Applying Java Functional Programming Features & Threads in ThreadJoinTest

Douglas C. Schmidt
d.schmidt@vanderbilt.edu
www.dre.vanderbilt.edu/~schmidt

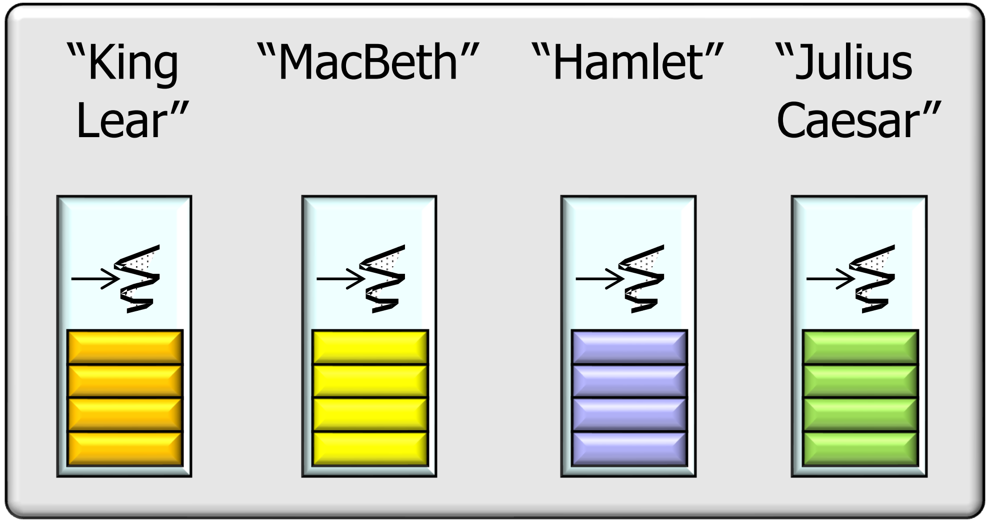
Professor of Computer Science

Institute for Software Integrated Systems

Vanderbilt University
Nashville, Tennessee, USA

Learning Objectives in this Part of the Lesson

- Understand how Java functional features are applied in an "embar-rassingly parallel" program
- Know how to create, start, process, & join Java Thread objects via functional programming features



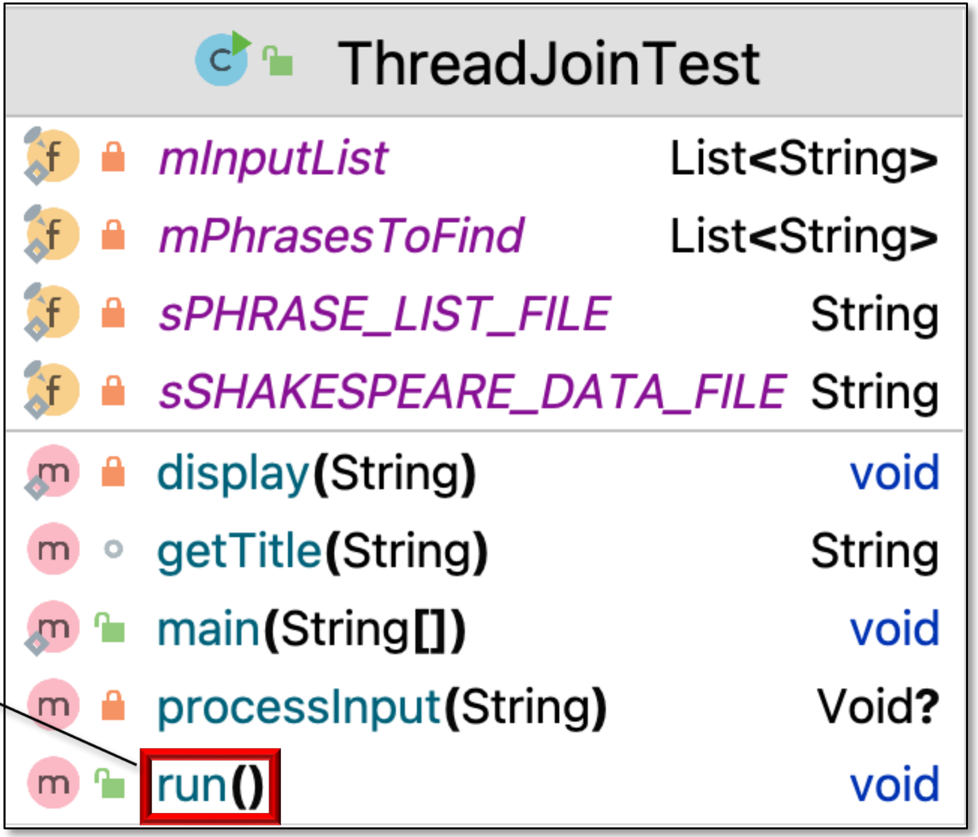
See github.com/douglascraigschmidt/ModernJava/tree/main/CS/ThreadJoinTest

