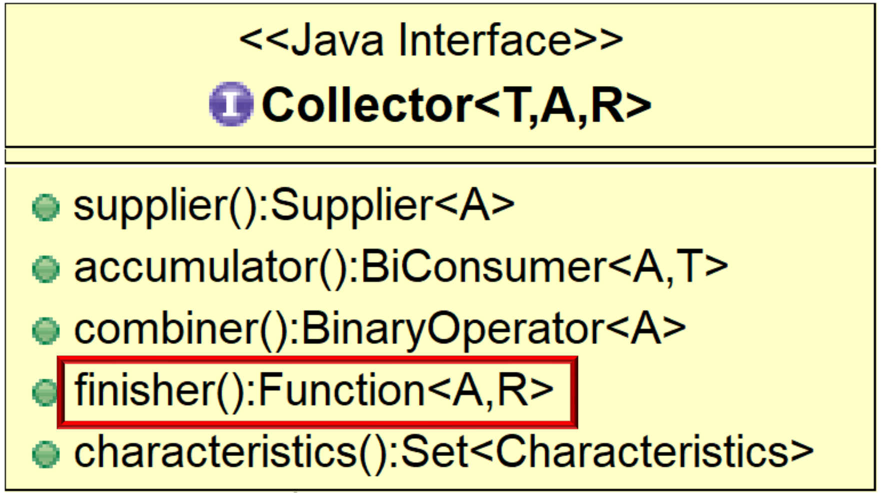
The Non-Concurrent Collector API

- Five factory 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.
    - `return 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>

The Non-Concurrent Collector API

- Five factory 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.
    - `return 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

The Non-Concurrent Collector API

- Five factory 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.
    - `return Function.identity()`
    - `return null;`

```
Stream
  .generate(() ->
     makeBigFraction
       (new Random(), false))
  .limit(sMAX_FRACTIONS)

  .map(reduceAndMultiplyFraction)
  .collect(FuturesCollector
             .toFuture())
```

finisher() can also be much more interesting!

```
  .thenAccept
     (this::sortAndPrintList);
```

See Java8/ex19/src/main/java/utils/FuturesCollector.java

End of Understanding the Java Streams Non-Concurrent Collector API