# Evaluate the Benefits of Java Parallel Streams


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

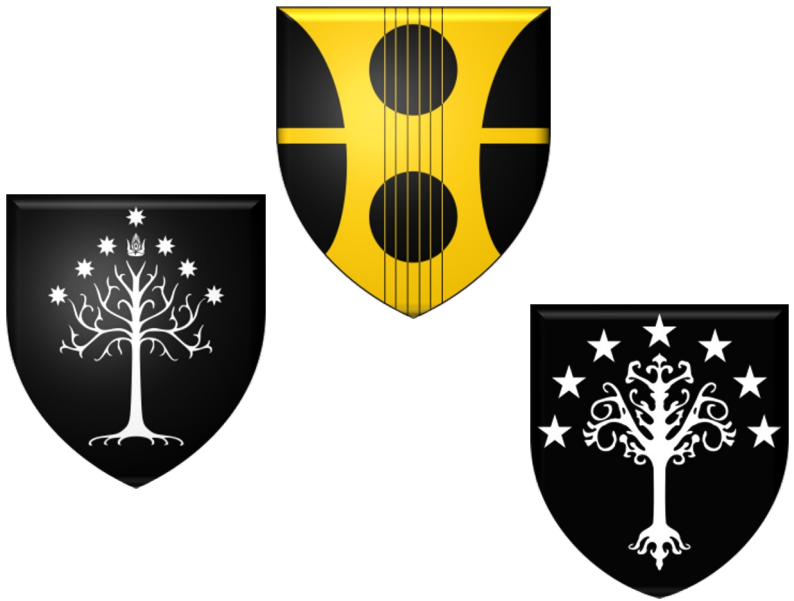

# Learning Objectives in this Part of the Lesson

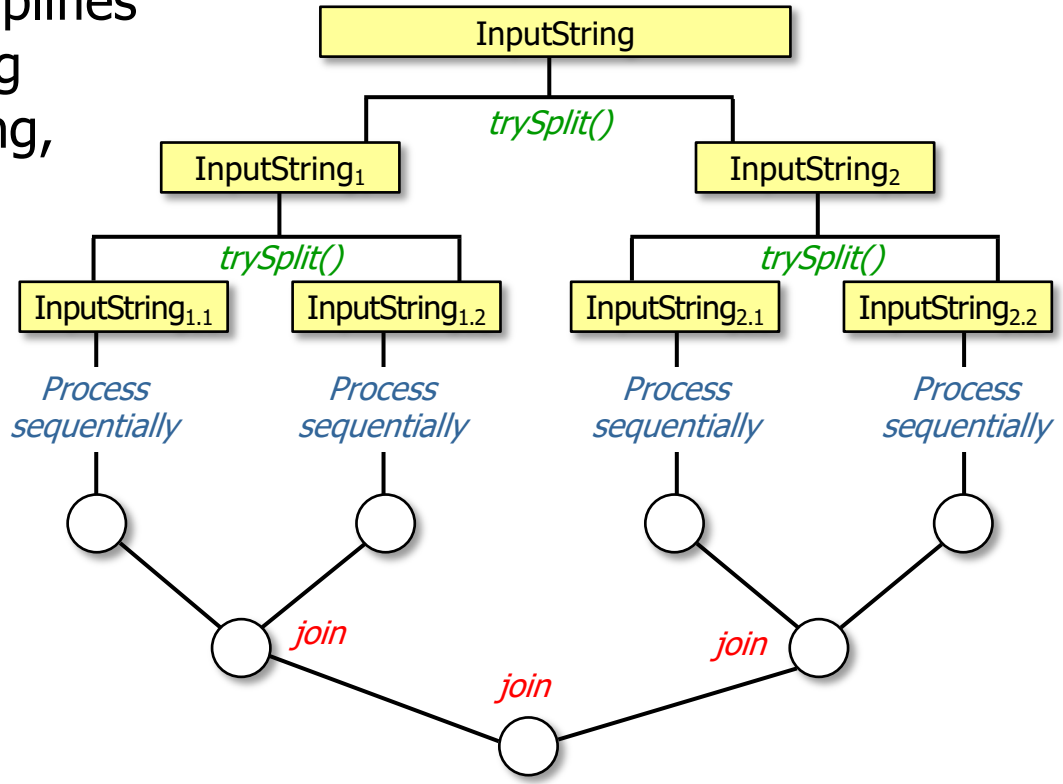
- Evaluate the benefits of Java parallel streams

