**Java 8 Collectors mainly consist of three things –**

1. **Stream.collect() method**, collect () method is a terminal operation in Stream interface.
2. **Collector interface**, Collector is an interface in java.util.stream package.
3. **Collectors class**, Collectors class, also a member of java.util.stream package, is a utility class containing many static methods which perform some common reduction operations.

**Stream.collect() Method**

collect () method is a terminal operation in Stream interface. It is a special case of reduction operation called **mutable reduction operation** because it returns mutable result container such as List, Set or Map according to supplied Collector.

**java.util.stream.Collector Interface**

java.util.stream.Collector interface contains four functions that work together to accumulate input elements into a mutable result container and optionally performs a final transformation on the result.

**Those four functions are,**

1. **Supplier (): -** function that creates and returns a new mutable result container.
2. **Accumulator (): -** A function that accumulates a value into a mutable result container.
3. **Combiner (): -** A function that accepts two partial results and merges them.
4. **finisher (): -** A function that performs final transformation from the intermediate accumulation type to the final result type.

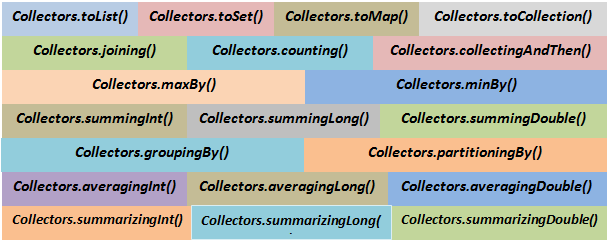
Collector Interface Methods,

1. **BiConsumer<A, T> accumulator ()** A function that folds a value into a mutable result container.
2. **Set<Collector.Characteristics> characteristics ()** Returns a Set of Collector.Characteristics indicating the characteristics of this Collector.
3. **BinaryOperator<A> combiner ()** A function that accepts two partial results and merges them.
4. **Function<A, R> finisher ()** Perform the final transformation from the intermediate accumulation type A to the final result type R.
5. **static <T, A, R> Collector<T, A, R> of (Supplier<A> supplier, BiConsumer<A, T> accumulator, BinaryOperator<A> combiner, Function<A, R> finisher, Collector.Characteristics... characteristics)** Returns a new Collector described by the given supplier, accumulator, combiner, and finisher functions.
6. **static <T, R> Collector<T, R, R> of (Supplier<R> supplier, BiConsumer<R, T> accumulator, BinaryOperator<R> combiner, Collector.Characteristics... characteristics)** Returns a new Collector described by the given supplier, accumulator, and combiner functions.
7. **Supplier<A> supplier ()** A function that creates and returns a new mutable result container.

**java.util.stream.Collectors Class**

java.util.stream.Collectors class contains static factory methods which perform some common reduction operations such as accumulating elements into Collection, finding min, max, average, sum of elements etc.

All the methods of Collectors class return Collector type which will be supplied to collect () method as an argument.



**Collectors Class Methods: -**

1. **Collectors.toList():** It returns a Collector which collects all input elements into a new List.
2. **Collectors.toSet():** It returns a Collector which collects all input elements into a new Set.
3. **Collectors.toMap() :** This method returns a Collector which collects input elements into a Map whose keys and values are the result of applying mapping functions to input elements.
4. **Collectors.toCollection():** This method returns a Collector which collects all input elements into a new Collection.
5. **Collectors.joining():** This method returns a Collector which concatenates input elements separated by the specified delimiter.
6. **Collectors.counting():** It returns a Collector that counts number of input elements.
7. **Collectors.maxBy():** This method returns a Collector that collects largest element in a stream according to supplied Comparator.
8. **Collectors.minBy():** This method returns a Collector which collects smallest element in a stream according to supplied Comparator.
9. **summingInt(), summingLong(), summingDouble():** These methods return a Collector which collects sum of all input elements.
10. **averagingInt(), averagingLong(), averagingDouble():** These methods return a Collector which collects average of input elements.
11. **summarizingInt(), summarizingLong(), summarizingDouble():** These methods return a special class called Int/Long/ DoubleSummaryStatistics which contain statistical information like sum, max, min, average etc. of input elements.
12. **Collectors.groupingBy():** This method groups the input elements according supplied classifier and returns the results in a Map.
13. **Collectors.partitioningBy():** This method partitions the input elements according to supplied Predicate and returns a Map<Boolean, List<T>>. Under the true key, you will find elements which match given Predicate and under the false key, you will find the elements which doesn’t match given Predicate.
14. **Collectors.collectingAndThen():** This is a special method which lets you to perform one more action on the result after collecting the result.

