Java 8 Sequential SearchStreamGang Example (Part 1)

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

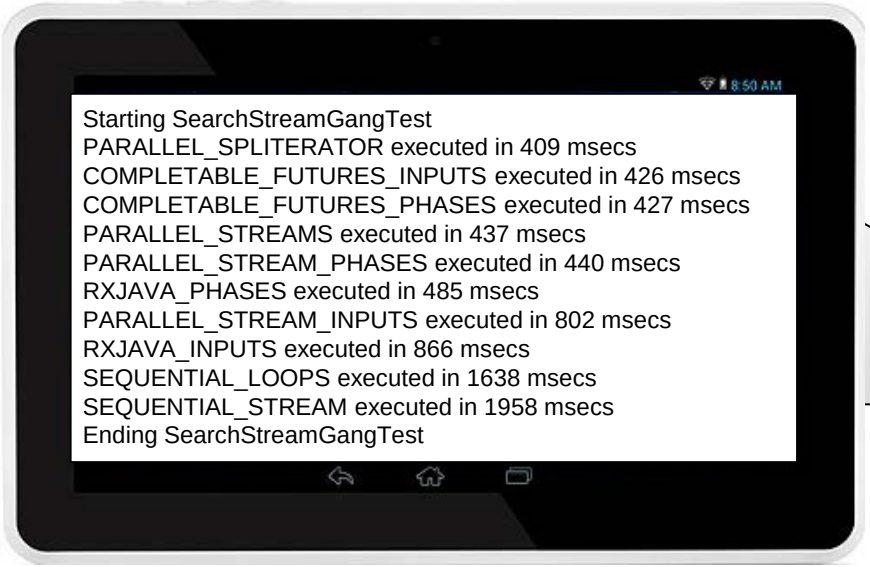
Professor of Computer Science

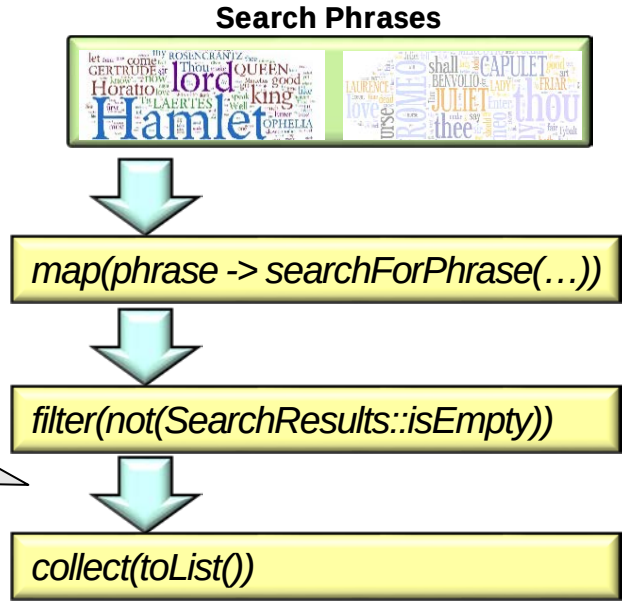
Institute for Software Integrated Systems

Vanderbilt University
Nashville, Tennessee, USA

Learning Objectives in this Part of the Lesson

- Know how to apply sequential streams to the SearchStreamGang program

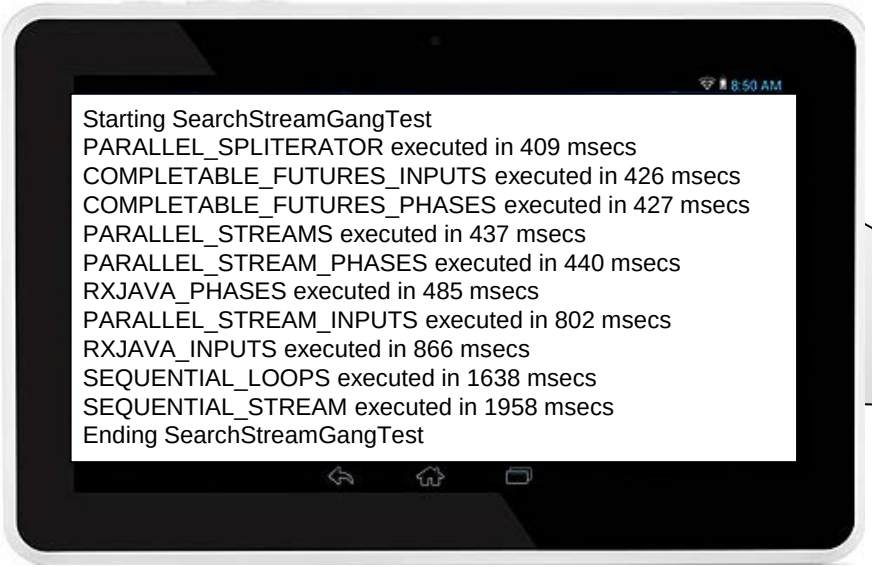


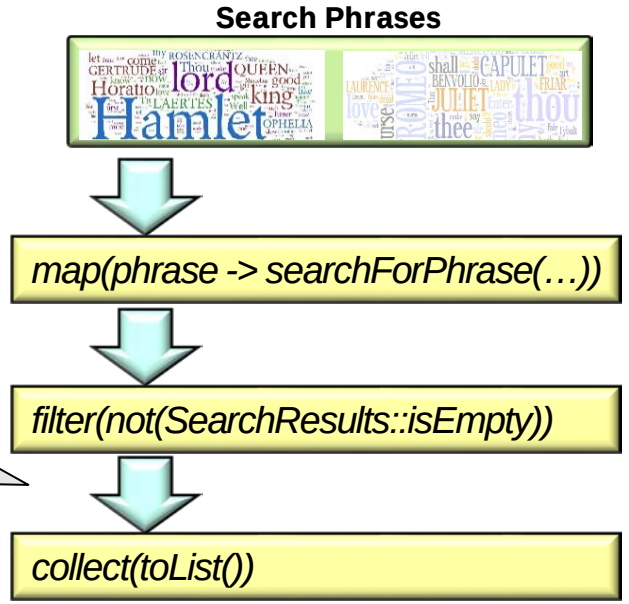


See github.com/douglascraigschmidt/LiveLessons/tree/master/SearchStreamGang

Learning Objectives in this Part of the Lesson

- Know how to apply sequential streams to the SearchStreamGang program
  - Understand the SearchStreamGang processStream() & processInput() methods





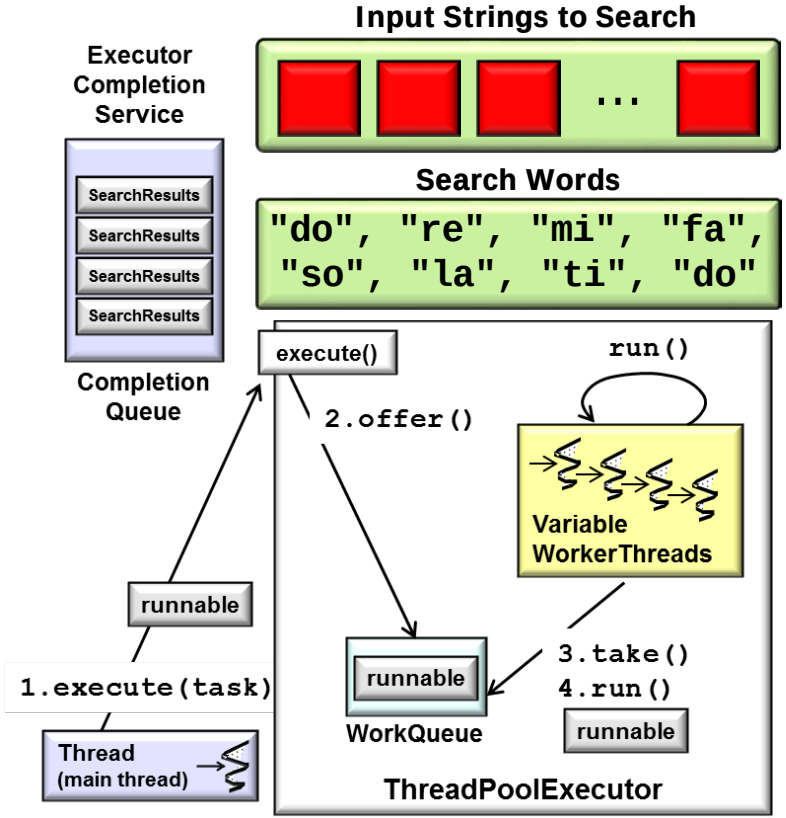
Learning Objectives in this Part of the Lesson

- Know how to apply sequential streams to the SearchStreamGang program
  - Understand the SearchStreamGang processStream() & processInput() methods
  - This program is more interesting than the SimpleSearchStream program

Overview of SearchStreamGang

Overview of SearchStreamGang

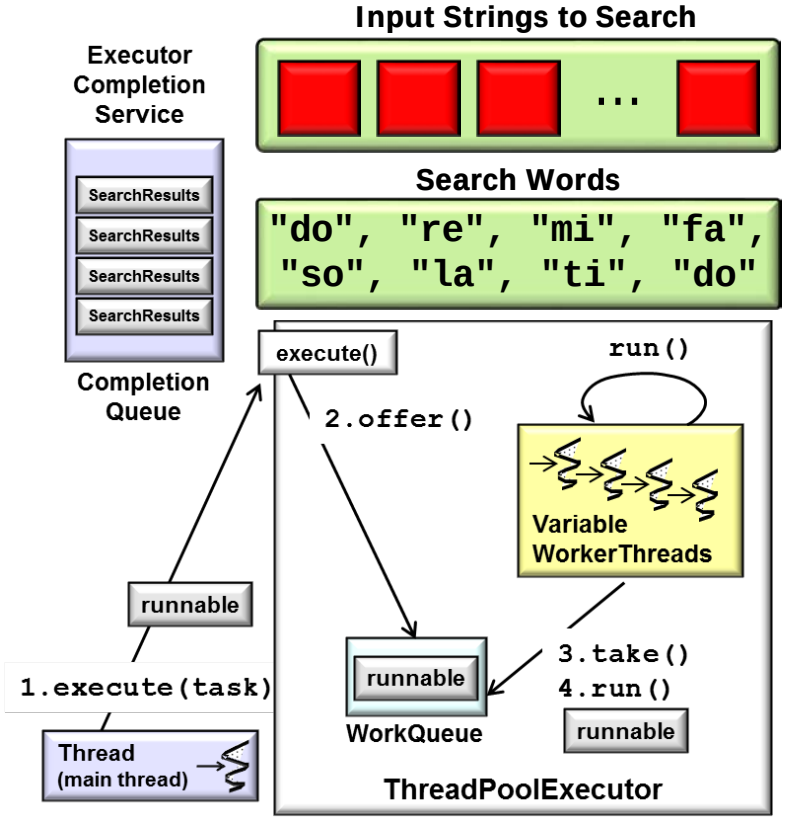
- SearchStreamGang is a Java 8 revision of SearchTaskGang



