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

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

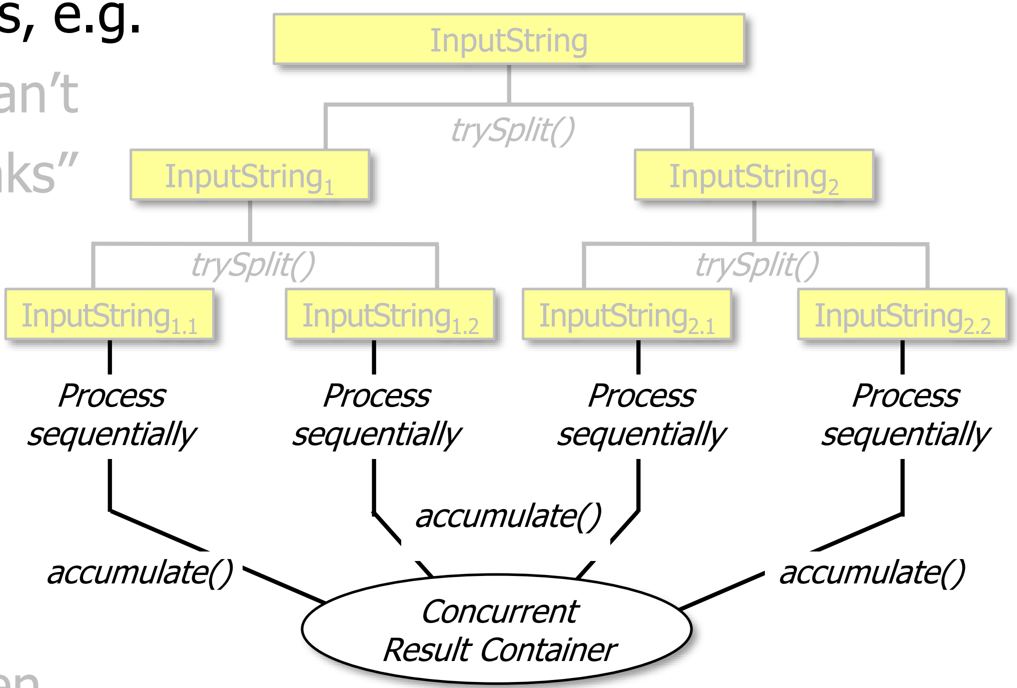
Professor of Computer Science

Institute for Software Integrated Systems

Vanderbilt University
Nashville, Tennessee, USA

Learning Objectives in this Part of the Lesson

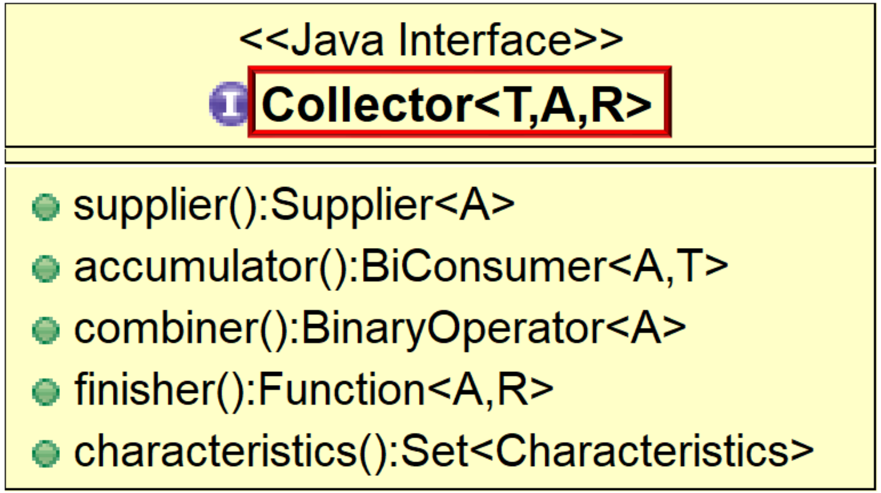
- Understand parallel stream internals, e.g.
  - Know what can change & what can't
  - Partition a data source into "chunks"
  - Process chunks in parallel via the common fork-join pool
  - Configure the Java parallel stream common fork-join pool
  - Perform a reduction to combine partial results into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



