Understand Java Parallel Streams Internals: Parallel Processing w/the Common Fork-Join Pool

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

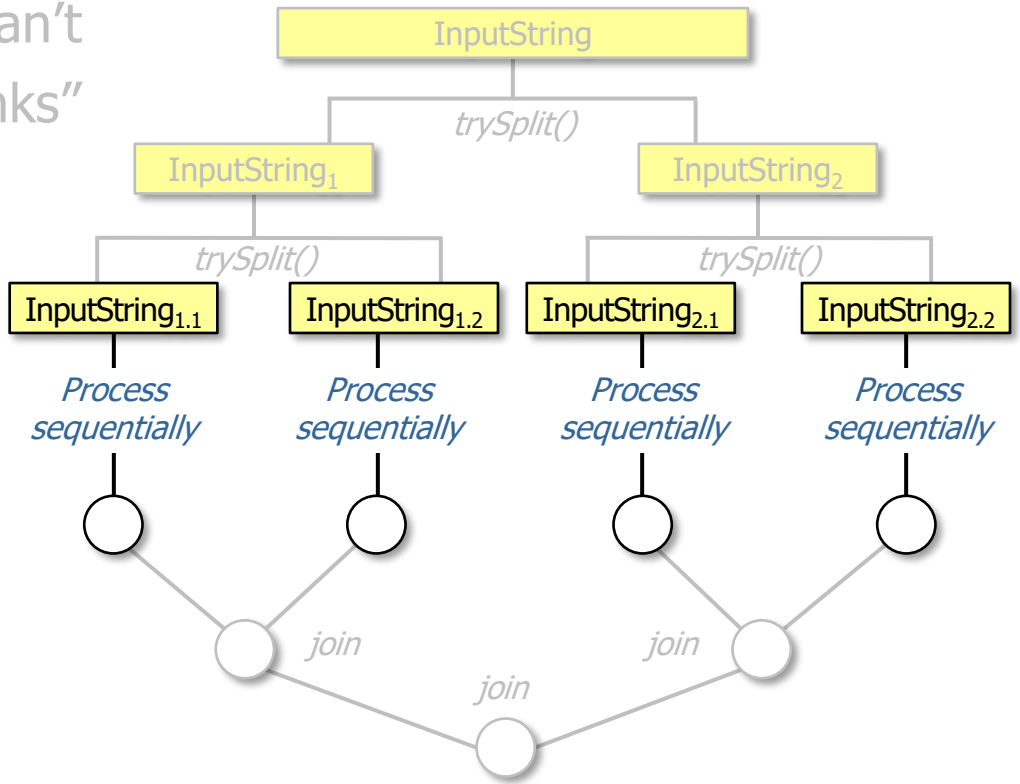
Professor of Computer Science

Institute for Software Integrated Systems

Vanderbilt University
Nashville, Tennessee, USA

Learning Objectives in this Part of the Lesson

- Understand parallel stream internals, e.g.
  - Know what can change & what can't
  - Partition a data source into "chunks"
  - Process chunks in parallel via the common fork-join pool

