Java Sequential SearchStreamGang Example: Introduction

Douglas C. Schmidt

<u>d.schmidt@vanderbilt.edu</u>

www.dre.vanderbilt.edu/~schmidt



Professor of Computer Science

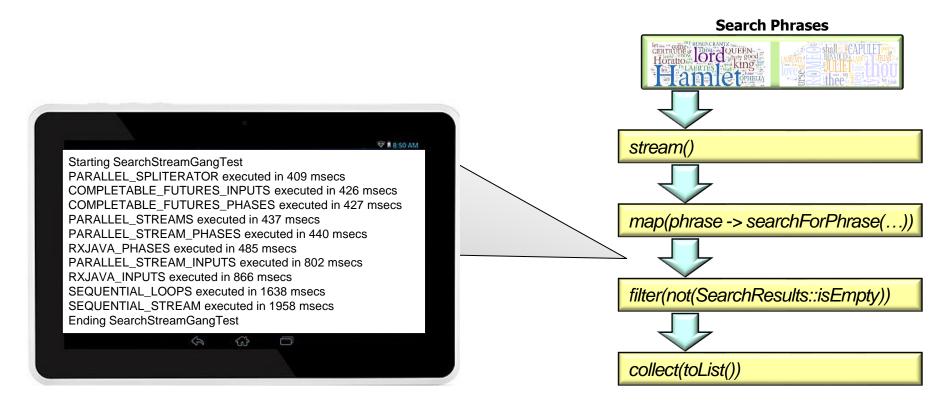
Institute for Software Integrated Systems

Vanderbilt University Nashville, Tennessee, USA



Learning Objectives in this Part of the Lesson

Understand the design of the SearchStreamGang program

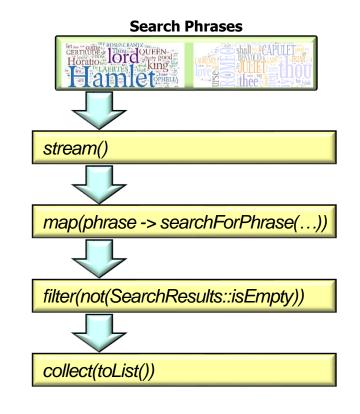


See github.com/douglascraigschmidt/LiveLessons/tree/master/SearchStreamGang

Learning Objectives in this Part of the Lesson

Understand the design of the SearchStreamGang program





This example is more interesting than the SimpleSearchStream program

SearchStreamGang revises SearchTaskGang to use functional programming & streams instead of OO prorgramming

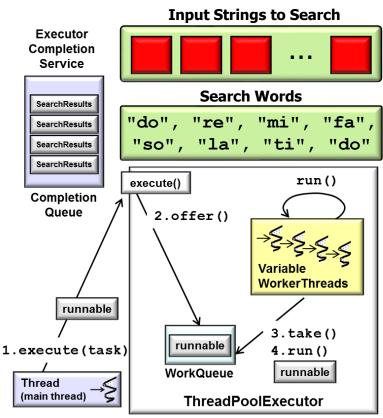
Input Strings to Search Executor Completion . . . Service **Search Words** SearchResults "re", "mi", "fa", SearchResults "so", "la", "ti", "do" SearchResults SearchResults run() execute() Completion Queue 2.offer() WorkerThreads runnable 3.take() runnable 1.execute(task) 4.run() WorkQueue runnable Thread (main thread) **ThreadPoolExecutor**

SearchStreamGang revises SearchTaskGang to use functional programming &

streams instead of OO prorgramming

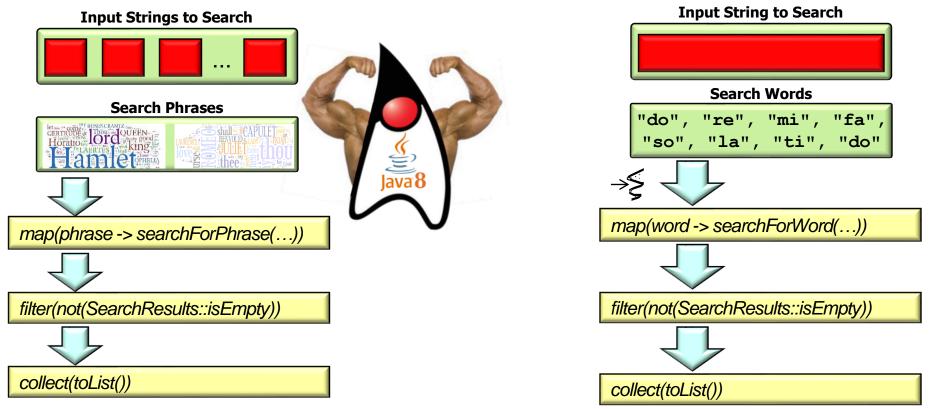
 SearchTaskGang showcases the Java executor framework for tasks that are "embarrassingly parallel"