# Benefits of Java Parallel Streams

# Benefits of Java Parallel Streams

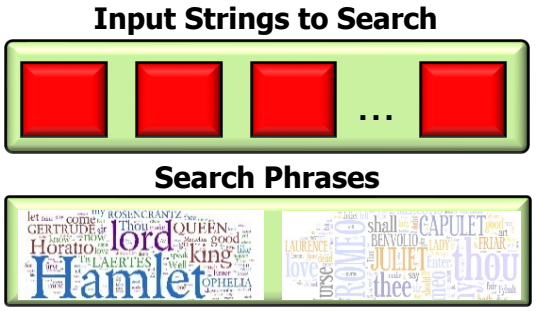
- The Java streams framework simplifies parallel programming by shielding developers from details of splitting, applying, & combining results

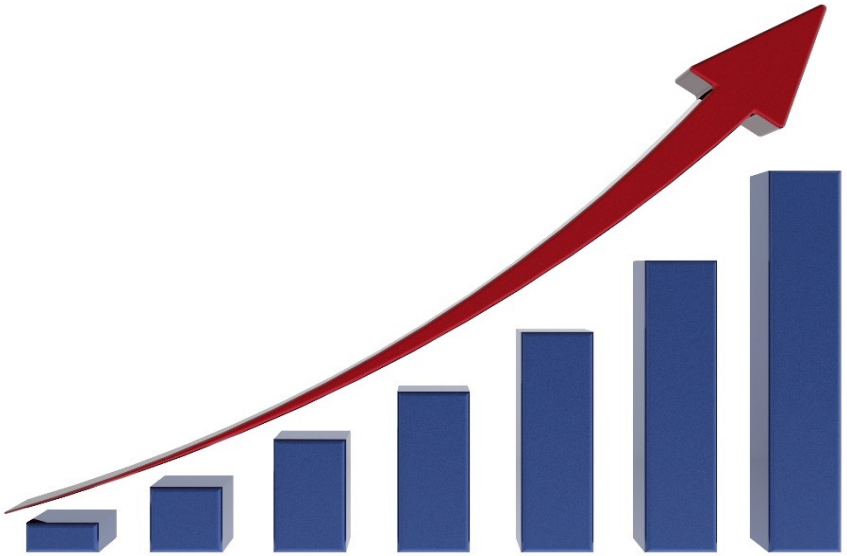


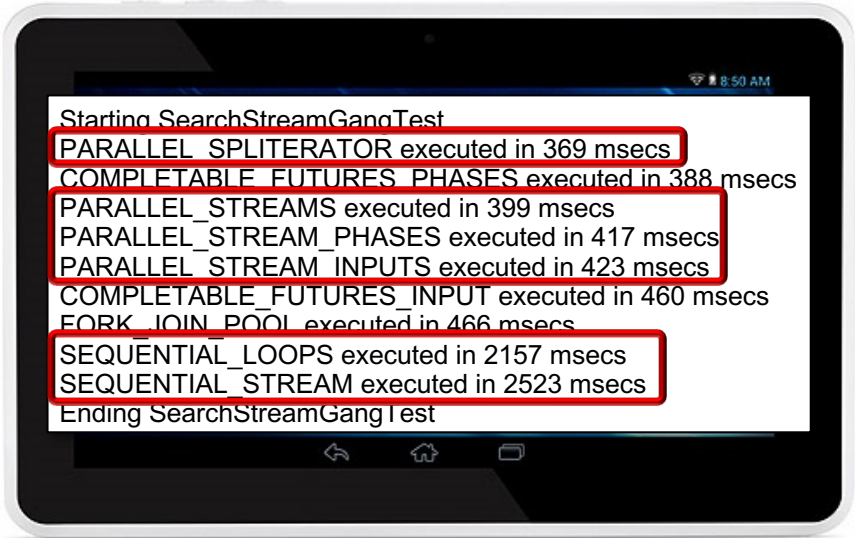


# Benefits of Java Parallel Streams

- Parallel stream implementations are often (much) faster & more scalable than sequential (stream & loops) implementations