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

Overview of SearchStreamGang

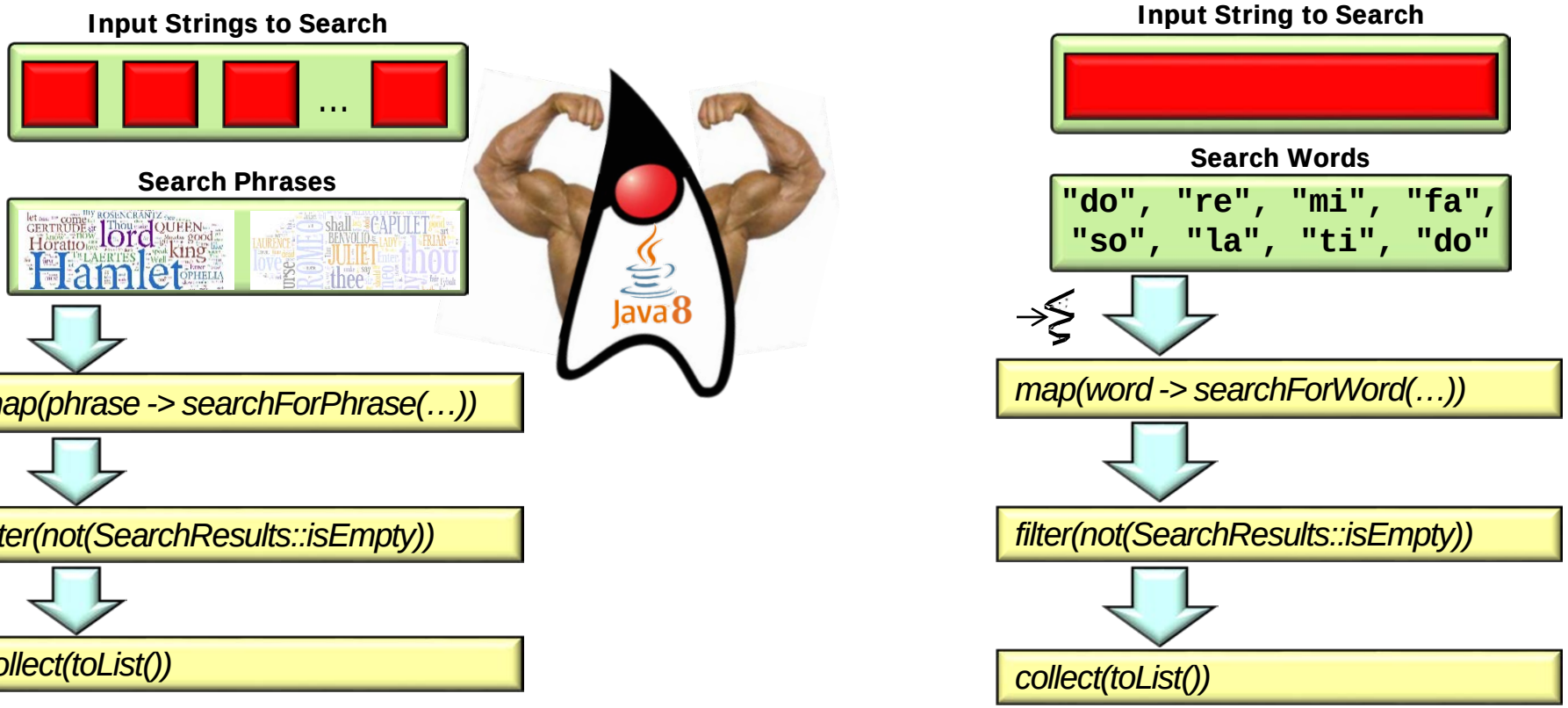
- SearchStreamGang is a Java 8 revision of SearchTaskGang
  - SearchTaskGang showcases the Java executor framework for tasks that are “embarrassingly parallel”



e.g., Executor, Executor Service, Executor Completion Service

Overview of SearchStreamGang

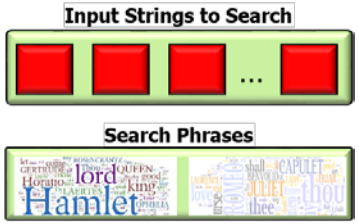
- SearchStreamGang is a more powerful revision of SimpleSearchStream



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

Overview of SearchStreamGang

- SearchStreamGang is a more powerful revision of SimpleSearchStream, e.g.
  - It uses regular expressions to find phrases in works of Shakespeare



The Complete Works of William Shakespeare

Welcome to the Web's first edition of the Complete Works of William Shakespeare. This site has offered Shakespeare's plays and poetry to the Internet community since 1993.

For other Shakespeare resources, visit the Mr. William Shakespeare and the Internet Web site.

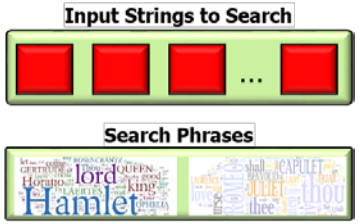
The original electronic source for this server was the Complete Moby(tm) Shakespeare. The HTML versions of the plays provided here are placed in the public domain.

Older news items

See shakespeare.mit.edu

Overview of SearchStreamGang

- SearchStreamGang is a more powerful revision of SimpleSearchStream, e.g.
  - It uses regular expressions to find phrases in works of Shakespeare



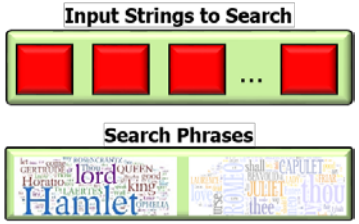
"...
My liege, and madam, to expostulate
What majesty should be, what duty is,
Why day is day, night is night, and time is time.
Were nothing but to waste night, day, and time.
Therefore, since brevity is the soul of wit,
And tediousness the limbs and outward flourishes,
I will be brief. ..."

"Brevity is the soul of wit"

A phrase can match anywhere within a line

Overview of SearchStreamGang

- SearchStreamGang is a more powerful revision of SimpleSearchStream, e.g.
  - It uses regular expressions to find phrases in works of Shakespeare



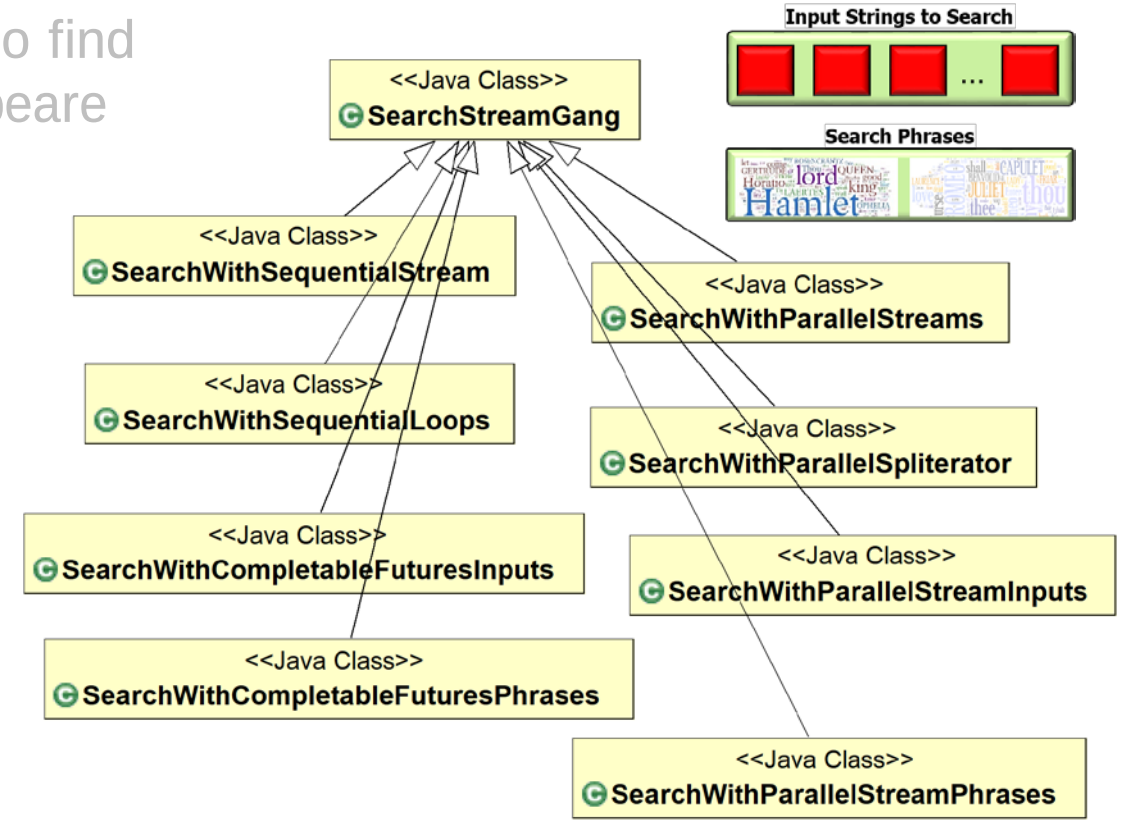
"...
What's in a name? That which we call a rose
By any other name would smell as sweet.
So Romeo would, were he not Romeo call'd,
Retain that dear perfection which he owes
Without that title. ..."

"What's in a name? That which we call a rose
By any other name would smell as sweet."

The phrases can also match across multiple lines

Overview of SearchStreamGang

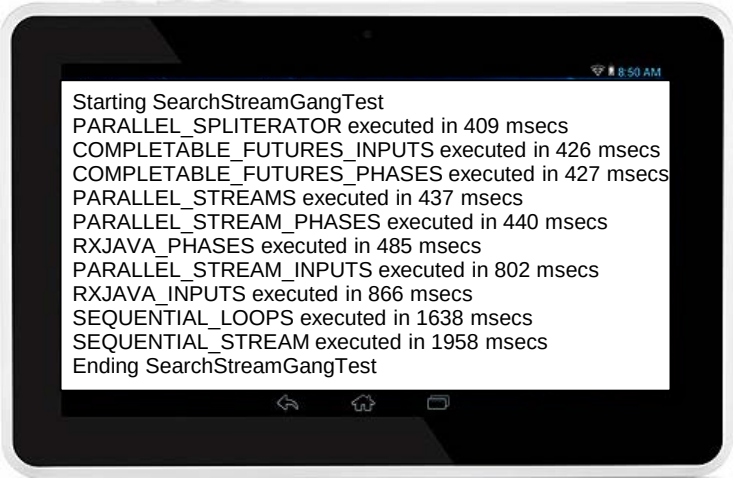
- SearchStreamGang is a more powerful revision of SimpleSearchStream, e.g.
  - It uses regular expressions to find phrases in works of Shakespeare
  - It defines a framework for Java 8 concurrency & parallelism strategies

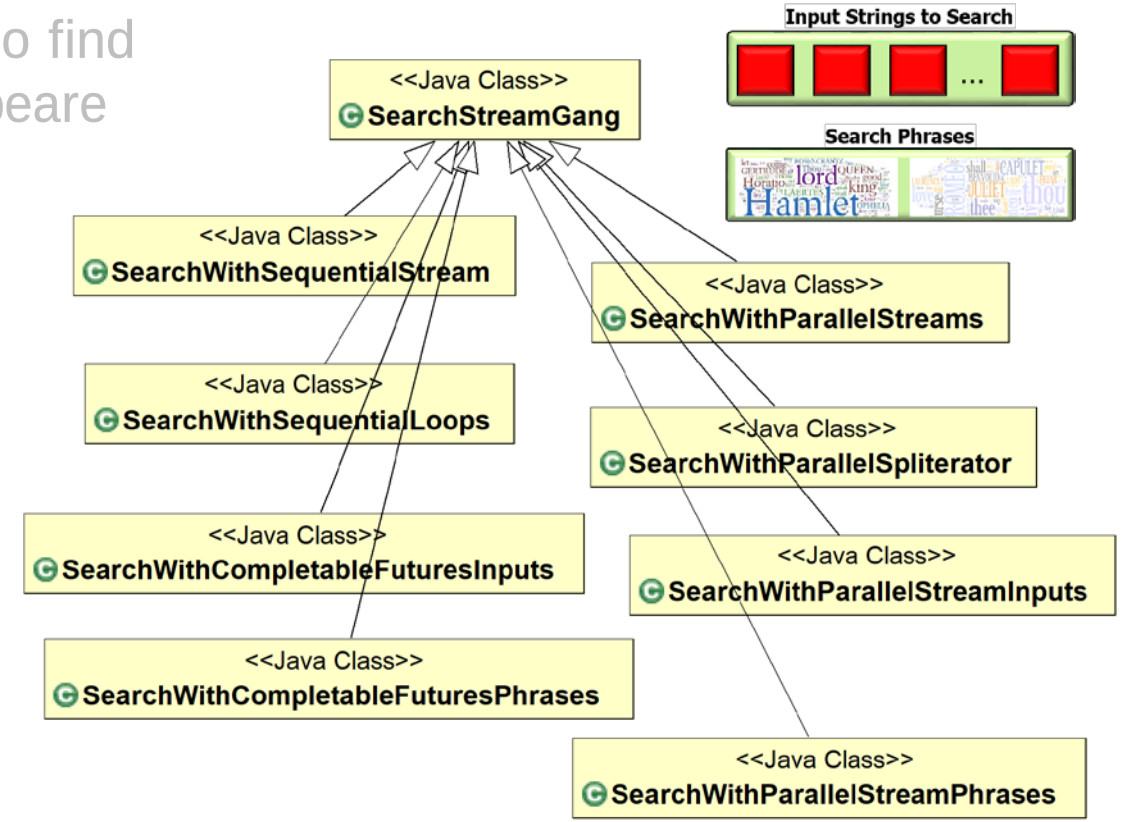


e.g., parallel streams, parallel spliterator, & completable futures

Overview of SearchStreamGang

- SearchStreamGang is a more powerful revision of SimpleSearchStream, e.g.
  - It uses regular expressions to find phrases in works of Shakespeare
  - It defines a framework for Java 8 concurrency & parallelism strategies