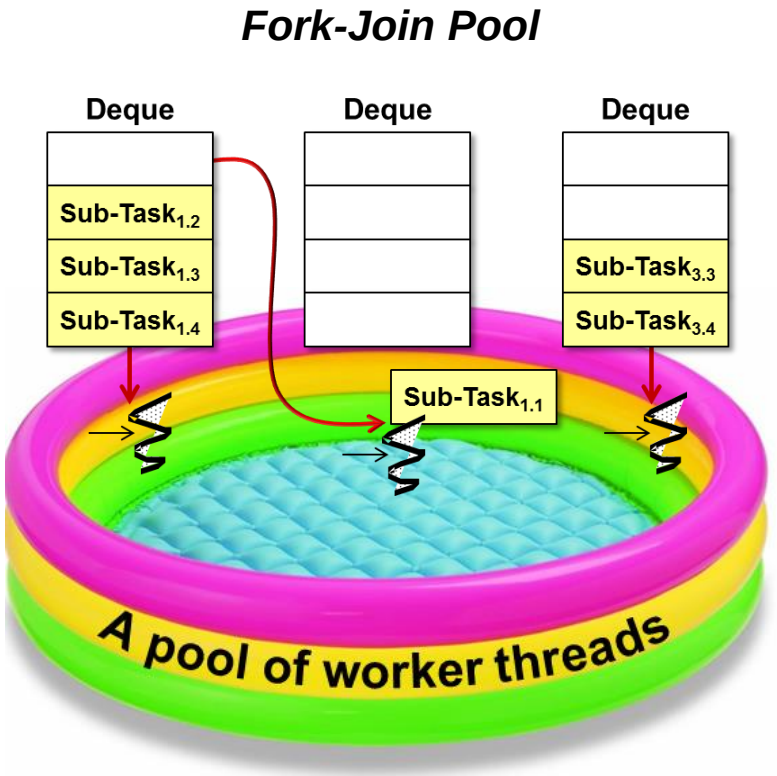


See developer.ibm.com/languages/java/articles/j-java-streams-3-brian-goetz

Processing Chunks in Parallel via the Common Fork-Join Pool

Processing Chunks in Parallel via the Common Fork-Join Pool

- Chunks created by a spliterator are processed in the common fork-join pool



See gee.cs.oswego.edu/dl/papers/fj.pdf

Processing Chunks in Parallel via the Common Fork-Join Pool

- A fork-join pool provides a high performance, fine-grained task execution framework for Java data parallelism

Class ForkJoinPool

java.lang.Object
    java.util.concurrent.AbstractExecutorService
        java.util.concurrent.ForkJoinPool

All Implemented Interfaces:
`Executor, ExecutorService`

```
public class ForkJoinPool
extends AbstractExecutorService
```

An `ExecutorService` for running `ForkJoinTasks`. A `ForkJoinPool` provides the entry point for submissions from non-`ForkJoinTask` clients, as well as management and monitoring operations.

A `ForkJoinPool` differs from other kinds of `ExecutorService` mainly by virtue of employing *work-stealing*: all threads in the pool attempt to find and execute tasks submitted to the pool and/or created by other active tasks (eventually blocking waiting for work if none exist). This enables efficient processing when most tasks spawn other subtasks (as do most `ForkJoinTasks`), as well as when many small tasks are submitted to the pool from external clients. Especially when setting *asyncMode* to true in constructors, `ForkJoinPools` may also be appropriate for use with event-style tasks that are never joined.

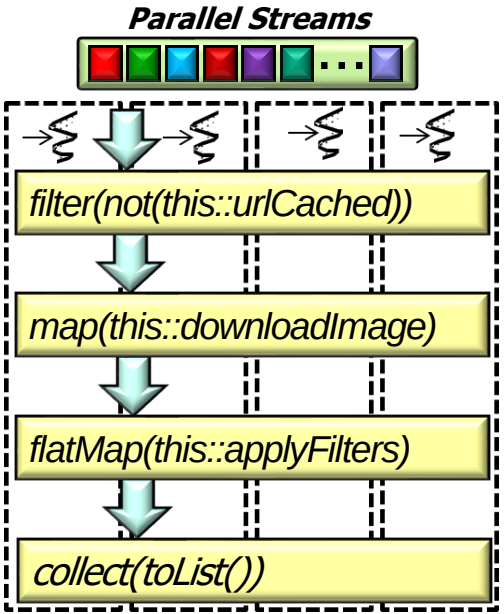
A static `commonPool()` is available and appropriate for most applications. The common pool is used by any ForkJoinTask that is not explicitly submitted to a specified pool. Using the common pool normally reduces resource usage (its threads are slowly reclaimed during periods of non-use, and reinstated upon subsequent use).

