Java 8 Sequential SearchStreamGang Example (Part 3)

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

Professor of Computer Science

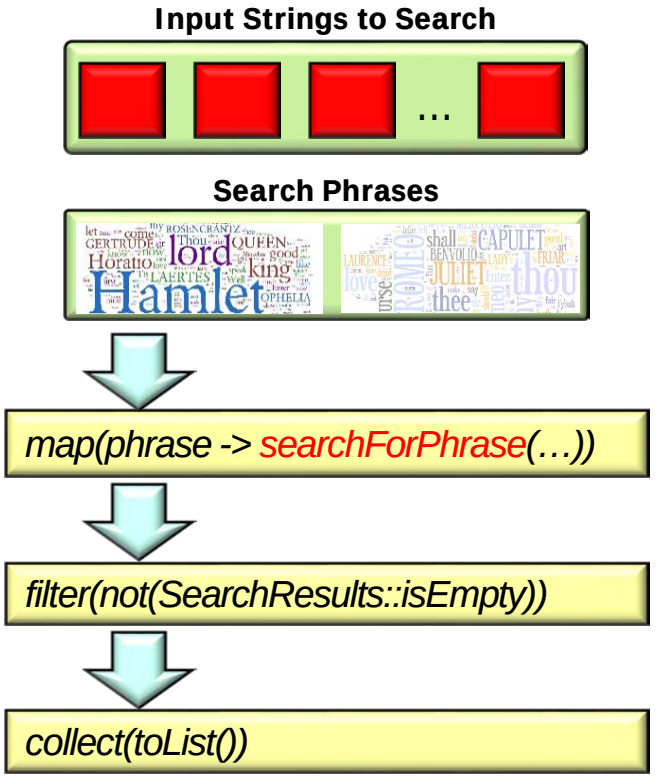
Institute for Software Integrated Systems

Vanderbilt University
Nashville, Tennessee, USA

Learning Objectives in this Part of the Lesson

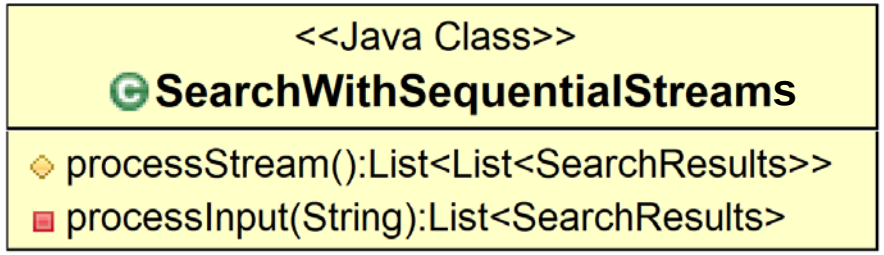
- Know how to apply sequential streams to the SearchStreamGang program
- Recognize how a Spliterator is used in SearchWithSequentialStreams

```
SearchResults searchForPhrase
  (String phrase, CharSequence input,
   String title, boolean parallel) {
  return new SearchResults
    (..., phrase, ..., StreamSupport
      .stream(new PhraseMatchSpliterator
                        (input, phrase),
              parallel)
      .collect(toList()));
}
```



Learning Objectives in this Part of the Lesson

- Know how to apply sequential streams to the SearchStreamGang program
- Recognize how a Spliterator is used in SearchWithSequentialStreams
- Understand the pros & cons of the SearchWithSequentialStreams class

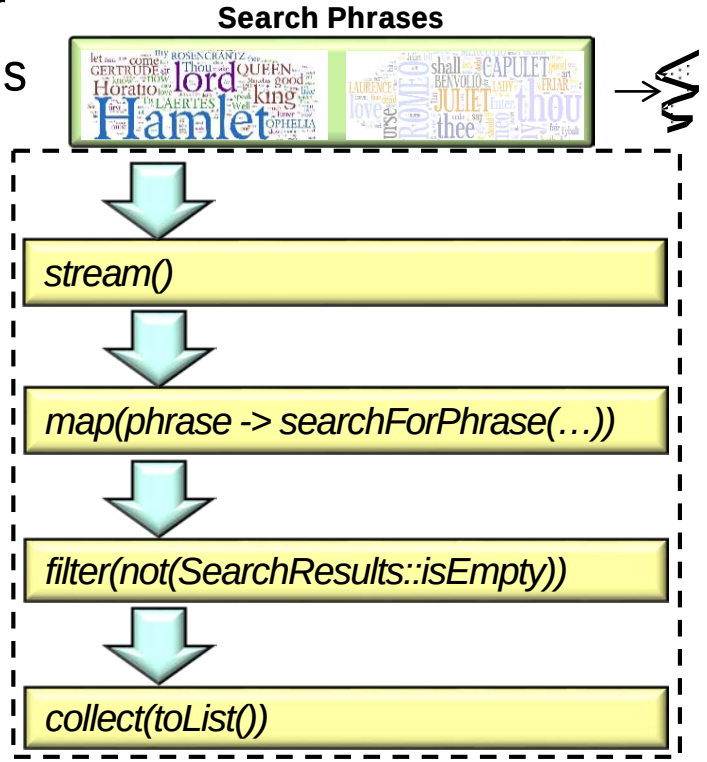


See SearchStreamGang/src/main/java/livelessons/streamgangs/SearchWithSequentialStreams.java

