Java Parallel Streams Internals: Demo'ing Spliterator Performance

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
  - Partition a data source into "chunks"
    - Know the impact of different Java collections on the performance of different spliterators

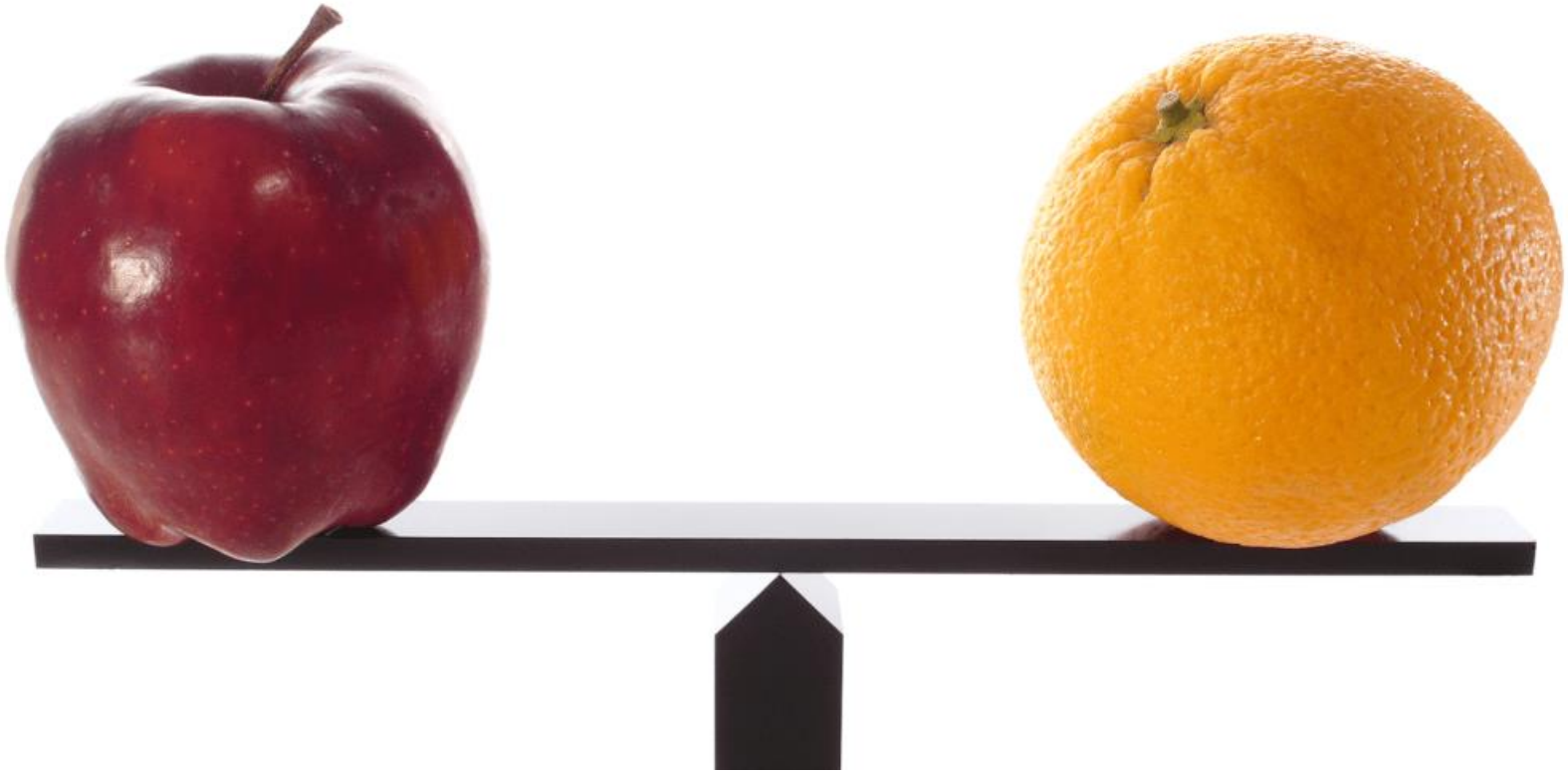
```
Starting spliterator tests for 1000 words..
..printing results
    17 msecs: ArrayList parallel
    19 msecs: LinkedList parallel

Starting spliterator tests for 10000 words..
..printing results
    88 msecs: LinkedList parallel
    90 msecs: ArrayList parallel

Starting spliterator tests for 100000 words..
..printing results
   599 msecs: ArrayList parallel
   701 msecs: LinkedList parallel

Starting spliterator tests for 883311 words..
..printing results
  5718 msecs: ArrayList parallel
 31226 msecs: LinkedList parallel
```