Overview of the Concurrency Model

Overview of the Concurrency Model

- This case study shows functional programming features the context of Java Thread objects

The run() method creates a List of Java Thread objects & then starts & waits for the Thread objects to complete

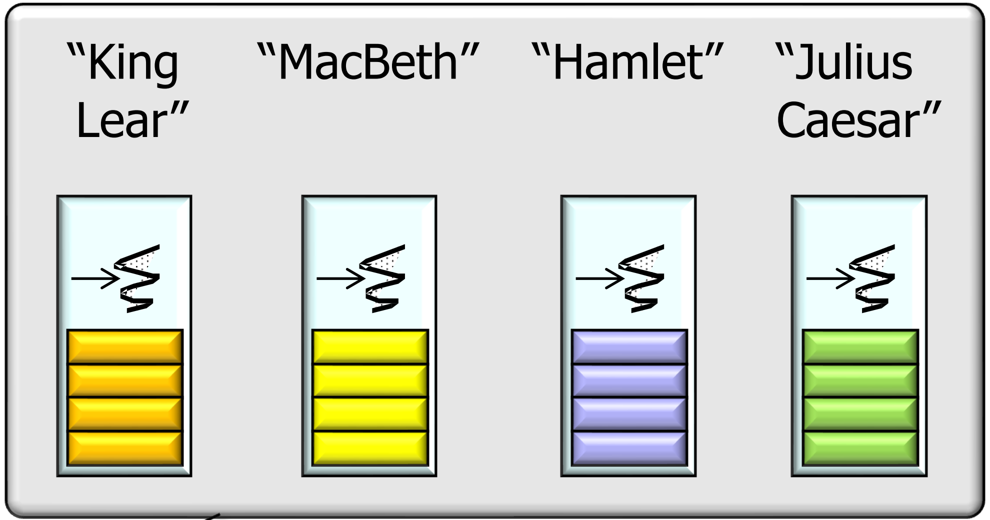


ThreadJoinTest	
mInputList	List<String>
mPhrasesToFind	List<String>
sPHRASE_LIST_FILE	String
sSHAKESPEARE_DATA_FILE	String
display(String)	void
getTitle(String)	String
main(String[])	void
processInput(String)	Void?
run()	void

See CS/ThreadJoinTest/src/main/java/ThreadJoinTest.java

Overview of the Concurrency Model

- This case study shows functional programming features the context of Java Thread objects

