Learning Objectives in this Part of the Lesson

- Know how Java 8 parallel streams are applied in the SearchStreamGang
- Understand the pros & cons of the SearchWithParallelStreams class
- Recognize how a parallel spliterator can improve parallel stream performance

```java
SearchResults searchForPhrase(..., boolean parallel) {
    return new SearchResults(..., StreamSupport.stream(new PhraseMatchSpliterator(...), parallel)
        .collect(toList()));
}
```

This solution addresses a “con” covered in the first part of this lesson
Learning Objectives in this Part of the Lesson

• Know how Java 8 parallel streams are applied in the SearchStreamGang
• Understand the pros & cons of the SearchWithParallelStreams class
• Recognize how a parallel spliterator can improve parallel stream performance
• Understand the pros & cons of the SearchWithParallelSpliterator class

<<Java Class>>

`SearchWithParallelSpliterator`

- processStream(): List<List<SearchResults>>
- processInput(CharSequence): List<SearchResults>
Overview of SearchWith ParallelSpliterator
SearchWithParallelSpliterator is another implementation strategy in the SearchStreamGang program.

See `SearchStreamGang/src/main/java/livelessons/streamgangs/SearchWithParallelSpliterator.java`
Overview of SearchWithParallelSpliterator

- SearchWithParallelSpliterator uses parallel streams in three ways

<<Java Class>>

SearchWithParallelSpliterator

- processStream(): List<List<SearchResults>>
- processInput(CharArray): List<SearchResults>
Overview of SearchWithParallelSpliterator

- SearchWithParallelSpliterator uses parallel streams in three ways
- Search chunks of input in parallel

Diagram:
- Input Strings
  - Input Strings₁
    - Input Strings₁₁
      - Process sequentially
    - Input Strings₁₂
      - Process sequentially
  - Input Strings₂
    - Input Strings₂₁
      - Process sequentially
    - Input Strings₂₂
      - Process sequentially

Legend:
- A pool of worker threads
Overview of SearchWithParallelSpliterator

• SearchWithParallelSpliterator uses parallel streams in three ways

  • Search chunks of input in parallel

  • Search chunks of phrases in parallel

Phrases

Phrases_1

Phrases_1.1

Phrases_1.2

Phrases_2

Phrases_2.1

Phrases_2.2

A pool of worker threads

1.1

1.2

2.1

2.2

Process sequentially
Overview of SearchWithParallelSpliterator

- SearchWithParallelSpliterator uses parallel streams in three ways
  - Search chunks of input in parallel
  - Search chunks of phrases in parallel
  - Search chunks of each input string in parallel
Overview of SearchWithParallelSpliterator

- SearchWithParallelSpliterator uses parallel streams in three ways
  - Search chunks of input in parallel
  - Search chunks of phrases in parallel
  - Search chunks of each input string in parallel

SearchWithParallelSpliterator is thus the most aggressively parallelism strategy!
Overview of SearchWithParallelSpliterator

- The relative contribution of each parallel streams model is shown here:
  - Time for 38 strings = 703 ms (parallelSpliterator|parallelPhrases|parallelInput)
  - Time for 38 strings = 706 ms (sequentialSpliterator|parallelPhrases|parallelInput)
  - Time for 38 strings = 726 ms (parallelSpliterator|sequentialPhrases|parallelInput)
  - Time for 38 strings = 739 ms (sequentialSpliterator|parallelPhrases|sequentialInput)
  - Time for 38 strings = 749 ms (sequentialSpliterator|parallelPhrases|sequentialInput)
  - Time for 38 strings = 759 ms (parallelSpliterator|sequentialPhrases|sequentialInput)
  - Time for 38 strings = 1760 ms (parallelSpliterator|sequentialPhrases|sequentialInput)
  - Time for 38 strings = 3000 ms (sequentialSpliterator|sequentialPhrases|sequentialInput)
Longer input strings leverage the parallel spliterator even better:

