Learning Objectives in this Part of the Lesson

• Know how aggregate operations & functional programming features are applied in the parallel streams framework
Learning Objectives in this Part of the Lesson

• Know how aggregate operations & functional programming features are applied in the parallel streams framework

• Understand how a parallel stream works
Overview of Java 8
Parallel Streams
Overview of Java 8 Parallel Streams

- A Java 8 parallel stream splits its elements into multiple chunks & uses a thread pool to process these chunks independently.

See docs.oracle.com/javase/tutorial/collections STREAMS/parallelism.html
Overview of Java 8 Parallel Streams

- A Java 8 parallel stream splits its elements into multiple chunks & uses a thread pool to process these chunks independently
- This splitting & thread pool are often invisible to programmers
Overview of Java 8 Parallel Streams

• A Java 8 parallel stream splits its elements into multiple chunks & uses a thread pool to process these chunks independently
  • This splitting & thread pool are often invisible to programmers
  • The *order* in which chunks are processed is likely non-deterministic

![Diagram of parallel stream processing](image)

i.e., programmers often have little/no control over how chunks are processed
Overview of Java 8 Parallel Streams

• A Java 8 parallel stream splits its elements into multiple chunks & uses a thread pool to process these chunks independently
  • This splitting & thread pool are often invisible to programmers
• The order in which chunks are processed is likely non-deterministic
  • This non-determinism is usually a good thing!
Overview of Java 8 Parallel Streams

- A Java 8 parallel stream splits its elements into multiple chunks & uses a thread pool to process these chunks independently.
  - This splitting & thread pool are often invisible to programmers.
  - The order in which chunks are processed is likely non-deterministic.
  - The results of the processing are likely deterministic.

Programmers have more control over how the results are presented.
Overview of Java 8 Parallel Streams

- When a stream executes sequentially all of its aggregate operations run in a single thread

```
List <String>  \rightarrow
Stream <String>  \rightarrow
Stream <SearchResults>  \rightarrow
Stream <SearchResults>  \rightarrow
List <SearchResults>
```

```
Search Phrases

- stream()
- map(phrase -> searchForPhrase(…))
- filter(not(SearchResults::isEmpty))
- collect(toList())
```
Overview of Java 8 Parallel Streams

- When a stream executes in parallel, it is partitioned into multiple substream “chunks” that run in a common fork-join pool

See docs.oracle.com/javase/8/docs/api/java/util/concurrent/ForkJoinPool.html
Overview of Java 8 Parallel Streams

- When a stream executes in parallel, it is partitioned into multiple substream "chunks" that run in a common fork-join pool.

```
Stream <SearchResults>
Stream <String>
List <SearchResults>
List <String>
```

```
parallelStream()
map(phrase -> searchForPhrase(…))
filter(not(SearchResults::isEmpty))
collect(toList())
```

Threads in the pool process different chunks in a non-deterministic order.
Overview of Java 8 Parallel Streams

- When a stream executes in parallel, it is partitioned into multiple substream “chunks” that run in a common fork-join pool.

```
List <String>

Stream <String>

Stream <SearchResults>

Stream <SearchResults>

List <SearchResults>
```

Intermediate operations iterate over & process these chunks in parallel.

```
parallelStream()

map(phrase -> searchForPhrase(…))

filter(not(SearchResults::isEmpty))

collect(toList())
```
Overview of Java 8 Parallel Streams

- When a stream executes in parallel, it is partitioned into multiple substream “chunks” that run in a common fork-join pool.

A terminal operation then combines the chunks into a single result.
Overview of Java 8 Parallel Streams

- When a stream executes in parallel, it is partitioned into multiple substream “chunks” that run in a common fork-join pool

(Stateless) Java 8 lambda expressions & method references are used to pass behaviors
Overview of Java 8 Parallel Streams

- The same aggregate operations can be used for sequential & parallel streams

<table>
<thead>
<tr>
<th>Modifier and Type</th>
<th>Method and Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>boolean</td>
<td>allMatch(Predicate&lt;? super T&gt; predicate) Returns whether all elements of this stream match the provided predicate.</td>
</tr>
<tr>
<td>boolean</td>
<td>anyMatch(Predicate&lt;? super T&gt; predicate) Returns whether any elements of this stream match the provided predicate.</td>
</tr>
<tr>
<td>static &lt;T&gt; Stream.Builder&lt;T&gt;</td>
<td>builder() Returns a builder for a Stream.</td>
</tr>
<tr>
<td>&lt;R,A&gt; R</td>
<td>collect(Collectors&lt;? super T,A,R&gt; collector) Performs a mutable reduction operation on the elements of this stream using a Collector.</td>
</tr>
<tr>
<td>&lt;R&gt; R</td>
<td>collect(Supplier&lt;? extends R&gt; supplier, BiConsumer&lt;? super T,R&gt; accumulator, BiConsumer&lt;? super R,R&gt; combiner) Performs a mutable reduction operation on the elements of this stream.</td>
</tr>
<tr>
<td>static &lt;T&gt; Stream&lt;T&gt;</td>
<td>concat(Stream&lt;? extends T&gt; a, Stream&lt;? extends T&gt; 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.</td>
</tr>
<tr>
<td>long</td>
<td>count() Returns the count of elements in this stream.</td>
</tr>
<tr>
<td>Stream&lt;T&gt;</td>
<td>distinct() Returns a stream consisting of the distinct elements (according to Object.equals(Object)) of this stream.</td>
</tr>
<tr>
<td>static &lt;T&gt; Stream&lt;T&gt;</td>
<td>empty() Returns an empty sequential Stream.</td>
</tr>
<tr>
<td>Stream&lt;T&gt;</td>
<td>filter(Predicate&lt;? super T&gt; predicate) Returns a stream consisting of the elements of this stream that match the given predicate.</td>
</tr>
<tr>
<td>Optional&lt;T&gt;</td>
<td>findAny() Returns an Optional describing some element of the stream, or an empty Optional if the stream is empty.</td>
</tr>
<tr>
<td>Optional&lt;T&gt;</td>
<td>findFirst() Returns an Optional describing the first element of this stream, or an empty Optional if the stream is empty.</td>
</tr>
<tr>
<td>&lt;R&gt; Stream&lt;R&gt;</td>
<td>flatMap(Function&lt;? super T,? extends R&gt; 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.</td>
</tr>
</tbody>
</table>