Using Java Spliterator in SearchStreamGang

Using Java Spliterator in SearchStreamGang

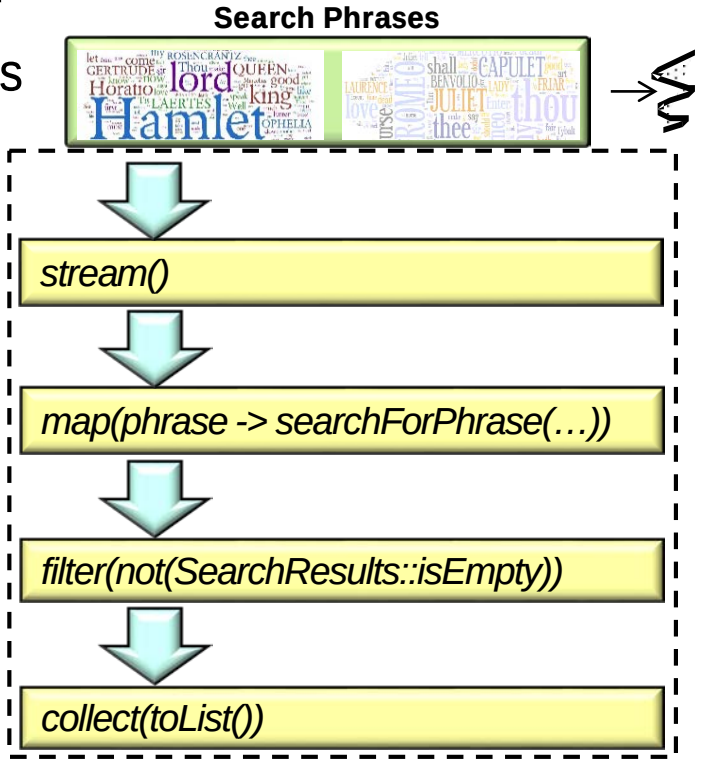
- SearchStreamGang uses PhraseMatchSpliterator that works for both sequential & parallel streams



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

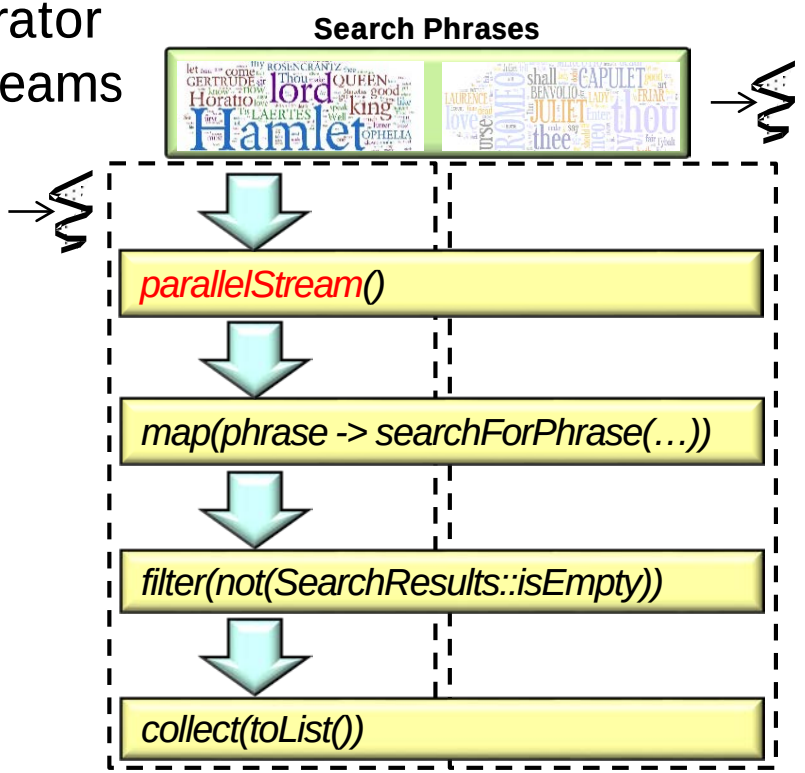
Using Java Spliterator in SearchStreamGang

- SearchStreamGang uses PhraseMatchSpliterator that works for both sequential & parallel streams
  - We focus on the sequential portions now



Using Java Spliterator in SearchStreamGang

- SearchStreamGang uses PhraseMatchSpliterator that works for both sequential & parallel streams
  - We focus on the sequential portions now
  - We'll cover the parallel portions later

