Java Streams: the 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()

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

We showcase collect() using the Hamlet program

See github.com/douglascraigschmidt/LiveLessons/tree/master/Java8/ex12

A Stream Terminal Operation That Returns Collections

A Stream Terminal Operation That Returns Collections

- The collect() terminal operation typically returns a collection

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

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

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

```
void runCollectTo*() {
  List<String> characters =
    Arrays.asList("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

A Stream Terminal Operation That Returns Collections

- The collect() terminal operation typically returns a collection

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

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

A Stream Terminal Operation That Returns Collections

- The collect() terminal operation typically returns a collection

```
void runCollectTo*() {
  List<String> characters =
    Arrays.asList("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 type of collector

See docs.oracle.com/javase/8/docs/api/java/util/stream/Stream.html#collect

A Stream Terminal Operation That Returns Collections

- The collect() terminal operation typically returns a collection

```
void runCollectTo*() {
  List<String> characters =
    Arrays.asList("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.

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

```
void runCollectToList() {
  List<String> characters =
    Arrays.asList("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

A Stream Terminal Operation That Returns Collections

- The collect() terminal operation typically returns a collection

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

Using collect() is less error-prone than forEach() since initialization is implicit & it's inherently thread-safe

See earlier lessons on "Java Streams: the forEach() Terminal Operation

A Stream Terminal Operation That Returns Collections

- The collect() terminal operation typically returns a collection

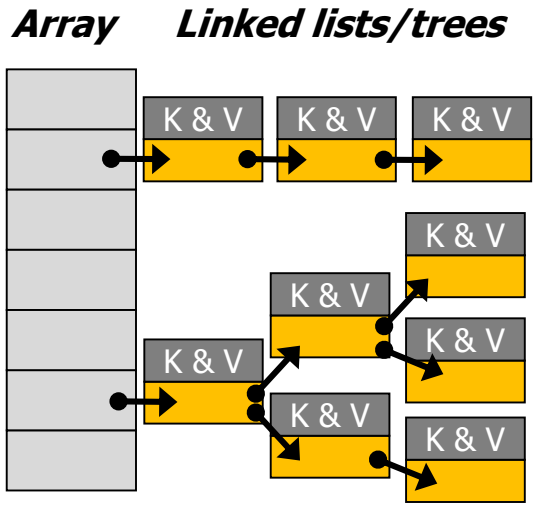
```
void runCollectToSet() {
  List<String> characters =
    Arrays.asList("horatio",
                  "laertes",
                  "Hamlet", ...);
  Set<String> results =
    characters
      .stream()
      .filter(s ->
        toLowerCase(…) =='h')
      .map(this::capitalize)
      .collect(toSet());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

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



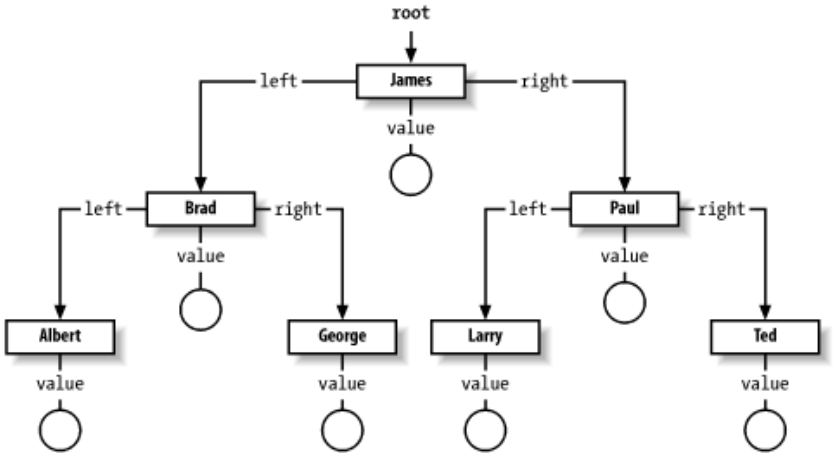