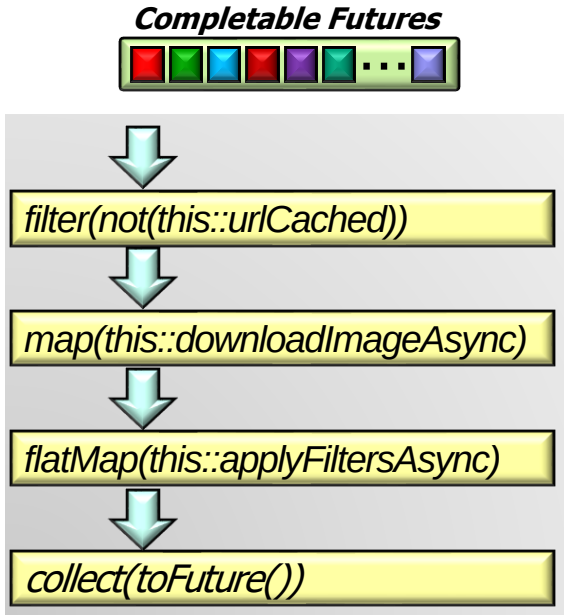
For applications that require separate or custom pools, a `ForkJoinPool` may be constructed with a given target parallelism level; by default, equal to the number of available processors. The pool attempts to maintain enough active (or available) threads by dynamically adding, suspending, or resuming internal worker threads, even if some tasks are stalled waiting to join others. However, no such adjustments are guaranteed in the face of blocked I/O or other unmanaged synchronization. The nested `ForkJoinPool.ManagedBlocker` interface enables extension of the kinds of synchronization accommodated.

See docs.oracle.com/javase/8/docs/api/java/util/concurrent/ForkJoinPool.html

Processing Chunks in Parallel via the Common Fork-Join Pool

- A fork-join pool provides a high performance, fine-grained task execution framework for Java data parallelism
  - It provides a parallel computing engine for many higher-level frameworks