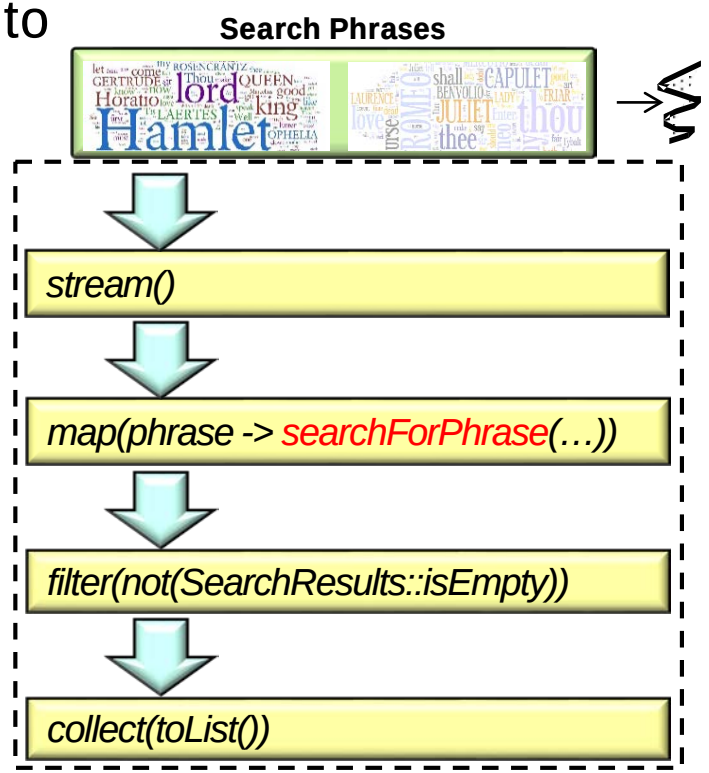


See *"Java 8 Parallel SearchStreamGang Example (Part 2)"*

Using Java Spliterator in SearchStreamGang

- searchForPhrase() uses PhraseMatchSpliterator to find all phrases in input & return SearchResults

```
SearchResults searchForPhrase
  (String phrase, CharSequence input,
   String title, boolean parallel) {
  return new SearchResults
    (..., phrase, ..., StreamSupport
      .stream(new PhraseMatchSpliterator
                        (mInput, word),
              parallel)
      .collect(toList()));
}
```



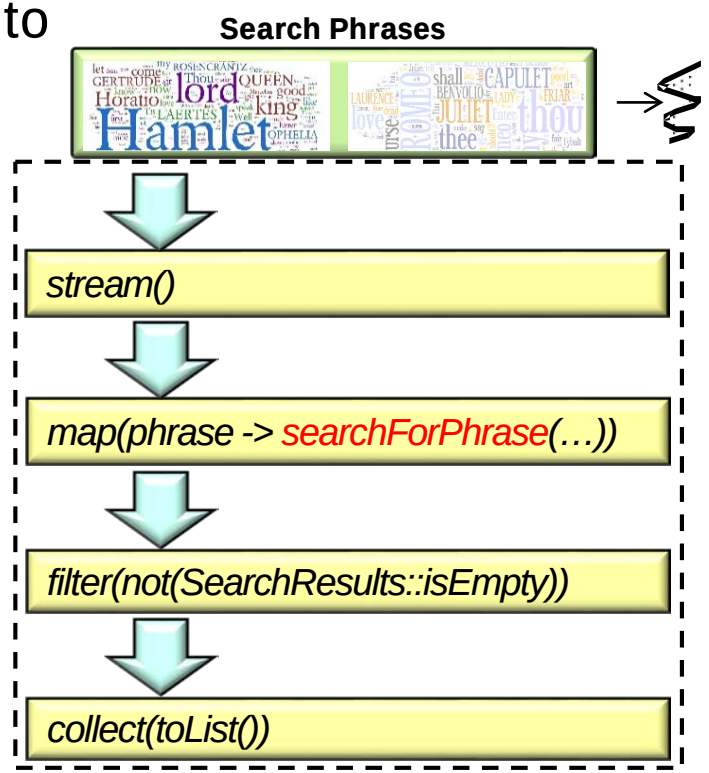
See SearchStreamGang/src/main/java/livelessons/streamgangs/SearchStreamGang.java

Using Java Spliterator in SearchStreamGang

- searchForPhrase() uses PhraseMatchSpliterator to find all phrases in input & return SearchResults

```
SearchResults searchForPhrase
  (String phrase, CharSequence input,
   String title, boolean parallel) {
  return new SearchResults
    (..., phrase, ..., StreamSupport
      .stream(new PhraseMatchSpliterator
                        (input, phrase),
              parallel)
      .collect(toList()));
}
```

StreamSupport.stream() creates a sequential or parallel stream via PhraseMatchSpliterator