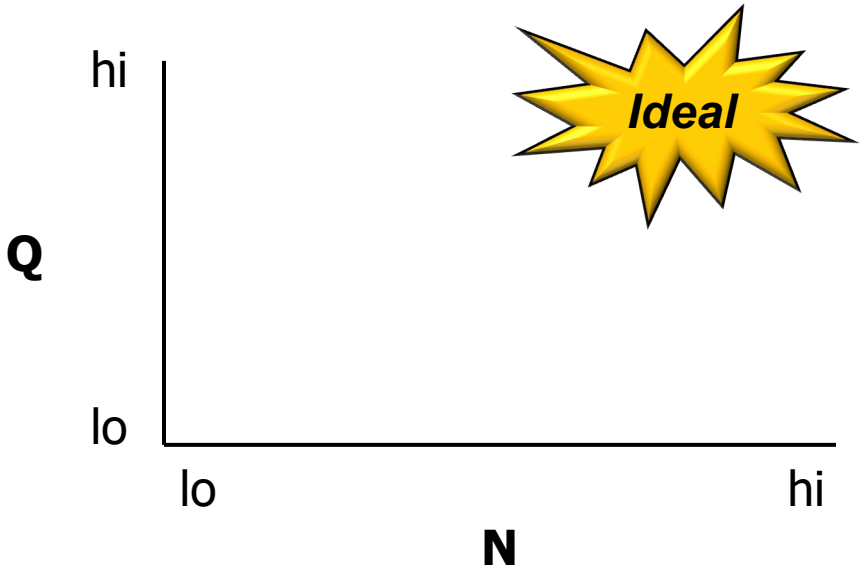






Tests conducted on a 3.2GHz 10-core MacBook Pro with 64 Gbytes of RAM

# Benefits of Java Parallel Streams

- The performance speedup is a largely a function of the partitioning strategy for the input (N), the amount of work performed (Q), & the # of cores




*The NQ model*

- *N is the # of data elements to process per thread*
- *Q quantifies how CPU-intensive the processing is*

# Benefits of Java Parallel Streams

- Apps often don't need explicit synchronization or threading

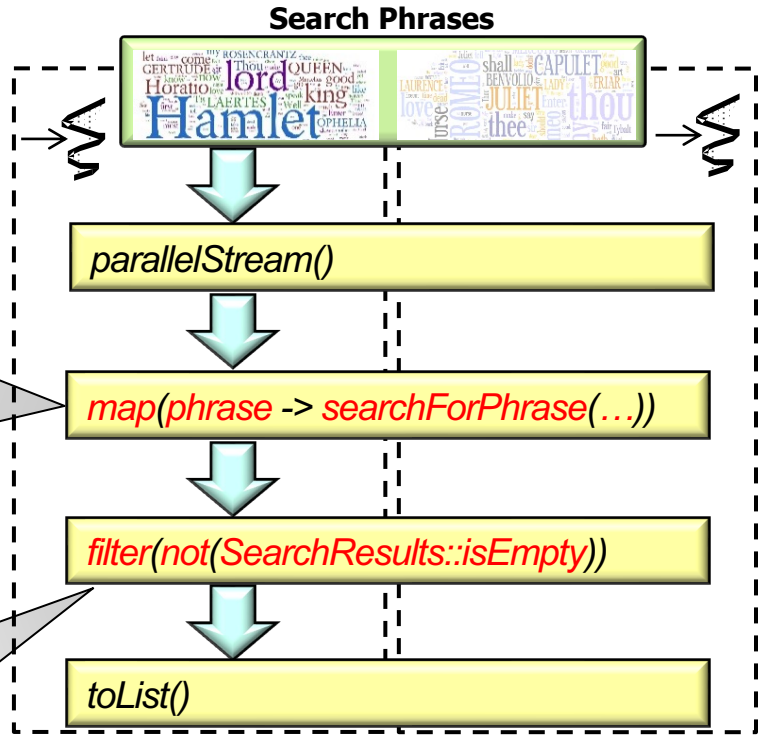
Alleviates many accidental & inherent complexities of concurrency/parallelism