See github.com/douglascraigschmidt/LiveLessons/tree/master/Java8/ex14

Demonstrating Spliterator Performance

Demonstrating Spliterator Performance

- Spliterators for ArrayList & LinkedList partition data quite differently



See earlier lesson on "*Java Parallel Streams Internals: Partitioning*"

Demonstrating Spliterator Performance

- Spliterators for ArrayList & LinkedList partition data quite differently

```
ArrayListSpliterator<E> trySplit() {
  int hi = getFence(), lo = index, mid = (lo + hi) >>> 1;
  // divide range in half unless too small
  return lo >= mid ? null : new ArrayListSpliterator<E>
                                  (list, lo, index = mid, ...);
}
```

ArrayList's spliterator splits evenly & efficiently (e.g., doesn't copy data)

See openjdk/8u40-b25/java/util/ArrayList.java

Demonstrating Spliterator Performance

- Spliterators for ArrayList & LinkedList partition data quite differently

```
Spliterator<E> trySplit() { ...
  int n = batch + BATCH_UNIT, j = 0; Object[] a = new Object[n];
  do { a[j++] = p.item; }
  while ((p = p.next) != null && j < n); ...
  return Spliterators.spliterator(a, 0, j, Spliterator.ORDERED);
}
```

LinkedList's spliterator does not split evenly & efficiently (e.g., it copies data)

See openjdk/8u40-b25/java/util/LinkedList.java

Demonstrating Spliterator Performance

- This demo program shows the performance difference of parallel spliterators for ArrayList & LinkedList when processing the complete works of Shakepeare

```
void timeParallelStreamUppercase(String testName,
                                 List<CharSequence> words) {
  ...
  List<String> list = new ArrayList<>();

  for (int i = 0; i < sMAX_ITERATIONS; i++)
    list
      .addAll(words
                .parallelStream()
                .map(charSeq ->
                     charSeq.toString().toUpperCase())
                .collect(toList())); ...
```

Demonstrating Spliterator Performance

- This demo program shows the performance difference of parallel spliterators for ArrayList & LinkedList when processing the complete works of Shakepeare

```
void timeParallelStreamUppercase(String testName,
                                 List<CharSequence> words) {
  ...
  List<String> list = new ArrayList<>();

  for (int i = 0; i < sMAX_ITERATIONS; i++)
    list
      .addAll(words
                .parallelStream()
                .map(charSeq ->
                     charSeq.toString().toUpperCase())
                .collect(toList())); ...
```

The words param is passed an ArrayList & a LinkedList

Demonstrating Spliterator Performance

- This demo program shows the performance difference of parallel spliterators for ArrayList & LinkedList when processing the complete works of Shakepeare

```
void timeParallelStreamUppercase(String testName,
                                 List<CharSequence> words) {
  ...
  List<String> list = new ArrayList<>();

  for (int i = 0; i < sMAX_ITERATIONS; i++)
    list
      .addAll(words
                .parallelStream()
                .map(charSeq ->
                     charSeq.toString().toUpperCase())
                .collect(toList())); ...
```