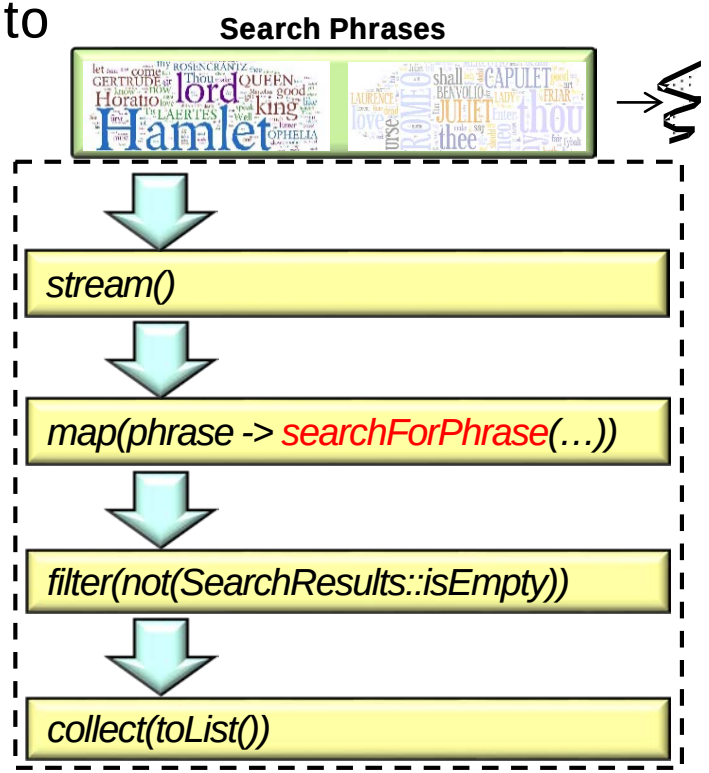
See docs.oracle.com/javase/8/docs/api/java/util/stream/StreamSupport.html#stream

Using Java Spliterator in SearchStreamGang

- searchForPhrase() uses PhraseMatchSpliterator to find all phrases in input & return SearchResults

```
SearchResults searchForPhrase
   (String phrase, CharSequence input,
    String title, boolean parallel) {
   return new SearchResults
     (..., phrase, ..., StreamSupport
       .stream(new PhraseMatchSpliterator
                        (input, phrase),
               parallel)
       .collect(toList()));
}
```

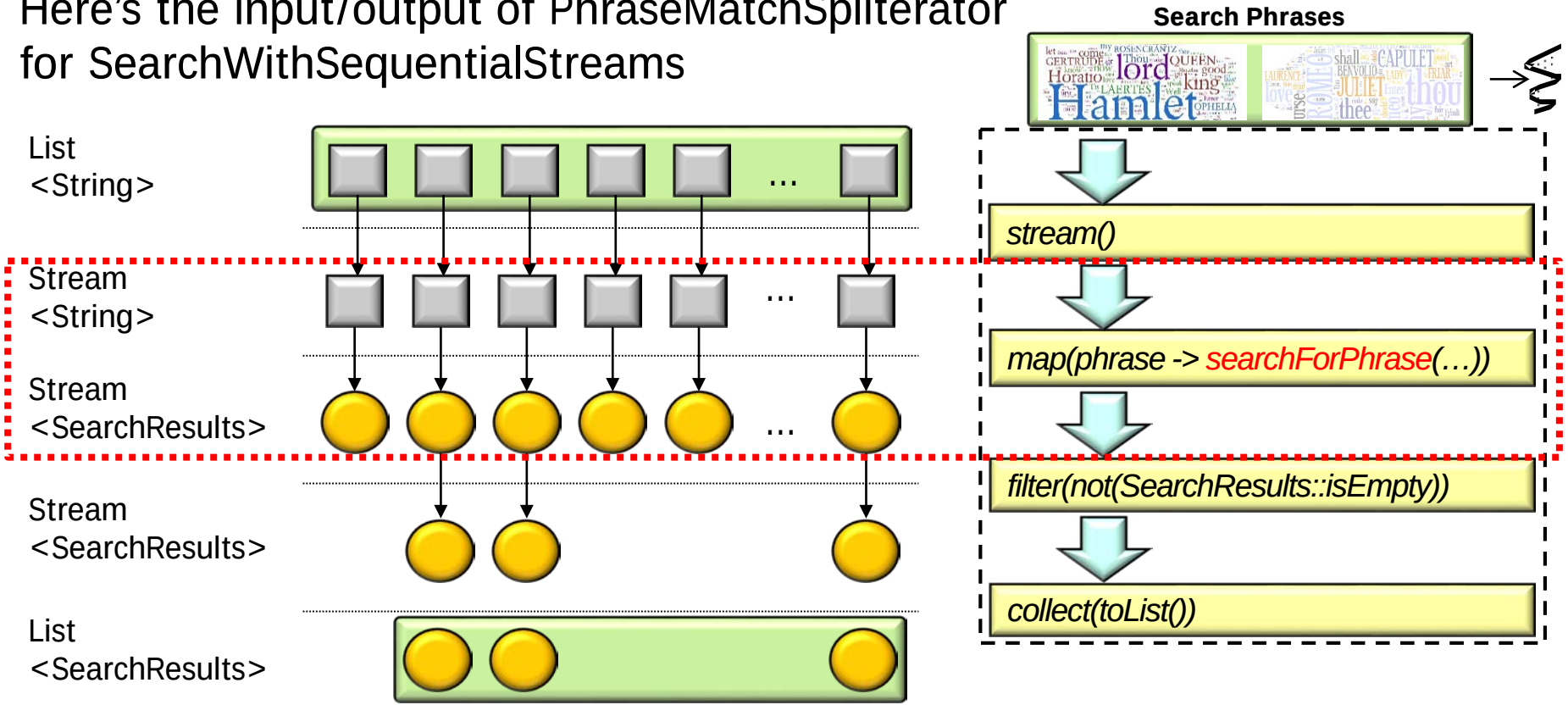
For SearchWithSequentialStreams "parallel" is false, so we'll use a sequential spliterator