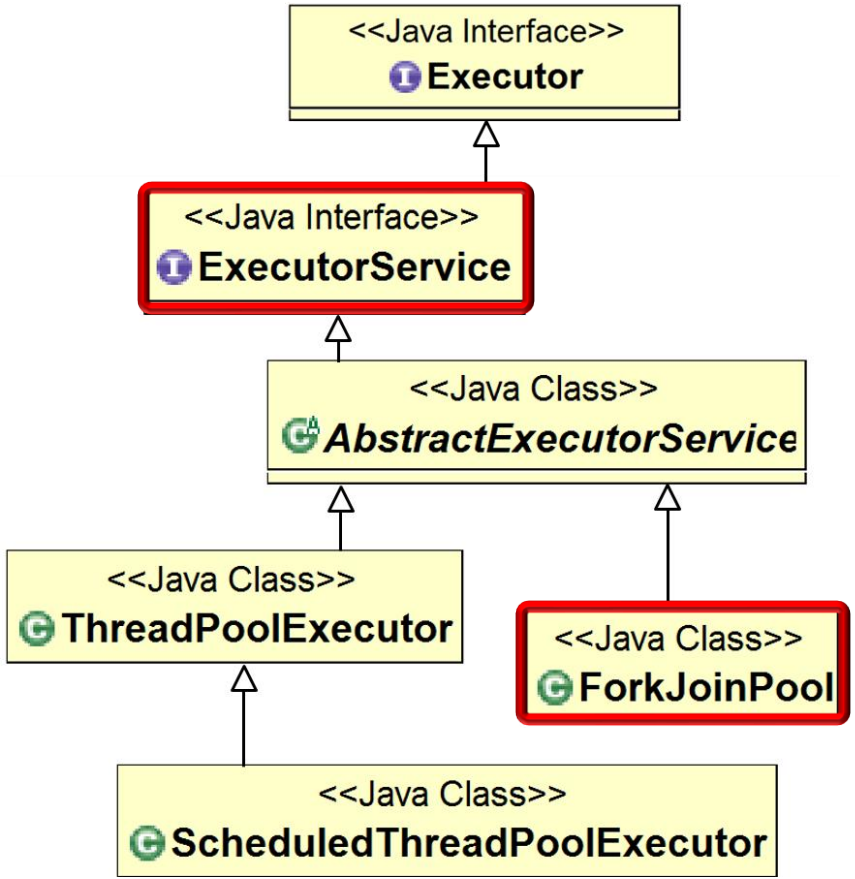




See www.infoq.com/interviews/doug-lea-fork-join

Processing Chunks in Parallel via the Common Fork-Join Pool

- ForkJoinPool implements the Executor Service interface



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

Processing Chunks in Parallel via the Common Fork-Join Pool

- ForkJoinPool implements the Executor Service interface
  - A ForkJoinPool executes ForkJoinTasks

Class ForkJoinTask<V>

java.lang.Object
    java.util.concurrent.ForkJoinTask<V>

All Implemented Interfaces:

`Serializable, Future<V>`

Direct Known Subclasses:

`CountedCompleter, RecursiveAction, RecursiveTask`

```
public abstract class ForkJoinTask<V>
extends Object
implements Future<V>, Serializable
```

Abstract base class for tasks that run within a `ForkJoinPool`. A `ForkJoinTask` is a thread-like entity that is much lighter weight than a normal thread. Huge numbers of tasks and subtasks may be hosted by a small number of actual threads in a ForkJoinPool, at the price of some usage limitations.

See docs.oracle.com/javase/8/docs/api/java/util/concurrent/ForkJoinTask.html

Processing Chunks in Parallel via the Common Fork-Join Pool

- ForkJoinPool implements the Executor Service interface
  - A ForkJoinPool executes ForkJoinTasks
  - ForkJoinTask associates a chunk of data along with a computation on that data to enable fine-grained parallelism

Class ForkJoinTask<V>

java.lang.Object
  java.util.concurrent.ForkJoinTask<V>

All Implemented Interfaces:

`Serializable, Future<V>`

Direct Known Subclasses:

`CountedCompleter, RecursiveAction, RecursiveTask`

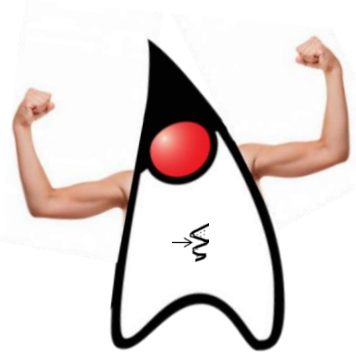
```
public abstract class ForkJoinTask<V>
extends Object
implements Future<V>, Serializable
```

Abstract base class for tasks that run within a `ForkJoinPool`. A `ForkJoinTask` is a thread-like entity that is much lighter weight than a normal thread. Huge numbers of tasks and subtasks may be hosted by a small number of actual threads in a ForkJoinPool, at the price of some usage limitations.

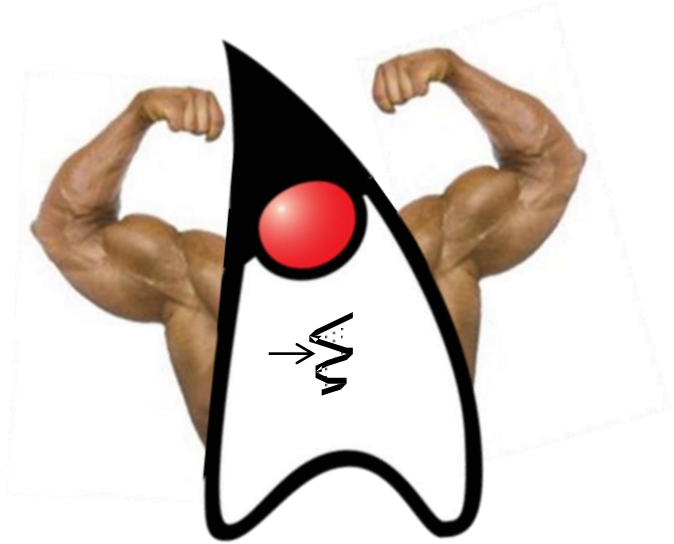
See www.dre.vanderbilt.edu/~schmidt/PDF/DataParallelismInJava.pdf