Implementing Non-Concurrent & Concurrent Collectors

Implementing Non-Concurrent & Concurrent Collectors

- The Collector interface defines three generic types



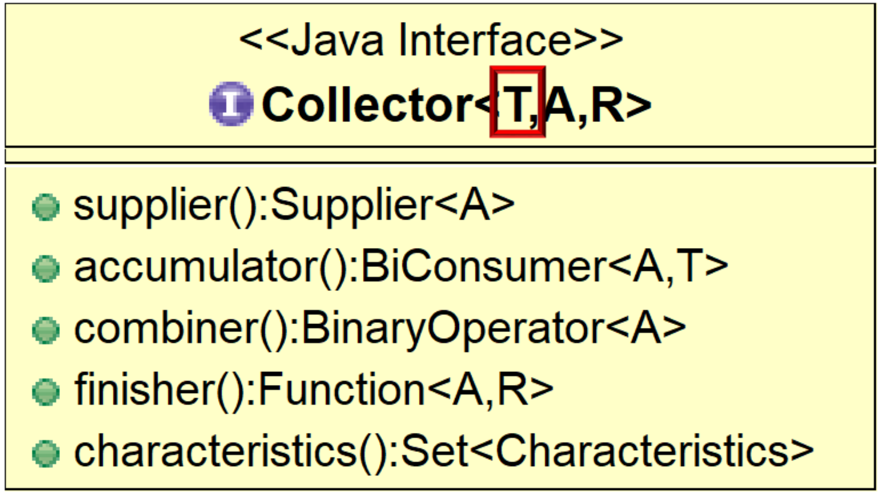
```
<<Java Interface>>
Collector<T,A,R>

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

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

Implementing Non-Concurrent & Concurrent Collectors

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

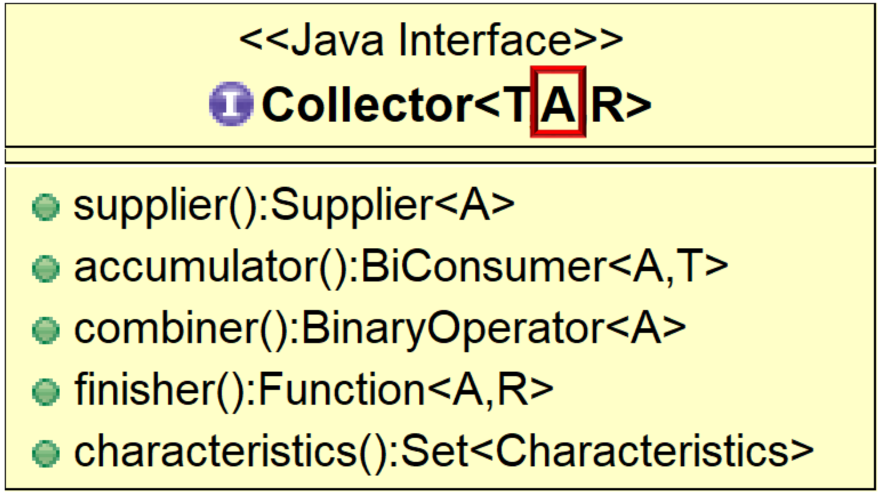

<<Java Interface>>

Collector<T,A,R>

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

Implementing Non-Concurrent & Concurrent Collectors

- The Collector interface defines three generic types
  - T
  - **A** – The type of a mutable accumulator object for collection
    - e.g., ConcurrentHashSet, List of T, Future of T, etc.
      - Lists can be implemented by ArrayList, LinkedList, etc.



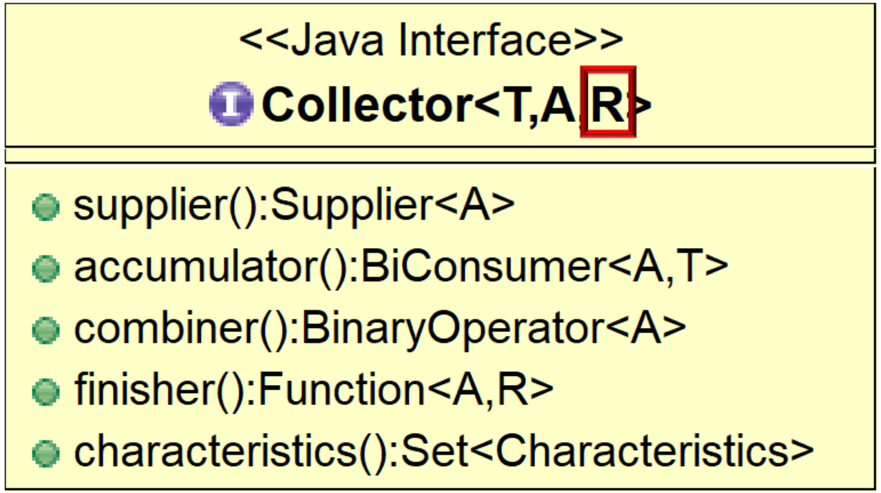
<<Java Interface>>
Collector<T,A,R>

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

See [Java8/ex14/src/main/java/utils/ConcurrentHashSet.java]

Implementing Non-Concurrent & Concurrent Collectors

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



<<Java Interface>>

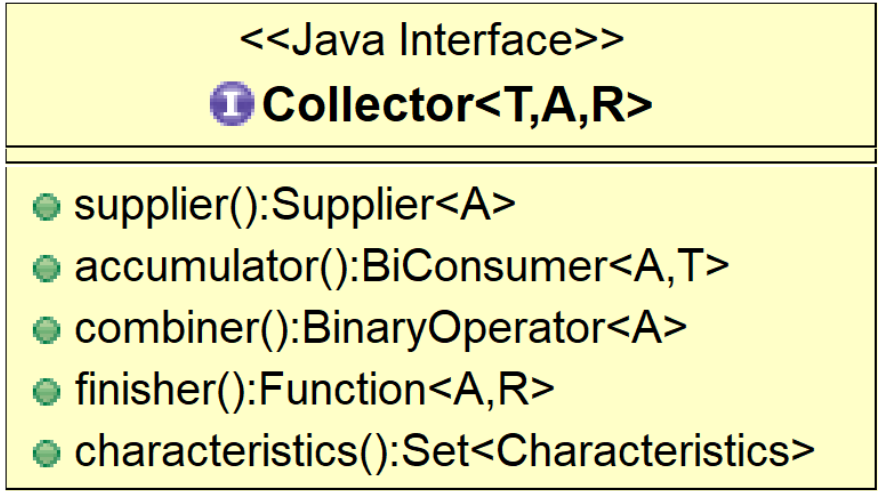
Collector<T,A,R>

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

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

