

The Java Streams collect() Terminal Operation (Part 1)

Douglas C. Schmidt

d.schmidt@vanderbilt.edu

www.dre.vanderbilt.edu/~schmidt

Professor of Computer Science

**Institute for Software
Integrated Systems**

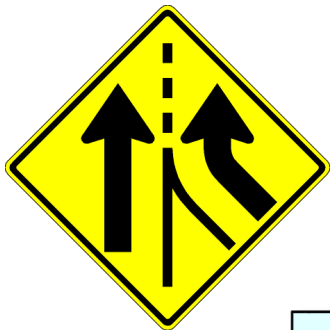
**Vanderbilt University
Nashville, Tennessee, USA**



Learning Objectives in this Part of the Lesson

- Understand common terminal operations, e.g.

- `forEach()`
- `collect()`
 - Know what a collector does in the context of `collect()`



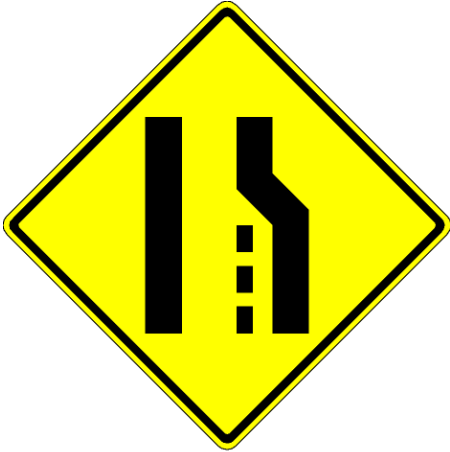
*We showcase `collect()`
using the Hamlet program*

```
void runCollect*() {  
    List<String> characters =  
        List.of("horatio",  
                "laertes",  
                "Hamlet", ...);  
    ...<String> results =  
        characters  
            .stream()  
            .filter(s ->  
                toLowerCase(...) == 'h')  
            .map(this::capitalize)  
            .sorted()  
            .collect(...); ...  
}
```

A Stream Terminal Operation That Returns Collections

A Stream Terminal Operation That Returns Collections

- A collector performs reduction operations



Interface `Collector<T,A,R>`

Type Parameters:

`T` - the type of input elements to the reduction operation

`A` - the mutable accumulation type of the reduction operation (often hidden as an implementation detail)

`R` - the result type of the reduction operation

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

A [mutable reduction operation](#) 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` provides implementations of many common mutable reductions.

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

A Stream Terminal Operation That Returns Collections

- A collector performs reduction operations, e.g.
- Summarizing elements according to various criteria



SUMMARY

```
public static <T> Collector<T,?,IntSummaryStatistics>  
    summarizingInt(ToIntFunction<? super T> mapper)
```

```
public static <T> Collector<T,?,IntSummaryStatistics>
```

Returns a Collector which applies an int-producing mapping function to each input element, and returns summary statistics for the resulting values.

Type Parameters:

T - the type of the input elements

Parameters:

mapper - a mapping function to apply to each element

Returns:

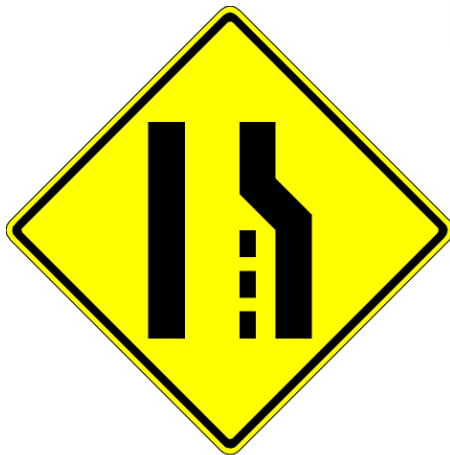
a Collector implementing the summary-statistics reduction

See Also:

```
summarizingDouble(ToDoubleFunction),  
summarizingLong(ToLongFunction)
```

A Stream Terminal Operation That Returns Collections

- A collector performs reduction operations, e.g.
 - Summarizing elements according to various criteria
 - Accumulating elements into various collections or single objects, etc.



