The Java Streams collect()
Terminal Operation

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()

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

See [github.com/douglascraigschmidt/LiveLessons/tree/master/Java8/ex12](https://github.com/douglascraigschmidt/LiveLessons/tree/master/Java8/ex12)
A Stream Terminal Operation That Returns Collections
void runCollectTo*() {
    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

```java
void runCollectTo*() {
    List<String> characters =
        List.of("horatio",
                "laertes",
                "Hamlet", ...);

    ...<String> results =
        characters
            .stream()
            .filter(s ->
                toLowerCase(...) =='h')
            .map(this::capitalize)
            .sorted()
            .collect(...); ...
}
```

Many variants of collect() are showcased in this example.

See [github.com/douglascairnschmidt/LiveLessons/tree/master/Java8/ex12](github.com/douglascairnschmidt/LiveLessons/tree/master/Java8/ex12)
A Stream Terminal Operation That Returns Collections

- The collect() terminal operation typically returns a collection

```java
void runCollectTo*() {
    List<String> characters = List.of("horatio",
                                        "laertes",
                                        "Hamlet", ...);

    ...<String> results = characters
                        .stream()
                        .filter(s -> s.toLowerCase(...) == 'h')
                        .map(this::capitalize)
                        .sorted()
                        .collect(...); ...
```

Create & process a stream consisting of characters from the play "Hamlet".
The collect() terminal operation typically returns a collection.

```java
go void runCollectTo*() {  
    List<String> characters =  
        List.of("horatio",  
               "laertes",  
               "Hamlet", ...);  

    ...<String> results =  
        characters  
            .stream()  
            .filter(s ->  
                    toLowerCase(...) == 'h')  
            .map(this::capitalize)  
            .sorted()  
            .collect(...);  ...
```

Performs a mutable reduction on all elements of this stream using some collector & returns a single result.

See docs.oracle.com/javase/8/docs/api/java/util/stream/Stream.html#collect
The collect() terminal operation typically returns a collection.

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

A collector performs reduction operations, e.g., summarizing elements according to various criteria, accumulating elements into various types of collections, etc.
The collect() terminal operation typically returns a collection

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

Collect results into a ArrayList, which can contain duplicates.

See docs.oracle.com/javase/8/docs/api/java/util/stream/Collectors.html#toList
The `collect()` terminal operation typically returns a collection.

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

`collect()` is much less error-prone than `forEach()` since initialization is implicit & it’s thread-safe.

See earlier lesson on “Java Streams: the `forEach()` Terminal Operation'
The collect() terminal operation typically returns a collection.

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

Collect the results into a HashSet, which can contain no duplicates.

See docs.oracle.com/javase/8/docs/api/java/util/stream/Collectors.html#toSet
A Stream Terminal Operation That Returns Collections

- The collect() terminal operation typically returns a collection

```java
toMap()
```

```java
toMap()
```

```java
List<String> characters = List.of("horatio", "laertes", "Hamlet", ...);
Map<String, Integer> results = characters
    .stream()
    .filter(s -> s.toLowerCase() == 'h')
    .map(this::capitalize)
    .collect(toMap(identity(), String::length, Integer::sum));
```

Collect results into a HashMap, along with the length of (merged duplicate) entries.

See docs.oracle.com/javase/8/docs/api/java/util/stream/Collectors.html#toMap
The collect() terminal operation typically returns a collection

```java
collectGroupingBy()
```

```java
List<String> characters = List.of("horatio", "laertes", "Hamlet", ...);
Map<String, Long> results =
... .collect
  (groupingBy
   (identity(),
    TreeMap::new,
    summingLong
     (String::length)));
... Collect the results into a TreeMap by grouping elements according to
      name (key) & name length (value).
```
The `collect()` terminal operation typically returns a collection.

```java
void runCollectGroupingBy() {
    List<String> characters = List.of("horatio",
                                         "laertes",
                                         "Hamlet", ...);
    Map<String, Long> results = ...
        .collect
            (groupingBy
                (identity(),
                 TreeMap::new,
                 summingLong
                    (String::length)));
    ...
}
```

`groupingBy()` partitions a stream via a “classifier” function (identity() always returns its input argument).

See [docs.oracle.com/javase/8/docs/api/java/util/function/Function.html#identity](http://docs.oracle.com/javase/8/docs/api/java/util/function/Function.html#identity)
The `collect()` terminal operation typically returns a collection.

```java
class void runCollectGroupingBy() {  
List<String> characters = 
  List.of("horatio",  
    "laertes",  
    "Hamlet", ...);  
Map<String, Long> results = 
  ...  
  .collect  
    (groupingBy  
      (identity(),  
        TreeMap::new,  
        summingLong  
          (String::length)))  
  ...  
```

A constructor reference is used to create a `TreeMap`.

See [docs.oracle.com/javase/8/docs/api/java/util/TreeMap.html](http://docs.oracle.com/javase/8/docs/api/java/util/TreeMap.html)
A Stream Terminal Operation That Returns Collections

The collect() terminal operation typically returns a collection.

```java
void runCollectGroupingBy() {
    List<String> characters =
        List.of("horatio",
                "laertes",
                "Hamlet", ...);
    Map<String, Long> results =
        ...;
    .collect
        (groupingBy
            (identity(),
             TreeMap::new,
             summingLong
                (String::length)));
    ...
}
```

This “downstream collector” defines a `summingLong()` collector that’s applied to the results of the classifier function.

See [www.baeldung.com/java-groupingby-collector](http://www.baeldung.com/java-groupingby-collector)
A Stream Terminal Operation That Returns Collections

- The collect() terminal operation typically returns a collection

```java
void runCollectReduce() {
    Map<String, Long> matchingCharactersMap = Pattern.compile(",")
        .splitAsStream("horatio,Hamlet,...")
    ...
    .collect(
        groupingBy
        (identity(),
        TreeMap::new,
        summingLong
        (String::length)));
```

Convert a string into a stream via regular expression splitting!

See [docs.oracle.com/javase/8/docs/api/java/util/regex/Pattern.html#splitAsStream](docs.oracle.com/javase/8/docs/api/java/util/regex/Pattern.html#splitAsStream)
A Stream Terminal Operation That Returns Collections

• The collect() terminal operation typically returns a collection

```java
void runCollectReduce() {
    Map<String, Long> matchingCharactersMap = 
        Pattern.compile("","").splitAsStream("horatio,Hamlet,...")
        .collect(
            groupingBy
                (identity(),
                TreeMap::new,
                summingLong
                    (String::length)));
}```

Collect the results into a TreeMap by grouping elements according to name (key) & name length (value).

See [docs.oracle.com/javase/8/docs/api/java/util/stream/Collectors.html#groupingBy](https://docs.oracle.com/javase/8/docs/api/java/util/stream/Collectors.html#groupingBy)
End of the Java Streams collect() Terminal Operation