Understand the Java Sequential SearchStream Gang Object-Oriented Implementation

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

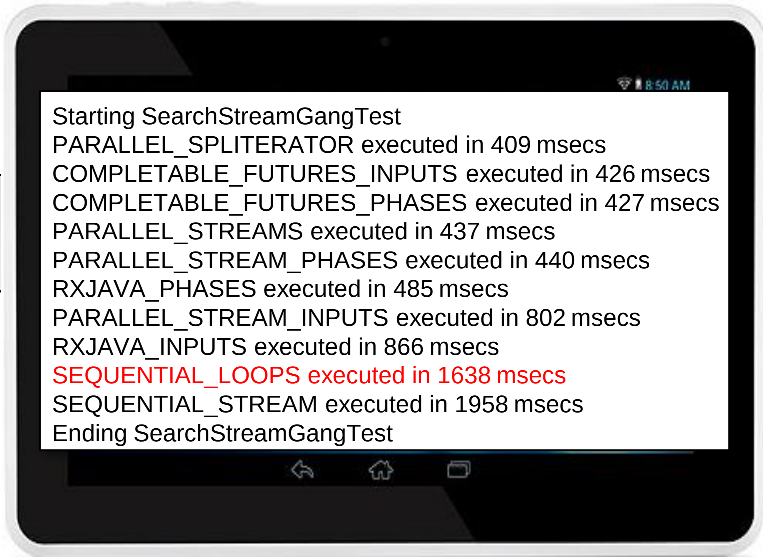
Learning Objectives in this Lesson

- Know how to apply object-oriented features to the SearchStreamGang program

```
protected List<List<SearchResults>> processStream() {
  List<List<SearchResults>> listOfListOfResults =
    new ArrayList<>();

  for (CharSequence inputSeq
         : getInput())
    listOfListOfResults
      .add(processInput(inputSeq);

  return listOfListOfResults;
}
```



```
Starting SearchStreamGangTest
PARALLEL_SPLITERATOR executed in 409 msecs
COMPLETABLE_FUTURES_INPUTS executed in 426 msecs
COMPLETABLE_FUTURES_PHASES executed in 427 msecs
PARALLEL_STREAMS executed in 437 msecs
PARALLEL_STREAM_PHASES executed in 440 msecs
RXJAVA_PHASES executed in 485 msecs
PARALLEL_STREAM_INPUTS executed in 802 msecs
RXJAVA_INPUTS executed in 866 msecs
SEQUENTIAL_LOOPS executed in 1638 msecs
SEQUENTIAL_STREAM executed in 1958 msecs
Ending SearchStreamGangTest
```