Implementing Non-Concurrent & Concurrent Collectors

- Five methods are defined in the Collector interface



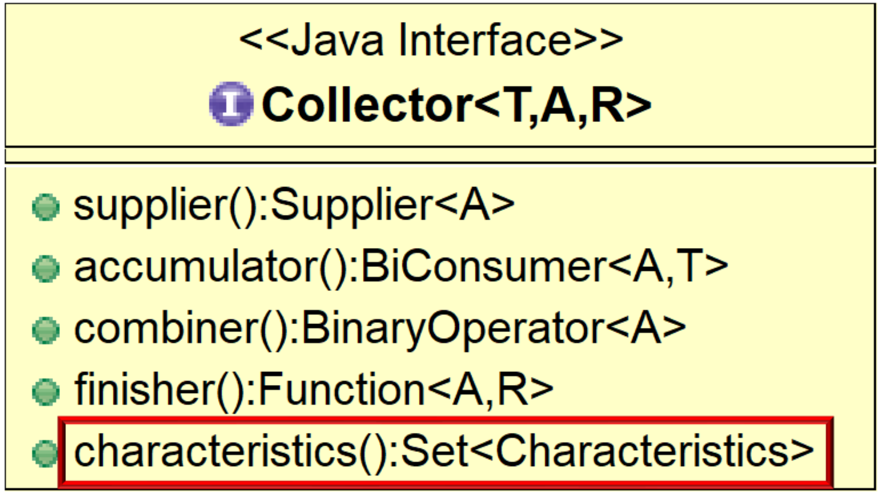
<<Java Interface>>

Collector<T,A,R>

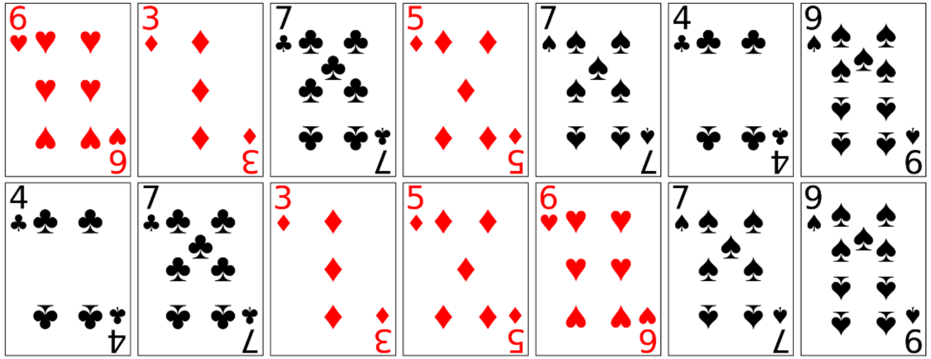
- supplier():Supplier<A>
- accumulator():BiConsumer<A,T>
- combiner():BinaryOperator<A>
- finisher():Function<A,R>
- characteristics():Set<Characteristics>

Implementing Non-Concurrent & Concurrent Collectors

- 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



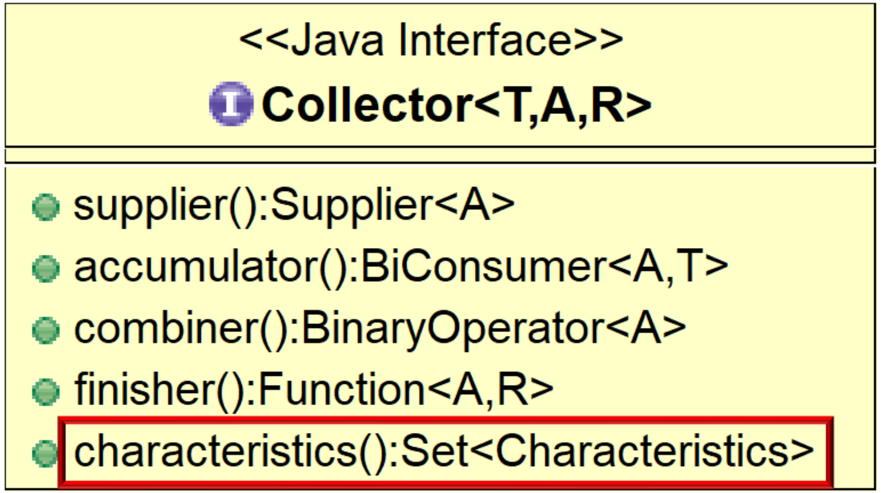
<<Java Interface>>
Collector<T,A,R>
supplier():Supplier<A>
accumulator():BiConsumer<A,T>
combiner():BinaryOperator<A>
finisher():Function<A,R>
characteristics():Set<Characteristics>



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

Implementing Non-Concurrent & Concurrent Collectors

- 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

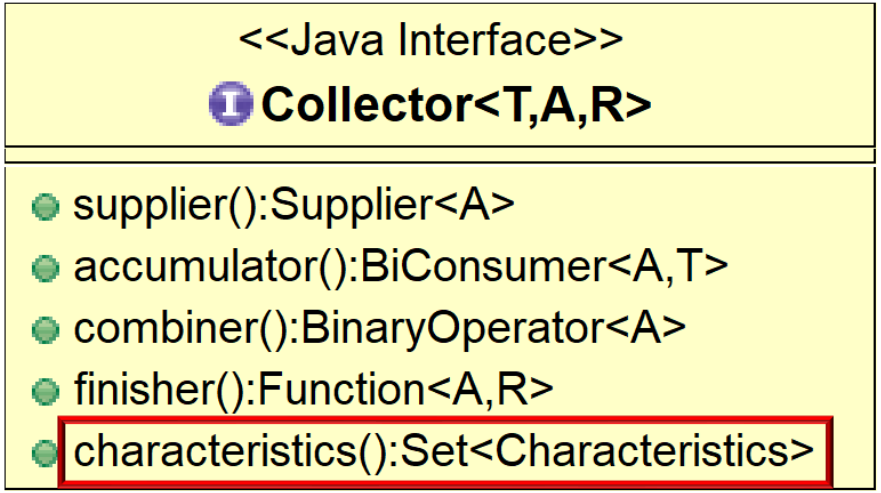


<<Java Interface>>
Collector<T,A,R>

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

Implementing Non-Concurrent & Concurrent Collectors

- 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>

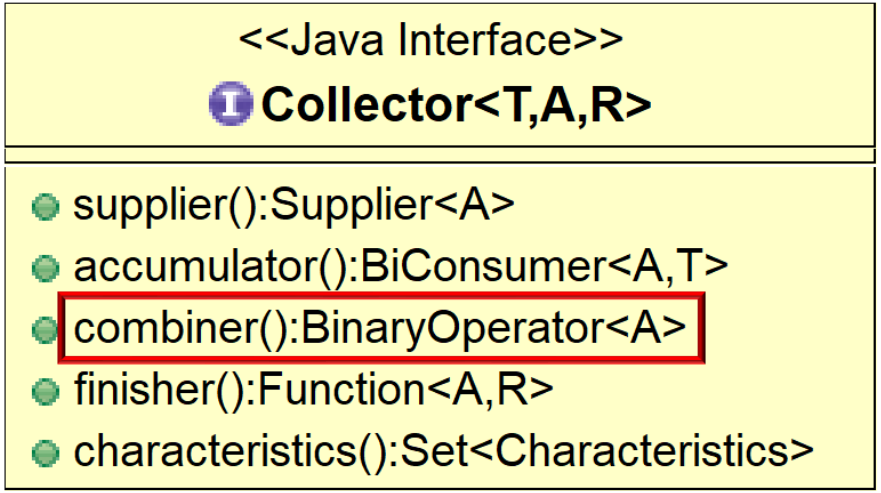
- supplier():Supplier<A>
- accumulator():BiConsumer<A,T>
- combiner():BinaryOperator<A>
- finisher():Function<A,R>
- characteristics():Set<Characteristics>

The mutable result container must be synchronized!!