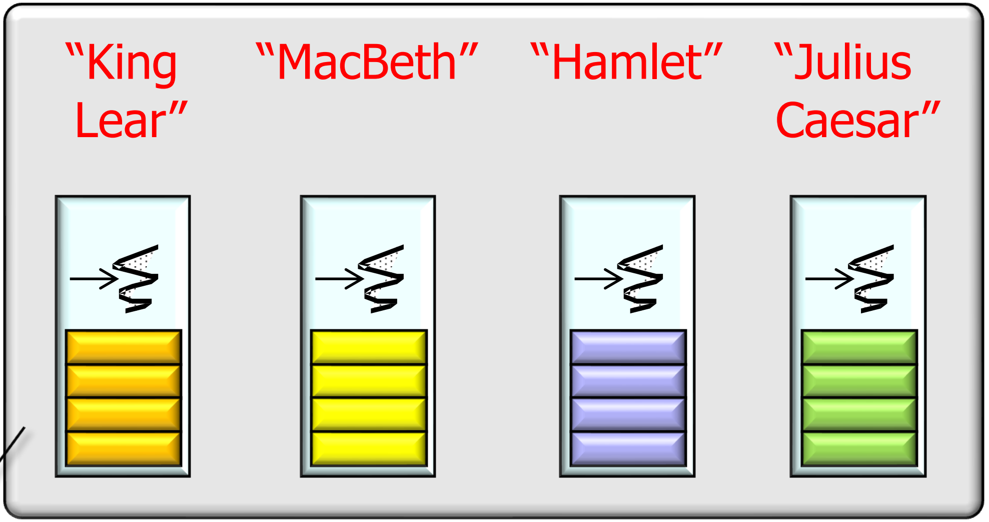


Each Java Thread objects searches for Bard phrases concurrently

We use Java platform Thread objects in this case study

Overview of the Concurrency Model

- This case study shows functional programming features the context of Java Thread objects



A "thread-per-work-of-Shakespeare" concurrency model is thus applied

This model scales well due limits on the number of works of Shakespeare

Overview of the Concurrency Model

```
[42] "All that glisters is not gold" appears at offset 52032 in "The Merchant of Venice"
[44] "The course of true love never did run smooth" appears at offset 7544 in "A Midsummer Night's Dream"
[55] "Better a witty fool than a foolish wit" appears at offset 16295 in "Twelfth Night; or, What You Will"
[38] "Sit you down, father; rest you" appears at offset 143305 in "The Tragedy of King Lear"
[44] "Lord, what fools these mortals be!" appears at offset 55498 in "A Midsummer Night's Dream"
[55] "If music be the food of love, play on" appears at offset 820 in "Twelfth Night; or, What You Will"
[43] "I cannot tell what the dickens his name is" appears at offset 67121 in "The Merry Wives of Windsor"
[29] "The better part of valour is discretion" appears at offset 149185 in "The First Part of King Henry IV"
[48] "Now is the winter of our discontent" appears at offset 1804 in "King Richard III"
[30] "Uneasy lies the head that wears a crown" appears at offset 76264 in "Second Part of King Henry IV"
[28] "Get thee to a nunnery" appears at offset 86103 in "The Tragedy of Hamlet, Prince of Denmark"
[28] "Get thee to a nunnery" appears at offset 86985 in "The Tragedy of Hamlet, Prince of Denmark"
[48] "A horse! a horse! my kingdom for a horse!" appears at offset 193548 in "King Richard III"
[48] "A horse! a horse! my kingdom for a horse!" appears at offset 193848 in "King Richard III"
[48] "Off with his head!" appears at offset 103021 in "King Richard III"
[49] "What light through yonder window breaks" appears at offset 41234 in "The Tragedy of Romeo and Juliet"
```