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

Using Java Spliterator in SearchStreamGang

- Here's the input/output of PhraseMatchSpliterator for SearchWithSequentialStreams

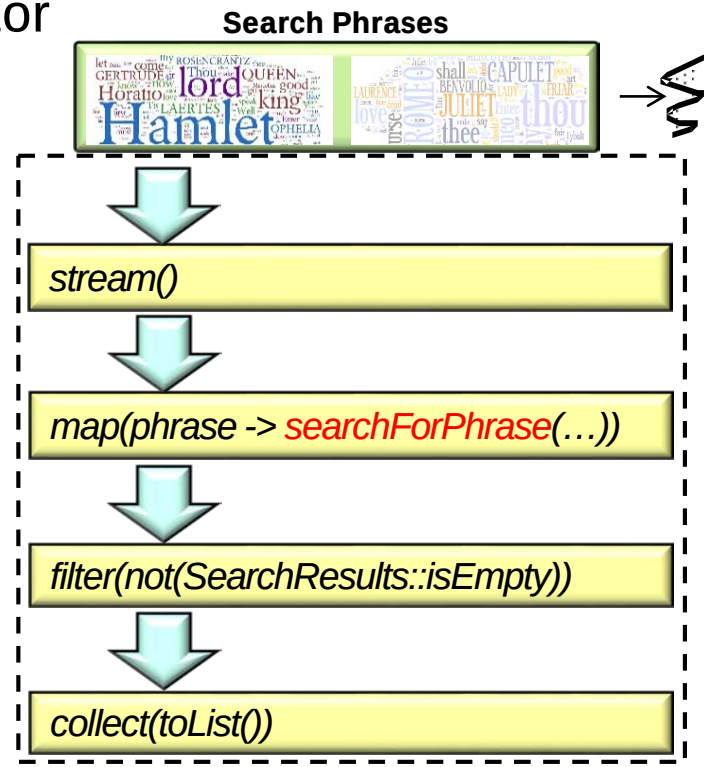


Using Java Spliterator in SearchStreamGang

- Here's the input/output of PhraseMatchSpliterator for SearchWithSequentialStreams

"...
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" matches at index [54739]



Using Java Spliterator in SearchStreamGang

- PhraseMatchSpliterator uses Java regex to create a stream of SearchResults Result objects that match the # of times a phrase appears in an input string

```
class PhraseMatchSpliterator implements Spliterator<Result> {
  private CharSequence mInput;

  private final String mPhrase;

  private final Pattern mPattern;

  private Matcher mPhraseMatcher;

  private final int mMinSplitSize;

  private int mOffset = 0;
  ...
```

Spliterator is an interface that defines eight methods, including tryAdvance() & trySplit()

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

Using Java Spliterator in SearchStreamGang

- PhraseMatchSpliterator uses Java regex to create a stream of SearchResults Result objects that match the # of times a phrase appears in an input string

```
class PhraseMatchSpliterator implements Spliterator<Result> {
  private CharSequence mInput;

  private final String mPhrase;

  private final Pattern mPattern;

  private final Matcher mPhraseMatcher;

  private final int mMinSplitSize;

  private int mOffset = 0;
  ...
```

These fields implement PhraseMatchSpliterator for both sequential & parallel use-cases

Some fields are updated in the trySplit() method, which is why they aren't final

Using Java Spliterator in SearchStreamGang

- PhraseMatchSpliterator uses Java regex to create a stream of SearchResults Result objects that match the # of times a phrase appears in an input string

```
class PhraseMatchSpliterator implements Spliterator<Result> {
  ...
  PhraseMatchSpliterator(CharSequence input, String phrase) {
    String regexPhrase = "\\b" + phrase.trim().replaceAll
                                   ("\\s+", "\\\\b\\\\s+\\\\b")
                       + "\\b"; ...
    mPattern = Pattern.compile(regexPhrase,
                               Pattern.CASE_INSENSITIVE | Pattern.DOTALL);
    mPhraseMatcher = mPattern.matcher(input);
    mInput = input; mPhrase = phrase;
    mMinSplitSize = input.length() / 2;
  } ...
```

A regex is compiled into a pattern that matches a phrase across lines

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

Using Java Spliterator in SearchStreamGang

- PhraseMatchSpliterator uses Java regex to create a stream of SearchResults Result objects that match the # of times a phrase appears in an input string

```
class PhraseMatchSpliterator implements Spliterator<Result> {
  ...
  PhraseMatchSpliterator(CharSequence input, String phrase) {
    String regexPhrase = "\\b" + phrase.trim().replaceAll
                                    ("\\s+", "\\\\b\\\\s+\\\\b")
                               + "\\b"; ...

    mPattern = Pattern.compile(regexPhrase,
                  Pattern.CASE_INSENSITIVE | Pattern.DOTALL);
    mPhraseMatcher = mPattern.matcher(input);
    mInput = input; mPhrase = phrase;
    mMinSplitSize = input.length() / 2;
  } ...
```