- Time for 2 strings = 700 ms ([parallelSpliterator|parallelPhrases|parallelInput])
- Time for 2 strings = 738 ms ([parallelSpliterator|parallelPhrases|sequentialInput])
- Time for 2 strings = 761 ms ([sequentialSpliterator|parallelPhrases|parallelInput])
- Time for 2 strings = 780 ms ([sequentialSpliterator|parallelPhrases|sequentialInput])
- Time for 2 strings = 1008 ms ([parallelSpliterator|sequentialPhrases|parallelInput])
- Time for 2 strings = 1617 ms ([parallelSpliterator|sequentialPhrases|sequentialInput])
- Time for 2 strings = 1986 ms ([sequentialSpliterator|sequentialPhrases|parallelInput])
- Time for 2 strings = 2870 ms ([sequentialSpliterator|sequentialPhrases|sequentialInput])

Longer strings may provide better opportunity to leverage benefits of parallelism.
Overview of SearchWithParallelSpliterator

- SearchWithParallelSpliterator `processInput()` has just one minuscule change

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

    List<SearchResults> results = mPhrasesToFind
        .parallelStream()
        .map(phase ->
            searchForPhrase(phase, input, title, true))
        .filter(not(SearchResults::isEmpty))
        .collect(toList());
    return results;
}
```

The value of “true” triggers the use of a parallel search for a phrase in an input string
Overview of SearchWithParallelSpliterator

- searchForPhrase() uses a parallel spliterator to break the input into “chunks” that are processed in parallel

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

Overview of SearchWithParallelSpliterator

- searchForPhrase() uses a parallel spliterator to break the input into “chunks” that are processed in parallel

```java
SearchResults searchForPhrase(String phrase, CharSequence input,
                               String title, boolean parallel) {
    return new SearchResults(..., ..., phrase, title,
                               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
Overview of SearchWithParallelSpliterator

- searchForPhrase() uses a parallel spliterator to break the input into “chunks” that are processed in parallel

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

The value of “parallel” is true when searchForPhrase() is called in the SearchWithParallelSpliterator program

See docs.oracle.com/javase/8/docs/api/java/util/stream/StreamSupport.html#stream
Using Parallel Spliterator in SearchStreamGang
Using a Parallel Spliterator in SearchStreamGang

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

\[
\text{map}(\text{phrase} \rightarrow \text{searchForPhrase}(\ldots)) \\
\text{filter}(\text{not(SearchResults::isEmpty)}) \\
\text{collect(toList())}
\]

Using a Parallel Spliterator in SearchStreamGang

• SearchStreamGang uses PhraseMatchSpliterator that works for both sequential & parallel streams
• We focused on the sequential portions earlier

See “Java 8 Sequential SearchStreamGang Example (Part 2)”
Using a Parallel Spliterator in SearchStreamGang

- SearchStreamGang uses PhraseMatchSpliterator that works for both sequential & parallel streams
- We focused on the sequential portions earlier
- We’ll cover the parallel portions now

The goal is to further optimize the performance of the parallel streams solution
Using a Parallel Spliterator in SearchStreamGang

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

```
List <String>
```

```
Stream <String>
```

```
Stream <SearchResults>
```

```
List <SearchResults>
```

Stream

<SearchResults>

List

<SearchResults>

• Here's the input/output of PhraseMatchSpliterator for SearchWithParallelSpliterator
Here’s the input/output of PhraseMatchSpliterator for SearchWithParallelSpliterator

“...
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. Your noble son is mad.
Mad call I it; for, to define true madness,
What is't but to be nothing else but mad?
But let that go….”
Using a Parallel Spliterator in SearchStreamGang

- Here’s the input/output of PhraseMatchSpliterator for SearchWithParallelSpliterator

```
“Brevity is the soul of wit” at [54739]

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. Your noble son is mad.
Mad call I it; for, to define true madness,
What is’t but to be nothing else but mad?
But let that go.…”

“Brevity is the soul of wit” at [54739]

When the split occurs efficiently/evenly the speedups can be substantial!
Here’s the input/output of PhraseMatchSpliterator for SearchWithParallelSpliterator

“… 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. Your noble son is mad.
Mad call I it; for, to define true madness,
What is’t but to be nothing else but mad?
But let that go.…”

However, the spliterator must be careful not to split input across phrases…
PhraseMatchSpliterator uses Java regex to create a stream of SearchResults Result objects that match the # of times a phrase appears in an input string.