**Module** [**java.base**](https://docs.oracle.com/en/java/javase/16/docs/api/java.base/module-summary.html)

**Package** [**java.util.stream**](https://docs.oracle.com/en/java/javase/16/docs/api/java.base/java/util/stream/package-summary.html)

**Since: 1.8**

# **Interface Collector<T, A, R>**

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

**public interface Collector<T, A, R>**

A [mutable reduction operation](https://docs.oracle.com/en/java/javase/16/docs/api/java.base/java/util/stream/package-summary.html#Reduction) that accumulates input elements into a mutable result container, optionally transforming the accumulated result into a final representation after all input elements have been processed. Reduction operations can be performed either sequentially or in parallel.

Examples of mutable reduction operations include: accumulating elements into a Collection; concatenating strings using a StringBuilder; computing summary information about elements such as sum, min, max, or average; computing "pivot table" summaries such as "maximum valued transaction by seller", etc. The class [Collectors](https://docs.oracle.com/en/java/javase/16/docs/api/java.base/java/util/stream/Collectors.html) provides implementations of many common mutable reductions.

A Collector is specified by four functions that work together to accumulate entries into a mutable result container, and optionally perform a final transform on the result. They are:

* creation of a new result container ([supplier()](https://docs.oracle.com/en/java/javase/16/docs/api/java.base/java/util/stream/Collector.html#supplier()))
* incorporating a new data element into a result container ([accumulator()](https://docs.oracle.com/en/java/javase/16/docs/api/java.base/java/util/stream/Collector.html#accumulator()))
* combining two result containers into one ([combiner()](https://docs.oracle.com/en/java/javase/16/docs/api/java.base/java/util/stream/Collector.html#combiner()))
* performing an optional final transform on the container ([finisher()](https://docs.oracle.com/en/java/javase/16/docs/api/java.base/java/util/stream/Collector.html#finisher()))

Collectors also have a set of characteristics, such as [Collector.Characteristics.CONCURRENT](https://docs.oracle.com/en/java/javase/16/docs/api/java.base/java/util/stream/Collector.Characteristics.html#CONCURRENT), that provide hints that can be used by a reduction implementation to provide better performance.

A sequential implementation of a reduction using a collector would create a single result container using the supplier function, and invoke the accumulator function once for each input element. A parallel implementation would partition the input, create a result container for each partition, accumulate the contents of each partition into a subresult for that partition, and then use the combiner function to merge the subresults into a combined result.

Libraries that implement reduction based on Collector, such as [Stream.collect(Collector)](https://docs.oracle.com/en/java/javase/16/docs/api/java.base/java/util/stream/Stream.html#collect(java.util.stream.Collector)), must adhere to the following constraints:

* The first argument passed to the accumulator function, both arguments passed to the combiner function, and the argument passed to the finisher function must be the result of a previous invocation of the result supplier, accumulator, or combiner functions.
* The implementation should not do anything with the result of any of the result supplier, accumulator, or combiner functions other than to pass them again to the accumulator, combiner, or finisher functions, or return them to the caller of the reduction operation.
* If a result is passed to the combiner or finisher function, and the same object is not returned from that function, it is never used again.
* Once a result is passed to the combiner or finisher function, it is never passed to the accumulator function again.
* For non-concurrent collectors, any result returned from the result supplier, accumulator, or combiner functions must be serially thread-confined. This enables collection to occur in parallel without the Collector needing to implement any additional synchronization. The reduction implementation must manage that the input is properly partitioned, that partitions are processed in isolation, and combining happens only after accumulation is complete.
* For concurrent collectors, an implementation is free to (but not required to) implement reduction concurrently. A concurrent reduction is one where the accumulator function is called concurrently from multiple threads, using the same concurrently-modifiable result container, rather than keeping the result isolated during accumulation. A concurrent reduction should only be applied if the collector has the [Collector.Characteristics.UNORDERED](https://docs.oracle.com/en/java/javase/16/docs/api/java.base/java/util/stream/Collector.Characteristics.html#UNORDERED) characteristics or if the originating data is unordered.