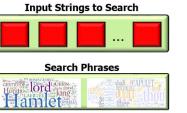
e.g., Executor, ExecutorService, ExecutorCompletionService

SearchStreamGang is also a more powerful revision of SimpleSearchStream

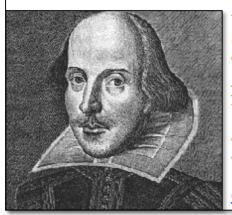


See github.com/douglascraigschmidt/LiveLessons/tree/master/SimpleSearchStream

- SearchStreamGang is also a more powerful revision of SimpleSearchStream, e.g.
 - It uses regular expressions to find phrases in works of Shakespeare



The Complete Works of William Shakespeare



Welcome to the Web's first edition of the Complete Works of William Shakespeare. This site has offered Shakespeare's plays and poetry to the Internet community since 1993.

For other Shakespeare resources, visit the Mr. William Shakespeare and the Internet Web site.

The original electronic source for this server was the Complete Moby(tm) Shakespeare. The HTML versions of the plays provided here are placed in the public domain.

Older news items

See shakespeare.mit.edu

- SearchStreamGang is also a more powerful revision of SimpleSearchStream, e.g.
 - It uses regular expressions to find phrases in works of Shakespeare

"...
My liege, and madam, to expostulate

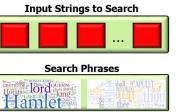
What majesty should be, what duty is, Why day is day, night is night, and time is time.

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. ..."



"Brevity is the soul of wit"

- SearchStreamGang is also a more powerful revision of SimpleSearchStream, e.g.
 - It uses regular expressions to find phrases in works of Shakespeare

Input Strings to Search

Search Phrases

Tannet me

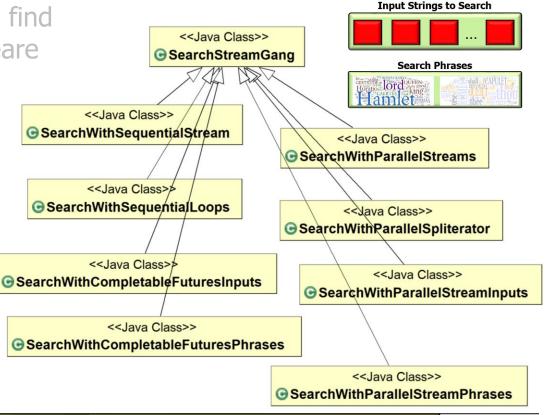
What's in a name? That which we call a rose By any other name would smell as sweet.

So Romeo would, were he not Romeo call'd, Retain that dear perfection which he owes Without that title. ..."

"What's in a name? That which we call a rose By any other name would smell as sweet."

The phrases can also match across multiple lines

- SearchStreamGang is also a more powerful revision of SimpleSearchStream, e.g.
 - It uses regular expressions to find phrases in works of Shakespeare
 - It defines a framework for comparing Java concurrency & parallelism strategies



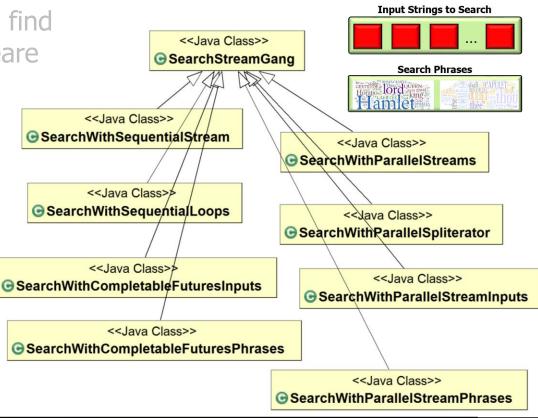
e.g., parallel streams, parallel spliterator, & completable futures

• SearchStreamGang is also a more powerful revision of SimpleSearchStream, e.g.

• It uses regular expressions to find phrases in works of Shakespeare

 It defines a framework for comparing Java concurrency & parallelism strategies

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



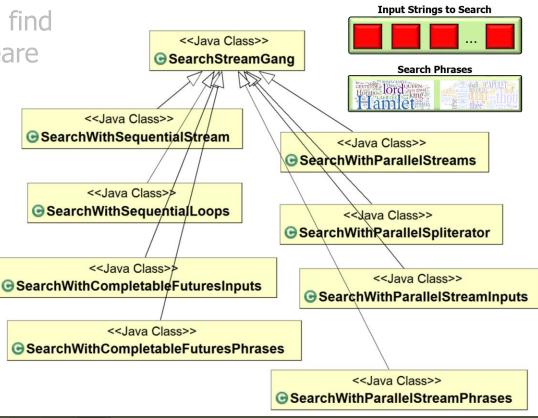
This framework enables "apples-to-apples" performance comparisons

• SearchStreamGang is also a more powerful revision of SimpleSearchStream, e.g.