A matcher is created to search the input for the regex pattern

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

Using Java Spliterator in SearchStreamGang

- PhraseMatchSpliterator uses Java regex to create a stream of SearchResults Result objects that match the # of times a phrase appears in an input string

```
class PhraseMatchSpliterator implements Spliterator<Result> {
  ...
  PhraseMatchSpliterator(CharSequence input, String phrase) {
    String regexPhrase = "\\b" + phrase.trim().replaceAll
                                    ("\\s+", "\\\\b\\\\s+\\\\b")
                               + "\\b"; ...

    mPattern = Pattern.compile(regexPhrase,
                  Pattern.CASE_INSENSITIVE | Pattern.DOTALL);
    mPhraseMatcher = mPattern.matcher(input);
    mInput = input; mPhrase = phrase;
    mMinSplitSize = input.length() / 2;
  } ...
```

Define the min split size

Using Java Spliterator in SearchStreamGang

- PhraseMatchSpliterator uses Java regex to create a stream of SearchResults Result objects that match the # of times a phrase appears in an input string

```
class PhraseMatchSpliterator implements Spliterator<Result> {
  ...
  boolean tryAdvance(Consumer<? super Result> action) {
    if (!mPhraseMatcher.find())
        return false;

    else {
        action.accept(new Result
                              (mOffset + mPhraseMatcher.start()));
        return true;
    }
  }
  ...
```

This method plays the role of hasNext() & next() in Java's Iterator interface

Using Java Spliterator in SearchStreamGang

- PhraseMatchSpliterator uses Java regex to create a stream of SearchResults Result objects that match the # of times a phrase appears in an input string

```
class PhraseMatchSpliterator implements Spliterator<Result> {
  ...
  boolean tryAdvance(Consumer<? super Result> action) {
    if (!mPhraseMatcher.find())
        return false;

    else {
        action.accept(new Result
                              (mOffset + mPhraseMatcher.start()));
        return true;
    }
  }
  ...
```

It first checks if there are any remaining phrases in the input that match the regex

Using Java Spliterator in SearchStreamGang

- PhraseMatchSpliterator uses Java regex to create a stream of SearchResults Result objects that match the # of times a phrase appears in an input string

```
class PhraseMatchSpliterator implements Spliterator<Result> {
  ...
  boolean tryAdvance(Consumer<? super Result> action) {
    if (!mPhraseMatcher.find())
        return false;

    else {
        action.accept(new Result
                              (mOffset + mPhraseMatcher.start()));
        return true;
    }
  }
  ...
```

If there is a match, then accept() keeps track of which index in the input string the match occurred

Using Java Spliterator in SearchStreamGang

- PhraseMatchSpliterator uses Java regex to create a stream of SearchResults Result objects that match the # of times a phrase appears in an input string

```
class PhraseMatchSpliterator implements Spliterator<Result> {
  ...
  public Spliterator<SearchResults.Result> trySplit() {
    ...
  }
  ...
```

We'll analyze the trySplit() method when we discuss SearchWithParallelStreams (it's not used for the sequential version)

Pros of the SearchWith SequentialStreams Class

Pros of the SearchWithSequentialStreams Class

- There are several benefits with this sequential streams implementation

```
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;
}
```

Pros of the SearchWithSequentialStreams Class

- There are several benefits with this sequential streams implementation

```
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;
}
```

Streams use "internal" iterators versus "external" iterators used by collections

Internal iterators shield programs from streams processing implementation details

Pros of the SearchWithSequentialStreams Class

- There are several benefits with this sequential streams implementation

```
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 pipeline is declarative since it's a series of transformations performed by aggregate operations

Pros of the SearchWithSequentialStreams Class

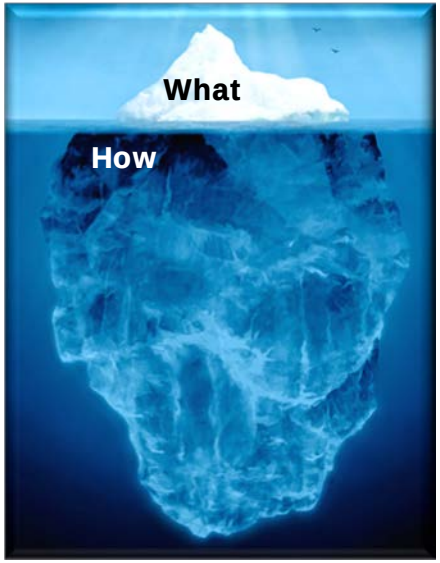
- There are several benefits with this sequential streams implementation

```
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;
}
```



Focus on “what” operations to perform, rather than on “how” they’re implemented

Pros of the SearchWithSequentialStreams Class

- There are several benefits with this sequential streams implementation