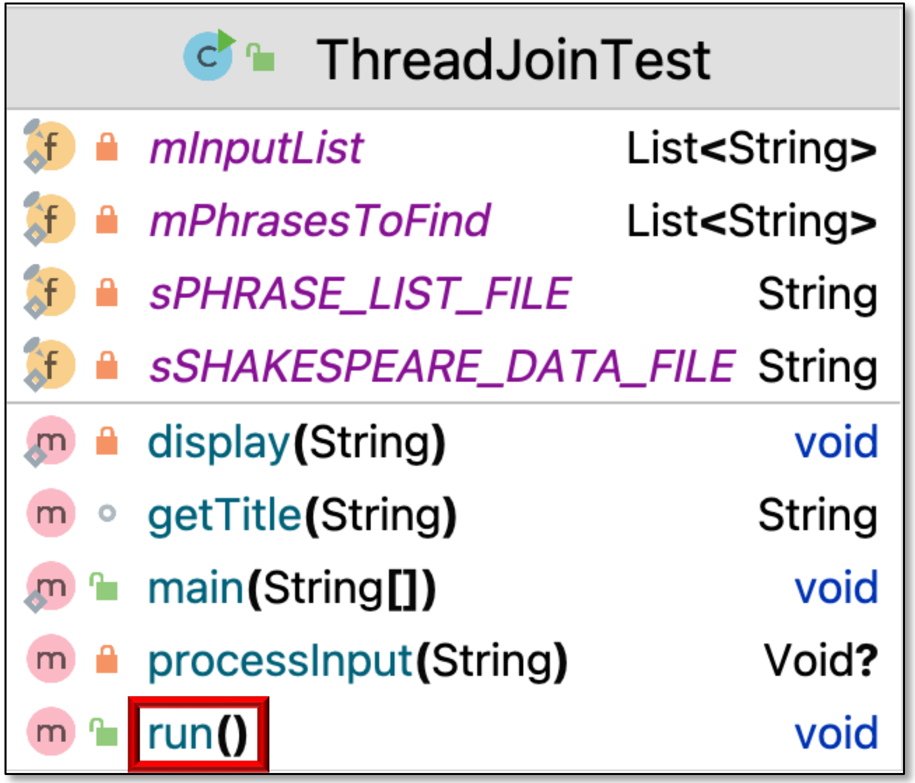
The output displays the id for each Thread that found a match, demonstrating the "embarrassingly parallel" design

See en.wikipedia.org/wiki/Embarrassingly_parallel

Creating & Starting Java Thread Objects

Creating & Starting Thread Objects

- The ThreadJoinTest.run() method starts Thread objects to perform the concurrent Bard phrase searches



ThreadJoinTest

mInputList	List<String>
mPhrasesToFind	List<String>
sPHRASE_LIST_FILE	String
sSHAKESPEARE_DATA_FILE	String
display(String)	void
getTitle(String)	String
main(String[])	void
processInput(String)	Void?
run()	void

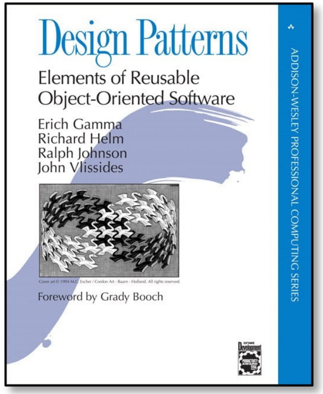
See CS/ThreadJoinTest/src/main/java/ThreadJoinTest.java

Creating & Starting Thread Objects

- Thread objects are created via the makeThreads() method called in run()

```
public void run() {
  var workerThreads =
    makeThreads
      (this::processInput);
  ...
```

makeThreads() applies the Factory Method pattern to create a List of worker Thread objects


Design Patterns
Elements of Reusable Object-Oriented Software
Erich Gamma
Richard Helm
Ralph Johnson
John Vlissides
Foreword by Grady Booch
ADDISON-WESLEY PROFESSIONAL COMPUTING SERIES

See en.wikipedia.org/wiki/Factory_method_pattern

Creating & Starting Thread Objects

- Thread objects are created via the makeThreads() method called in run()
  - A method reference to the processInput() method is passed as a param

```
public void run() {
  var workerThreads =
    makeThreads
      (this::processInput);
  ...
```

```
Void processInput(String input)
{ ... }
```

Creating & Starting Thread Objects

- Thread objects are created via the makeThreads() method called in run()
  - A method reference to the processInput() method is passed as a param

```
public void run() {
  var workerThreads =
    makeThreads
      (this::processInput);
  ...
```

```
Void processInput(String input)
{ ... }
```

