Java SearchWithParallelStreams Example: Implementing Hook Methods

Douglas C. Schmidt
d.schmidt@vanderbilt.edu
www.dre.vanderbilt.edu/~schmidt

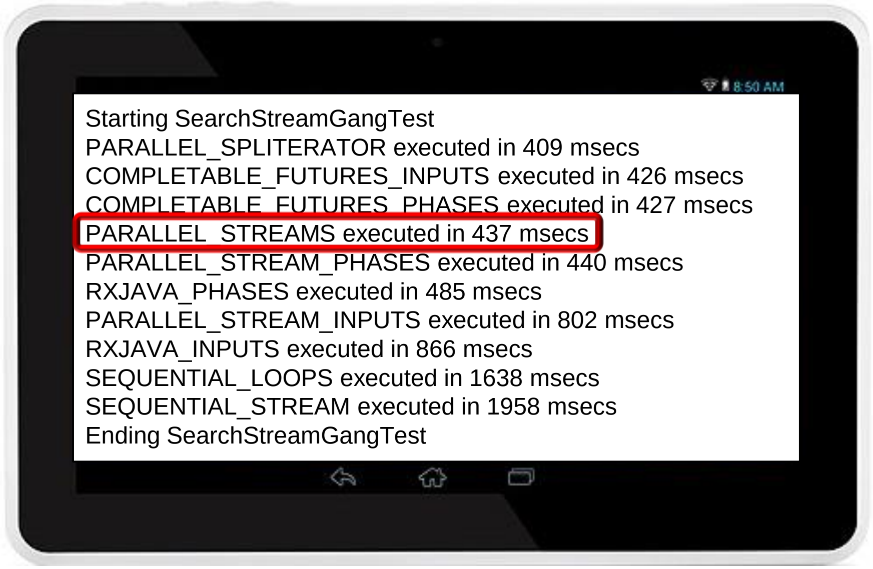
Professor of Computer Science

Institute for Software Integrated Systems

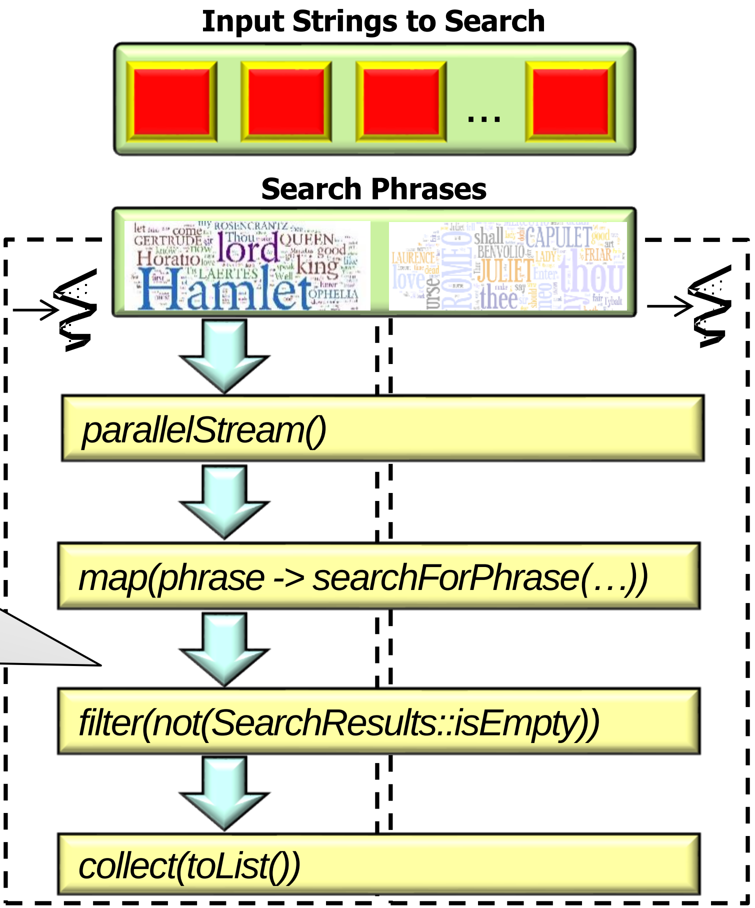
Vanderbilt University
Nashville, Tennessee, USA