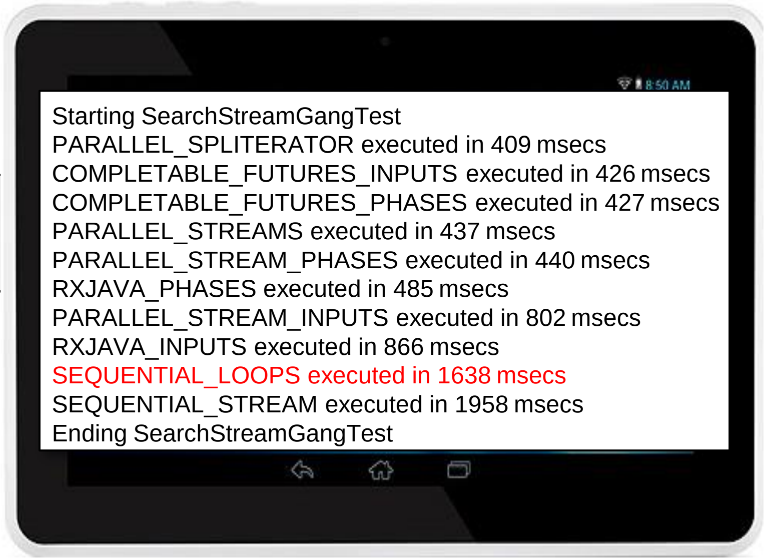
Learning Objectives in this Lesson

- Know how to apply object-oriented features to the SearchStreamGang program

```
protected List<List<SearchResults>> processStream() {
  List<List<SearchResults>> listOfListOfResults =
    new ArrayList<>();

  for (CharSequence inputSeq
         : getInput())
    listOfListOfResults
      .add(processInput(inputSeq);

  return listOfListOfResults;
}
```



```
Starting SearchStreamGangTest
PARALLEL_SPLITERATOR executed in 409 msecs
COMPLETABLE_FUTURES_INPUTS executed in 426 msecs
COMPLETABLE_FUTURES_PHASES executed in 427 msecs
PARALLEL_STREAMS executed in 437 msecs
PARALLEL_STREAM_PHASES executed in 440 msecs
RXJAVA_PHASES executed in 485 msecs
PARALLEL_STREAM_INPUTS executed in 802 msecs
RXJAVA_INPUTS executed in 866 msecs
SEQUENTIAL_LOOPS executed in 1638 msecs
SEQUENTIAL_STREAM executed in 1958 msecs
Ending SearchStreamGangTest
```

Also understand the limitations with object-oriented programming..

An Object-Oriented Implementation of processStream()

Object-Oriented Implementation of processStream()

- processStream() sequentially searches for phrases in lists of input "strings"

```
protected List<List<SearchResults>> processStream() {

  List<List<SearchResults>> listOfListOfResults =
    new ArrayList<>();

  for (CharSequence inputSeq : getInput())
    listOfListOfResults
      .add(processInput(inputSeq));

  return listOfListOfResults;
}
```

This method is not actually implemented with a stream..

See livelessons/streamgangs/SearchWithSequentialLoops.java

Object-Oriented Implementation of processStream()

- processStream() sequentially searches for phrases in lists of input "strings"

```
protected List<List<SearchResults>> processStream() {

  List<List<SearchResults>> listOfListOfResults =
    new ArrayList<>();

  for (CharSequence inputSeq : getInput())
    listOfListOfResults
      .add(processInput(inputSeq));

  return listOfListOfResults;
}
```

Results are stored in a list of lists of search results

Object-Oriented Implementation of processStream()

- processStream() sequentially searches for phrases in lists of input "strings"

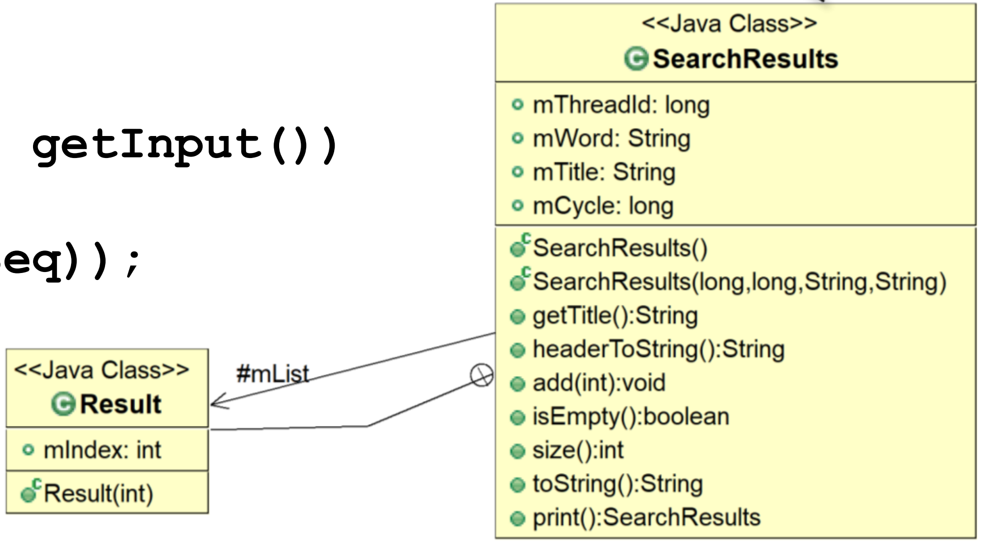
```
protected List<List<SearchResults>> processStream() {

  List<List<SearchResults>> listOfListOfResults =
    new ArrayList<>();

  for (CharSequence inputSeq : getInput())
    listOfListOfResults
      .add(processInput(inputSeq));

  return listOfListOfResults;
}
```

Stores # of times a phrase appeared in an input string



See livelessons/utils/SearchResults.java

Object-Oriented Implementation of processStream()

- processStream() sequentially searches for phrases in lists of input "strings"