Class Collectors

```
java.lang.Object  
    java.util.stream.Collectors
```

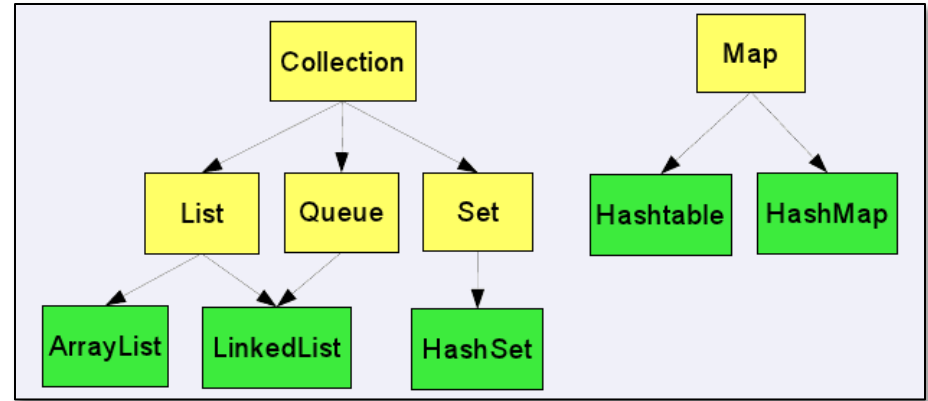
```
public final class Collectors  
    extends Object
```

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

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

A Stream Terminal Operation That Returns Collections

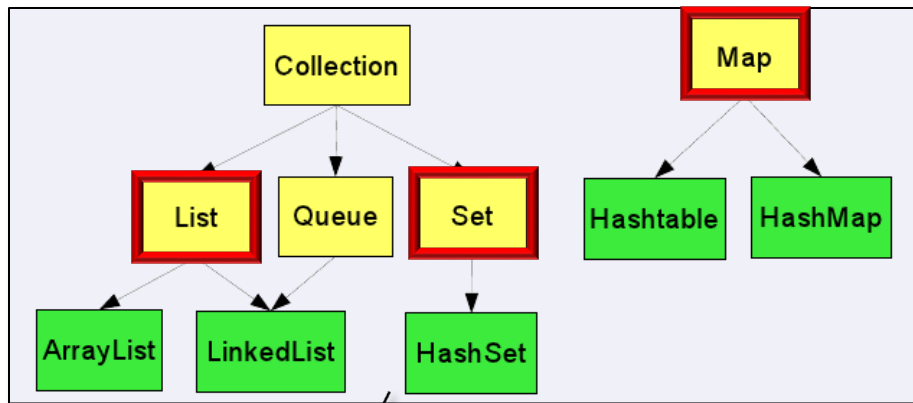
- The `collect()` terminal operation typically returns a collection



See www.concretepage.com/java/jdk-8/java-8-stream-collect-example

A Stream Terminal Operation That Returns Collections

- The `collect()` terminal operation typically returns a collection



We focus on the most common pre-defined collectors in this lesson

A Stream Terminal Operation That Returns Collections

- The collect() terminal operation typically returns a collection

This example demonstrates many variants of Collectors.

```
void runCollect*() {  
    List<String> characters =  
        List.of("horatio",  
                "laertes",  
                "Hamlet", ...);  
    ...<String> results =  
        characters  
            .stream()  
            .filter(s ->  
                toLowerCase(...) == 'h')  
            .map(this::capitalize)  
            .sorted()  
            .collect(...); ...  
}
```

A Stream Terminal Operation That Returns Collections

- The collect() terminal operation typically returns a collection

Create & process a stream consisting of characters from the play "Hamlet".

```
void runCollect*() {  
    List<String> characters =  
        List.of("horatio",  
                "laertes",  
                "Hamlet", ...);  
    ...<String> results =  
        characters  
            .stream()  
            .filter(s ->  
                toLowerCase(...) == 'h')  
            .map(this::capitalize)  
            .sorted()  
            .collect(...); ...  
}
```

A Stream Terminal Operation That Returns Collections

- The `collect()` terminal operation typically returns a collection



collect() performs a mutable reduction on all stream elements using some collector & returns a single collection.

```
void runCollect*() {  
    List<String> characters =  
        List.of("horatio",  
                "laertes",  
                "Hamlet", ...);  
    ...<String> results =  
        characters  
            .stream()  
            .filter(s ->  
                toLowerCase(...) == 'h')  
            .map(this::capitalize)  
            .sorted()  
            .collect(...); ...  
}
```

A Stream Terminal Operation That Returns Collections

- The collect() terminal operation typically returns a collection

```
void runCollectJoining() {  
    List<String> characters =  
        List.of("horatio",  
                "laertes",  
                "Hamlet", ...);  
  
    String results =  
        characters  
            .stream()  
            .filter(s ->  
                toLowerCase(...) == 'h')  
            .map(this::capitalize)  
            .sorted()  
            .collect(joining(" ")); ...  
}
```

*A collector can also
return a single object.*

End of the Java Streams collect() Terminal Operation (Part 1)