Learning Objectives in this Part of the Lesson

- Know how Java parallel streams are applied in SearchWithParallelStreams



```
Starting SearchStreamGangTest
PARALLEL_SPLITERATOR executed in 409 msecs
COMPLETABLE_FUTURES_INPUTS executed in 426 msecs
COMPLETABLE_FUTURES_PHASES executed in 427 msecs
PARALLEL_STREAMS executed in 437 msecs
PARALLEL_STREAM_PHASES executed in 440 msecs
RXJAVA_PHASES executed in 485 msecs
PARALLEL_STREAM_INPUTS executed in 802 msecs
RXJAVA_INPUTS executed in 866 msecs
SEQUENTIAL_LOOPS executed in 1638 msecs
SEQUENTIAL_STREAM executed in 1958 msecs
Ending SearchStreamGangTest
```



See github.com/douglascraigschmidt/LiveLessons/tree/master/SearchStreamGang

Implementing processStream() as a Parallel Stream

Implementing processStream() as a Parallel Stream

- Parallel processStream() has one minuscule change wrt the sequential version

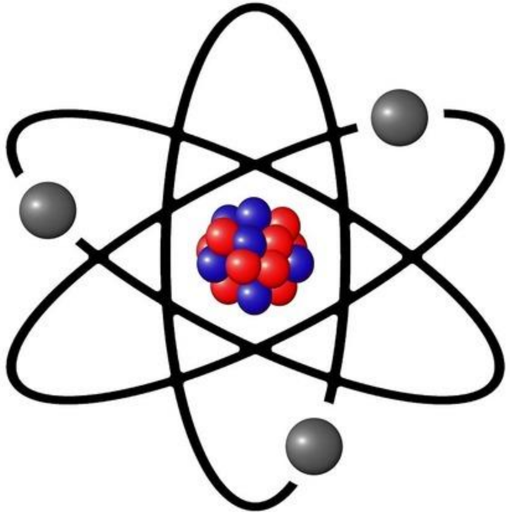
```
protected List<List<SearchResults>> processStream() {
  List<CharSequence> inputList =
    getInput();

  return inputList

    .parallelStream()

    .map(this::processInput)

    .collect(toList());
}
```



Implementing processStream() as a Parallel Stream

- Parallel processStream() has one minuscule change wrt the sequential version

```
protected List<List<SearchResults>> processStream() {
  List<CharSequence> inputList =
    getInput();

  return inputList

    .parallelStream()

    .map(this::processInput)

    .collect(toList());
}
```

Uses the ArrayList spliterator to create a parallel stream that searches an arraylist of input strings in multiple worker threads

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

Implementing processStream() as a Parallel Stream

- Parallel processStream() has one minuscule change wrt the sequential version

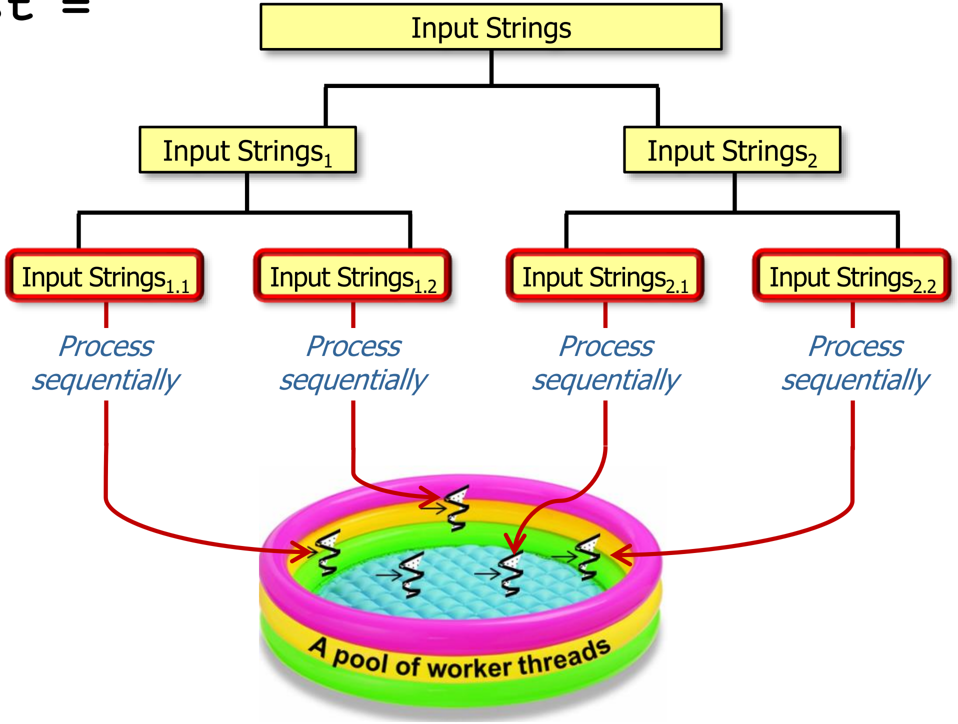
```
protected List<List<SearchResults>> processStream() {
  List<CharSequence> inputList =
    getInput();

  return inputList

    .parallelStream()

    .map(this::processInput)

    .collect(toList());
}
```