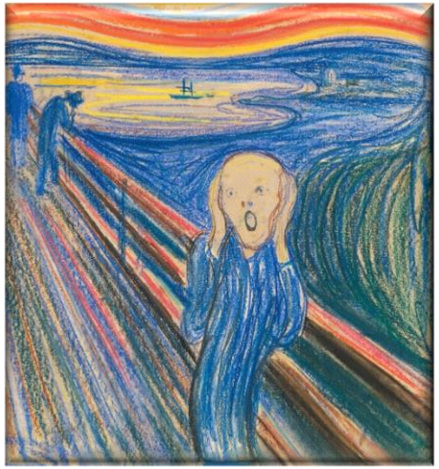
```
protected List<List<SearchResults>> processStream() {
```

Must remember to initialize this list or chaos & insanity will result..

```
  List<List<SearchResults>> listOfListOfResults =
    new ArrayList<>();

  for (CharSequence inputSeq : getInput())
    listOfListOfResults
      .add(processInput(inputSeq));

  return listOfListOfResults;
}
```



Object-Oriented Implementation of processStream()

- processStream() sequentially searches for phrases in lists of input "strings"

```
protected List<List<SearchResults>> processStream() {


  List<List<SearchResults>> listOfListOfResults =
    new ArrayList<>();

  for (CharSequence inputSeq : getInput())
    listOfListOfResults
      .add(processInput(inputSeq));

  return listOfListOfResults;
}
```

Explicitly loop thru all the works of Shakespeare

Object-Oriented Implementation of processStream()

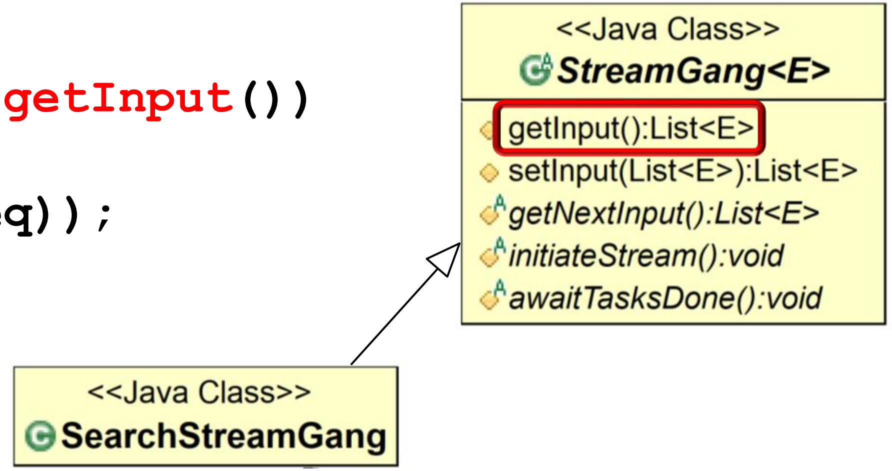
- processStream() sequentially searches for phrases in lists of input "strings"

```
protected List<List<SearchResults>> processStream() {


  List<List<SearchResults>> listOfListOfResults =
    new ArrayList<>();

  for (CharSequence inputSeq : getInput())
    listOfListOfResults
      .add(processInput(inputSeq));

  return listOfListOfResults;
}
```



The getInput() method is defined in the StreamGang framework

Object-Oriented Implementation of processStream()

- processStream() sequentially searches for phrases in lists of input "strings"

```
protected List<List<SearchResults>> processStream() {


  List<List<SearchResults>> listOfListOfResults =
    new ArrayList<>();

  for (CharSequence inputSeq : getInput())
    listOfListOfResults
      .add(processInput(inputSeq));

  return listOfListOfResults;
}
```

CharSequence optimizes subSequence() to avoid memory copies (cf. String substring())

See www.javaspecialists.eu/archive/Issue230.html

Object-Oriented Implementation of processStream()

- processStream() sequentially searches for phrases in lists of input “strings”

```
protected List<List<SearchResults>> processStream() {


   List<List<SearchResults>> listOfListOfResults =
     new ArrayList<>();

   for (CharSequence inputSeq : getInput())
     listOfListOfResults
       .add(processInput(inputSeq));

   return listOfListOfResults;
}
```

Applying processInput() to each work

processInput() returns a list of SearchResults—one list for each input string

Object-Oriented Implementation of processStream()

