The Java Streams collect() Terminal Operation (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
We showcase `collect()` using the Hamlet program

Learning Objectives in this Part of the Lesson

- Understand common terminal operations, e.g.
  - `forEach()`
  - `collect()`

- Know what a collector does

- Recognize common Java pre-defined collectors & know how to use them in conjunction with `collect()`

```java
void runCollect*() {
  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 Summary of Pre-defined Collectors
A Summary of Pre-defined Collectors

- 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 ->
            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
A Summary of Pre-defined Collectors

- 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 -> 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](https://docs.oracle.com/javase/8/docs/api/java/util/stream/Collectors.html#toSet)
void runCollectToMap() {
    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
A Summary of Pre-defined Collectors

- 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)));
    ...
}
```

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](http://docs.oracle.com/javase/8/docs/api/java/util/stream/Collectors.html#groupingBy)
A Summary of Pre-defined Collectors

- 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](docs.oracle.com/javase/8/docs/api/java/util/function/Function.html#identity)
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)));
```
• 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 Summary of Pre-defined Collectors

- 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](https://docs.oracle.com/javase/8/docs/api/java/util/regex/Pattern.html#splitAsStream)
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
End of the Java Streams collect() Terminal Operation (Part 2)