Implementing the Java SearchWith ParallelStreams 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 SearchWithParallelStreams are implemented with Java parallel streams

Implementing `processStream()` as a Parallel Stream
Implementing processStream() as a Parallel Stream

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

```java
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

```java
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](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

```java
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

```java
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 phrases

Implementing `processStream()` as a Parallel Stream

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

```java
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 phrases
Implementing `processStream()` as a Parallel Stream

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

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

    return inputList
        .parallelStream()
        .map(this::processInput)
        .collect(toList());
}
```

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

Trigger intermediate operation processing & merge partial results into a single list of lists
Implementing `processInput()` as a Parallel Stream
Implementing processInput() as a Parallel Stream

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

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

    List<SearchResults> results = mPhrasesToFind
        .parallelStream()
        .map(phrase ->
            searchForPhrase(phrase, 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.

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

    List<SearchResults> results = mPhrasesToFind
        .parallelStream()
        .map(phrase ->
            searchForPhrase(phrase, 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 phrase occurrences*

See [docs.oracle.com/javase/8/docs/api/java/util/Spliterator.html](http://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

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

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

The PhraseMatchSpliterator breaks the input into "chunks" that are processed sequentially
Implementing `processInput()` as a Parallel Stream

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

```java
List<SearchResults> processInput(CharSequence inputSeq) {
    String title = getTitle(inputSeq);
    CharSequence input = inputSeq.
    List<SearchResults> results =
        .parallelStream()
        .map(phrase ->
            searchForPhrase(phrase
                .filter(not(SearchResults::
                    .collect(toList()));
        return results;
    }
```

Each phrase (& each input string) is processed in parallel in the common fork-join pool.
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(phrase ->
            searchForPhrase(phrase, input, title, false)
        )
        .filter(not(SearchResults::isEmpty))
        .collect(toList());
    return results;
}
```

Implementing `processInput()` as a Parallel Stream

- Trigger intermediate operation processing
- Merge partial results into a single list

Collectors.toList() returns a non-concurrent collector that obeys encounter order
Implementing `processInput()` as a Parallel Stream

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

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

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

    return results;
}
```

*Return the list of search results*
End of Implementing the Java SearchWithParallelStreams Hook Methods