- processStream() sequentially searches for phrases in lists of input "strings"

```
protected List<List<SearchResults>> processStream() {


  List<List<SearchResults>> listOfListOfResults =
    new ArrayList<>();

  for (CharSequence inputSeq : getInput())
    listOfListOfResults
      .add(processInput(inputSeq));

  return listOfListOfResults;
}
```

Add the list of search results to the list of lists

Object-Oriented Implementation of processStream()

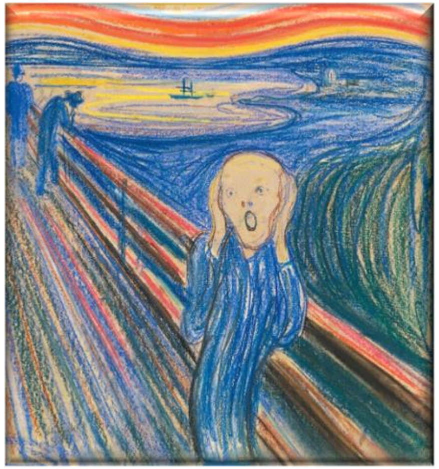
- processStream() sequentially searches for phrases in lists of input "strings"

```
protected List<List<SearchResults>> processStream() {


  List<List<SearchResults>> listOfListOfResults =
    new ArrayList<>();

  for (CharSequence inputSeq : getInput())
    listOfListOfResults
      .add(processInput(inputSeq));

  return listOfListOfResults;
}
```



If `listOfListOfResults` isn't initialized properly chaos & insanity will result..

Object-Oriented Implementation of processStream()

- processStream() sequentially searches for phrases in lists of input "strings"

```
protected List<List<SearchResults>> processStream() {


  List<List<SearchResults>> listOfListOfResults =
    new ArrayList<>();

  for (CharSequence inputSeq : getInput())
    listOfListOfResults
      .add(processInput(inputSeq));

  return listOfListOfResults;
}
```

This code implements the accumulator anti-pattern, which doesn't scale easily

Object-Oriented Implementation of processStream()

- processStream() sequentially searches for phrases in lists of input "strings"

```
protected List<List<SearchResults>> processStream() {


  List<List<SearchResults>> listOfListOfResults =
    new ArrayList<>();

  for (CharSequence inputSeq : getInput())
    listOfListOfResults
      .add(processInput(inputSeq));

  return listOfListOfResults;
}
```

Return the search results

Object-Oriented Implementation of processStream()

- processStream() sequentially searches for phrases in lists of input "strings"

```
protected List<List<SearchResults>> processStream() {
```

Indicates how many times a search phrase appeared in each input string

```
  List<List<SearchResults>> listOfListOfResults =
    new ArrayList<>();

  for (CharSequence inputSeq : getInput())
    listOfListOfResults
      .add(processInput(inputSeq));

  return listOfListOfResults;
}
```

Object-Oriented Implementation of processInput()

Object-Oriented Implementation of processInput()

- processInput() searches an input string for all occurrences of phrases to find

```
List<SearchResults> processInput(CharSequence inputSeq) {
  String title = getTitle(inputSeq);
  CharSequence input = inputSeq.
    subSequence(title.length(), inputSeq.length());
  List<SearchResults> listOfResults = new ArrayList<>();

  for (String phrase : mPhrasesToFind) {

    SearchResults res =
      searchForPhrase(phrase, input, title);
    if (res.size() > 0) listOfResults.add(res);
  }
  return listOfResults;
}
```

Object-Oriented Implementation of processInput()

- processInput() searches an input string for all occurrences of phrases to find

```
List<SearchResults> processInput(CharSequence inputSeq) {
  String title = getTitle(inputSeq);
  CharSequence input = inputSeq.
    subSequence(title.length(), inputSeq.length());
  List<SearchResults> listOfResults = new ArrayList<>();

  for (String phrase : mPhrasesToFind) {

    SearchResults res =
      searchForPhrase(phrase, input, title);
    if (res.size() > 0) listOfResults.add(res);
  }
  return listOfResults;
}
```

The input is a section of a text file managed by the test driver program

Object-Oriented Implementation of processInput()