See docs.oracle.com/javase/8/docs/api/java/util/stream/Stream.html
Overview of Java 8 Parallel Streams

- The same aggregate operations can be used for sequential & parallel streams

**e.g., SearchStreamGang uses the same aggregate operations for both SearchWithSequentialStreams & SearchWithParallelStreams implementations**

- **Search Phrases**
  - stream() vs. parallelStream()
  - map(phrase -> searchForPhrase(…))
  - filter(not(SearchResults::isEmpty))
  - collect(toList())

Overview of Java 8 Parallel Streams

- The same aggregate operations can be used for sequential & parallel streams
- Java 8 streams can thus treat parallelism as an optimization & leverage all available cores!

Overview of Java 8 Parallel Streams

- The same aggregate operations can be used for sequential & parallel streams
- Java 8 streams can thus treat parallelism as an optimization & leverage all available cores!
- Naturally, behaviors run by these aggregate operations must be designed carefully to avoid accessing unsynchronized shared state..

See henrikeichenhardt.blogspot.com/2013/06/why-shared-mutable-state-is-root-of-all.html
Overview of How a Parallel Stream Works
A Java 8 parallel stream implements a "map/reduce" variant optimized for multi-core processors. See en.wikipedia.org/wiki/MapReduce for an overview of how a parallel stream works.
A Java 8 parallel stream implements a “map/reduce” variant optimized for multi-core processors.

It’s actually more like the “split-apply-combine” data analysis strategy.

See www.jstatsoft.org/article/view/v040i01
Overview of How a Parallel Stream Works

- Split-apply-combine works as follows:
  1. **Split** – Recursively partition a data source into independent “chunks”

Overview of How a Parallel Stream Works

- Split-apply-combine works as follows:
  
  1. Split – Recursively partition a data source into independent “chunks”
  
  - Spliterators are defined to partition collections in Java 8

```java
public interface Spliterator<T> {
    boolean tryAdvance(Consumer<? super T> action);
    Spliterator<T> trySplit();
    long estimateSize();
    int characteristics();
}
```

See [docs.oracle.com/javase/8/docs/api/java/util/Spliterator.html](docs.oracle.com/javase/8/docs/api/java/util/Spliterator.html)
Overview of How a Parallel Stream Works

- Split-apply-combine works as follows:

  1. Split – Recursively partition a data source into independent “chunks”
  - Spliterators are defined to partition collections in Java 8
  - You can also define custom spliterators

See [github.com/douglascraigschmidt/LiveLessons/tree/master/SearchStreamSpliterator](https://github.com/douglascraigschmidt/LiveLessons/tree/master/SearchStreamSpliterator)
Overview of How a Parallel Stream Works

- Split-apply-combine works as follows:
  1. Split - Recursively partition a data source into independent “chunks”
    - Spliterators are defined to partition collections in Java 8
    - You can also define custom spliterators
    - Parallel streams perform better on data sources that can be split efficiently & evenly

Split-apply-combine works as follows:

1. **Split** - Recursively partition a data source into independent “chunks”

2. **Apply** - Process chunks independently in a pool of threads

Splitting & applying run simultaneously (after certain limit met), not sequentially

Overview of How a Parallel Stream Works
Overview of How a Parallel Stream Works

• Split-apply-combine works as follows:
  1. Split – Recursively partition a data source into independent “chunks”
  2. Apply – Process chunks independently in a pool of threads

• Programmers have some control over how many threads are in the pool
Overview of How a Parallel Stream Works

- Split-apply-combine works as follows:
  1. Split - Recursively partition a data source into independent “chunks”
  2. Apply - Process chunks independently in a pool of threads
  3. Combine - Join partial results into a single result
Overview of How a Parallel Stream Works

• Split-apply-combine works as follows:
  1. Split – Recursively partition a data source into independent “chunks”
  2. Apply – Process chunks independently in a pool of threads
  3. Combine – Join partial results into a single result
     • Performed by terminal operations like collect() & reduce()

See [www.codejava.net/java-core/collections/java-8-stream-terminal-operations-examples](http://www.codejava.net/java-core/collections/java-8-stream-terminal-operations-examples)
End of Overview of Java 8 Parallel Streams (Part 1)