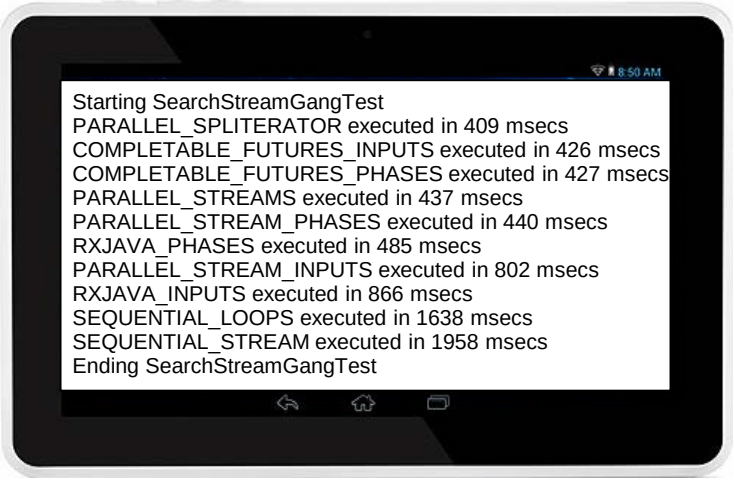


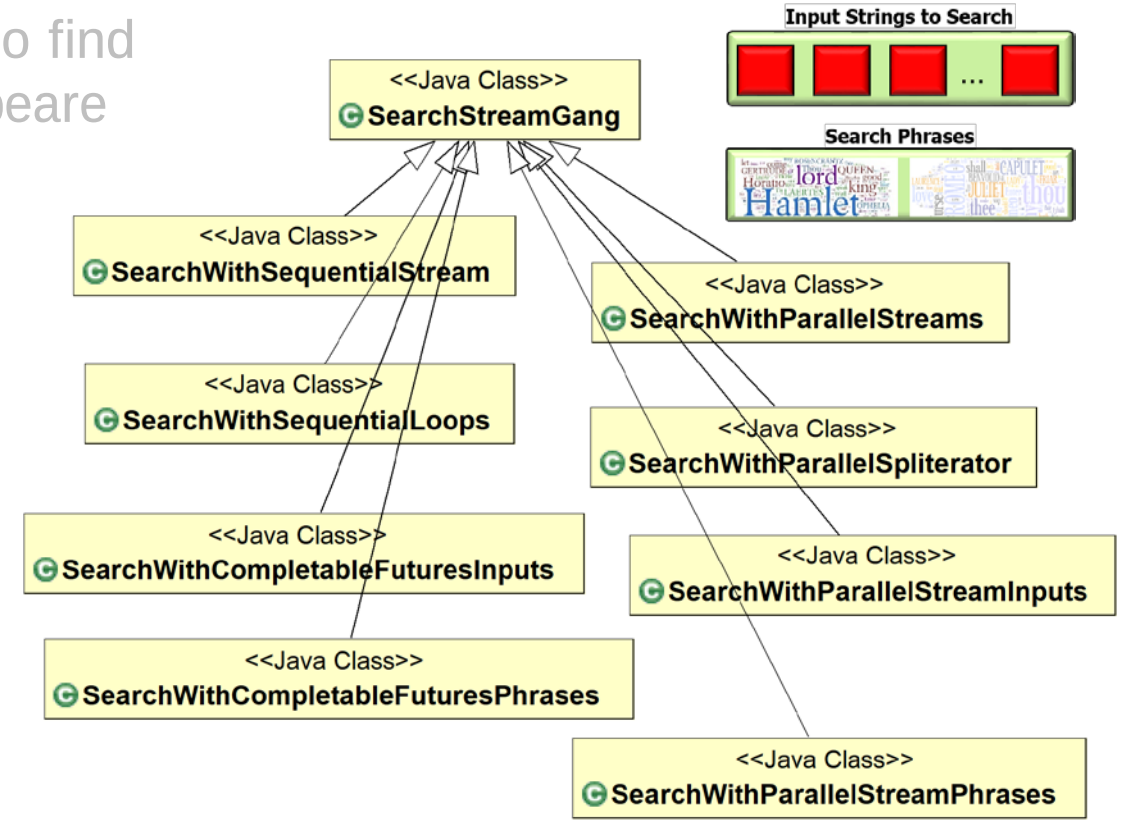


This framework enables “apples-to-apples” performance comparisons

Overview of SearchStreamGang

- SearchStreamGang is a more powerful revision of SimpleSearchStream, e.g.
  - It uses regular expressions to find phrases in works of Shakespeare
  - It defines a framework for Java 8 concurrency & parallelism strategies



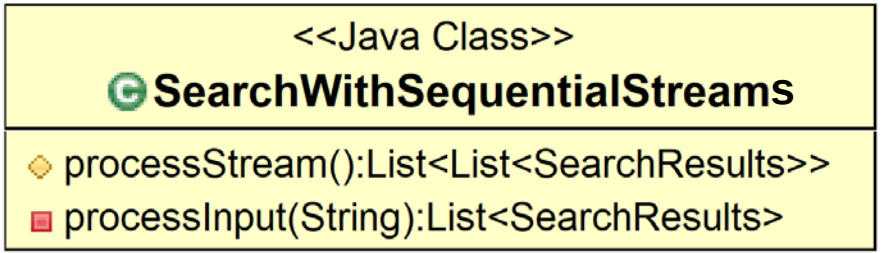


We'll cover Java 8 concurrency/parallel strategies after covering sequential streams

Visualizing processStream() & processInput()

Visualizing processStream() & processInput()

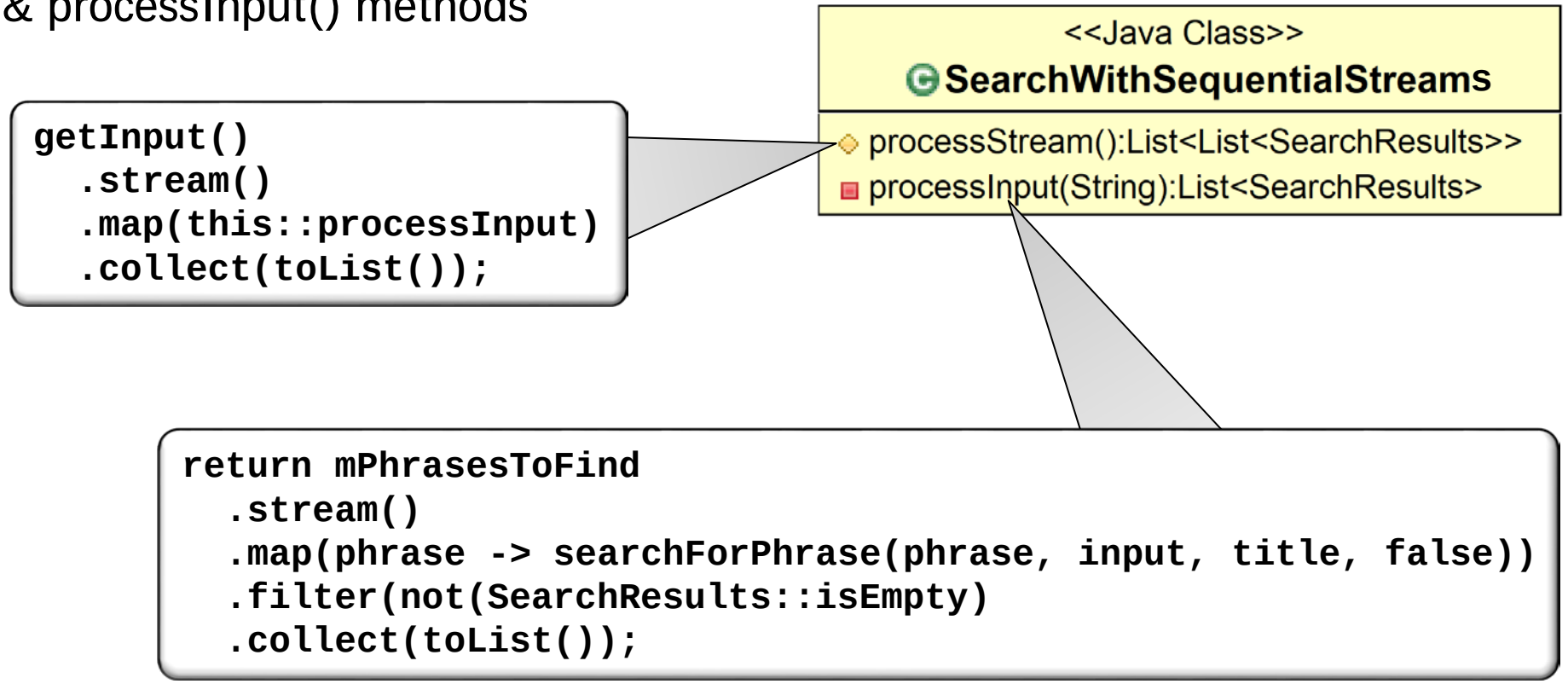
- We show aggregate operations in the SearchStreamGang's processStream() & processInput() methods



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

Visualizing processStream() & processInput()

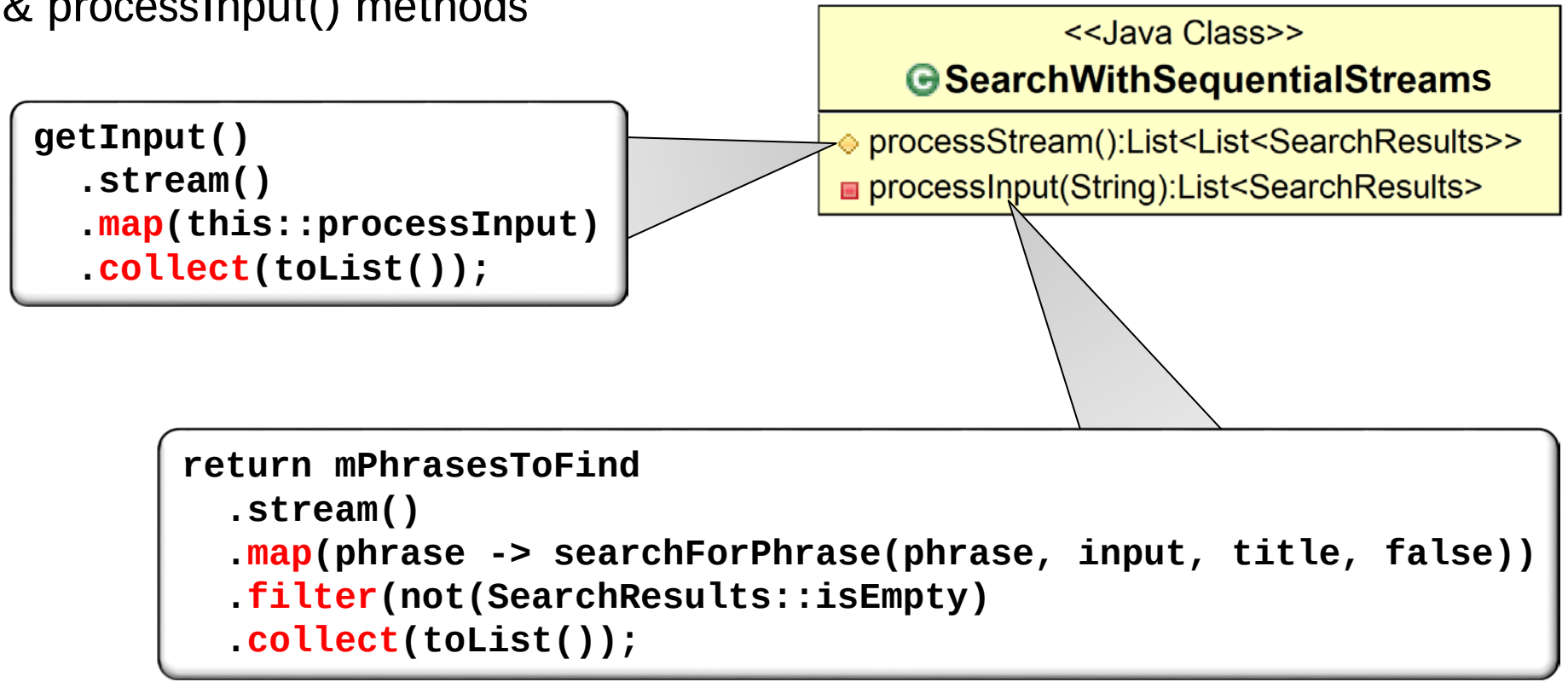
- We show aggregate operations in the SearchStreamGang's processStream() & processInput() methods