This method searches for Bard phrases in a single work of William Shakespeare

We'll examine the processInput() method implementation shortly

Creating & Starting Thread Objects

- Thread objects are created via the makeThreads() method called in run()
  - A method reference to the processInput() method is passed as a param
  - makeThreads() expects a Function functional param

```
public void run() {
  var workerThreads =
    makeThreads
      (this::processInput);
  ...
```

```
List<Thread> makeThreads
  (Function<String, Void> task)
{ ... }
```

See docs.oracle.com/javase/8/docs/api/java/util/function/Function.html

Creating & Starting Thread Objects

- Thread objects are created via the makeThreads() method called in run()
  - A method reference to the processInput() method is passed as a param
  - makeThreads() expects a Function functional param

```
public void run() {
  var workerThreads =
    makeThreads
      (this::processInput);
  ...
```

```
List<Thread> makeThreads
  (Function<String, Void> task)
{ ... }
```

This functional interface makes it simple to change the Function passed to makeThreads() if necessary

See docs.oracle.com/javase/8/docs/api/java/util/function/Function.html

Creating & Starting Thread Objects

- The makeThreads() factory method applies the Function to create a Runnable for a Thread

```
<T, R> List<Thread> makeThreads
   (List<T> inputList,
    Function<T, R> task) {
  List<Thread> workerThreads =
    new ArrayList<>();

  inputList.forEach(input ->
    workerThreads.add(new Thread
      (() -> task.apply(input))));

  return workerThreads;
}
```

This generic factory method creates a List of Thread objects that will be joined when their processing is done

See CS/ThreadJoinTest/src/main/java/utils/ThreadUtils.java

Creating & Starting Thread Objects

- The makeThreads() factory method applies the Function to create a Runnable for a Thread

```
<T, R> List<Thread> makeThreads
   (List<T> inputList,
    Function<T, R> task) {
  List<Thread> workerThreads =
    new ArrayList<>();



  inputList.forEach(input ->
    workerThreads.add(new Thread
      (() -> task.apply(input))));

  return workerThreads;
}
```

The 'inputList' param contains a List of items to process (e.g., a work of Shakespeare)

Creating & Starting Thread Objects

- The makeThreads() factory method applies the Function to create a Runnable for a Thread

```
<T, R> List<Thread> makeThreads
  (List<T> inputList,
   Function<T, R> task) {
  List<Thread> workerThreads =
    new ArrayList<>();

  inputList.forEach(input ->
    workerThreads.add(new Thread
      (() -> task.apply(input))));

  return workerThreads;
}
```

The 'task' param is bound to the Function to perform on each input element (e.g., search for Bard phrases)

Creating & Starting Thread Objects

- The makeThreads() factory method applies the Function to create a Runnable for a Thread

```
<T, R> List<Thread> makeThreads
   (List<T> inputList,
    Function<T, R> task) {
   List<Thread> workerThreads =
     new ArrayList<>();
```

Create an empty List of Thread objects

```
   inputList.forEach(input ->
     workerThreads.add(new Thread
       (() -> task.apply(input))));

   return workerThreads;
}
```

Creating & Starting Thread Objects

- The makeThreads() factory method applies the Function to create a Runnable for a Thread

```
<T, R> List<Thread> makeThreads
   (List<T> inputList,
    Function<T, R> task) {
  List<Thread> workerThreads =
    new ArrayList<>();




  inputList.forEach(input ->
    workerThreads.add(new Thread
      (() -> task.apply(input))));

  return workerThreads;
}
```

Iterate through 'inputList' & create a new Thread

Creating & Starting Thread Objects

- The makeThreads() factory method applies the Function to create a Runnable for a Thread

```
<T, R> List<Thread> makeThreads
   (List<T> inputList,
    Function<T, R> task) {
  List<Thread> workerThreads =
    new ArrayList<>();

  inputList.forEach(input ->
    workerThreads.add(new Thread
      (() -> task.apply(input))));

  return workerThreads;
}
```

task.apply() creates a runnable that provides the computation for each of the Thread objects