- processInput() searches an input string for all occurrences of phrases to find

```
List<SearchResults> processInput(CharSequence inputSeq) {
  String title = getTitle(inputSeq);
  CharSequence input = inputSeq.
    subSequence(title.length(), inputSeq.length());
  List<SearchResults> listOfResults = new ArrayList<>();

  for (String phrase : mPhrasesToFind) {

    SearchResults res =
      searchForPhrase(phrase, input, title);
    if (res.size() > 0) listOfResults.add(res);
  }
  return listOfResults;
}
```

The input is split into two parts

Object-Oriented Implementation of processInput()

- processInput() searches an input string for all occurrences of phrases to find

```
List<SearchResults> processInput(CharSequence inputSeq) {
  String title = getTitle(inputSeq);
  CharSequence input = inputSeq.
    subSequence(title.length(), inputSeq.length());
  List<SearchResults> listOfResults = new ArrayList<>();

  for (String phrase : mPhrasesToFind) {

    SearchResults res =
      searchForPhrase(phrase, input, title);
    if (res.size() > 0) listOfResults.add(res);
  }
  return listOfResults;
}
```

subSequence() avoids overhead

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

Object-Oriented Implementation of processInput()

- processInput() searches an input string for all occurrences of phrases to find

```
List<SearchResults> processInput(CharSequence inputSeq) {
  String title = getTitle(inputSeq);
  CharSequence input = inputSeq.
    subSequence(title.length(), inputSeq.length());
  List<SearchResults> listOfResults = new ArrayList<>();

  for (String phrase : mPhrasesToFind) {

    SearchResults res =
      searchForPhrase(phrase, input, title);
    if (res.size() > 0) listOfResults.add(res);
  }
  return listOfResults;
}
```

Create empty list to store results

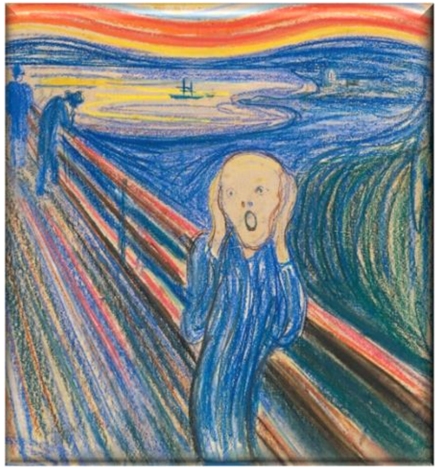
Object-Oriented Implementation of processInput()

- processInput() searches an input string for all occurrences of phrases to find

```
List<SearchResults> processInput(CharSequence inputSeq) {
  String title = getTitle(inputSeq);
  CharSequence input = inputSeq.
    subSequence(title.length(), inputSeq.length());
  List<SearchResults> listOfResults = new ArrayList<>();

  for (String phrase : mPhrasesToFind) {

    SearchResults res =
      searchForPhrase(phrase, input, title);
    if (res.size() > 0) listOfResults.add(res);
  }
  return listOfResults;
}
```



Again, you must remember to initialize this list or chaos & insanity will result..

Object-Oriented Implementation of processInput()

- processInput() searches an input string for all occurrences of phrases to find

```
List<SearchResults> processInput(CharSequence inputSeq) {
  String title = getTitle(inputSeq);
  CharSequence input = inputSeq.
    subSequence(title.length(), inputSeq.length());
  List<SearchResults> listOfResults = new ArrayList<>();

  for (String phrase : mPhrasesToFind) {

    SearchResults res =
      searchForPhrase(phrase, input, title);
    if (res.size() > 0) listOfResults.add(res);
  }
  return listOfResults;
}
```

Explicitly loop thru all phrases to search input

Object-Oriented Implementation of processInput()

- processInput() searches an input string for all occurrences of phrases to find

```
List<SearchResults> processInput(CharSequence inputSeq) {
  String title = getTitle(inputSeq);
  CharSequence input = inputSeq.
    subSequence(title.length(), inputSeq.length());
  List<SearchResults> listOfResults = new ArrayList<>();

  for (String phrase : mPhrasesToFind) {

    SearchResults res =
      searchForPhrase(phrase, input, title);
    if (res.size() > 0) listOfResults.add(res);
  }
  return listOfResults;
}
```

Apply this function to all phrases in the input

Object-Oriented Implementation of processInput()