Processing Chunks in Parallel via the Common Fork-Join Pool

- A ForkJoinTask is similar to—but lighter weight—than a Java Thread



ForkJoinTask

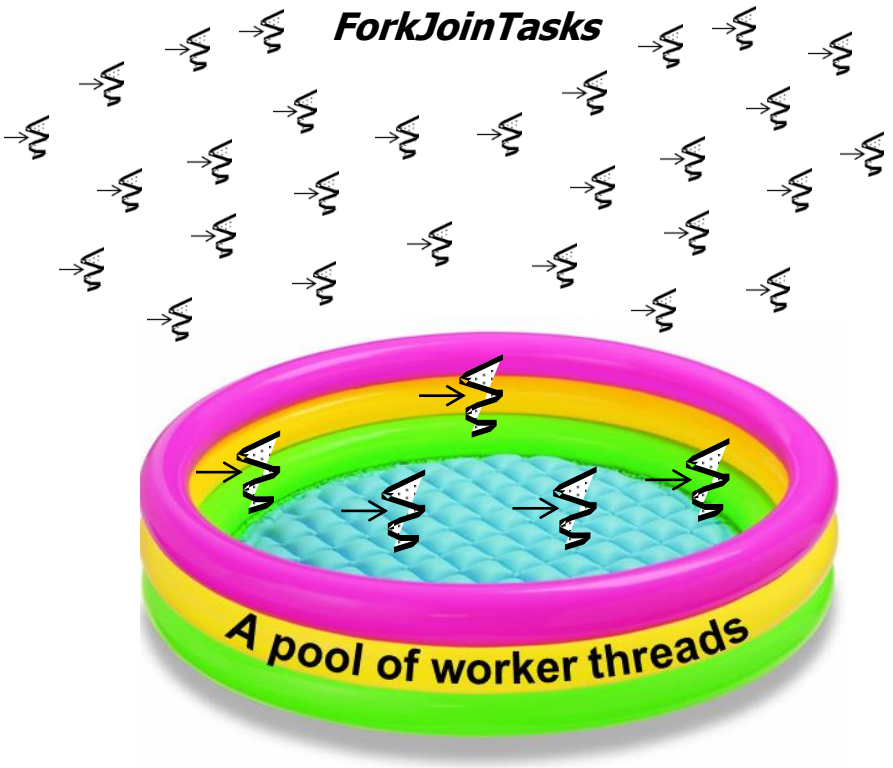


Thread

e.g., it omits its own run-time stack, registers, thread-local storage, etc.