# Benefits of Java Parallel Streams

- Apps often don't need explicit synchronization or threading
  - Stateless behaviors alleviate the need to access shared mutable state

```
return new SearchResults
   (Thread.currentThread().getId(),
    currentCycle(), phrase, title,
    StreamSupport
      .stream(new PhraseMatchSpliterator
                    (input, phrase),
              parallel)
      .collect(toList()));
```

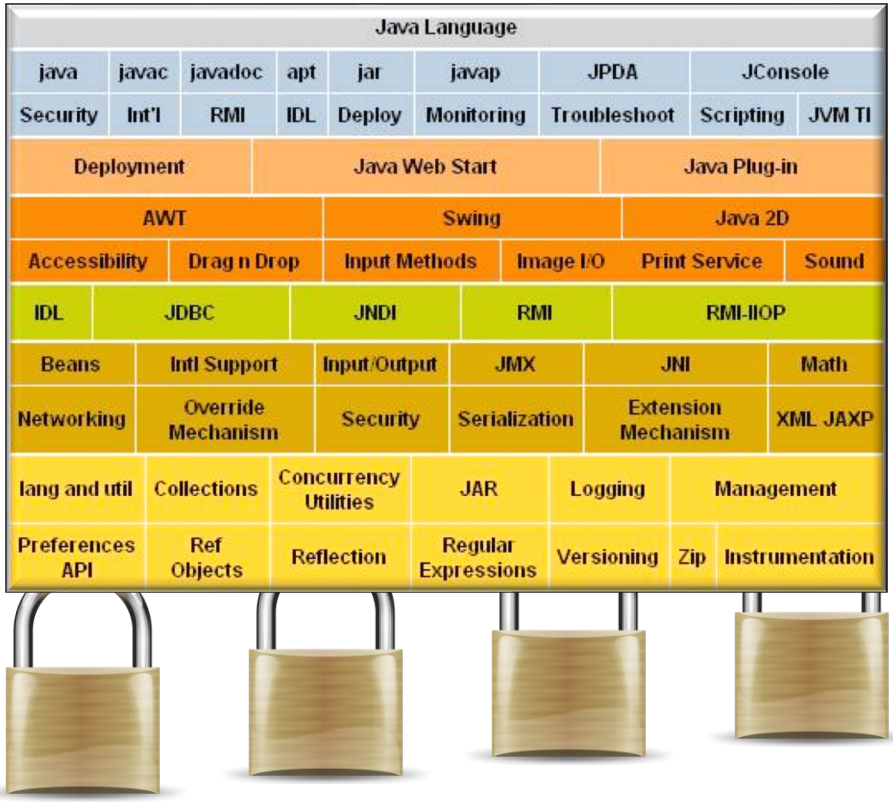
```
return mList.size() == 0;
```




See en.wikipedia.org/wiki/Pure_function

# Benefits of Java Parallel Streams

- Apps often don't need explicit synchronization or threading
  - Stateless behaviors alleviate the need to access shared mutable state
  - The Java class library can be used to handle locking needed to protect shared mutable state



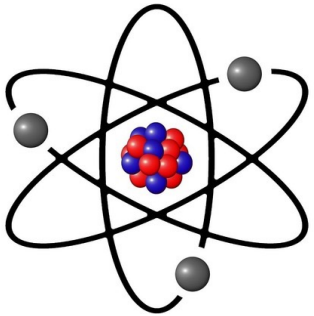

See docs.oracle.com/javase/tutorial/essential/concurrency/collections.html

# Benefits of Java Parallel Streams

- Streams ensures that the structure of sequential & parallel code is the same

```
List<List<SearchResults>>
            processStream() {
  return getInput()
    .stream()
    .map(this::processInput)
    .toList();
}
```

```
List<List<SearchResults>>
            processStream() {
  return getInput()
    .parallelStream()
    .map(this::processInput)
    .toList();
}
```



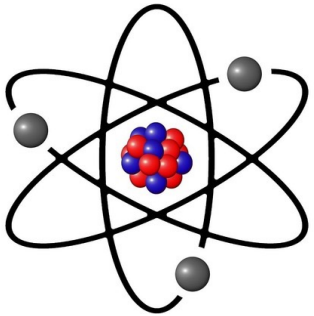
Converting sequential to parallel streams only require minuscule changes!