- processInput() searches an input string for all occurrences of phrases to find

```
List<SearchResults> processInput(CharSequence inputSeq) {
  String title = getTitle(inputSeq);
  CharSequence input = inputSeq.
    subSequence(title.length(), inputSeq.length());
  List<SearchResults> listOfResults = new ArrayList<>();

  for (String phrase : mPhrasesToFind) {

    SearchResults res =
      searchForPhrase(phrase, input, title);
    if (res.size() > 0) listOfResults.add(res);
  }
  return listOfResults;
}
```

Explicitly check if results are non-empty

Object-Oriented Implementation of processInput()

- processInput() searches an input string for all occurrences of phrases to find

```
List<SearchResults> processInput(CharSequence inputSeq) {
  String title = getTitle(inputSeq);
  CharSequence input = inputSeq.
    subSequence(title.length(), inputSeq.length());
  List<SearchResults> listOfResults = new ArrayList<>();

  for (String phrase : mPhrasesToFind) {

    SearchResults res =
      searchForPhrase(phrase, input, title);
    if (res.size() > 0) listOfResults.add(res);
  }
  return listOfResults;
}
```

Add non-empty SearchResults to the list of search results

Object-Oriented Implementation of processInput()

- processInput() searches an input string for all occurrences of phrases to find

```
List<SearchResults> processInput(CharSequence inputSeq) {
  String title = getTitle(inputSeq);
  CharSequence input = inputSeq.
    subSequence(title.length(), inputSeq.length());
  List<SearchResults> listOfResults = new ArrayList<>();

  for (String phrase : mPhrasesToFind) {

    SearchResults res =
      searchForPhrase(phrase, input, title);
    if (res.size() > 0) listOfResults.add(res);
  }
  return listOfResults;
}
```

This code has control constructs, which makes it hard to read linearly!

Object-Oriented Implementation of processInput()

- processInput() searches an input string for all occurrences of phrases to find

```
List<SearchResults> processInput(CharSequence inputSeq) {
  String title = getTitle(inputSeq);
  CharSequence input = inputSeq.
    subSequence(title.length(), inputSeq.length());
  List<SearchResults> listOfResults = new ArrayList<>();

  for (String phrase : mPhrasesToFind) {

    SearchResults res =
      searchForPhrase(phrase, input, title);
    if (res.size() > 0) listOfResults.add(res);
  }
  return listOfResults;
}
```

This code implements the accumulator anti-pattern, which doesn't scale easily

Object-Oriented Implementation of processInput()

- processInput() searches an input string for all occurrences of phrases to find

```
List<SearchResults> processInput(CharSequence inputSeq) {
  String title = getTitle(inputSeq);
  CharSequence input = inputSeq.
    subSequence(title.length(), inputSeq.length());
  List<SearchResults> listOfResults = new ArrayList<>();

  for (String phrase : mPhrasesToFind) {

    SearchResults res =
      searchForPhrase(phrase, input, title);
    if (res.size() > 0) listOfResults.add(res);
  }
  return listOfResults;
}
```

Return list result to processStream()

Object-Oriented Implementation of searchForPhrase()

Object-Oriented Implementation of searchForPhrase()

- searchForPhrase() uses a regex to find phrases in an input string

```
SearchResults searchForPhrase(String phr, String in, String ti){
  List<SearchResults.Result> results = new ArrayList<>();
  String regex = phr.trim()
    .replaceAll("\\s+", "\\\\s+")
    .replace("?", "\\?");

  Pattern pat = compile(regex, CASE_INSENSITIVE | DOTALL);
  Matcher match = pat.matcher(in);


  while (match.find()) results.add(new Result(match.start()));

  return new SearchResults(Thread.currentThread().getId(),
                                         0, phr, ti, results); ...
```

Object-Oriented Implementation of searchForPhrase()

- searchForPhrase() uses a regex to find phrases in an input string

```
SearchResults searchForPhrase(String phr, String in, String ti){
  List<SearchResults.Result> results = new ArrayList<>();
  String regex = phr.trim()
    .replaceAll("\\s+", "\\\\s+")
    .replace("?", "\\?");

  Pattern pat = compile(regex, CASE_INSENSITIVE | DOTALL);
  Matcher match = pat.matcher(in);


  while (match.find()) results.add(new Result(match.start()));

  return new SearchResults(Thread.currentThread().getId(),
                                   0, phr, ti, results); ...
```