```java
class PhraseMatchSpliterator implements Spliterator<Result> {
    ...
    PhraseMatchSpliterator(CharSequence input, String phrase) { ... }
    boolean tryAdvance(Consumer<? super Result> action) {
        ... }
    ...
}
```

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

See [SearchStreamGang/src/main/java/livelessons/utils/PhraseMatchSpliterator.java](SearchStreamGang/src/main/java/livelessons/utils/PhraseMatchSpliterator.java)
Using a Parallel 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

```java
class PhraseMatchSpliterator implements Spliterator<Result> {
    ...
    PhraseMatchSpliterator (CharSequence input, String phrase) { ... }
    boolean tryAdvance (Consumer<? super Result> action) {
        ... }
    ...
```

Earlier we analyzed several of its methods that are used in sequential streams

See “Java 8 Sequential SearchStreamGang Example (Part 2)”
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> {
    ...
    Spliterator<Result> trySplit() { ... }
    int computeStartPos(int splitPos) { ... }
    int tryToUpdateSplitPos(int startPos, int splitPos) { ... }
    PhraseMatchSpliterator splitInput(int splitPos) { ... }
    ...
}

We'll now explore its methods that are used for parallel streams.
Using a Parallel 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> {
...
  Spliterator<Result> trySplit() { ... }

  int computeStartPos(int splitPos) { ... }

  int tryToUpdateSplitPos(int startPos,
                          int splitPos) { ... }

  PhraseMatchSpliterator splitInput(int splitPos) { ... }
...

We’ll now explore its methods that are used for parallel streams!

Note that there is no synchronization in any of these methods!!!
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> {
    ...
    Spliterator<Result> trySplit() {
        if (input is below minimum size) return null
        else {
            split input in 2 relatively even-sized chunks
            return a spliterator for "left chunk"
        }
    }
    ...
}

trySplit() attempts to split the input “evenly” so phrases can be matched in parallel
Using a Parallel 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

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

Splits don’t needn’t be perfectly equal in order for the spliterator to be efficient
Using a Parallel Spliterator in SearchStreamGang

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

class PhraseMatchSpliterator implements Spliterator<Result> {
    ...
    Spliterator<Result> trySplit() {
        if (mInput.length() <= mMinSplitSize) return null;
        int startPos, splitPos = mInput.length() / 2;
        if ((startPos = computeStartPos(splitPos)) < 0) return null;
        if ((splitPos = tryToUpdateSplitPos(startPos, splitPos)) < 0) return null;
        return splitInput(splitPos); ...
    }

This code is heavily commented, so please check it out
Using a Parallel 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

```java
class PhraseMatchSpliterator implements Spliterator<Result> {
    ... 
    Spliterator<Result> trySplit() {
        if (mInput.length() <= mMinSplitSize) return null;
        int startPos, splitPos = mInput.length() / 2;

        if ((startPos = computeStartPos(splitPos)) < 0) return null;
        if ((splitPos = tryToUpdateSplitPos(startPos, splitPos)) < 0)
            return null;