Processing Chunks in Parallel via the Common Fork-Join Pool

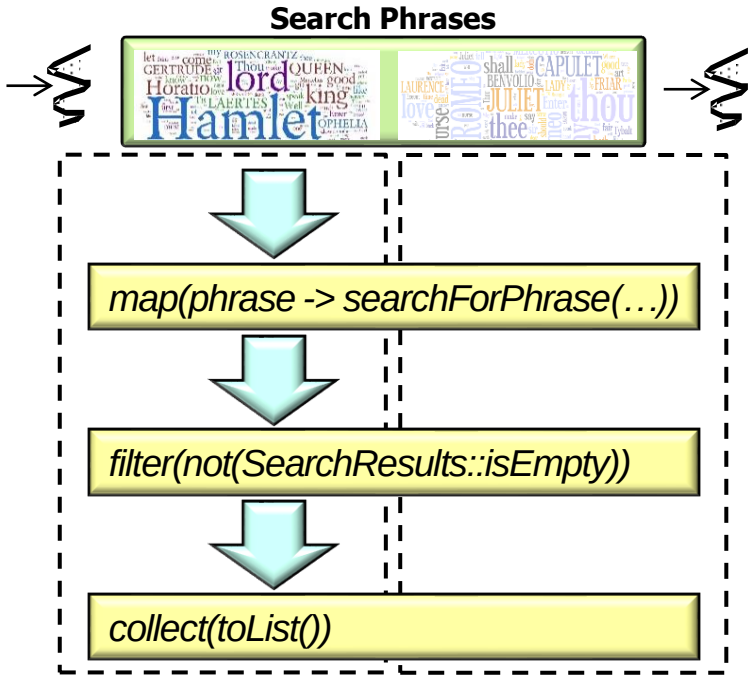
- A ForkJoinTask is similar to—but lighter weight—than a Java Thread
  - A large # of ForkJoinTasks can thus run in a small # of Java worker threads in a ForkJoinPool