Creating & Starting Thread Objects

- The makeThreads() factory method applies the Function to create a Runnable for a Thread

```
<T, R> List<Thread> makeThreads
   (List<T> inputList,
    Function<T, R> task) {
   List<Thread> workerThreads =
     new ArrayList<>();







   inputList.forEach(input ->
     workerThreads.add(new Thread
       (() -> task.apply(input))));

   return workerThreads;
}
```

Add each new Thread to the List of Thread objects

Creating & Starting Thread Objects

- The makeThreads() factory method applies the Function to create a Runnable for a Thread

```
<T, R> List<Thread> makeThreads
   (List<T> inputList,
    Function<T, R> task) {
  List<Thread> workerThreads =
    new ArrayList<>();






  inputList.forEach(input ->
    workerThreads.add(new Thread
      (() -> task.apply(input))));

  return workerThreads;
}
```

Return the List of Thread objects

Creating & Starting Thread Objects

- The run() method uses forEach() & a method reference to start all the Thread objects

```
public void run() {
  List<Thread> workerThreads =
    makeThreads
      (this::processInput);

  workerThreads
    .forEach(Thread::start);
  ...
```

Each started Thread then searches for Bard phrases in its work of Shakespeare

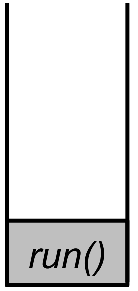
See CS/ThreadJoinTest/src/main/java/ThreadJoinTest.java

Creating & Starting Thread Objects

- The run() method uses forEach() & a method reference to start all the Thread objects

```
public void run() {
  List<Thread> workerThreads =
    makeThreads
      (this::processInput);

  workerThreads
    .forEach(Thread::start);
  ...
```



Runtime thread stack

Each call to Thread::start creates a new platform Thread object that has its own runtime stack

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

Processing & Joining Java Thread Objects

Processing & Joining Java Thread Objects

- The processInput() method was passed to the makeThreads() factory method, which bound it to a Thread object for each work of Shakepeare

```
public void run() {
  var workerThreads =
    makeThreads
      (this::processInput);
  ...
```

Processing & Joining Java Thread Objects

- The processInput() method was passed to the makeThreads() factory method, which bound it to a Thread object for each work of Shakepeare
  - Each Thread object was then started via forEach()

```
public void run() {
  List<Thread> workerThreads =
    makeThreads
      (this::processInput);

  workerThreads
    .forEach(Thread::start);
  ...
```

Processing & Joining Java Thread Objects

- processInput() runs in a Thread & searches its input param for all the occurrences of Bard phrases to find

```
Void processInput(String input) {
  var title = getTitle(input);

  for (var phrase :
           mPhrasesToFind) {
    for (int offset = input
                 .indexOf(phrase);
             offset != -1;
             offset = input
                 .indexOf(phrase,
                          offset
                          + phrase
                           .length())){
         display(...);
    } ...
```

Processing & Joining Java Thread Objects

- processInput() runs in a Thread & searches its input param for all the occurrences of Bard phrases to find

The 'input' param contains a work of Shakespeare

```
Void processInput(String input) {
  var title = getTitle(input);

  for (var phrase :
           mPhrasesToFind) {
    for (int offset = input
               .indexOf(phrase);
         offset != -1;
         offset = input
               .indexOf(phrase,
                        offset
                        + phrase
                         .length())){
         display(...);
    } ...
```

Processing & Joining Java Thread Objects

- processInput() runs in a Thread & searches its input param for all the occurrences of Bard phrases to find

```
Void processInput(String input) {
  var title = getTitle(input);

  for (var phrase :
         mPhrasesToFind) {
    for (int offset = input
             .indexOf(phrase);
         offset != -1;
         offset = input
             .indexOf(phrase,
                      offset
                      + phrase
                       .length())){
      display(...);
    } ...
```