        return splitInput(splitPos); ...
    }
}
```

Bail out if input is too small to split further
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> {
    ...
    Spliterator<Result> trySplit() {
        if (mInput.length() <= mMinSplitSize) return null;
        int startPos,
            splitPos = mInput.length() / 2;
        if ((startPos = computeStartPos(splitPos)) < 0) return null;
        if ((splitPos = tryToUpdateSplitPos(startPos, splitPos)) < 0)
            return null;
        return splitInput(splitPos); ...
    }

    Initial guess at the split position

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> {

    ...  

    Spliterator<Result> trySplit() {
        if (mInput.length() <= mMinSplitSize) return null;
        int startPos,
            splitPos = mInput.length() / 2;

        if (startPos = computeStartPos(splitPos)) < 0) return null;

        if (splitPos = tryToUpdateSplitPos(startPos, splitPos)) < 0)
            return null;

        return splitInput(splitPos); ...

    }


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

class PhraseMatchSpliterator implements Spliterator<Result> {
    ...
        Spliterator<Result> trySplit() {
            if (mInput.length() <= mMinSplitSize) return null;
            int startPos,
                splitPos = mInput.length() / 2;

            if ((startPos = computeStartPos(splitPos)) < 0) return null;

            if ((splitPos = tryToUpdateSplitPos(startPos, splitPos)) < 0)
                return null;

            return splitInput(splitPos); ...

        }

    Update splitPos if phrase spans the initial splitPos
Using a Parallel 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.

```java
class PhraseMatchSpliterator implements Spliterator<Result> {
    ...
    Spliterator<Result> trySplit() {
        if (mInput.length() <= mMinSplitSize) return null;
        int startPos,
            splitPos = mInput.length() / 2;
        if ((startPos = computeStartPos(splitPos)) < 0) return null;
        if ((splitPos = tryToUpdateSplitPos(startPos, splitPos)) < 0)
            return null;
        return splitInput(splitPos);...
    }
    Create & return a new spliterator
```
Using a Parallel 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

```java
class PhraseMatchSpliterator implements Spliterator<Result> {
    ...  
    int tryToUpdateSplitPos(int startPos, int splitPos) {
        int endPos = splitPos + mPattern.toString().length();
        if (endPos >= mInput.length()) return -1;
        CharSequence substr = mInput.subSequence(startPos, endPos);
        Matcher pm = mPattern.matcher(substr);
        if (pm.find()) splitPos = startPos + pm.start() + pm.group().length();
        return splitPos;
    }
```

Don’t split a string across a phrase
Using a Parallel 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> {
    ...
    int tryToUpdateSplitPos(int startPos, int splitPos) {
        int endPos =
            splitPos + mPattern.toString().length();
        if (endPos >= mInput.length()) return -1;
        CharSequence substr =
            mInput.subSequence(startPos, endPos);
        Matcher pm = mPattern.matcher(substr);
        if (pm.find()) splitPos = startPos
            + pm.start() + pm.group().length();
        return splitPos;
    }

    Ensure phrase isn’t longer than the input string!
Using a Parallel 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