• It uses regular expressions to find phrases in works of Shakespeare

 It defines a framework for comparing Java concurrency & parallelism strategies

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



We'll cover Java concurrency/parallel strategies after covering sequential streams

 We show aggregate operations in the SearchStreamGang's processStream()
 & processInput() methods

⊙ SearchWithSequentialStreams

<<Java Class>>

- processStream():List<List<SearchResults>>
- processInput(String):List<SearchResults>

 We show aggregate operations in the <<Java Class>>

SearchStreamGang's processStream() & processInput() methods

SearchWithSequentialStreams

oprocessStream():List<List<SearchResults>>

.stream()

.map(this::processInput)

.collect(toList());

getInput()

processInput(String):List<SearchResults>

return mPhrasesToFind .stream() .map(phrase -> searchForPhrase(phrase, input, title, false))

.filter(not(SearchResults::isEmpty) .collect(toList());

See SearchStreamGang/src/main/java/livelessons/streamgangs/SearchWithSequentialStreams.java

<<Java Class>>

oprocessStream():List<List<SearchResults>>

processInput(String):List<SearchResults>

SearchWithSequentialStreams

 We show aggregate operations in the SearchStreamGang's processStream()
 & processInput() methods

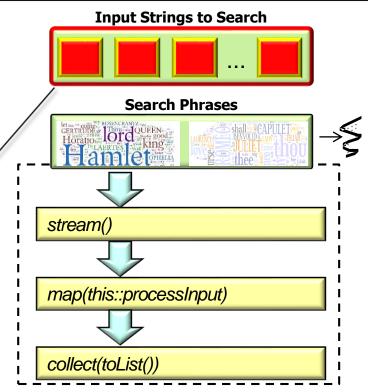
```
getInput()
    .stream()
    .map(this::processInput)
    .collect(toList());
```

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

i.e., the map(), filter(), & collect() aggregate operations

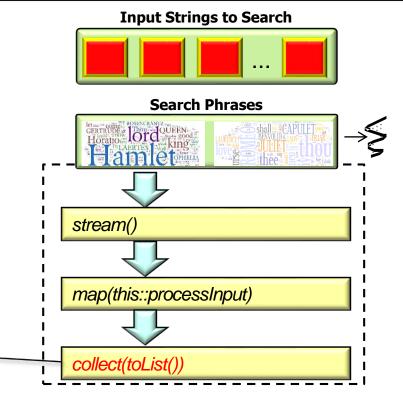
- We show aggregate operations in the SearchStreamGang's processStream()
 & processInput() methods
 - processStream()
 - Uses a sequential stream to search a list of input strings in one thread

Each input string contains a work of Shakespeare (e.g., Hamlet, MacBeth, etc.)

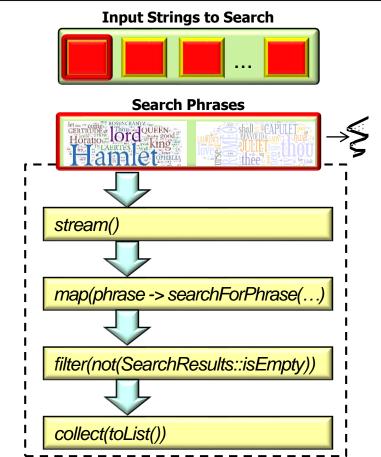


- We show aggregate operations in the SearchStreamGang's processStream()
 & processInput() methods
 - processStream()
 - Uses a sequential stream to search a list of input strings in one thread

Returns a list of lists of SearchResults

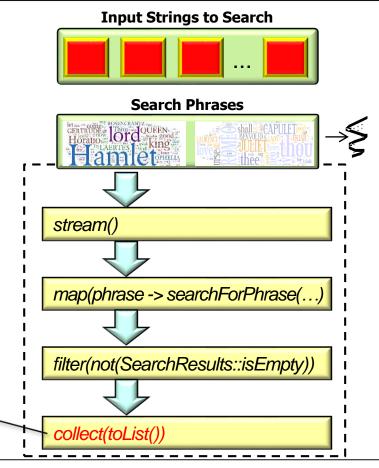


- We show aggregate operations in the SearchStreamGang's processStream()
 & processInput() methods
 - processStream()
 - processInput()
 - Uses a sequential stream to search a given input string & locate all the occurrences of phases in one thread



- We show aggregate operations in the SearchStreamGang's processStream()
 & processInput() methods
 - processStream()
 - processInput()
 - Uses a sequential stream to search a given input string & locate all the occurrences of phases in one thread

Returns a list of SearchResults



End of Java Sequential SearchStreamGang Example: Introduction