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

Implementing Non-Concurrent & Concurrent Collectors

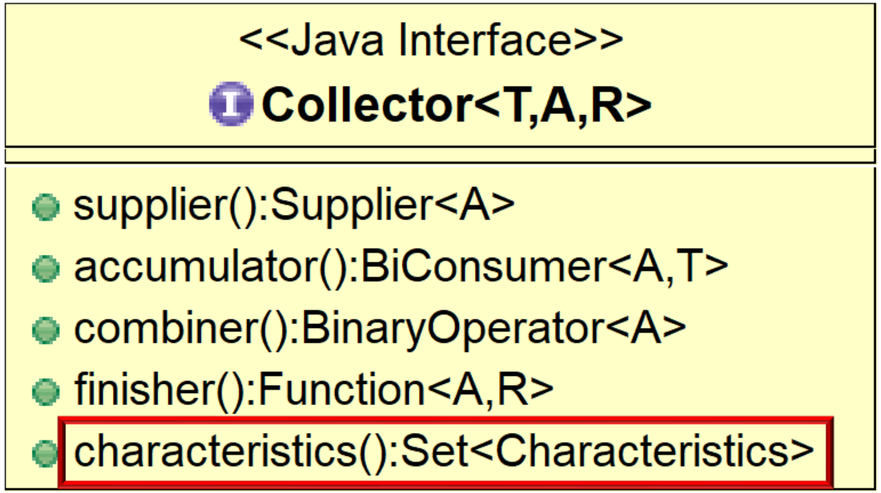
- Five methods are defined in the Collector interface
  - **characteristics()** – provides a stream with additional information used for internal optimizations, e.g.
    - UNORDERED
    - IDENTITY_FINISH
    - CONCURRENT
      - accumulator() is called concurrently on result container
      - The combiner()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

Implementing Non-Concurrent & Concurrent Collectors

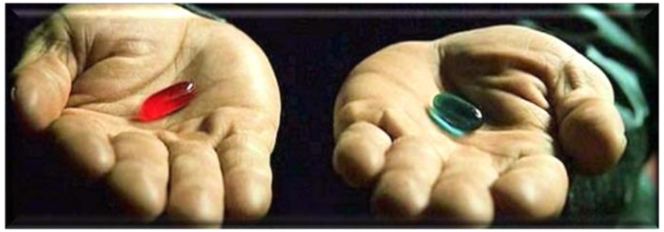
- Five methods are defined in the Collector interface
  - **characteristics()** – provides a stream with additional information used for internal optimizations, e.g.
    - UNORDERED
    - IDENTITY_FINISH
    - CONCURRENT
      - accumulator() is called concurrently on result container
      - The combiner()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



<<Java Interface>>

Collector<T,A,R>

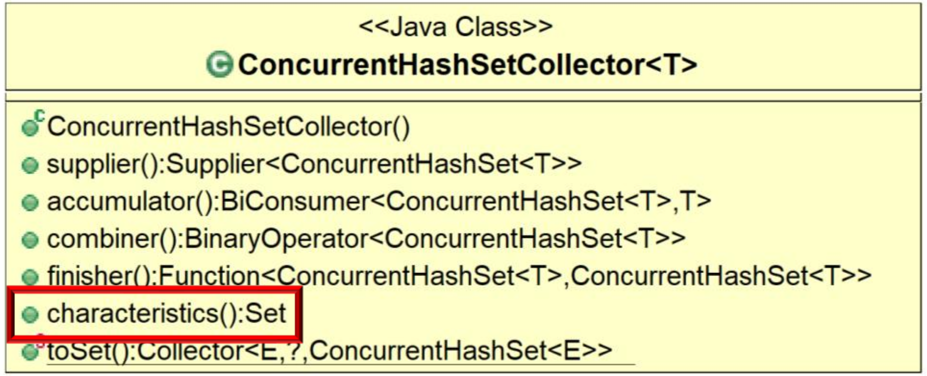
- supplier():Supplier<A>
- accumulator():BiConsumer<A,T>
- combiner():BinaryOperator<A>
- finisher():Function<A,R>
- characteristics():Set<Characteristics>



Internally, the streams framework decides how to ensure correct behavior

Implementing Non-Concurrent & Concurrent Collectors

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