See SearchStreamGang/src/main/java/livelessons/streamgangs/SearchWithSequentialStreams.java

Visualizing processStream() & processInput()

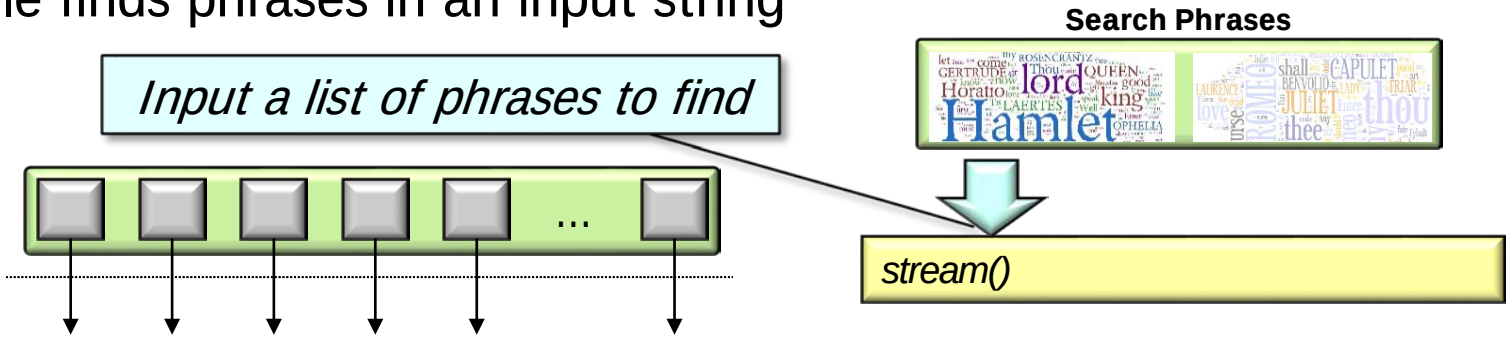
- We show aggregate operations in the SearchStreamGang's processStream() & processInput() methods



i.e., the map(), filter(), & collect() aggregate operations

Visualizing processStream() & processInput()

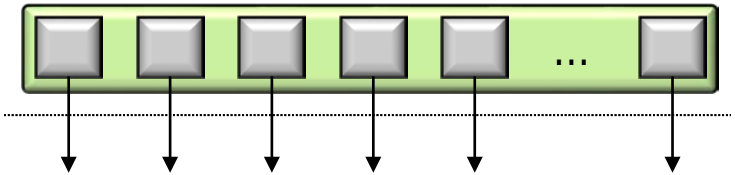
- This example finds phrases in an input string

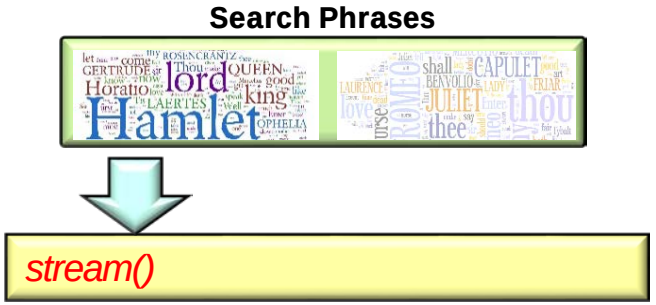


Visualizing processStream() & processInput()

- This example finds phrases in an input string

List
<String>

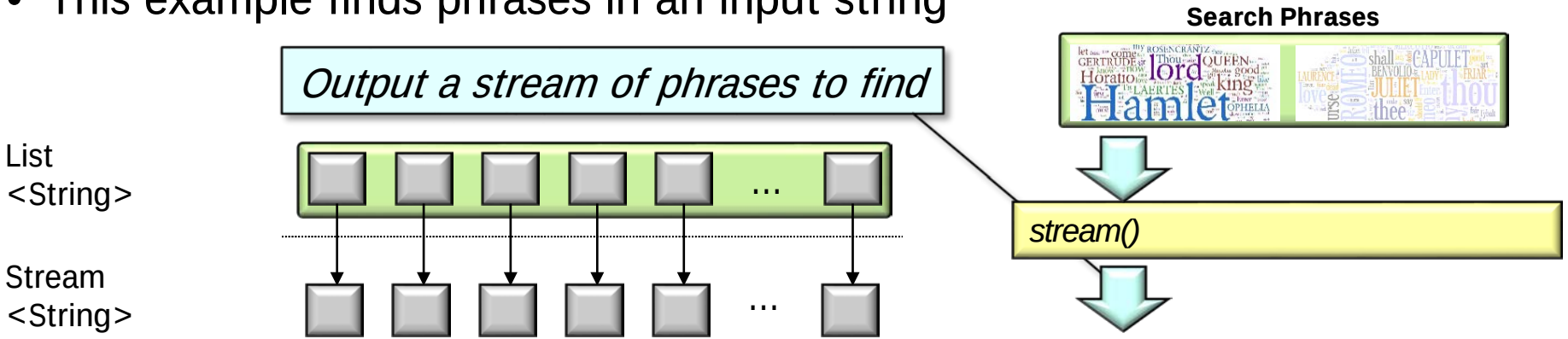




Convert collection to a (sequential) stream

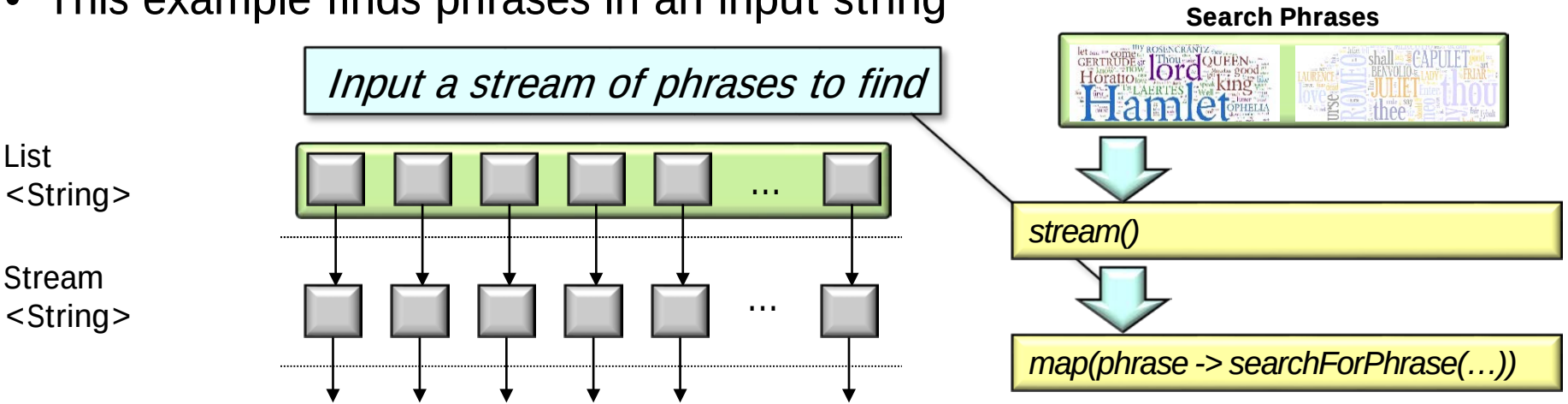
Visualizing processStream() & processInput()

- This example finds phrases in an input string



Visualizing processStream() & processInput()

- This example finds phrases in an input string

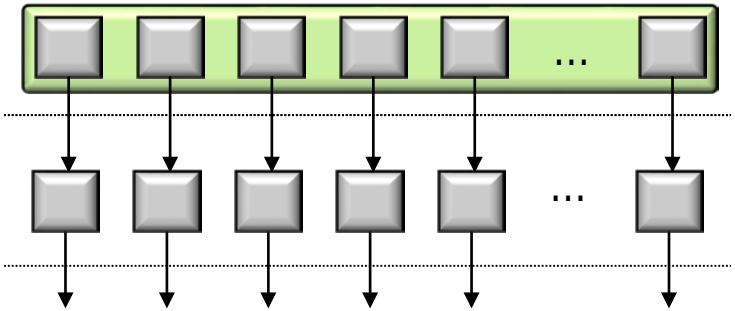


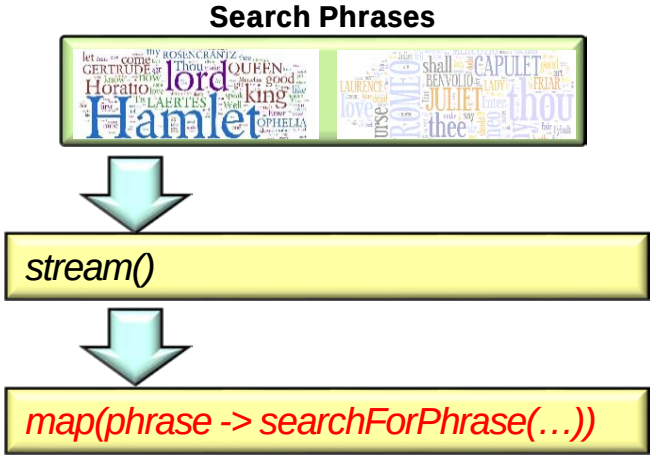
Visualizing processStream() & processInput()