Each input string is processed in parallel using the common fork-join pool

Implementing processStream() as a Parallel Stream

- Parallel processStream() has one minuscule change wrt the sequential version

```
protected List<List<SearchResults>> processStream() {
  List<CharSequence> inputList =
    getInput();

  return inputList

    .parallelStream()

    .map(this::processInput)

    .collect(toList());
}
```

Searches a given input string to locate all occurrences of phases

Implementing processStream() as a Parallel Stream

- Parallel processStream() has one minuscule change wrt the sequential version

```
protected List<List<SearchResults>> processStream() {
  List<CharSequence> inputList =
    getInput();

  return inputList

    .parallelStream()

    .map(this::processInput)

    .collect(toList());
}
```

Trigger intermediate operation processing & merge partial results into a single list of lists

Collectors.toList() returns a non-concurrent collector that obeys encounter order

Implementing processInput() as a Parallel Stream

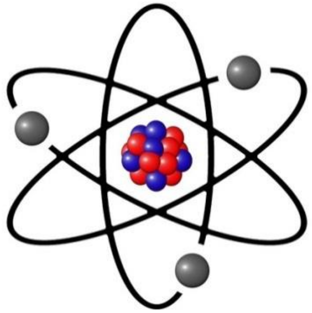
Implementing processInput() as a Parallel Stream

- Likewise, this processInput() implementation has just one minuscule change

```
List<SearchResults> processInput(CharSequence inputSeq) {
  String title = getTitle(inputSeq);
  CharSequence input = inputSeq.subSequence(...);

  List<SearchResults> results = mPhrasesToFind
    .parallelStream()
    .map(phase ->
           searchForPhrase(phase, input, title, false))
    .filter(not(SearchResults::isEmpty))

    .collect(toList());
  return results;
}
```



Implementing processInput() as a Parallel Stream

- Likewise, this processInput() implementation has just one minuscule change

```
List<SearchResults> processInput(CharSequence inputSeq) {
  String title = getTitle(inputSeq);
  CharSequence input = inputSeq.subSequence(...);

  List<SearchResults> results = mPhrasesToFind
    .parallelStream()
    .map(phase ->
         searchForPhrase(phase, input, title,
    .filter(not(SearchResults::isEmpty))

    .collect(toList());
  return results;
}
```

Uses ArrayList spliterator to create a parallel stream that searches an input string to locate all phase occurrences

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

Implementing processInput() as a Parallel Stream

- Likewise, this processInput() implementation has just one minuscule change

```
List<SearchResults> processInput(CharSequence inputSeq) {
  String title = getTitle(inputSeq);
  CharSequence input = inputSeq.subSequence(...);

  List<SearchResults> results = mPhrasesToFind
    .parallelStream()
    .map(phase ->
         searchForPhrase(phase, input, title, false))
    .filter(not(SearchResults::isEmpty))

    .collect(toList());
  return results;
}
```

The PhraseMatchSpliterator breaks the input into "chunks" that are processed sequentially