```
<<Java Class>>
ConcurrentHashSetCollector<T>

ConcurrentHashSetCollector()
supplier():Supplier<ConcurrentHashSet<T>>
accumulator():BiConsumer<ConcurrentHashSet<T>,T>
combiner():BinaryOperator<ConcurrentHashSet<T>>
finisher():Function<ConcurrentHashSet<T>,ConcurrentHashSet<T>>
characteristics():Set
toSet():Collector<E,?,ConcurrentHashSet<E>>
```

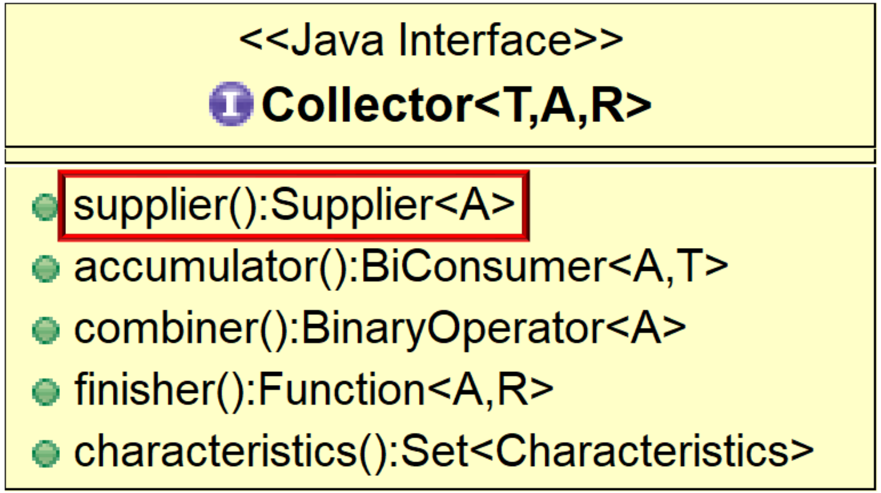
Any/all characteristics can be set using EnumSet.of()

```
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

Implementing Non-Concurrent & Concurrent Collectors

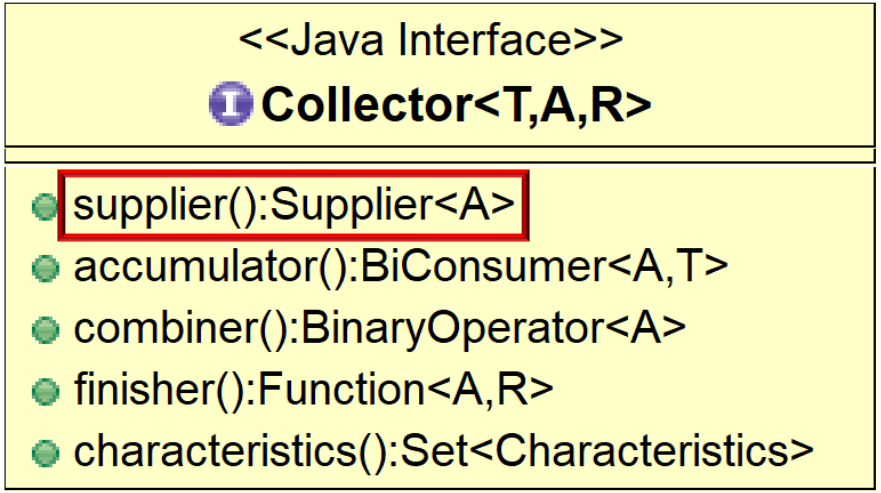
- Five methods are defined in the Collector interface
  - characteristics()
  - **supplier()** – returns a supplier that acts as a factory to generate an empty result container



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

Implementing Non-Concurrent & Concurrent Collectors

- 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`

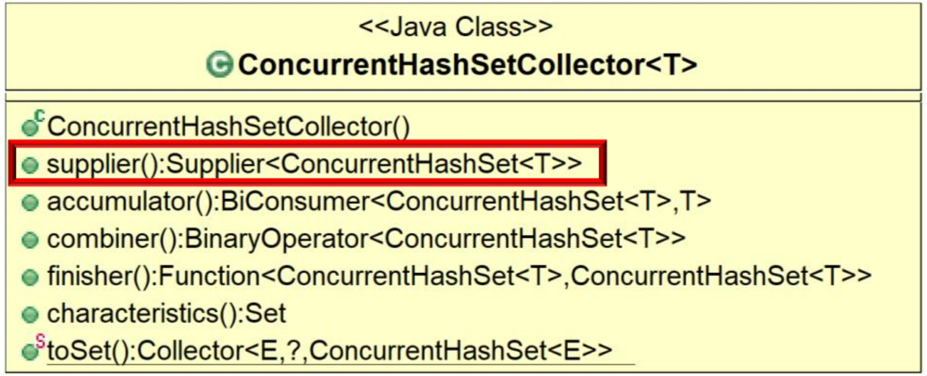


<<Java Interface>>
Collector<T,A,R>
supplier():Supplier<A>
accumulator():BiConsumer<A,T>
combiner():BinaryOperator<A>
finisher():Function<A,R>
characteristics():Set<Characteristics>

