Java Parallel Streams Internals:
Order of Results for Collections

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 parallel stream internals, e.g.
  - Know what can change & what can’t
    - Splitting, combining, & pooling mechanisms
  - Order of processing
  - Order of results
    - Overview
    - Collections that affect results order
Learning Objectives in this Part of the Lesson

- Understand parallel stream internals, e.g.
- Know what can change & what can’t
  - Splitting, combining, & pooling mechanisms
  - Order of processing
  - Order of results
  - Overview
  - Collections that affect results order

List<Integer> list = Arrays.asList(1, 2, ...);

Integer[] doubledList = list
  .parallelStream()
  .filter(x -> x % 2 == 0)
  .map(x -> x * 2)
  .toArray(Integer[]::new);

Multiple examples are analyzed in detail

See github.com/douglascraigschmidt/LiveLessons/tree/master/Java8/ex21
Collections that Affect Results Order
Collections that Affect Results Order

- Encounter order is maintained by
  - Ordered spliterators
  - Ordered collections
  - Static stream factory methods

```java
List<Integer> list = Arrays.asList(2, 3, 1, 4, 2);
Integer[] doubledList = list
    .parallelStream()
    .filter(x -> x % 2 == 0)
    .map(x -> x * 2)
    .toArray(Integer[]::new);
```

Collections that Affect Results Order

- Encounter order is maintained by
  - Ordered spliterators
  - Ordered collections
  - Static stream factory methods

List<Integer> list = Arrays.asList(2, 3, 1, 4, 2);

The encounter order is [2, 3, 1, 4, 2] since list is ordered & non-unique

Integer[] doubledList = list
  .parallelStream()
  .filter(x -> x % 2 == 0)
  .map(x -> x * 2)
  .toArray(Integer[]::new);

Recall that “ordered” isn’t the same as “sorted”!

Collections that Affect Results Order

- Encounter order is maintained by
  - Ordered spliterators
  - Ordered collections
  - Static stream factory methods

```java
List<Integer> list = Arrays.asList(2, 3, 1, 4, 2);

Integer[] doubledList = list.parallelStream()
    .filter(x -> x % 2 == 0)
    .map(x -> x * 2)
    .toArray(Integer[]::new);
```

*Only even values continue thru stream*
Collections that Affect Results Order

- Encounter order is maintained by
  - Ordered spliterators
  - Ordered collections
  - Static stream factory methods

```java
List<Integer> list = Arrays.asList(2, 3, 1, 4, 2);

Integer[] doubledList = list
    .parallelStream()
    .filter(x -> x % 2 == 0)
    .map(x -> x * 2)
    .toArray(Integer[]::new);
```

*Multiply each even number by 2*
Collections that Affect Results Order

- Encounter order is maintained by
  - Ordered spliterators
  - Ordered collections
  - Static stream factory methods

List<Integer> list = Arrays.asList(2, 3, 1, 4, 2);

Integer[] doubledList = list
  .parallelStream()
  .filter(x -> x % 2 == 0)
  .map(x -> x * 2)
  .toArray(Integer[]::new);

Convert stream into an array of integers
Collections that Affect Results Order

- Encounter order is maintained by
  - Ordered spliterators
  - Ordered collections
  - Static stream factory methods

List<Integer> list = Arrays.asList(2, 3, 1, 4, 2);

Integer[] doubledList = list
  .parallelStream()
  .filter(x -> x % 2 == 0)
  .map(x -> x * 2)
  .toArray(Integer[]::new);

*Result must be ordered as [4, 8, 4] since the list is an ordered collection*
Unordered collections don’t need to respect encounter order

Set<Integer> set = new HashSet<>(Arrays.asList(2, 3, 1, 4, 2))

Integer[] doubledSet = set
    .parallelStream()
    .filter(x -> x % 2 == 0)
    .map(x -> x * 2)
    .toArray(Integer[]::new);
Collections that Affect Results Order

- Unordered collections don’t need to respect encounter order

```
Set<Integer> set = new HashSet<>(Arrays.asList(2, 3, 1, 4, 2));
```

```
Integer[] doubledSet = set
    .parallelStream()
    .filter(x -> x % 2 == 0)
    .map(x -> x * 2)
    .toArray(Integer[]::new);
```

A HashSet is unordered & unique
Unordered collections don’t need to respect encounter order.

Set<Integer> set = new HashSet<>(Arrays.asList(2, 3, 1, 4, 2));

This code runs faster since encounter order need not be maintained in the results, which could be [8, 4] or [4, 8].

Integer[] doubledSet = set.parallelStream()
  .filter(x -> x % 2 == 0)
  .map(x -> x * 2)
  .toArray(Integer[]::new);
End of Java Parallel Streams Internals: Order of Results for Collections