```
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;
}
```

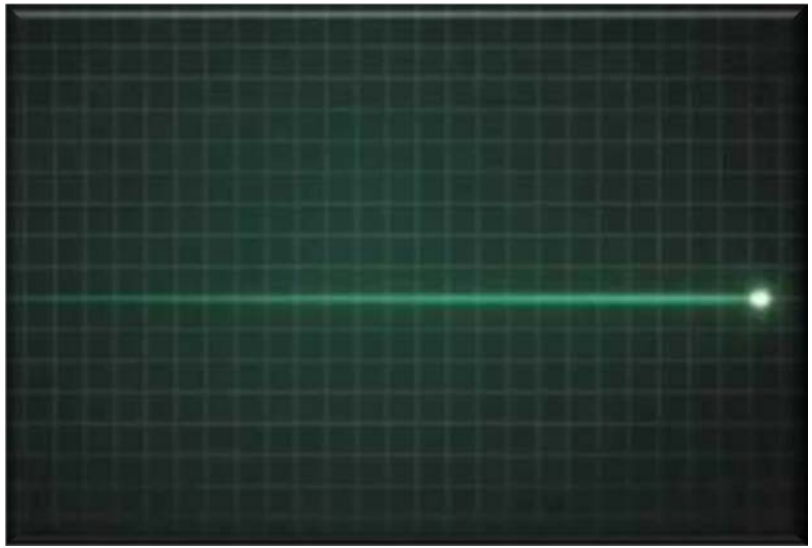
These lambda functions have no side-effects

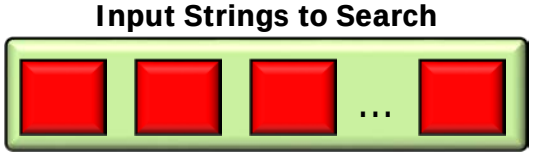
No side-effects makes it easier to reason about behavior & enables optimization

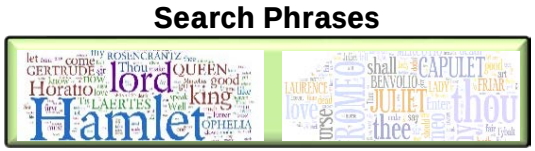
Cons of the SearchWith SequentialStreams Class

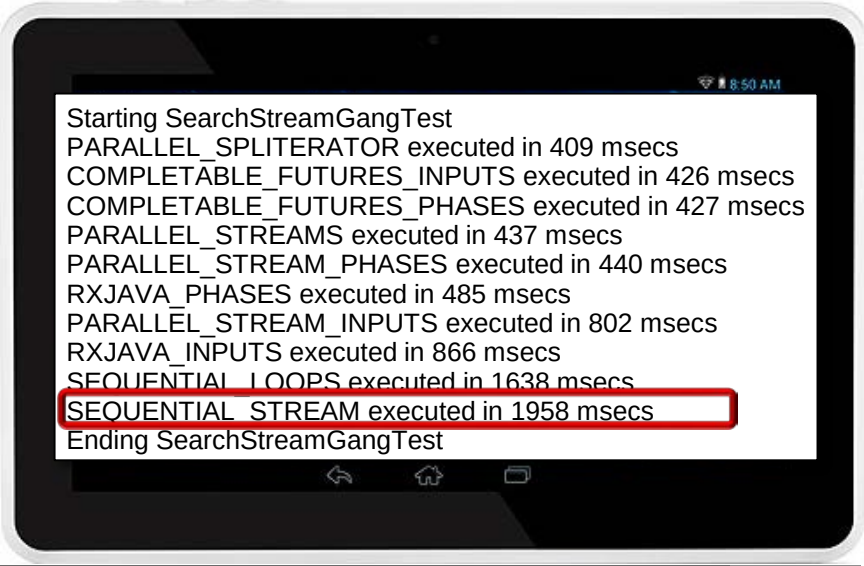
Cons of the SearchWithSequentialStreams Class

- The sequential implementation can't take advantage of multi-core processors









Tests conducted on a quad-core Lenovo P50 with 32 Gbytes of RAM

Cons of the SearchWithSequentialStreams Class

- This class only used a few Java 8 aggregate operations

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

  List<SearchResults> results = mPhrasesToFind
    .stream()
    .map(phrase
         -> searchForPhrase(phrase, input, title))

    .filter(not(SearchResults::isEmpty))

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

However, these aggregate operations are also useful for parallel streams

Cons of the SearchWithSequentialStreams Class

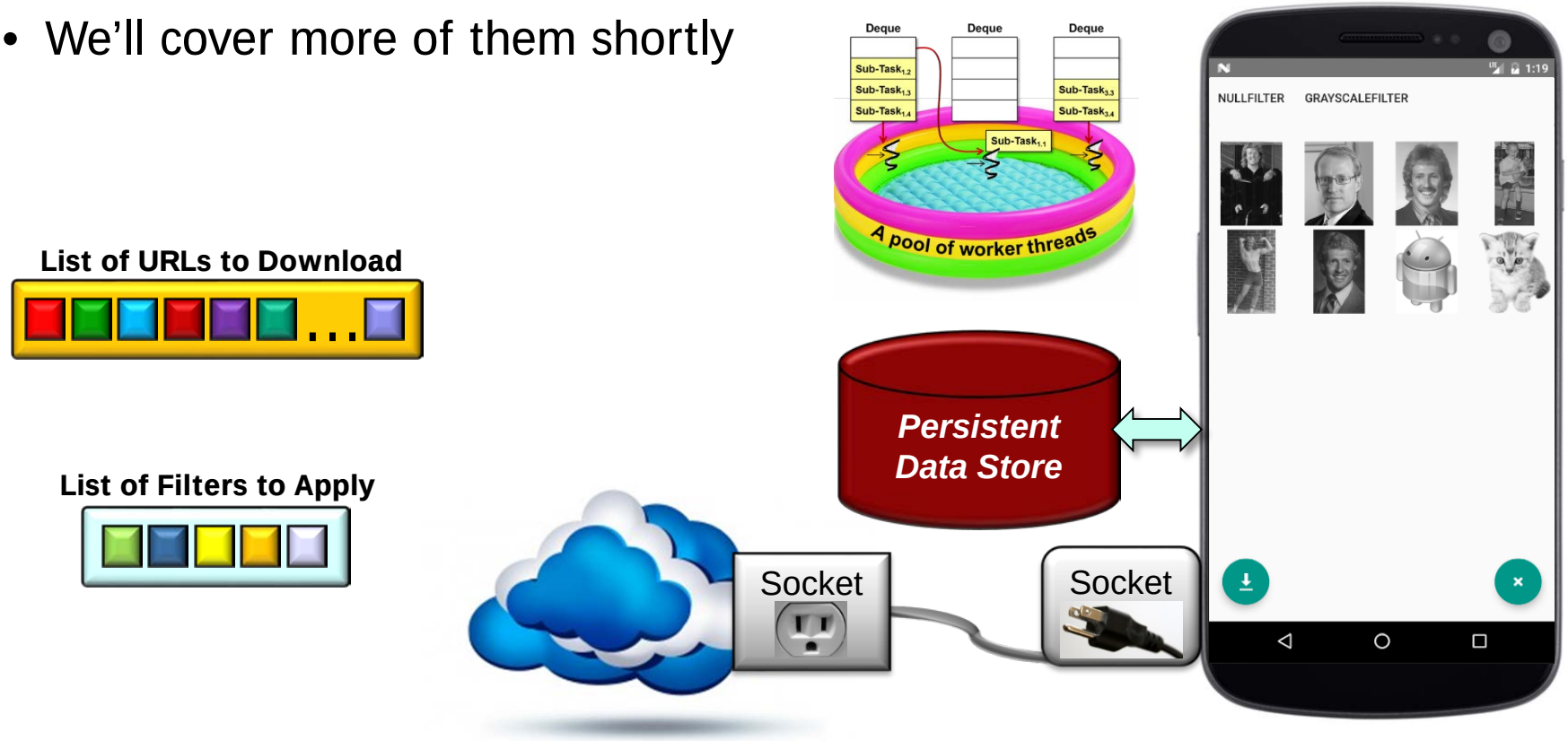
- *Many* other aggregate operations are part of the Java 8 stream API

Modifier and Type	Method and Description
boolean	**allMatch**(Predicate<? super T> predicate) Returns whether all elements of this stream match the provided predicate.
boolean	**anyMatch**(Predicate<? super T> predicate) Returns whether any elements of this stream match the provided predicate.
static <T> Stream.Builder<T>	**builder**() Returns a builder for a Stream.
<R,A> R	**collect**(Collector<? super T,A,R> collector) Performs a **mutable reduction** operation on the elements of this stream using a Collector.