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

Implementing Non-Concurrent & Concurrent Collectors

- 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 ConcurrentHashSet::new`

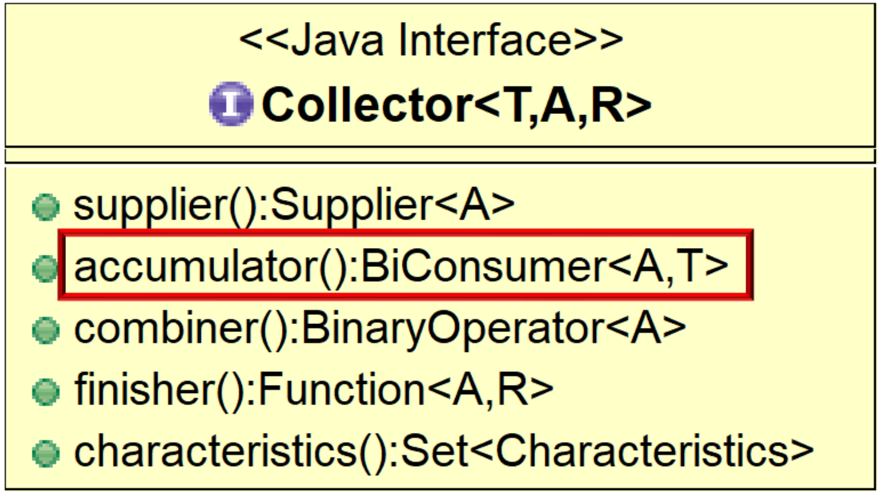

```
<<Java Class>>
ConcurrentHashSetCollector<T>

ConcurrentHashSetCollector()
supplier():Supplier<ConcurrentHashSet<T>>
accumulator():BiConsumer<ConcurrentHashSet<T>,T>
combiner():BinaryOperator<ConcurrentHashSet<T>>
finisher():Function<ConcurrentHashSet<T>,ConcurrentHashSet<T>>
characteristics():Set
toSet():Collector<E,?,ConcurrentHashSet<E>>
```

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

Implementing Non-Concurrent & Concurrent Collectors

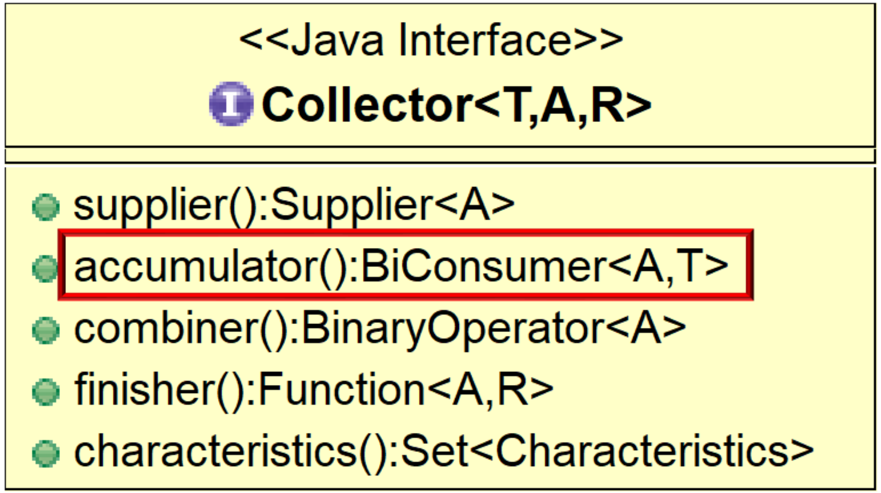
- Five methods are defined in the Collector interface
  - characteristics()
  - supplier()
  - **accumulator()** – returns a bi-consumer that adds a new element to an existing result container



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

Implementing Non-Concurrent & Concurrent Collectors

- 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`

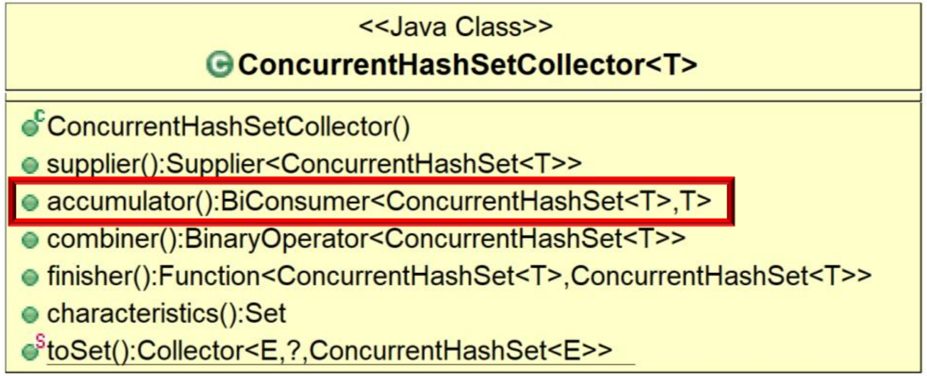

<<Java Interface>>
Collector<T,A,R>

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

A non-concurrent collector needs no synchronization

Implementing Non-Concurrent & Concurrent Collectors

- 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 ConcurrentHashSet::add`**