- This example finds phrases in an input string

List
<String>

Stream
<String>

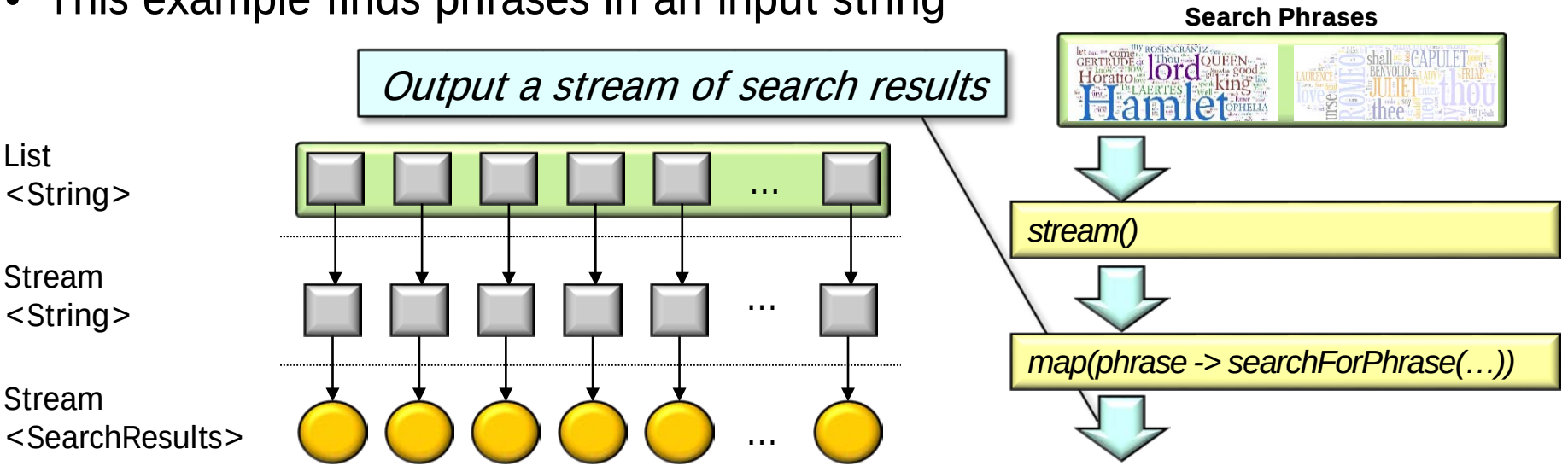




Search for the phrase in each input string

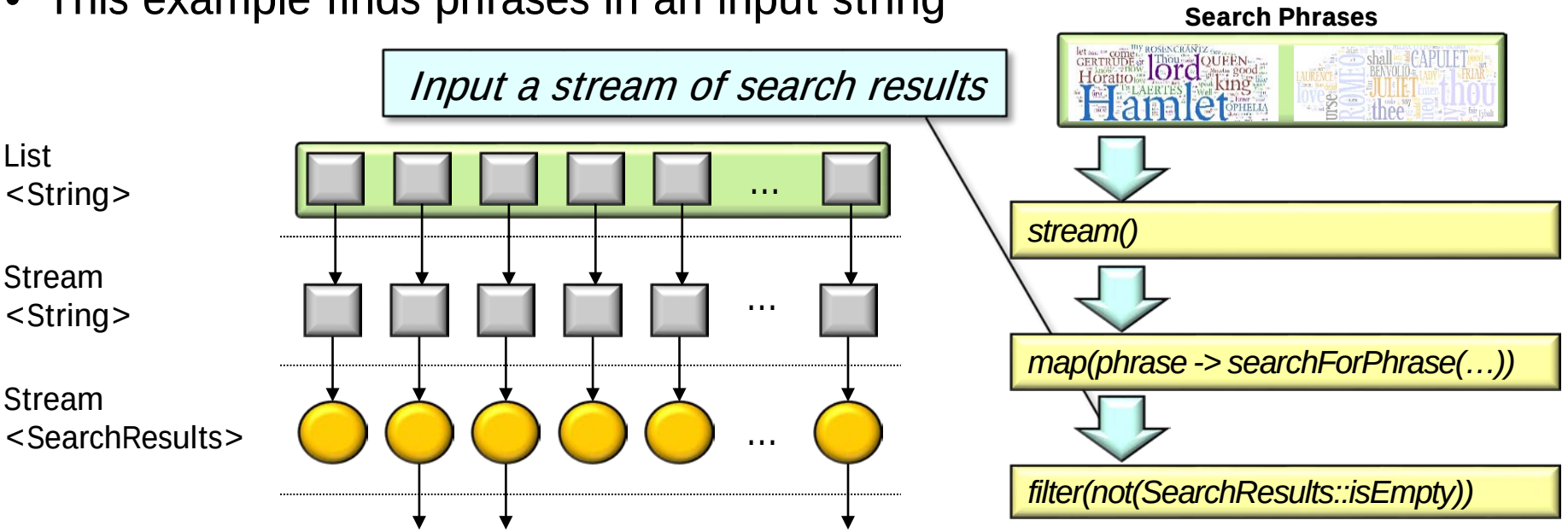
Visualizing processStream() & processInput()

- This example finds phrases in an input string



Visualizing processStream() & processInput()

- This example finds phrases in an input string



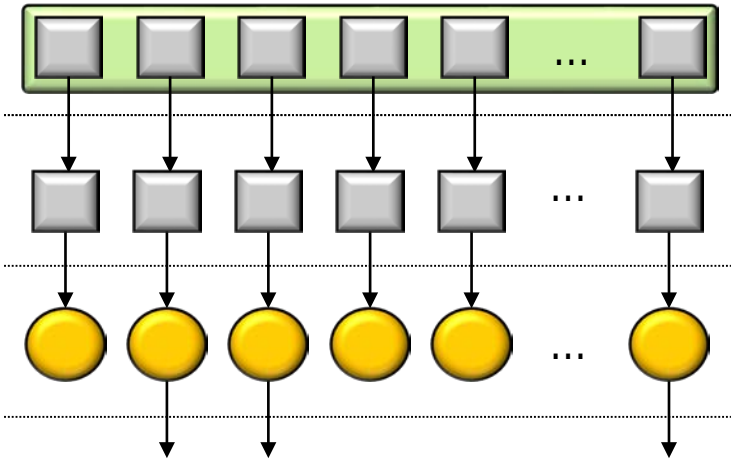
Visualizing processStream() & processInput()

- This example finds phrases in an input string

List
<String>

Stream
<String>

Stream
<SearchResults>



Search Phrases

stream()

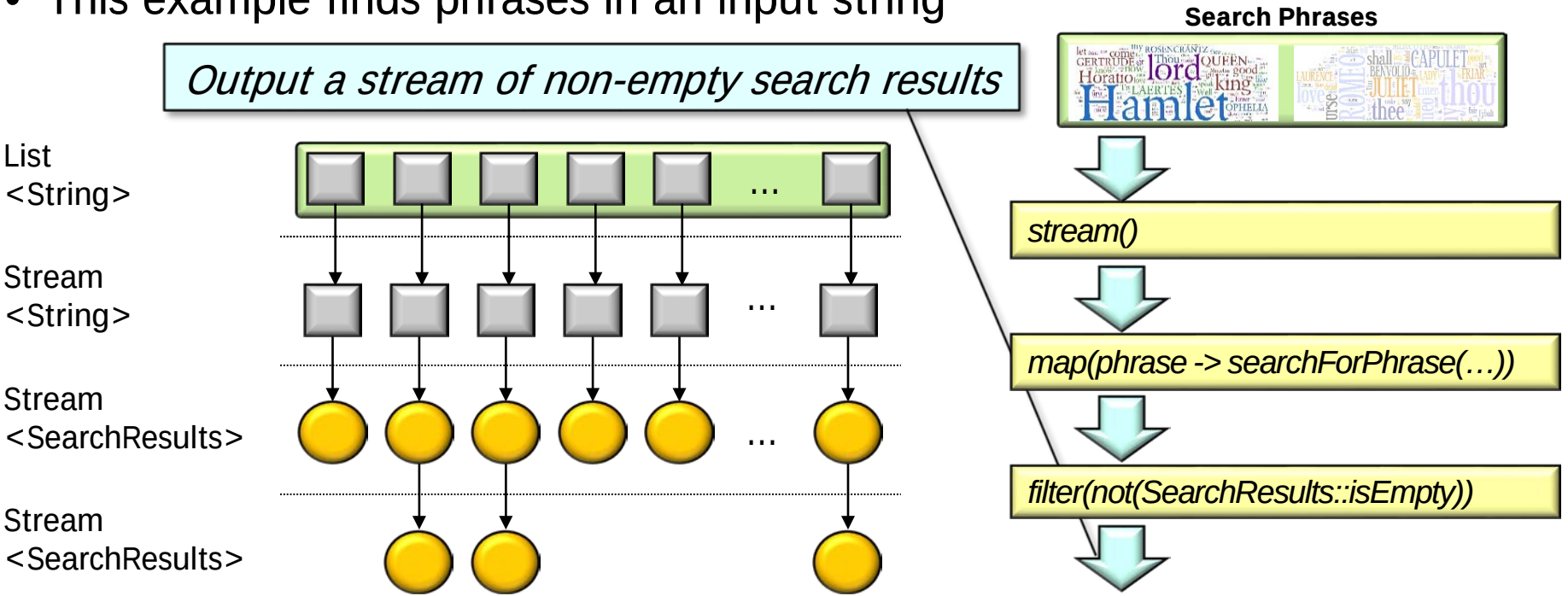
map(phrase -> searchForPhrase(…))

filter(not(SearchResults::isEmpty))

Remove empty search results from the stream

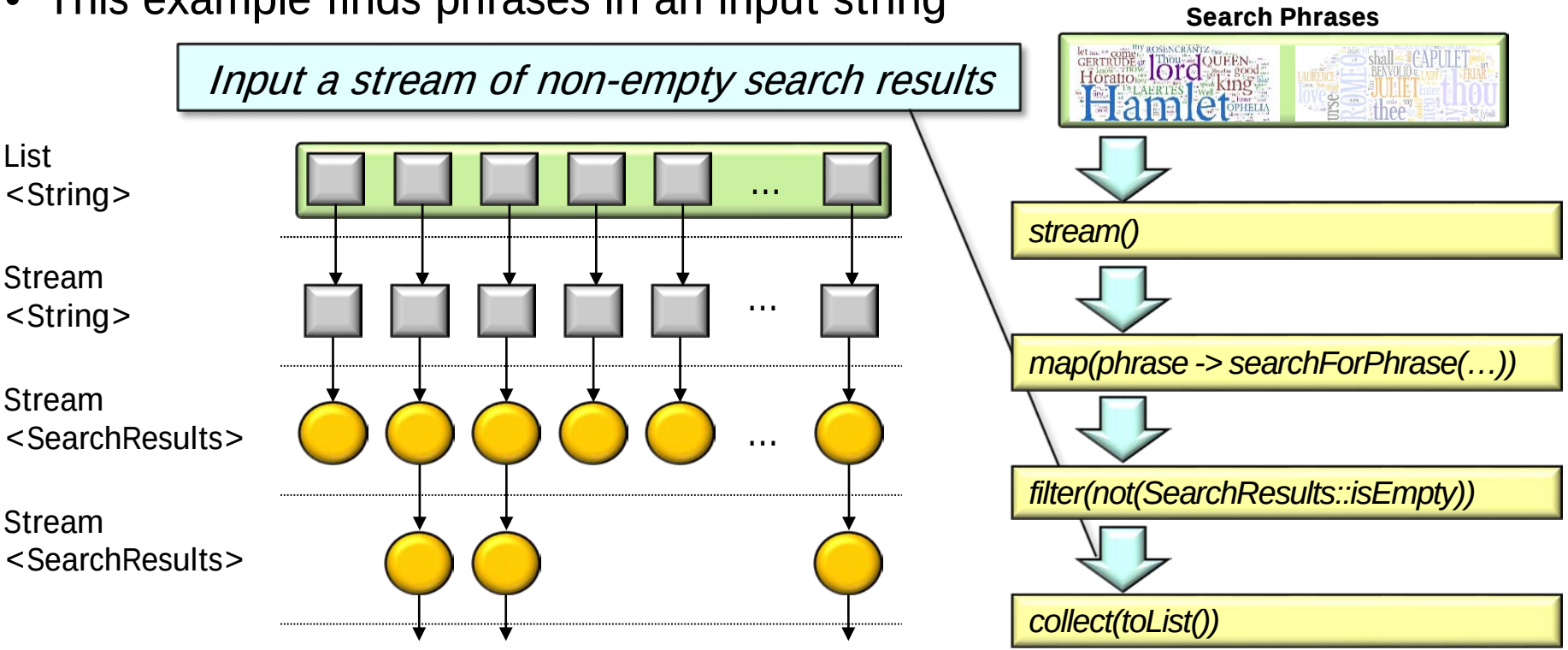
Visualizing processStream() & processInput()