```
void runCollectToMap() {
  List<String> characters =
    Arrays.asList("horatio",
                  "laertes",
                  "Hamlet", ...);
Map<String, Integer> results =
  characters
    .stream()
    .filter(s ->
      toLowerCase(…) =='h')
    .map(this::capitalize)
    .collect(toMap(identity(),
                   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 Stream Terminal Operation That Returns Collections

- The collect() terminal operation typically returns a collection



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

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

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

A Stream Terminal Operation That Returns Collections

- The collect() terminal operation typically returns a collection

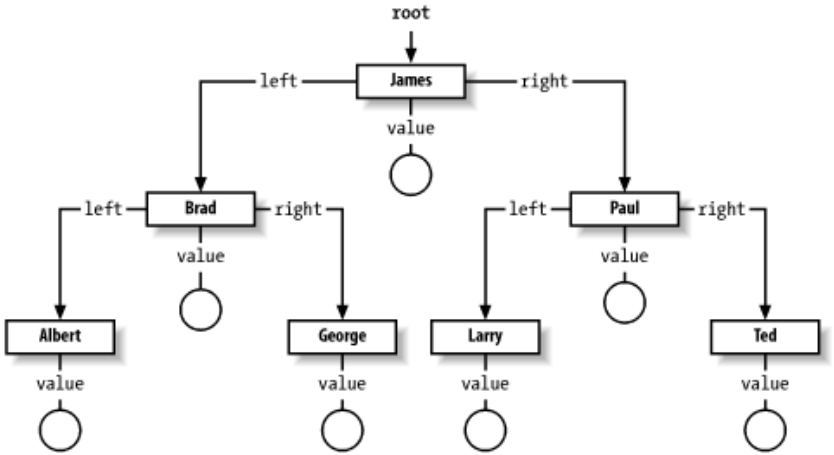
groupingBy() partitions a stream via a "classifier" function (identity() always returns its input argument).

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

See docs.oracle.com/javase/8/docs/api/java/util/function/Function.html#identity

A Stream Terminal Operation That Returns Collections

- The collect() terminal operation typically returns a collection




```
void runCollectGroupingBy() {
  List<String> characters =
    Arrays.asList("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

A Stream Terminal Operation That Returns Collections

- The collect() terminal operation typically returns a collection

```
void runCollectGroupingBy() {
  List<String> characters =
    Arrays.asList("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

A Stream Terminal Operation That Returns Collections

- The collect() terminal operation typically returns a collection

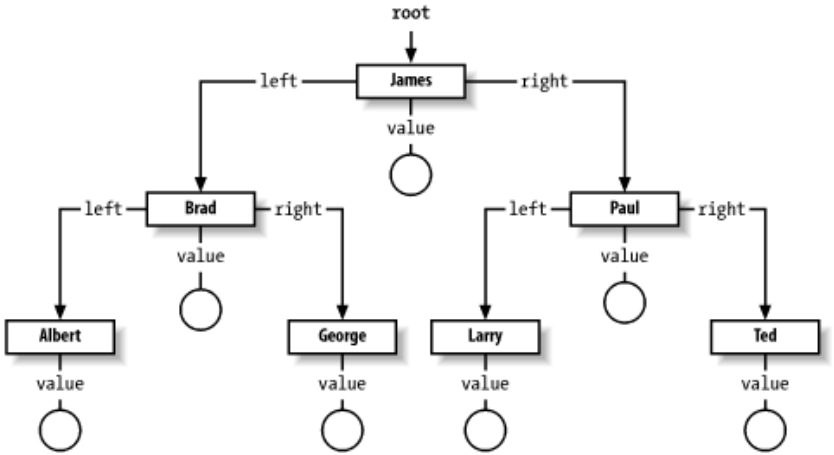
Convert a string into a stream via regular expression splitting!

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

See 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




```
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 Java Streams: the collect() Terminal Operation