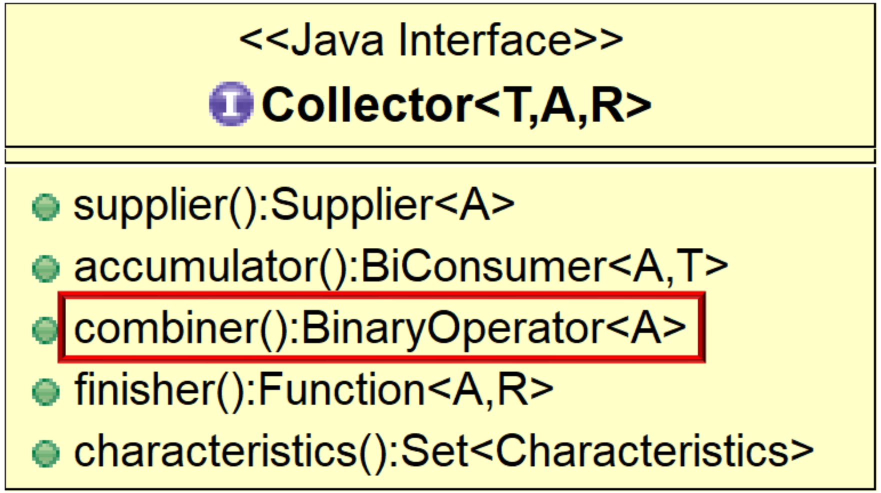
```
<<Java Class>>
ConcurrentHashSetCollector<T>

ConcurrentHashSetCollector()
supplier():Supplier<ConcurrentHashSet<T>>
accumulator():BiConsumer<ConcurrentHashSet<T>,T>
combiner():BinaryOperator<ConcurrentHashSet<T>>
finisher():Function<ConcurrentHashSet<T>,ConcurrentHashSet<T>>
characteristics():Set
toSet():Collector<E,?,ConcurrentHashSet<E>>
```

A concurrent collector must be synchronized

Implementing Non-Concurrent & Concurrent Collectors

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



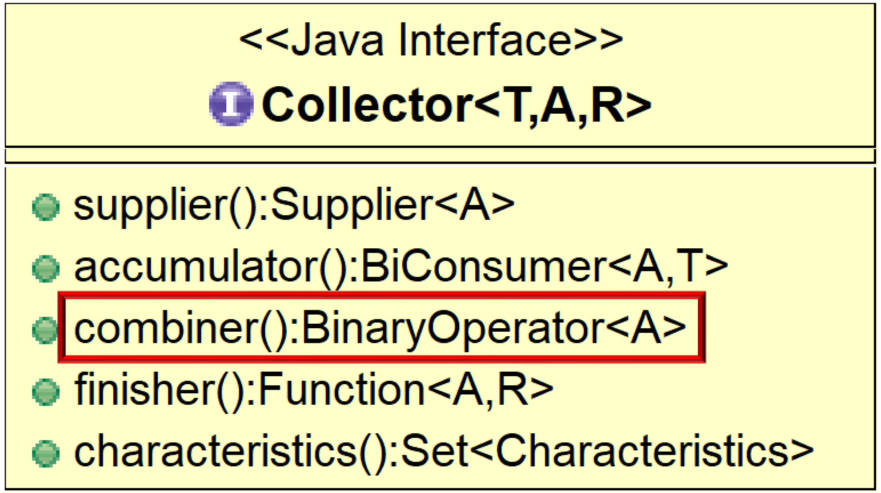
<<Java Interface>>
Collector<T,A,R>

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

Implementing Non-Concurrent & Concurrent Collectors

- 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;
                 }
      ```

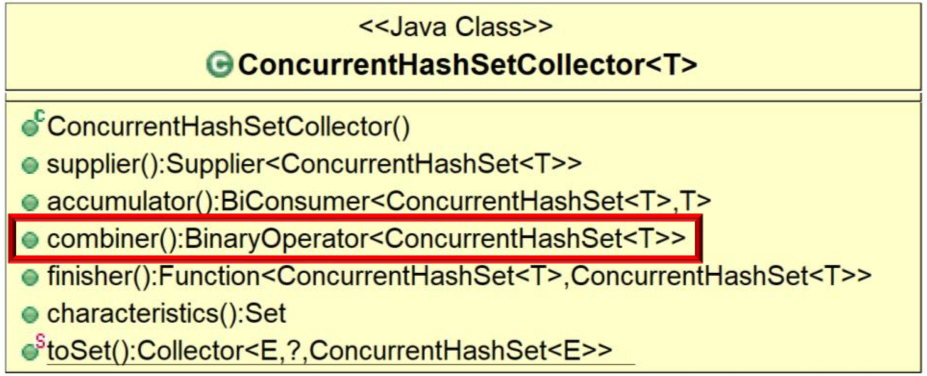


<<Java Interface>>
Collector<T,A,R>
supplier():Supplier<A>
accumulator():BiConsumer<A,T>
combiner():BinaryOperator<A>
finisher():Function<A,R>
characteristics():Set<Characteristics>