See SearchStreamGang/src/main/java/livelessons/utils/PhraseMatchSpliterator.java

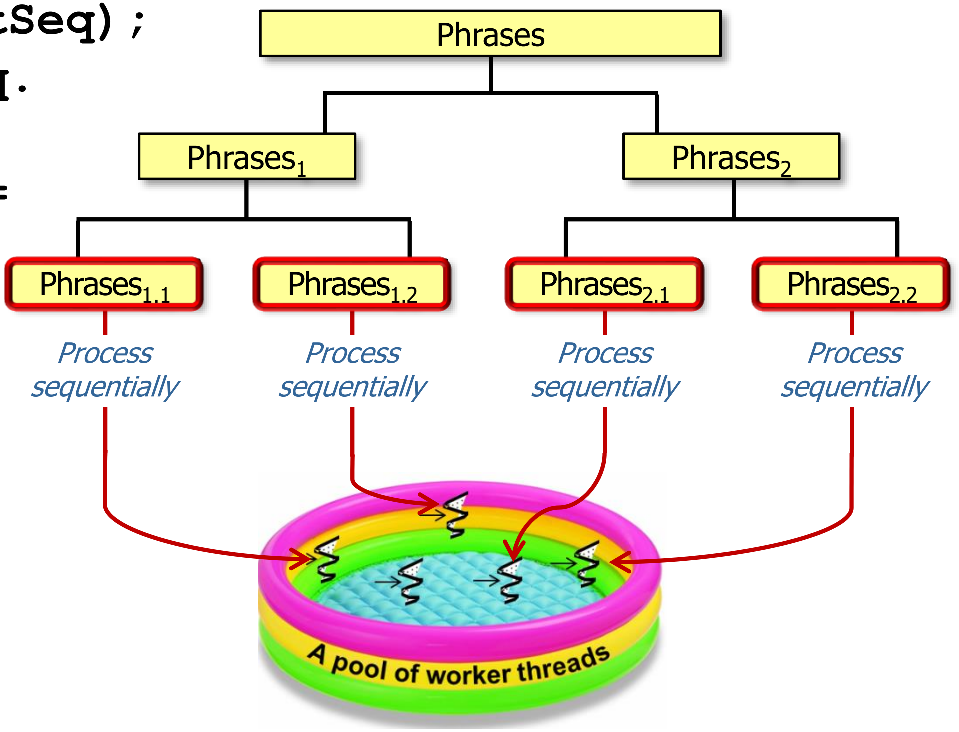
Implementing processInput() as a Parallel Stream

- Likewise, this processInput() implementation has just one minuscule change

```
List<SearchResults> processInput(CharSequence inputSeq) {
  String title = getTitle(inputSeq);
  CharSequence input = inputSeq.

  List<SearchResults> results =
    .parallelStream()
    .map(phase ->
          searchForPhrase(phase,
    .filter(not(SearchResults::

    .collect(toList());
  return results;
}
```



Each phrase (& each input string) is processed in parallel in the common fork-join pool

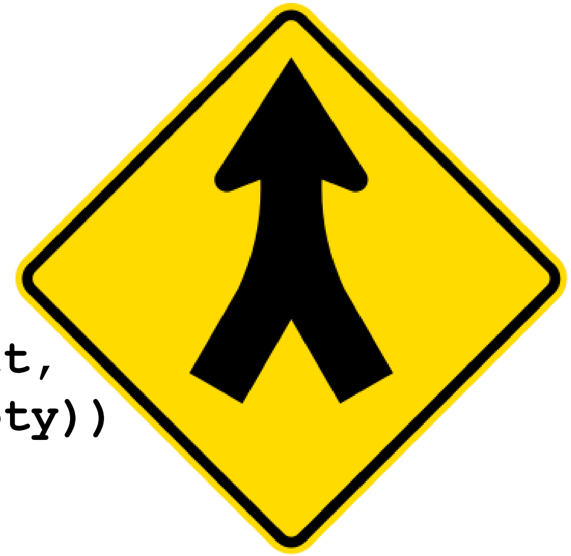
Implementing processInput() as a Parallel Stream

- Likewise, this processInput() implementation has just one minuscule change

```
List<SearchResults> processInput(CharSequence inputSeq) {
  String title = getTitle(inputSeq);
  CharSequence input = inputSeq.

  List<SearchResults> results =
    .parallelStream()
    .map(phase ->
          searchForPhrase(phase, input,
    .filter(not(SearchResults::isEmpty))

    .collect(toList());
  return results;
}
```



Trigger intermediate operation processing & merge partial results into a single list

Implementing processInput() as a Parallel Stream

- Likewise, this processInput() implementation has just one minuscule change

```
List<SearchResults> processInput(CharSequence inputSeq) {
  String title = getTitle(inputSeq);
  CharSequence input = inputSeq.

  List<SearchResults> results =
    .parallelStream()
    .map(phase ->
         searchForPhrase(phase, input, title, false))
    .filter(not(SearchResults::isEmpty))

    .collect(toList());
  return results;
}
```

Return the list of search results

End of Java SearchWith ParallelStreams Example: Implementing Hook Methods