**Module** [**java.base**](https://docs.oracle.com/en/java/javase/16/docs/api/java.base/module-summary.html)

**Package** [**java.util.stream**](https://docs.oracle.com/en/java/javase/16/docs/api/java.base/java/util/stream/package-summary.html)

**Since: 1.8**

# **Class Collectors**

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

**public final class Collectors**

Implementations of Collector that implement various useful reduction operations, such as accumulating elements into collections, summarizing elements according to various criteria, etc.

**Methods Added after 1.8: -**

### **ToUnmodifiableList**

public static <T> [Collector](https://docs.oracle.com/en/java/javase/16/docs/api/java.base/java/util/stream/Collector.html)<T,?, [List](https://docs.oracle.com/en/java/javase/16/docs/api/java.base/java/util/List.html)<T>> toUnmodifiableList()

Returns a Collector that accumulates the input elements into an [unmodifiable List](https://docs.oracle.com/en/java/javase/16/docs/api/java.base/java/util/List.html#unmodifiable) in encounter order. The returned Collector disallows null values and will throw NullPointerException if it is presented with a null value.

**Type Parameters:** T - the type of the input elements

**Returns:** A Collector that accumulates the input elements into an [unmodifiable List](https://docs.oracle.com/en/java/javase/16/docs/api/java.base/java/util/List.html#unmodifiable) in encounter order

**Since:** 10

### **ToUnmodifiableSet**

**public static <T>** [**Collector**](https://docs.oracle.com/en/java/javase/16/docs/api/java.base/java/util/stream/Collector.html)**<T, ?,** [**Set**](https://docs.oracle.com/en/java/javase/16/docs/api/java.base/java/util/Set.html)**<T>> toUnmodifiableSet()**

Returns a Collector that accumulates the input elements into an [unmodifiable Set](https://docs.oracle.com/en/java/javase/16/docs/api/java.base/java/util/Set.html#unmodifiable). The returned Collector disallows null values and will throw NullPointerException if it is presented with a null value. If the input contains duplicate elements, an arbitrary element of the duplicates is preserved.

This is an [unordered](https://docs.oracle.com/en/java/javase/16/docs/api/java.base/java/util/stream/Collector.Characteristics.html#UNORDERED) Collector.

**Type Parameters:**

T - the type of the input elements

**Returns:** a Collector that accumulates the input elements into an [unmodifiable Set](https://docs.oracle.com/en/java/javase/16/docs/api/java.base/java/util/Set.html#unmodifiable)

**Since:** 10

### **FlatMapping**

**public static <T, U, A, R>** [**Collector**](https://docs.oracle.com/en/java/javase/16/docs/api/java.base/java/util/stream/Collector.html)**<T, ?, R> flatMapping (**[**Function**](https://docs.oracle.com/en/java/javase/16/docs/api/java.base/java/util/function/Function.html)**<? super T, ? extends** [**Stream**](https://docs.oracle.com/en/java/javase/16/docs/api/java.base/java/util/stream/Stream.html)**<? extends U>> mapper,** [**Collector**](https://docs.oracle.com/en/java/javase/16/docs/api/java.base/java/util/stream/Collector.html)**<? super U, A, R> downstream)**

Adapts a Collector accepting elements of type U to one accepting elements of type T by applying a flat mapping function to each input element before accumulation. The flat mapping function maps an input element to a [stream](https://docs.oracle.com/en/java/javase/16/docs/api/java.base/java/util/stream/Stream.html) covering zero or more output elements that are then accumulated downstream. Each mapped stream is [closed](https://docs.oracle.com/en/java/javase/16/docs/api/java.base/java/util/stream/BaseStream.html#close()) after its contents have been placed downstream. (If a mapped stream is null an empty stream is used, instead.)

**API Note:** The flatMapping() collectors are most useful when used in a multi-level reduction, such as downstream of a groupingBy or partitioningBy. For example, given a stream of Order, to accumulate the set of line items for each customer: Map<String, Set<LineItem>> itemsByCustomerName = orders.stream().collect( groupingBy(Order::getCustomerName, flatMapping(order -> order.getLineItems().stream(), toSet())));

**Type Parameters:**

T - the type of the input elements

U - type of elements accepted by downstream collector

A - intermediate accumulation type of the downstream collector

R - result type of collector

**Parameters:**

mapper - a function to be applied to the input elements, which returns a stream of results

downstream - a collector which will receive the elements of the stream returned by mapper