# Benefits of Java Parallel Streams

- Streams ensures that the structure of sequential & parallel code is the same

```
List<SearchResults> results =
  mPhrasesToFind
    .parallelStream()
    .map(phase ->
         searchForPhrase(...,
                         false))
    .filter(not(SearchResults
                ::isEmpty))
    .toList();
```

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

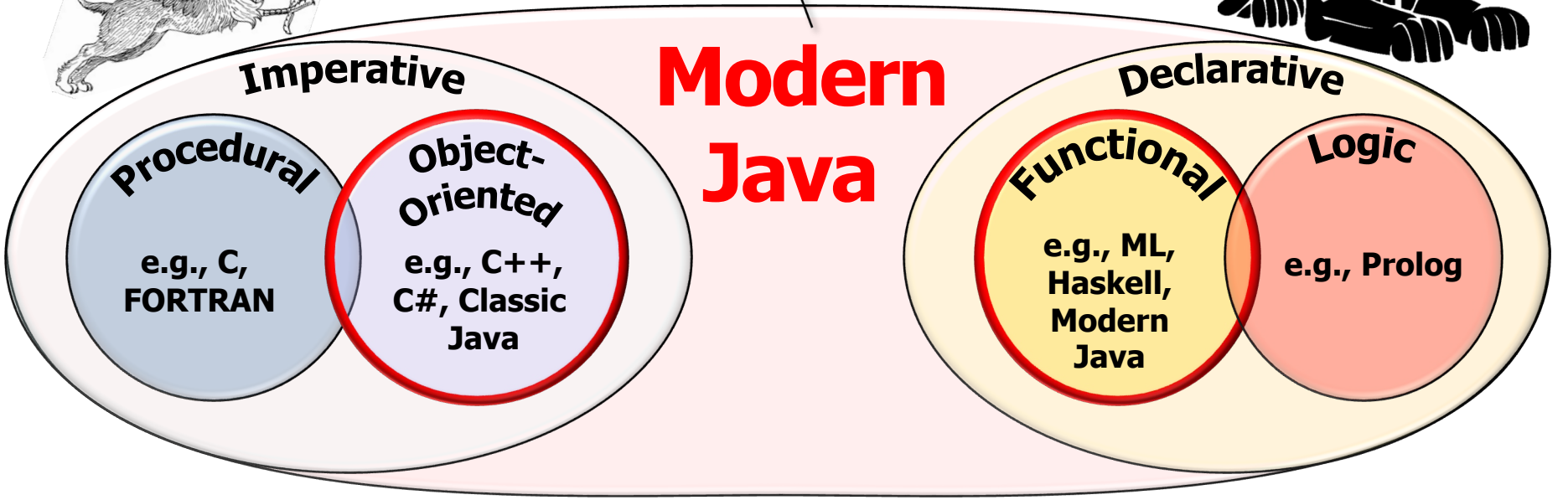


Converting sequential to parallel streams only require minuscule changes!

