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

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 & how to use them with collect()
      - These collectors were introduced in Java 8

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.

The following are examples of using the predefined collectors to perform common mutable reduction tasks:

```
// Accumulate names into a List
List<String> list = people.stream().map(Person::getName).collect(Collectors.toList());

// Accumulate names into a TreeSet
Set<String> set = people.stream().map(Person::getName).collect(Collectors.toCollection(TreeSet::new));

// Convert elements to strings and concatenate them, separated by commas
String joined = things.stream()
                      .map(Object::toString)
                      .collect(Collectors.joining(", "));

// Compute sum of salaries of employee
int total = employees.stream()
                     .collect(Collectors.summingInt(Employee::getSalary)));

// Group employees by department
Map<Department, List<Employee>> byDept
    = employees.stream()
               .collect(Collectors.groupingBy(Employee::getDepartment));

// Compute sum of salaries by department
Map<Department, Integer> totalByDept
    = employees.stream()
               .collect(Collectors.groupingBy(Employee::getDepartment,
                                              Collectors.summingInt(Employee::getSalary)));

// Partition students into passing and failing
Map<Boolean, List<Student>> passingFailing =
    students.stream()
            .collect(Collectors.partitioningBy(s -> s.getGrade() >= PASS_THRESHOLD));
```

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

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 & how to use them with collect()
      - These collectors were introduced in Java 8

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

We showcase collect() & these collectors via the Hamlet program

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

Pre-defined Collectors That Return Collections

Pre-defined Collectors That Return Collections

- The collect() terminal operation typically returns a collection

```
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

Pre-defined Collectors That Return Collections

- The collect() terminal operation typically returns a collection

```
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*"

Pre-defined Collectors That Return Collections

- The collect() terminal operation typically returns a collection

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

Java 16 adds the toList() terminal operator that returns an immutable List

See www.javacodegeeks.com/2020/12/jdk-16-stream-to-list-in-one-easy-call.html

Pre-defined Collectors That Return Collections

- The collect() terminal operation typically returns a collection

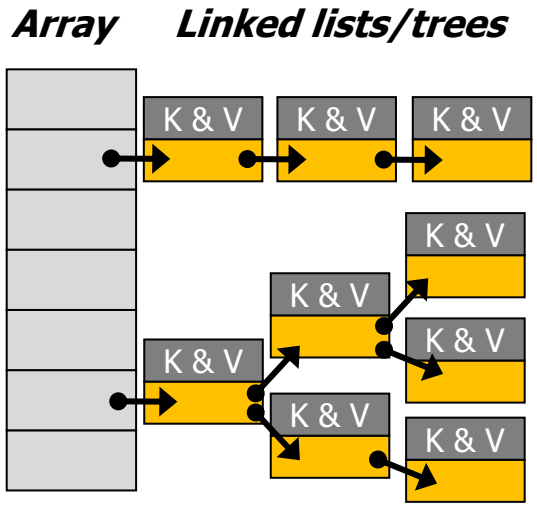
```
void runCollectToSet() {
  List<String> characters =
    List.of("horatio",
            "laertes",
            "Hamlet", ...);
 Set<String> results =
    characters
       .stream()
       .filter(s ->
         toLowerCase(…) =='h')
       .map(this::capitalize)
       .collect(toSet()); ...
```

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

Pre-defined Collectors That Return Collections

- The collect() terminal operation typically returns a collection



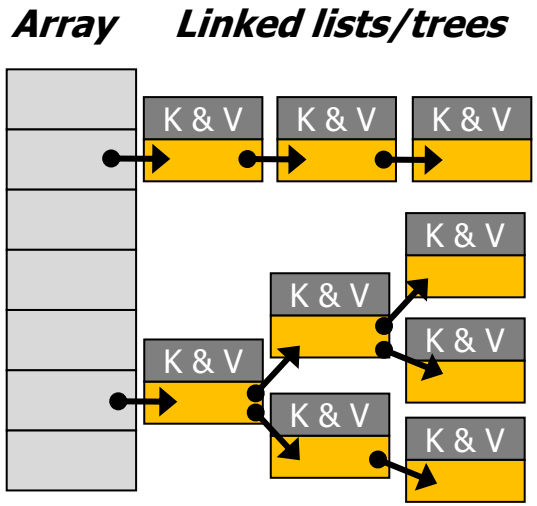

```
void runCollectToMap() {
  List<String> characters =
    List.of("horatio",
            "laertes",
            "Hamlet", ...);
  Map<String, Integer> results =
    characters
      .stream()
      .filter(s ->
        toLowerCase(…) =='h')
      .map(this::capitalize)
      .collect(toMap(identity(),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

Pre-defined Collectors That Return Collections

- The collect() terminal operation typically returns a collection



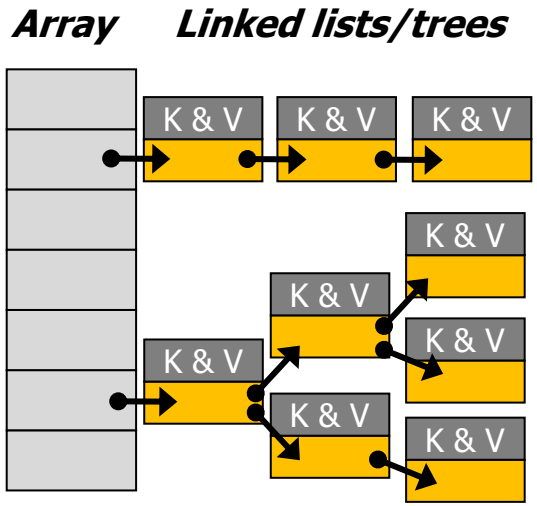