Split & uppercase the word list via a parallel spliterator

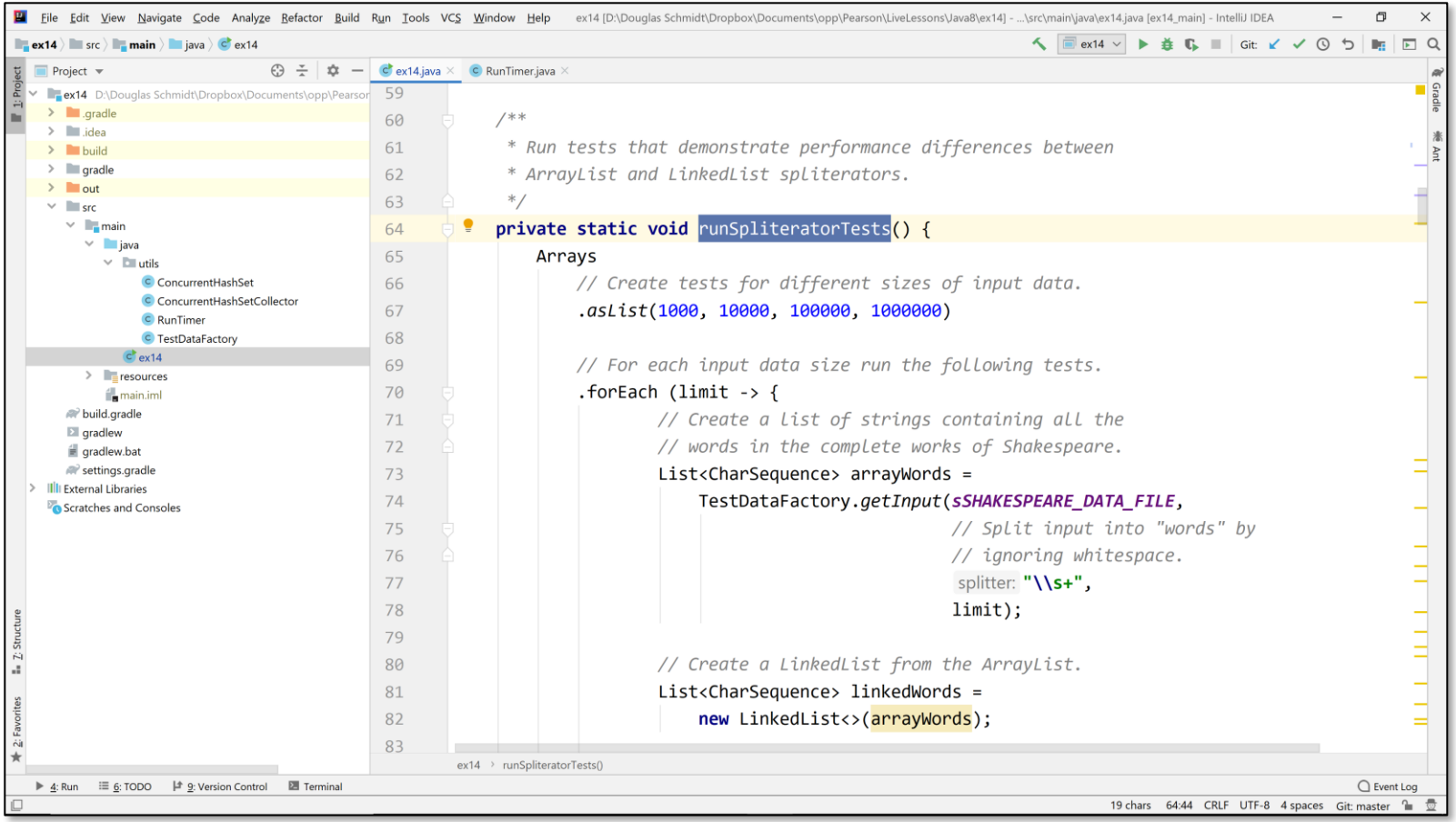
Demonstrating Spliterator Performance

- Results show spliterator differences become more significant as input grows

```
Starting spliterator tests for 1000 words....printing results
    17 msecs: ArrayList parallel
    19 msecs: LinkedList parallel

Starting spliterator tests for 10000 words....printing results
    88 msecs: ArrayList parallel
    90 msecs: LinkedList parallel

Starting spliterator tests for 100000 words....printing results
   599 msecs: ArrayList parallel
   701 msecs: LinkedList parallel

Starting spliterator tests for 883311 words....printing results
  5718 msecs: ArrayList parallel
 31226 msecs: LinkedList parallel
```

See upcoming lessons on "*When [Not] to Use Parallel Streams*"

Demonstrating Spliterator Performance



See github.com/douglascraigschmidt/LiveLessons/tree/master/Java8/ex14

End of Java Parallel Streams Internals: Demo'ing Spliterator Performance