Explicitly create an empty list to store search results

Object-Oriented Implementation of searchForPhrase()

- searchForPhrase() uses a regex to find phrases in an input string

```
SearchResults searchForPhrase(String phr, String in, String ti){
  List<SearchResults.Result> results = new ArrayList<>();
  String regex = phr.trim()
    .replaceAll("\\s+", "\\\\s+")
    .replace("?", "\\?");

  Pattern pat = compile(regex, CASE_INSENSITIVE | DOTALL);
  Matcher match = pat.matcher(in);


  while (match.find()) results.add(new Result(match.start()));

  return new SearchResults(Thread.currentThread().getId(),
                                         0, phr, ti, results); ...
```

Create regex to search for phrase by collapsing extraneous whitespace & quoting any/all question marks

Object-Oriented Implementation of searchForPhrase()

- searchForPhrase() uses a regex to find phrases in an input string

```
SearchResults searchForPhrase(String phr, String in, String ti){
  List<SearchResults.Result> results = new ArrayList<>();
  String regex = phr.trim()
    .replaceAll("\\s+", "\\\\s+")
    .replace("?", "\\?");

  Pattern pat = compile(regex, CASE_INSENSITIVE | DOTALL);
  Matcher match = pat.matcher(in);


  while (match.find()) results.add(new Result(match.start()));

  return new SearchResults(Thread.currentThread().getId(),
                                        0, phr, ti, results); ...
```

Compile the regex

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

Object-Oriented Implementation of searchForPhrase()

- searchForPhrase() uses a regex to find phrases in an input string

```
SearchResults searchForPhrase(String phr, String in, String ti){
  List<SearchResults.Result> results = new ArrayList<>();
  String regex = phr.trim()
    .replaceAll("\\s+", "\\\\s+")
    .replace("?", "\\?");

  Pattern pat = compile(regex, CASE_INSENSITIVE | DOTALL);
  Matcher match = pat.matcher(in);

  while (match.find()) results.add(new Result(match.start()));

  return new SearchResults(Thread.currentThread().getId(),
                                        0, phr, ti, results); ...
```

Create a matcher

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

Object-Oriented Implementation of searchForPhrase()

- searchForPhrase() uses a regex to find phrases in an input string

```
SearchResults searchForPhrase(String phr, String in, String ti){
  List<SearchResults.Result> results = new ArrayList<>();
  String regex = phr.trim()
    .replaceAll("\\s+", "\\\\s+")
    .replace("?", "\\?");

  Pattern pat = compile(regex, CASE_INSENSITIVE | DOTALL);
  Matcher match = pat.matcher(in);

  while (match.find()) results.add(new Result(match.start()));

  return new SearchResults(Thread.currentThread().getId(),
                                        0, phr, ti, results); ...
```

Explicitly loop thru all regex matches

Object-Oriented Implementation of searchForPhrase()

- searchForPhrase() uses a regex to find phrases in an input string

```
SearchResults searchForPhrase(String phr, String in, String ti){
  List<SearchResults.Result> results = new ArrayList<>();
  String regex = phr.trim()
    .replaceAll("\\s+", "\\\\s+")
    .replace("?", "\\?");

  Pattern pat = compile(regex, CASE_INSENSITIVE | DOTALL);
  Matcher match = pat.matcher(in);

  while (match.find()) results.add(new Result(match.start()));

  return new SearchResults(Thread.currentThread().getId(),
                                       0, phr, ti, results); ...
```

Explicitly add starting index of all regex matches

Object-Oriented Implementation of searchForPhrase()

- searchForPhrase() uses a regex to find phrases in an input string

```
SearchResults searchForPhrase(String phr, String in, String ti){
  List<SearchResults.Result> results = new ArrayList<>();
  String regex = phr.trim()
    .replaceAll("\\s+", "\\\\s+")
    .replace("?", "\\?");

  Pattern pat = compile(regex, CASE_INSENSITIVE | DOTALL);
  Matcher match = pat.matcher(in);


  while (match.find()) results.add(new Result(match.start()));

  return new SearchResults(Thread.currentThread().getId(),
                                        0, phr, ti, results); ...
```

Return search results to processInput()

End of Understand the Java Sequential SearchStreamGang Object-Oriented Implemention