<R> R	**collect**(Supplier<R> supplier, BiConsumer<R,? super T> accumulator, BiConsumer<R,R> combiner) Performs a **mutable reduction** operation on the elements of this stream.
static <T> Stream<T>	**concat**(Stream<? extends T> a, Stream<? extends T> b) Creates a lazily concatenated stream whose elements are all the elements of the first stream followed by all the elements of the second stream.
long	**count**() Returns the count of elements in this stream.
Stream<T>	**distinct**() Returns a stream consisting of the distinct elements (according to **Object.equals(Object)**) of this stream.
static <T> Stream<T>	**empty**() Returns an empty sequential Stream.
Stream<T>	**filter**(Predicate<? super T> predicate) Returns a stream consisting of the elements of this stream that match the given predicate.
Optional<T>	**findAny**() Returns an **Optional** describing some element of the stream, or an empty Optional if the stream is empty.
Optional<T>	**findFirst**() Returns an **Optional** describing the first element of this stream, or an empty Optional if the stream is empty.
<R> Stream<R>	**flatMap**(Function<? super T,? extends Stream<? extends R>> mapper) Returns a stream consisting of the results of replacing each element of this stream with the contents of a mapped stream produced by applying the provided mapping function to each element.

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

Cons of the SearchWithSequentialStreams Class

- *Many* other aggregate operations are part of the Java 8 stream API
  - We'll cover more of them shortly



See *"Java 8 Parallel ImageStreamGang Example"*

End of Java 8 Sequential SearchStreamGang Example (Part 3)