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

Implementing Non-Concurrent & Concurrent Collectors

- 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; }`
    - `return null`

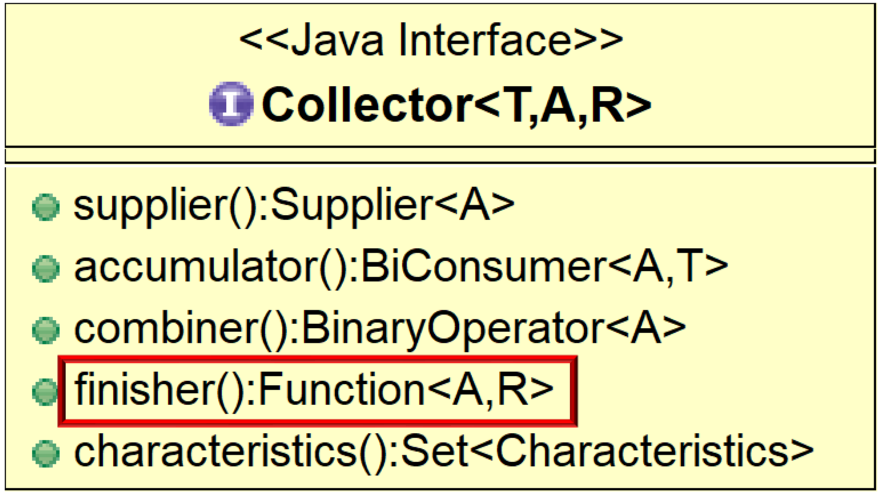


<<Java Class>>
ConcurrentHashSetCollector<T>
ConcurrentHashSetCollector()
supplier():Supplier<ConcurrentHashSet<T>>
accumulator():BiConsumer<ConcurrentHashSet<T>,T>
combiner():BinaryOperator<ConcurrentHashSet<T>>
finisher():Function<ConcurrentHashSet<T>,ConcurrentHashSet<T>>
characteristics():Set
toSet():Collector<E,?,ConcurrentHashSet<E>>

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

Implementing Non-Concurrent & Concurrent Collectors

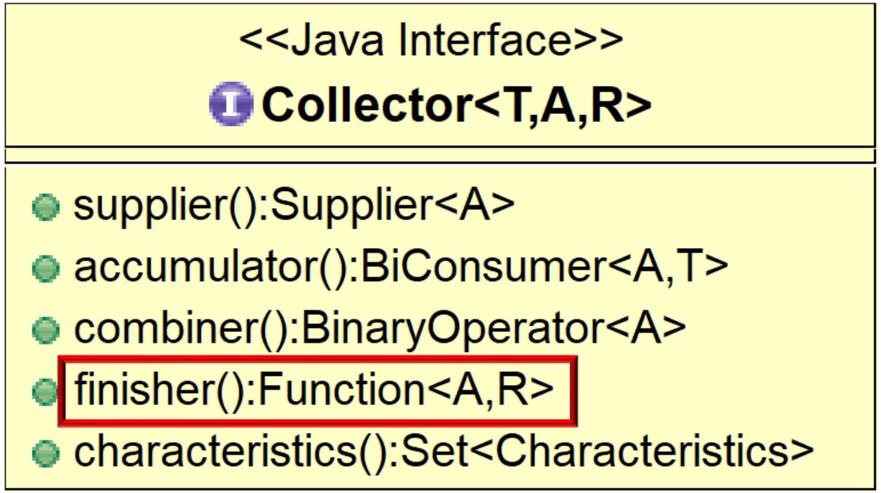
- Five methods are defined in the Collector interface
  - **characteristics()**
  - **supplier()**
  - **accumulator()**
  - **combiner()**
  - **finisher()** – returns a function that converts the result container to final result type



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

Implementing Non-Concurrent & Concurrent Collectors

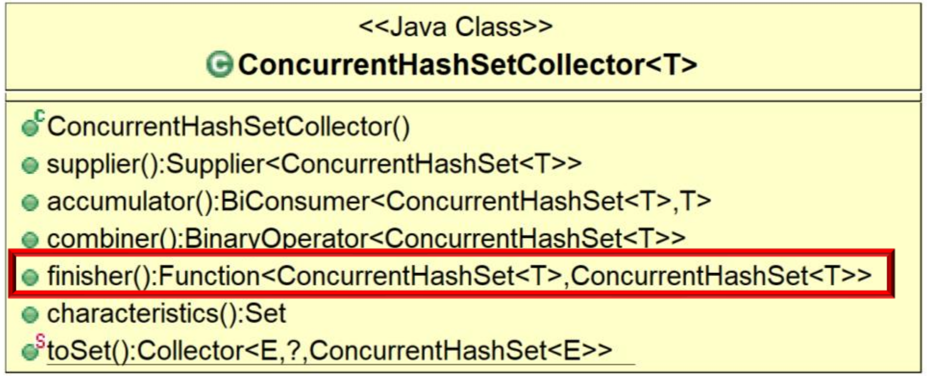
- Five methods are defined in the Collector interface
  - characteristics()
  - supplier()
  - accumulator()
  - combiner()
  - **finisher()** – returns a function that converts the result container to final result type, e.g.
    - **`Function.identity()`**



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

Implementing Non-Concurrent & Concurrent Collectors

- 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 Class>>
ConcurrentHashSetCollector<T>

ConcurrentHashSetCollector()
supplier():Supplier<ConcurrentHashSet<T>>
accumulator():BiConsumer<ConcurrentHashSet<T>,T>
combiner():BinaryOperator<ConcurrentHashSet<T>>
finisher():Function<ConcurrentHashSet<T>,ConcurrentHashSet<T>>
characteristics():Set
toSet():Collector<E,?,ConcurrentHashSet<E>>

Should be a no-op if IDENTITY_FINISH characteristic is set

Implementing Non-Concurrent & Concurrent Collectors

- 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`

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

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



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

finisher() can also be much more interesting!

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

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