**Returns:** a collector which applies the mapping function to the input elements and provides the flat mapped results to the downstream collector

**Since:** 9

### **Filtering**

**public static <T, A, R>** [**Collector**](https://docs.oracle.com/en/java/javase/16/docs/api/java.base/java/util/stream/Collector.html)**<T, ?, R> filtering (**[**Predicate**](https://docs.oracle.com/en/java/javase/16/docs/api/java.base/java/util/function/Predicate.html)**<? super T> predicate,** [**Collector**](https://docs.oracle.com/en/java/javase/16/docs/api/java.base/java/util/stream/Collector.html)**<? super T, A, R> downstream)**

Adapts a Collector to one accepting elements of the same type T by applying the predicate to each input element and only accumulating if the predicate returns true.

**API Note:**

The filtering() collectors are most useful when used in a multi-level reduction, such as downstream of a groupingBy or partitioningBy. For example, given a stream of Employee, to accumulate the employees in each department that have a salary above a certain threshold: Map<Department, Set<Employee>> wellPaidEmployeesByDepartment = employees.stream().collect( groupingBy(Employee::getDepartment, filtering(e -> e.getSalary() > 2000, toSet()))); A filtering collector differs from a stream's filter() operation. In this example, suppose there are no employees whose salary is above the threshold in some department. Using a filtering collector as shown above would result in a mapping from that department to an empty Set. If a stream filter() operation were done instead, there would be no mapping for that department at all.

**Type Parameters:**

T - the type of the input elements

A - intermediate accumulation type of the downstream collector

R - result type of collector

**Parameters:**

predicate - a predicate to be applied to the input elements

downstream - a collector which will accept values that match the predicate

**Returns: A** collector which applies the predicate to the input elements and provides matching elements to the downstream collector

**Since:** 9

### **ToUnmodifiableMap**

**public static <T, K, U>** [**Collector**](https://docs.oracle.com/en/java/javase/16/docs/api/java.base/java/util/stream/Collector.html)**<T, ?,** [**Map**](https://docs.oracle.com/en/java/javase/16/docs/api/java.base/java/util/Map.html)**<K, U>> toUnmodifiableMap (**[**Function**](https://docs.oracle.com/en/java/javase/16/docs/api/java.base/java/util/function/Function.html)**<? super T, ? extends K> keyMapper,** [**Function**](https://docs.oracle.com/en/java/javase/16/docs/api/java.base/java/util/function/Function.html)**<? super T, ? extends U> valueMapper)**

Returns a Collector that accumulates the input elements into an [unmodifiable Map](https://docs.oracle.com/en/java/javase/16/docs/api/java.base/java/util/Map.html#unmodifiable), whose keys and values are the result of applying the provided mapping functions to the input elements.

If the mapped keys contain duplicates (according to [Object.equals(Object)](https://docs.oracle.com/en/java/javase/16/docs/api/java.base/java/lang/Object.html#equals(java.lang.Object))), an IllegalStateException is thrown when the collection operation is performed. If the mapped keys might have duplicates, use [toUnmodifiableMap(Function, Function, BinaryOperator)](https://docs.oracle.com/en/java/javase/16/docs/api/java.base/java/util/stream/Collectors.html#toUnmodifiableMap(java.util.function.Function,java.util.function.Function,java.util.function.BinaryOperator)) to handle merging of the values.

The returned Collector disallows null keys and values. If either mapping function returns null, NullPointerException will be thrown.

**Type Parameters:**

T - the type of the input elements

K - the output type of the key mapping function

U - the output type of the value mapping function

**Parameters:**

keyMapper - a mapping function to produce keys, must be non-null

valueMapper - a mapping function to produce values, must be non-null

**Returns: A** Collector that accumulates the input elements into an [unmodifiable Map](https://docs.oracle.com/en/java/javase/16/docs/api/java.base/java/util/Map.html#unmodifiable), whose keys and values are the result of applying the provided mapping functions to the input elements

**Throws:** [NullPointerException](https://docs.oracle.com/en/java/javase/16/docs/api/java.base/java/lang/NullPointerException.html) - if either keyMapper or valueMapper is null