# Benefits of Java Parallel Streams

- Examples show synergies between functional & object-oriented programming

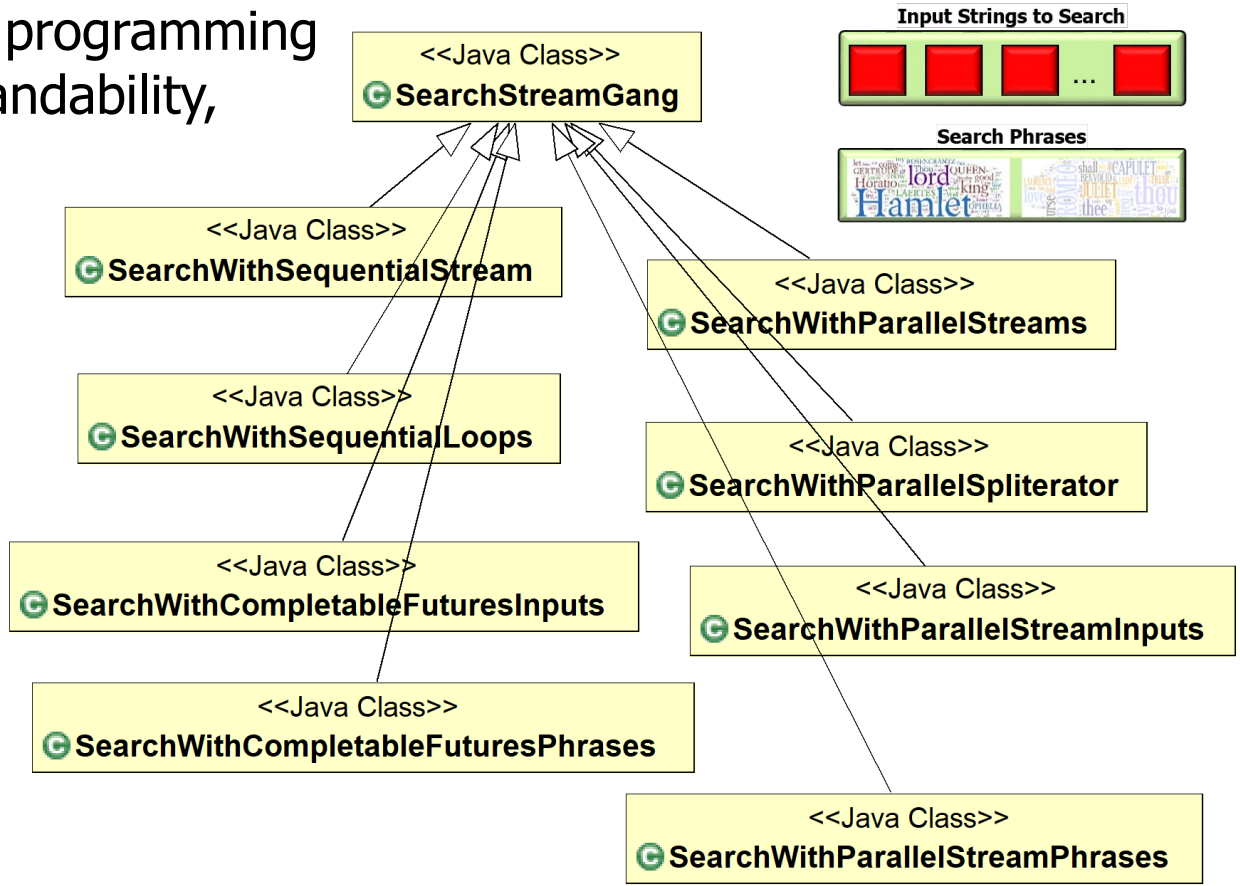
*Modern Java is a "hybrid" that combines both object-oriented & functional language features*




See imdanielsp.medium.com/hybrid-programming-languages-you-are-probably-using-one-77011e12363a

# Benefits of Java Parallel Streams

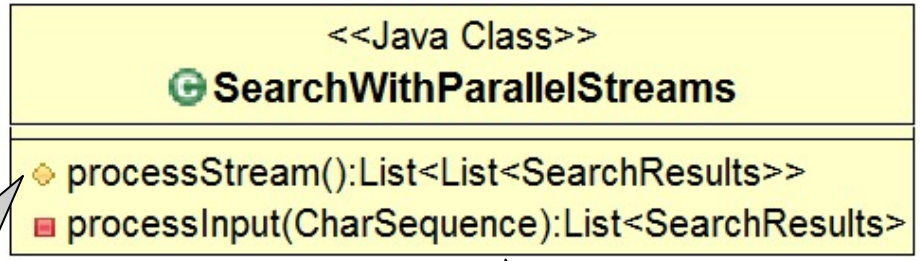
- Object-oriented design & programming features simplify understandability, reusability, & extensibility




Object-oriented techniques emphasize systematic reuse of *structure*

# Benefits of Java Parallel Streams

- Implementing object-oriented hook methods with functional programming features helps to close gap between domain intent & computations


```
getInput()
   .parallelStream()
   .map(this::processInput)
   .toList();
```

```
return mPhrasesToFind
   .parallelStream()
   .map(phrase -> searchForPhrase(phrase, input, title, false))
   .filter(not(SearchResults::isEmpty)
   .toList();
```

# End of Evaluate the Benefits of Java Parallel Streams