- This example finds phrases in an input string



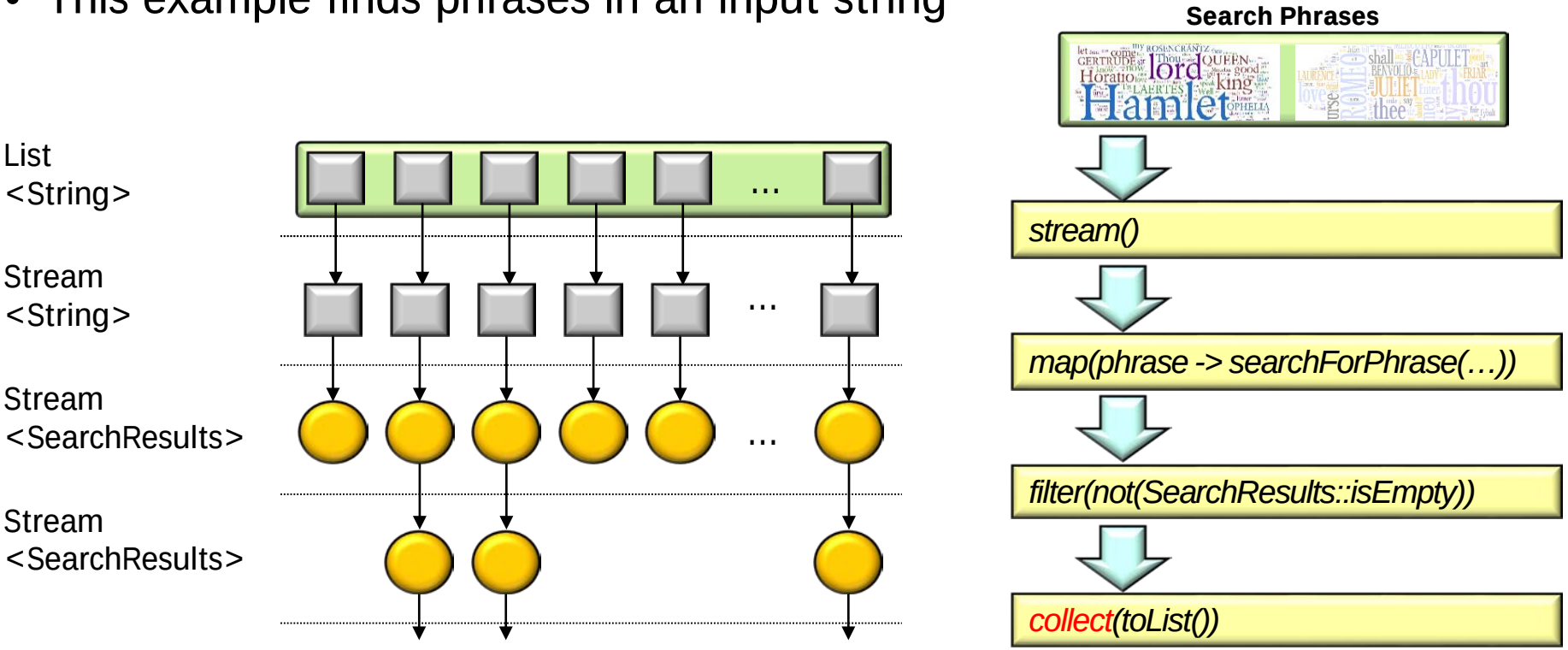
Visualizing processStream() & processInput()

- This example finds phrases in an input string



Visualizing processStream() & processInput()

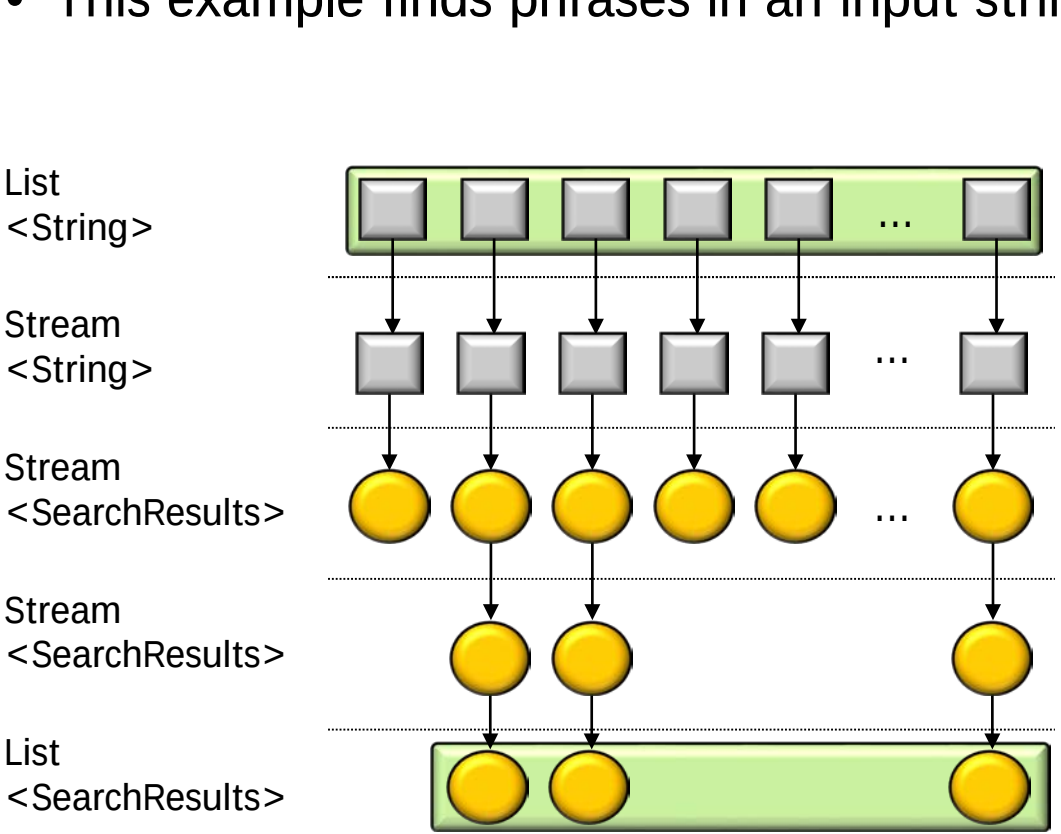
- This example finds phrases in an input string

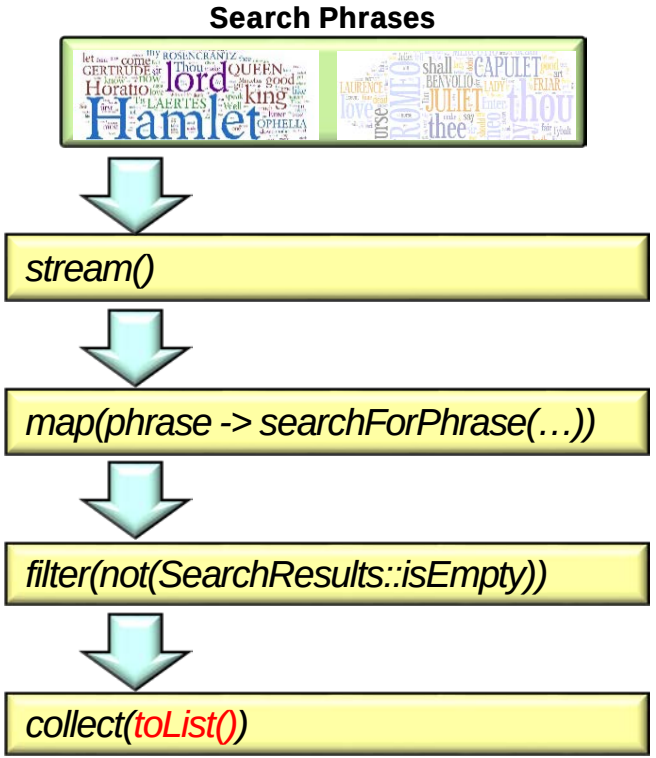


Trigger intermediate operation processing

Visualizing processStream() & processInput()

- This example finds phrases in an input string

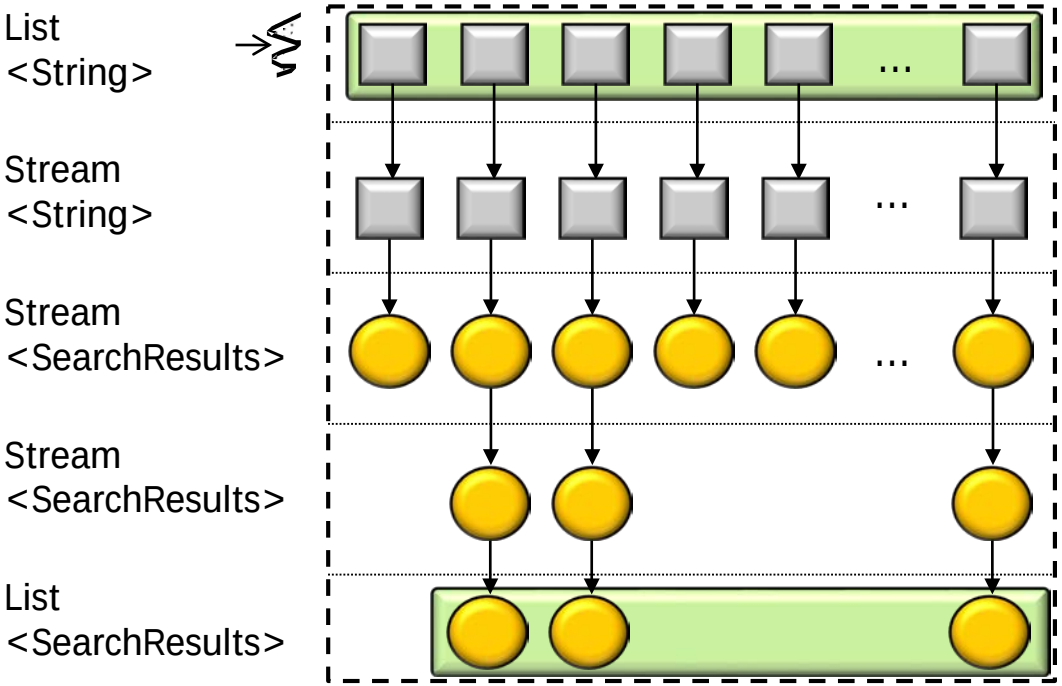


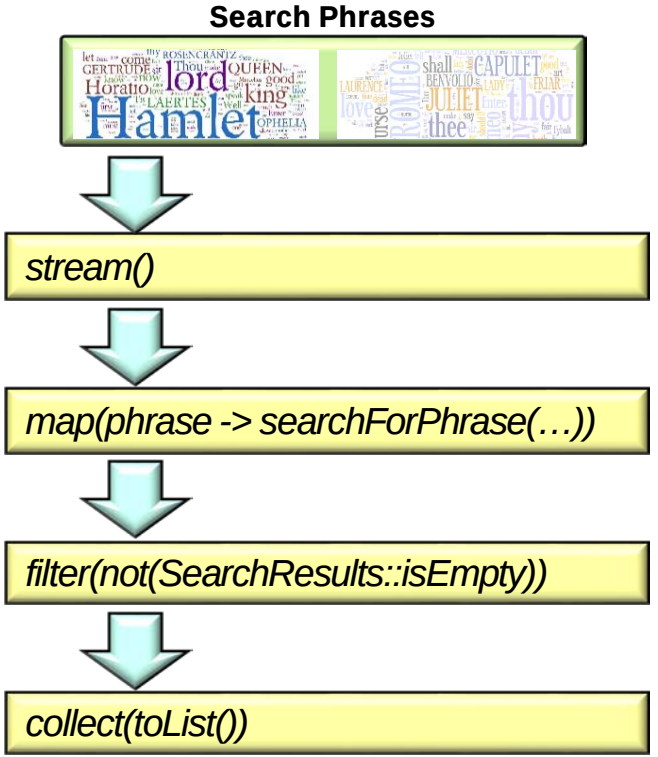


Return a list of search results

Visualizing processStream() & processInput()