Extract the title from the work (uses Java regular expressions)

Processing & Joining Java Thread Objects

- processInput() runs in a Thread & searches its input param for all the occurrences of Bard phrases to find

```
Void processInput(String input) {
  var title = getTitle(input);

  for (var phrase :
           mPhrasesToFind) {
    for (int offset = input
                .indexOf(phrase);
         offset != -1;
         offset = input
           .indexOf(phrase,
                    offset
                    + phrase
                     .length())){
      display(...);
    } ...
```

Iterate through all the Bard phrases to search for

Processing & Joining Java Thread Objects

- processInput() runs in a Thread & searches its input param for all the occurrences of Bard phrases to find

```
Void processInput(String input) {
  var title = getTitle(input);

  for (var phrase :
         mPhrasesToFind) {
    for (int offset = input
           .indexOf(phrase);
         offset != -1;
         offset = input
           .indexOf(phrase,
                    offset
                    + phrase
                     .length())){
         display(...);
    } ...
```

Check to see how many times (if any) 'phrase' appears in 'input' ('offset' != -1 indicates a match)

Processing & Joining Java Thread Objects

- processInput() runs in a Thread & searches its input param for all the occurrences of Bard phrases to find

```
Void processInput(String input) {
  var title = getTitle(input);

  for (var phrase :
           mPhrasesToFind) {
    for (int offset = input
                 .indexOf(phrase);
           offset != -1;
           offset = input
             .indexOf(phrase,
                      offset
                      + phrase
                       .length())){
      display(...);
    } ...
```

```
[42]  "All that glisters is not gold"
appears at offset 52032 in "The
Merchant of Venice"
```

Display results when ever a match occurs

Processing & Joining Java Thread Objects

- processInput() runs in a Thread & searches its input param for all the occurrences of Bard phrases to find

```
Void processInput(String input) {
  var title = getTitle(input);

  for (var phrase :
           mPhrasesToFind) {
    for (int offset = input
             .indexOf(phrase);
         offset != -1;
         offset = input
           .indexOf(phrase,
                    offset
                    + phrase
                     .length())){
      display(...);
    } ...
```

Update 'offset' to see if there are any more matches of 'phrase' in the 'input'

Processing & Joining Java Thread Objects

- The run() method waits for all worker Thread objects to finish via forEach() & a method reference

```
public void run() {
  List<Thread> workerThreads =
    makeThreads
      (this::processInput);

  workerThreads
    .forEach(Thread::start);

  workerThreads
    .forEach
       (rethrowConsumer
          (Thread::join));
```

Uses forEach() & a method reference

Processing & Joining Java Thread Objects

- The run() method waits for all worker Thread objects to finish via forEach() & a method reference

```
public void run() {
  List<Thread> workerThreads =
    makeThreads
      (this::processInput);

  workerThreads
    .forEach(Thread::start);

  workerThreads
    .forEach
      (rethrowConsumer
         (Thread::join));
```

Simple form of barrier synchronization

See en.wikipedia.org/wiki/Barrier_(computer_science)

Processing & Joining Java Thread Objects

- The run() method waits for all worker Thread objects to finish via forEach() & a method reference

```
public void run() {
  List<Thread> workerThreads =
    makeThreads
      (this::processInput);

  workerThreads
    .forEach(Thread::start);

  workerThreads
    .forEach
       (rethrowConsumer
          (Thread::join));
```

No other Java synchronizers are needed!

Processing & Joining Java Thread Objects

- The run() method waits for all worker Thread objects to finish via forEach() & a method reference

```
public void run() {
  List<Thread> workerThreads =
    makeThreads
      (this::processInput);

  workerThreads
    .forEach(Thread::start);

  workerThreads
    .forEach
      (rethrowConsumer
         (Thread::join));
```

Convert a checked exception to an unchecked exception

See stackoverflow.com/a/27644392/3312330

End of Applying Java Functional Programming Features & Threads in ThreadJoinTest