```java
class PhraseMatchSpliterator implements Spliterator<Result> {
    ...
    int tryToUpdateSplitPos(int startPos, int splitPos) {
        int endPos = splitPos + mPattern.toString().length();
        if (endPos >= mInput.length()) return -1;
        CharSequence substr = mInput.subSequence(startPos, endPos);
        Matcher pm = mPattern.matcher(substr);
        if (pm.find()) splitPos = startPos + pm.start() + pm.group().length();
        return splitPos;
    }
}
```

Check to see if the phrase spans across the initial splitPos
Using a Parallel 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> {
    ...
    int tryToUpdateSplitPos(int startPos, int splitPos) {
        int endPos =
            splitPos + mPattern.toString().length();
        if (endPos >= mInput.length()) return -1;
        CharSequence substr =
            mInput.subSequence(startPos, endPos);
        Matcher pm = mPattern.matcher(substr);
        if (pm.find()) splitPos = startPos + pm.start() + pm.group().length();
        return splitPos;
    }
Using a Parallel 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> {
    ...
    int tryToUpdateSplitPos(int startPos, int splitPos) {
        int endPos = 
            splitPos + mPattern.toString().length();
        if (endPos >= mInput.length()) return -1;
        CharSequence substr = 
            mInput.subSequence(startPos, endPos);
        Matcher pm = mPattern.matcher(substr);
        if (pm.find()) splitPos = startPos 
            + pm.start() + pm.group().length();
        return splitPos;
    }
    
    Return the final splitPos
Using a Parallel 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> {
    
    Spliterator<Result> splitInput(int splitPos) {
        CharSequence lhs = mInput.subSequence(0, splitPos);
        mInput = mInput.subSequence(splitPos, mInput.length());
        mPhraseMatcher = mPattern.matcher(mInput);

        return new PhraseMatchSpliterator(lhs, ...); 
    }

    
    Return a Spliterator that handles “left hand” portion of input, while "this" object handles “right hand” portion of input

    return new PhraseMatchSpliterator(lhs, ...); ...
}
Using a Parallel Spliterator in SearchStreamGang

- The Java 8 parallel streams runtime processes all the spliterator chunks in parallel in the common fork-join pool

This parallelism is in addition to parallelism of input string & phrase chunks!!
Pros of the SearchWithParallelSpliterator Class
Pros of the SearchWithParallelSpliterator Class

- This example shows how a parallel spliterator can help transparently improve program performance
Pros of the SearchWithParallelSpliterator Class

- This example shows how a parallel spliterator can help transparently improve program performance.
• This example shows how a parallel spliterator can help transparently improve program performance

Tests conducted on a 2.9GHz quad-core MacBook Pro with 16 Gbytes of RAM
Pros of the SearchWithParallelSpliterator Class

- This example shows how a parallel spliterator can help transparently improve program performance.
- These speedups occur since the granularity of parallelism is smaller & thus better able to leverage available cores.

See docs.oracle.com/javase/tutorial/collections/streams/parallelism.html
This example also shows that the difference between using sequential vs parallel spliterator can be minuscule!

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

Switching this boolean from “false” to “true” controls whether the spliterator runs sequentially or in parallel.
This example also shows that the difference between using sequential vs parallel spliterator can be minuscule!

SearchResults searchForPhrase(String phrase, CharSequence input, String title, boolean parallel) {
    return new SearchResults(..., ..., phrase, title, StreamSupport.
    stream(new PhraseMatchSpliterator(input, phrase),
    parallel).
    collect(toList()));
}
Cons of the SearchWith ParallelSpliterator Class
The parallel-related portions of PhraseMatchSpliterator are *much* more complicated to program than the sequential-related portions...

```java
class PhraseMatchSpliterator
   implements Spliterator<Result> {
   ...
   Spliterator<Result> trySplit() { ... }
   
   int computeStartPos(int splitPos) { ... }
   
   int tryToUpdateSplitPos(int startPos,
                           int splitPos)
   { ... }
   
   PhraseMatchSpliterator splitInput(int splitPos) { ... }
   ...
```
Cons of the SearchWithParallelSpliterator Class

- The parallel-related portions of PhraseMatchSpliterator are *much* more complicated to program than the sequential-related portions...

```java
class PhraseMatchSpliterator implements Spliterator<Result> {
    ...
    Spliterator<Result> trySplit() { ... }
    int computeStartPos(int splitPos) { ... }
    int tryToUpdateSplitPos(int startPos, int splitPos)
    { ... }
    PhraseMatchSpliterator splitInput(int splitPos) { ... }
    ...
}
```

JUnit tests are extremely useful.

Must split carefully.
Cons of the SearchWithParallelSpliterator Class

- The parallel-related portions of PhraseMatchSpliterator are much more complicated to program than the sequential-related portions...

```java
class PhraseMatchSpliterator
    implements Spliterator<Result> {

    Spliterator<Result> trySplit() { ... }

    int computeStartPos(int splitPos) { ... }

    int tryToUpdateSplitPos(int startPos, int splitPos)
    { ... }

    PhraseMatchSpliterator splitInput(int splitPos) { ... }

    ...

Writing the parallel spliterator took longer than writing the rest of the program!
End of Java 8 Parallel SearchStreamGang Example (Part 2)