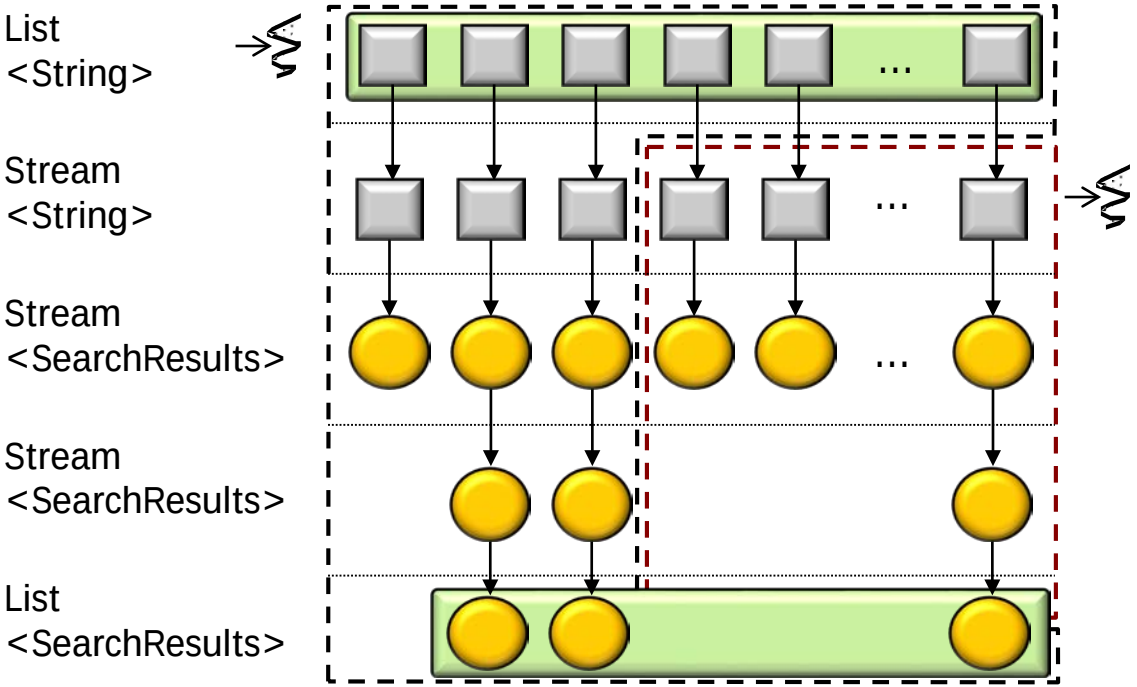
- Our focus here is on sequential streams

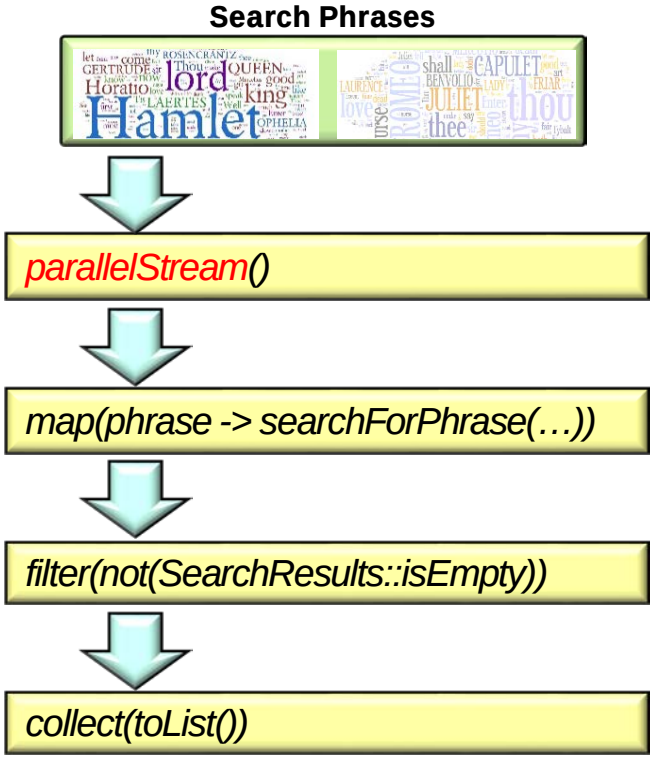




Visualizing processStream() & processInput()

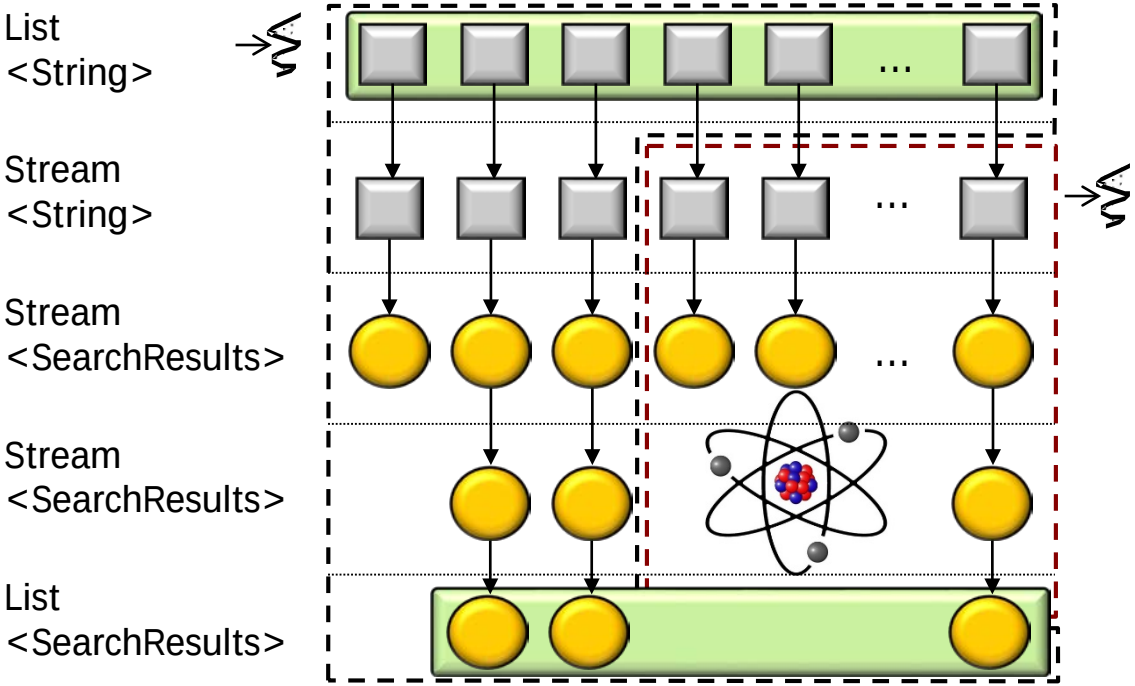
- Our focus here is on sequential streams
  - We'll cover parallel streams shortly

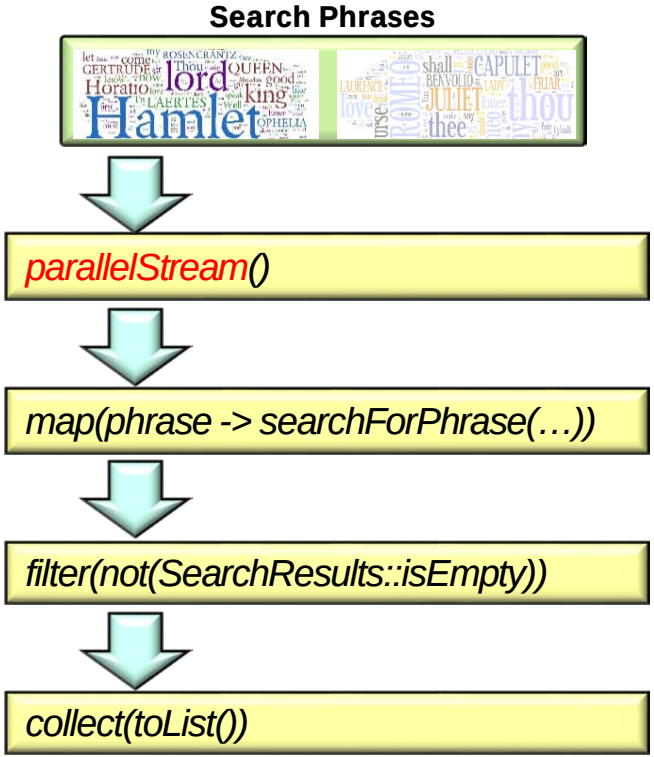




Visualizing processStream() & processInput()

- Our focus here is on sequential streams
  - We'll cover parallel streams shortly



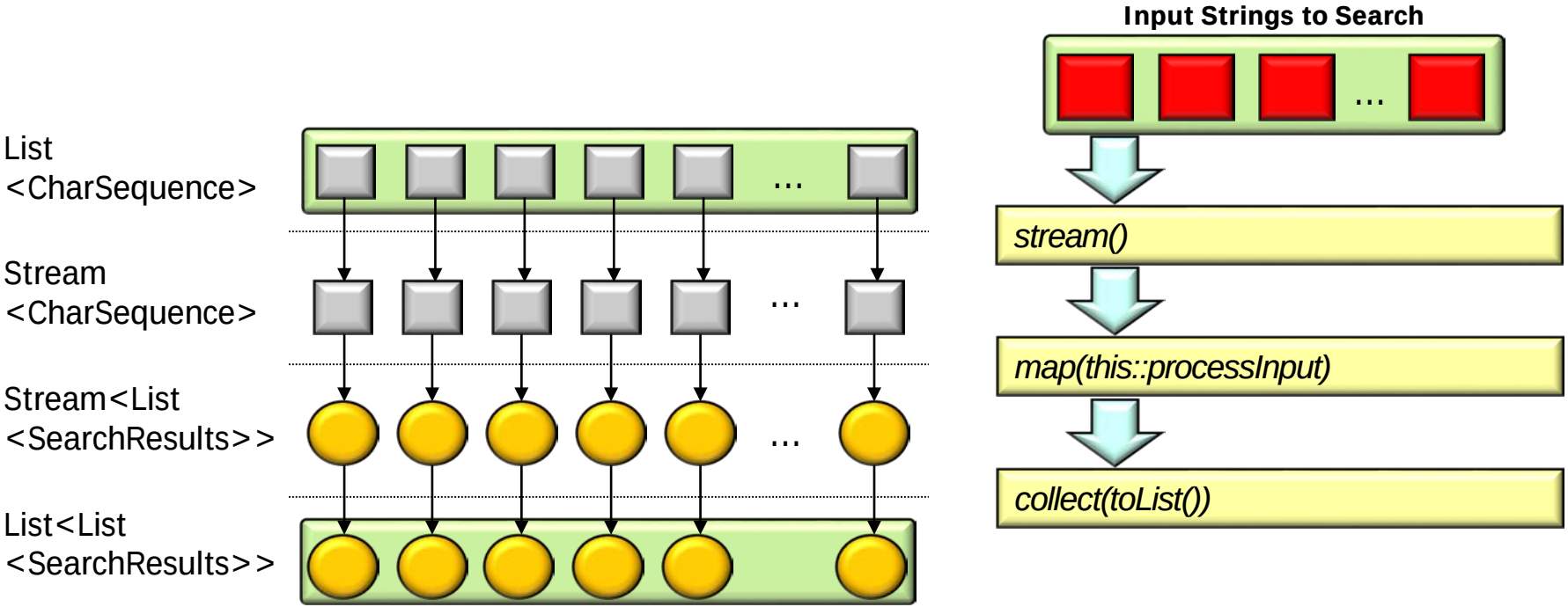


Minuscule changes are needed to transition from sequential to parallel streams!