```
void runCollectToMap() {
  List<String> characters =
    List.of("horatio",
            "laertes",
            "Hamlet", ...);
  Map<String, Integer> results =
    characters
      .stream()
      .filter(s ->
        toLowerCase(…) =='h')
      .map(this::capitalize)
      .collect(toMap(identity(),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

This mapping Function produces keys

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

Pre-defined Collectors That Return Collections

- The collect() terminal operation typically returns a collection



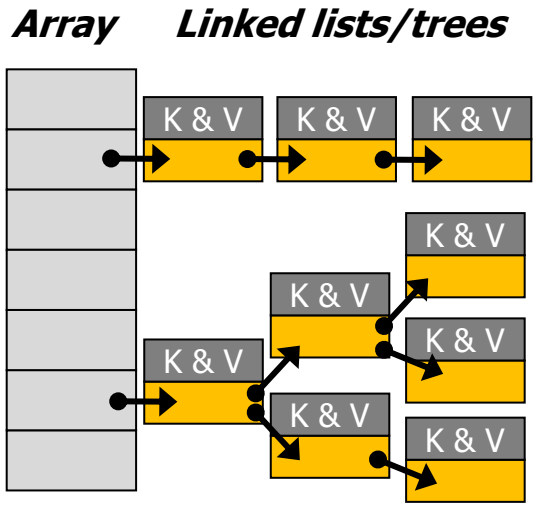

```
void runCollectToMap() {
  List<String> characters =
    List.of("horatio",
            "laertes",
            "Hamlet", ...);
  Map<String, Integer> results =
    characters
      .stream()
      .filter(s ->
        toLowerCase(…) =='h')
      .map(this::capitalize)
      .collect(toMap(identity(),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

This mapping Function produces values

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

Pre-defined Collectors That Return Collections

- The collect() terminal operation typically returns a collection



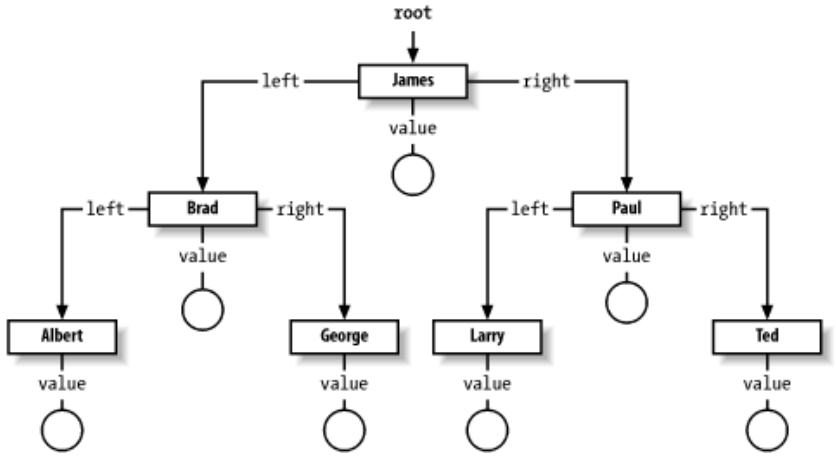

```
void runCollectToMap() {
  List<String> characters =
    List.of("horatio",
            "laertes",
            "Hamlet", ...);
  Map<String, Integer> results =
    characters
      .stream()
      .filter(s ->
        toLowerCase(…) =='h')
      .map(this::capitalize)
      .collect(toMap(identity(),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

This merge function resolves collisions between values that are associated with the same key

See docs.oracle.com/javase/8/docs/api/java/util/function/BinaryOperator.html

Pre-defined Collectors That Return 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 =
    List.of("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

Pre-defined Collectors That Return Collections

- The collect() terminal operation typically returns a collection

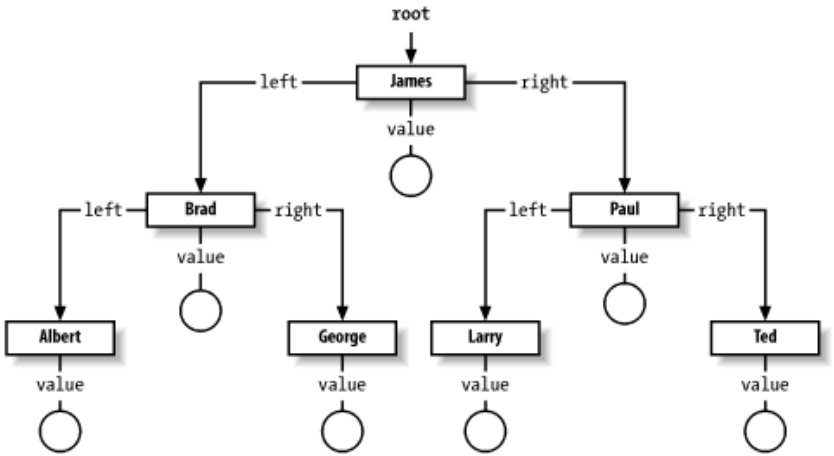
groupingBy() partitions a stream via a "classifier" function (identity() always returns its input argument).

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

Pre-defined Collectors That Return Collections

- The collect() terminal operation typically returns a collection




```
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

Pre-defined Collectors That Return Collections

- The collect() terminal operation typically returns a collection

```
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

Pre-defined Collectors That Return 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)));
```

Convert a string into a stream via regular expression splitting!

See docs.oracle.com/javase/8/docs/api/java/util/regex/Pattern.html#splitAsStream

End of the Java Streams collect() Terminal Operation (Part 2)