**Since:** 10

### **ToUnmodifiableMap**

**public static <T, K, U>** [**Collector**](https://docs.oracle.com/en/java/javase/16/docs/api/java.base/java/util/stream/Collector.html)**<T, ?,** [**Map**](https://docs.oracle.com/en/java/javase/16/docs/api/java.base/java/util/Map.html)**<K, U>> toUnmodifiableMap (**[**Function**](https://docs.oracle.com/en/java/javase/16/docs/api/java.base/java/util/function/Function.html)**<? super T, ? extends K> keyMapper,** [**Function**](https://docs.oracle.com/en/java/javase/16/docs/api/java.base/java/util/function/Function.html)**<? super T, ? extends U> valueMapper,** [**BinaryOperator**](https://docs.oracle.com/en/java/javase/16/docs/api/java.base/java/util/function/BinaryOperator.html)**<U> mergeFunction)**

Returns a Collector that accumulates the input elements into an [unmodifiable Map](https://docs.oracle.com/en/java/javase/16/docs/api/java.base/java/util/Map.html#unmodifiable), whose keys and values are the result of applying the provided mapping functions to the input elements.

If the mapped keys contain duplicates (according to [Object.equals(Object)](https://docs.oracle.com/en/java/javase/16/docs/api/java.base/java/lang/Object.html#equals(java.lang.Object))), the value mapping function is applied to each equal element, and the results are merged using the provided merging function.

The returned Collector disallows null keys and values. If either mapping function returns null, NullPointerException will be thrown.

**Type Parameters:**

T - the type of the input elements

K - the output type of the key mapping function

U - the output type of the value mapping function

**Parameters:**

keyMapper - a mapping function to produce keys, must be non-null

valueMapper - a mapping function to produce values, must be non-null

mergeFunction - a merge function, used to resolve collisions between values associated with the same key, as supplied to [Map.merge(Object, Object, BiFunction)](https://docs.oracle.com/en/java/javase/16/docs/api/java.base/java/util/Map.html#merge(K,V,java.util.function.BiFunction)), must be non-null

**Returns:** a Collector that accumulates the input elements into an [unmodifiable Map](https://docs.oracle.com/en/java/javase/16/docs/api/java.base/java/util/Map.html#unmodifiable), whose keys and values are the result of applying the provided mapping functions to the input elements

**Throws:** [NullPointerException](https://docs.oracle.com/en/java/javase/16/docs/api/java.base/java/lang/NullPointerException.html) - if the keyMapper, valueMapper, or mergeFunction is null

**Since:** 10

### **Teeing**

**public static <T, R1, R2, R>** [**Collector**](https://docs.oracle.com/en/java/javase/16/docs/api/java.base/java/util/stream/Collector.html)**<T, ?, R> teeing (**[**Collector**](https://docs.oracle.com/en/java/javase/16/docs/api/java.base/java/util/stream/Collector.html)**<? super T, ?, R1> downstream1,** [**Collector**](https://docs.oracle.com/en/java/javase/16/docs/api/java.base/java/util/stream/Collector.html)**<? super T, ?, R2> downstream2,** [**BiFunction**](https://docs.oracle.com/en/java/javase/16/docs/api/java.base/java/util/function/BiFunction.html)**<? super R1, ? super R2, R> merger)**

Returns a Collector that is a composite of two downstream collectors. Every element passed to the resulting collector is processed by both downstream collectors, then their results are merged using the specified merge function into the final result.

The resulting collector functions do the following:

* + supplier: creates a result container that contains result containers obtained by calling each collector's supplier
  + accumulator: calls each collector's accumulator with its result container and the input element
  + combiner: calls each collector's combiner with two result containers
  + finisher: calls each collector's finisher with its result container, then calls the supplied merger and returns its result.

The resulting collector is [Collector.Characteristics.UNORDERED](https://docs.oracle.com/en/java/javase/16/docs/api/java.base/java/util/stream/Collector.Characteristics.html#UNORDERED) if both downstream collectors are unordered and [Collector.Characteristics.CONCURRENT](https://docs.oracle.com/en/java/javase/16/docs/api/java.base/java/util/stream/Collector.Characteristics.html#CONCURRENT) if both downstream collectors are concurrent.

**Type Parameters:**

* + T - the type of the input elements
  + R1 - the result type of the first collector
  + R2 - the result type of the second collector
  + R - the final result type

**Parameters:**

* + downstream1 - the first downstream collector
  + downstream2 - the second downstream collector
  + merger - the function which merges two results into the single one

**Returns:** A Collector which aggregates the results of two supplied collectors.

**Since:** 12