Implementing processStream() as a Sequential Stream

Implementing processStream() as a Sequential Stream

- processStream() sequentially searches for phrases in lists of input “strings”



Implementing processStream() as a Sequential Stream

- processStream() sequentially searches for phrases in lists of input “strings”

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

  return inputList
    .stream()

    .map(this::processInput)

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

Implementing processStream() as a Sequential Stream

- processStream() sequentially searches for phrases in lists of input “strings”

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

  return inputList
    .stream()

    .map(this::processInput)

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

CharSequence enables optimizations that avoid excessive memory copies

Implementing processStream() as a Sequential Stream

- processStream() sequentially searches for phrases in lists of input “strings”

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

  return inputList
    .stream()

    .map(this::processInput)

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

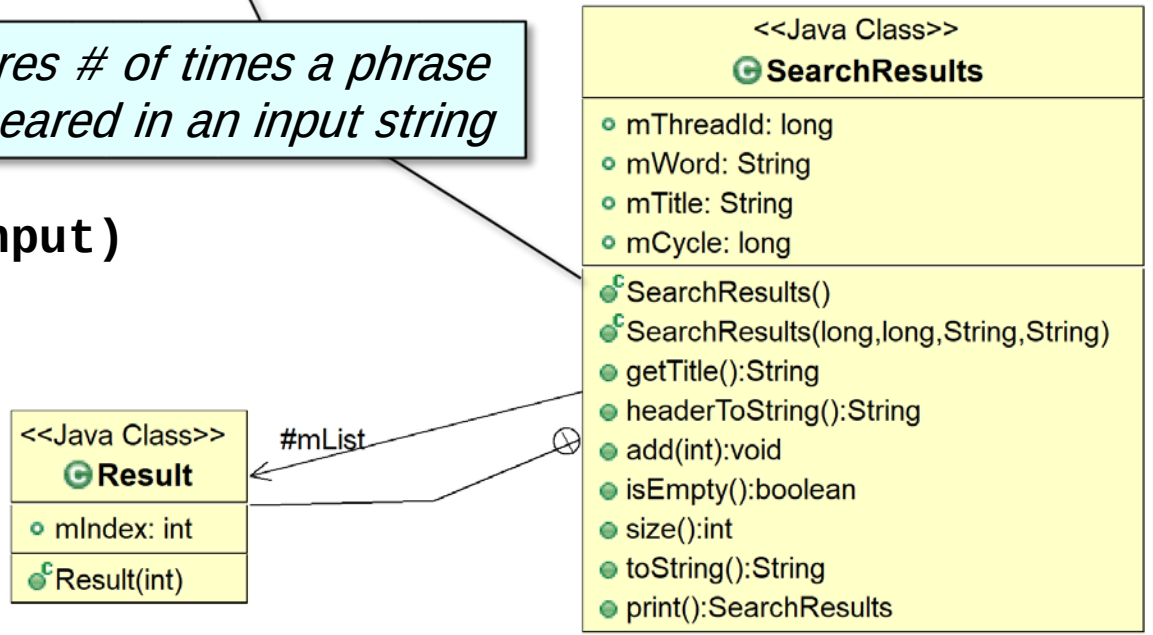
Returns a list of lists of search results denoting how many times a search phrase appeared in each input string

Implementing processStream() as a Sequential Stream

- processStream() sequentially searches for phrases in lists of input “strings”

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

  return inputList
    .stream()

    .map(this::processInput)

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

Stores # of times a phrase appeared in an input string

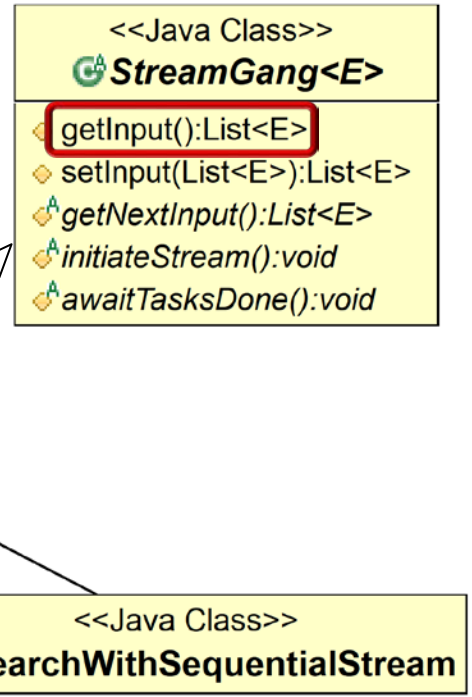


Implementing processStream() as a Sequential Stream

- processStream() sequentially searches for phrases in lists of input “strings”

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

  return inputList
    .stream()

    .map(this::processInput)

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

The input is structured as a list of CharSequences



Implementing processStream() as a Sequential Stream

- processStream() sequentially searches for phrases in lists of input “strings”

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

  return inputList
    .stream()

    .map(this::processInput)

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

Method is implemented via a sequential stream pipeline

Implementing processStream() as a Sequential Stream

- processStream() sequentially searches for phrases in lists of input “strings”

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

  return inputList
    .stream()

    .map(this::processInput)

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

This factory method converts the input list into a stream

The stream() factory method uses StreamSupport.stream(spliterator(), false)

Implementing processStream() as a Sequential Stream

- processStream() sequentially searches for phrases in lists of input “strings”

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

  return inputList
    .stream()

    .map(this::processInput)

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

Returns an output stream of SearchResults lists obtained by applying the processInput() method reference to each input in the stream

Implementing processStream() as a Sequential Stream

- processStream() sequentially searches for phrases in lists of input "strings"

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

  return inputList
    .stream()

    .map(this::processInput)

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

Returns an output stream of SearchResults lists obtained by applying the processInput() method reference to each input in the stream

processInput() returns a list of SearchResults—one list for each input string

Implementing processStream() as a Sequential Stream

- processStream() sequentially searches for phrases in lists of input “strings”

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

  return inputList
    .stream()

    .map(this::processInput)

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

This terminal operation triggers the intermediate operation processing & yields a list (of lists) result

Implementing processStream() as a Sequential Stream

- processStream() sequentially searches for phrases in lists of input “strings”

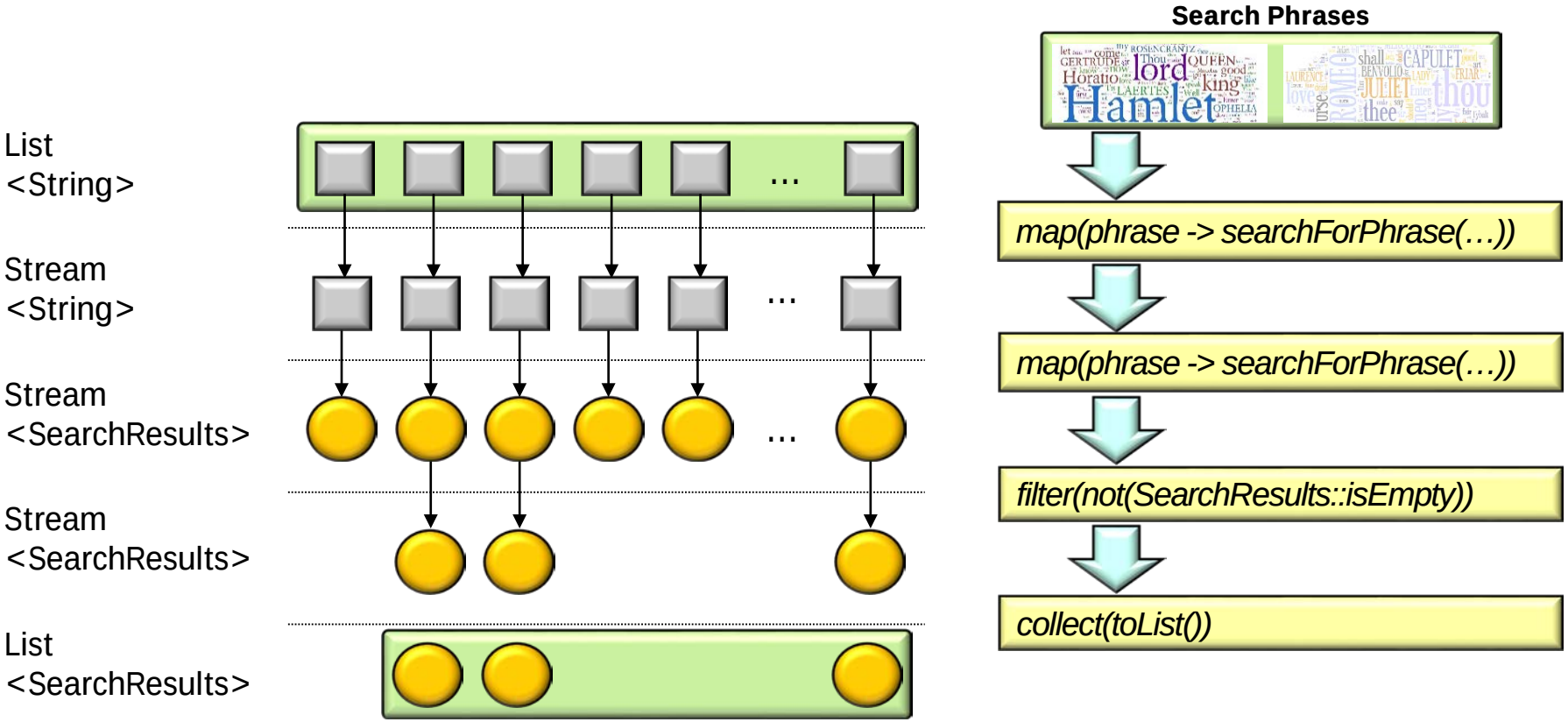
```
protected List<List<SearchResults>> processStream() {
  List<CharSequence> inputList = getInput();

  return inputList
    .stream()

    .map(this::processInput)

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

Returns a list of lists of search results denoting how many times a search phrase appeared in each input string

Implementing processInput() as a Sequential Stream

Implementing processInput() as a Sequential Stream

- processInput() searches an input string for all occurrences of phrases to find



Implementing processInput() as a Sequential Stream

- processInput() searches an input string for all occurrences of phrases to find

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


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

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

Implementing processInput() as a Sequential Stream

- processInput() searches an input string for all occurrences of phrases to find

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


  List<SearchResults> results = mPhrasesToFind
    .stream()
    .map(phrase
         -> searchForPhrase(phrase, input,
    .filter(not(SearchResults::isEmpty))

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

The input is a section of a text file managed by the test driver program

Implementing processInput() as a Sequential Stream

- processInput() searches an input string for all occurrences of phrases to find

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

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

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

The input string is split into two parts

Implementing processInput() as a Sequential Stream

- processInput() searches an input string for all occurrences of phrases to find

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

subSequence() is used to avoid memory copying overhead for substrings

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

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

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

Implementing processInput() as a Sequential Stream

- processInput() searches an input string for all occurrences of phrases to find

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


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

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

Convert a list of phrases into a stream

Implementing processInput() as a Sequential Stream

- processInput() searches an input string for all occurrences of phrases to find

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


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

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

Apply this function lambda to all phrases in input stream & return an output stream of SearchResults

Implementing processInput() as a Sequential Stream

- processInput() searches an input string for all occurrences of phrases to find

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


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

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

Returns output stream containing non-empty SearchResults from input stream

Implementing processInput() as a Sequential Stream

- processInput() searches an input string for all occurrences of phrases to find

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


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

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

Note use of a method reference & a negator predicate lambda

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

Implementing processInput() as a Sequential Stream

- processInput() searches an input string for all occurrences of phrases to find

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


  List<SearchResults> results = mPhrasesToFind
    .stream()
    .map(phrase
         -> searchForPhrase(phrase, input, title, false))
    .filter(result -> result.size() > 0)

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

Another approach using a lambda expression

Implementing processInput() as a Sequential Stream

- processInput() searches an input string for all occurrences of phrases to find

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


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

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

These are both intermediate operations

Implementing processInput() as a Sequential Stream

- processInput() searches an input string for all occurrences of phrases to find

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


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

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

This terminal operation triggers intermediate operation processing & yields a list result

Implementing processInput() as a Sequential Stream

- processInput() searches an input string for all occurrences of phrases to find

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


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

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

This terminal operation triggers intermediate operation processing & yields a list result

Implementing processInput() as a Sequential Stream

- processInput() searches an input string for all occurrences of phrases to find

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


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

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

The list result is returned back to the map() operation in processStream()

End of Java 8 Sequential SearchStreamGang Example (Part 1)