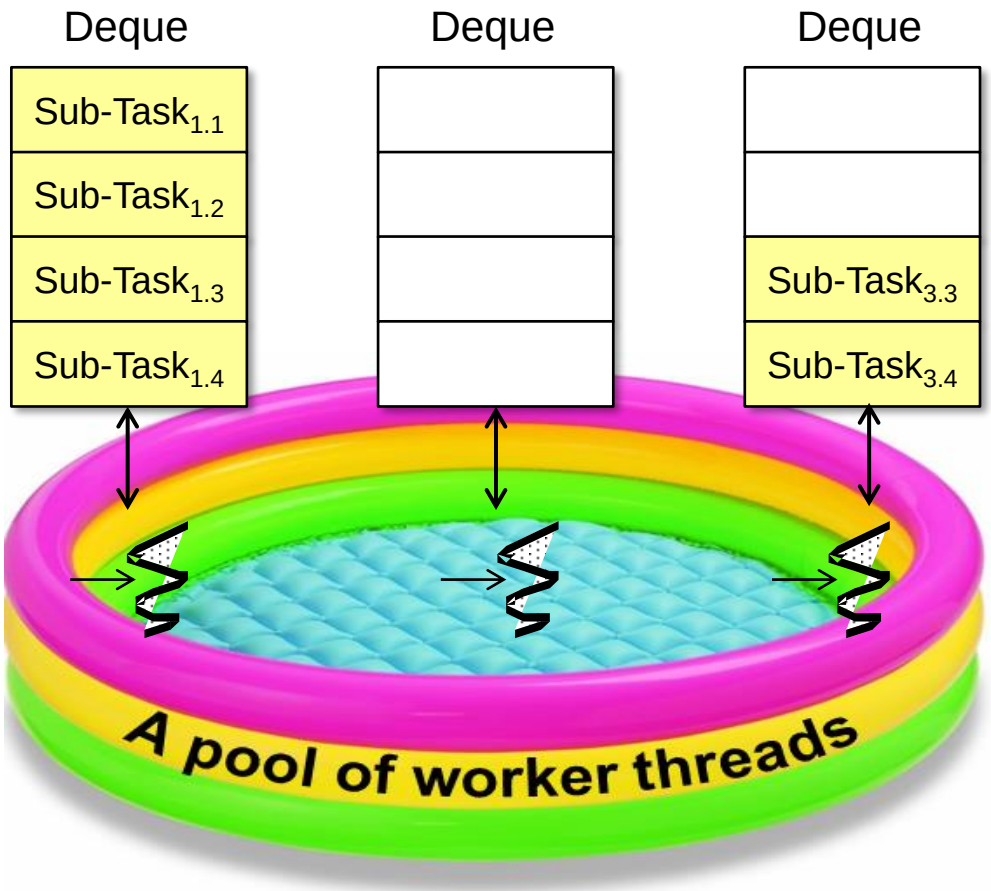


See www.infoq.com/interviews/doug-lea-fork-join

Processing Chunks in Parallel via the Common Fork-Join Pool

- Parallel streams are a "user friendly" ForkJoinPool façade





See en.wikipedia.org/wiki/Facade_pattern

Processing Chunks in Parallel via the Common Fork-Join Pool

- You can program directly to the ForkJoinPool API, though it can be somewhat painful!

```
List<List<SearchResults>>
  listOfListOfSearchResults =
    ForkJoinPool.commonPool()
      .invoke(new
         SearchWithForkJoinTask
           (inputList,
            mPhrasesToFind, ...));
```

I gave you the chance of programming Java streams willingly

But you have elected the way of pain!

See espressoprogrammer.com/fork-join-vs-parallel-stream-java-8

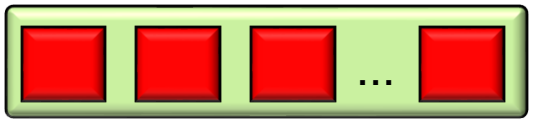
Processing Chunks in Parallel via the Common Fork-Join Pool

- You can program directly to the ForkJoinPool API, though it can be somewhat painful!

```
List<List<SearchResults>>
  listOfListOfSearchResults =
    ForkJoinPool.commonPool()
      .invoke(new
        SearchWithForkJoinTask
          (inputList,
           mPhrasesToFind, ...));
```

Use the common fork-join pool to search input strings for phrases that match

Input Strings to Search



Search Phrases

See livelessons/streamgangs/SearchWithForkJoin.java

Processing Chunks in Parallel via the Common Fork-Join Pool

- ForkJoinPool is best used for programs that don't match the parallel streams model

```
Long compute() {
  long count = 0L;
  List<RecursiveTask<Long>> forks =
    new LinkedList<>();
  for (Folder sub : mFolder.getSubs()){
    FolderSearchTask task = new
      FolderSearchTask(sub, mWord);
    forks.add(task); task.fork();
  }
  for (Doc doc : mFolder.getDocs()) {
    DocSearchTask task =
      new DocSearchTask(doc, mWord);
    forks.add(task); task.fork();
  }
  for (RecursiveTask<Long> task : forks)
    count += task.join();
  return count; ...
```

See en.wikipedia.org/wiki/Divide-and-conquer_algorithm

Processing Chunks in Parallel via the Common Fork-Join Pool

- ForkJoinPool is best used for programs that don't match the parallel streams model
  - e.g., this program counts the occurrence of a word in document folders

```
Long compute() {
  long count = 0L;
  List<RecursiveTask<Long>> forks =
    new LinkedList<>();
  for (Folder sub : mFolder.getSubs()){
    FolderSearchTask task = new
      FolderSearchTask(sub, mWord);
    forks.add(task); task.fork();
  }
  for (Doc doc : mFolder.getDocs()) {
    DocSearchTask task =
      new DocSearchTask(doc, mWord);
    forks.add(task); task.fork();
  }
  for (RecursiveTask<Long> task : forks)
    count += task.join();
  return count; ...
```

See www.oracle.com/technetwork/articles/java/fork-join-422606.html

Processing Chunks in Parallel via the Common Fork-Join Pool

- ForkJoinPool is best used for programs that don't match the parallel streams model
  - e.g., this program counts the occurrence of a word in document folders

Create a linked list of recursive task objects

```
Long compute() {
  long count = 0L;
  List<RecursiveTask<Long>> forks =
    new LinkedList<>();
  for (Folder sub : mFolder.getSubs()){
    FolderSearchTask task = new
      FolderSearchTask(sub, mWord);
    forks.add(task); task.fork();
  }
  for (Doc doc : mFolder.getDocs()) {
    DocSearchTask task =
      new DocSearchTask(doc, mWord);
    forks.add(task); task.fork();
  }
  for (RecursiveTask<Long> task : forks)
    count += task.join();
  return count; ...
```

Processing Chunks in Parallel via the Common Fork-Join Pool

- ForkJoinPool is best used for programs that don't match the parallel streams model
  - e.g., this program counts the occurrence of a word in document folders

Create & fork tasks to search folders recursively

```
Long compute() {
  long count = 0L;
  List<RecursiveTask<Long>> forks =
    new LinkedList<>();
  for (Folder sub : mFolder.getSubs()){
    FolderSearchTask task = new
      FolderSearchTask(sub, mWord);
    forks.add(task); task.fork();
  }
  for (Doc doc : mFolder.getDocs()) {
    DocSearchTask task =
      new DocSearchTask(doc, mWord);
    forks.add(task); task.fork();
  }
  for (RecursiveTask<Long> task : forks)
    count += task.join();
  return count; ...
```

Processing Chunks in Parallel via the Common Fork-Join Pool

- ForkJoinPool is best used for programs that don't match the parallel streams model
  - e.g., this program counts the occurrence of a word in document folders

```
Long compute() {
  long count = 0L;
  List<RecursiveTask<Long>> forks =
    new LinkedList<>();
  for (Folder sub : mFolder.getSubs()){
    FolderSearchTask task = new
      FolderSearchTask(sub, mWord);
    forks.add(task); task.fork();
  }
  for (Doc doc : mFolder.getDocs()) {
    DocSearchTask task =
      new DocSearchTask(doc, mWord);
    forks.add(task); task.fork();
  }
  for (RecursiveTask<Long> task : forks)
    count += task.join();
  return count; ...
```

Create & fork tasks to search documents

Processing Chunks in Parallel via the Common Fork-Join Pool

- ForkJoinPool is best used for programs that don't match the parallel streams model
  - e.g., this program counts the occurrence of a word in document folders

Join all the tasks together & count the # of search matches

```
Long compute() {
  long count = 0L;
  List<RecursiveTask<Long>> forks =
    new LinkedList<>();
  for (Folder sub : mFolder.getSubs()){
    FolderSearchTask task = new
      FolderSearchTask(sub, mWord);
    forks.add(task); task.fork();
  }
  for (Doc doc : mFolder.getDocs()) {
    DocSearchTask task =
      new DocSearchTask(doc, mWord);
    forks.add(task); task.fork();
  }
  for (RecursiveTask<Long> task : forks)
    count += task.join();
  return count; ...
```

Processing Chunks in Parallel via the Common Fork-Join Pool

- ForkJoinPool is best used for programs that don't match the parallel streams model
  - e.g., this program counts the occurrence of a word in document folders

```
Long compute() {
  long count = 0L;
  List<RecursiveTask<Long>> forks =
    new LinkedList<>();
  for (Folder sub : mFolder.getSubs()){
    FolderSearchTask task = new
      FolderSearchTask(sub, mWord);
    forks.add(task); task.fork();
  }
  for (Doc doc : mFolder.getDocs()) {
    DocSearchTask task =
      new DocSearchTask(doc, mWord);
    forks.add(task); task.fork();
  }
  for (RecursiveTask<Long> task : forks)
    count += task.join();
  return count; ...
```

Return the final count

Processing Chunks in Parallel via the Common Fork-Join Pool

- All parallel streams in a process share the common fork-join pool

See dzone.com/articles/common-fork-join-pool-and-streams

Processing Chunks in Parallel via the Common Fork-Join Pool

- All parallel streams in a process share the common fork-join pool
  - Helps optimize resource utilization by knowing what cores are being used globally within a process

See dzone.com/articles/common-fork-join-pool-and-streams

Processing Chunks in Parallel via the Common Fork-Join Pool

- All parallel streams in a process share the common fork-join pool
  - Helps optimize resource utilization by knowing what cores are being used globally within a process
    - This "global" vs "local" resource management tradeoff is common in computing & other domains

See blog.tsia.com/blog/local-or-global-resource-management-which-model-is-better

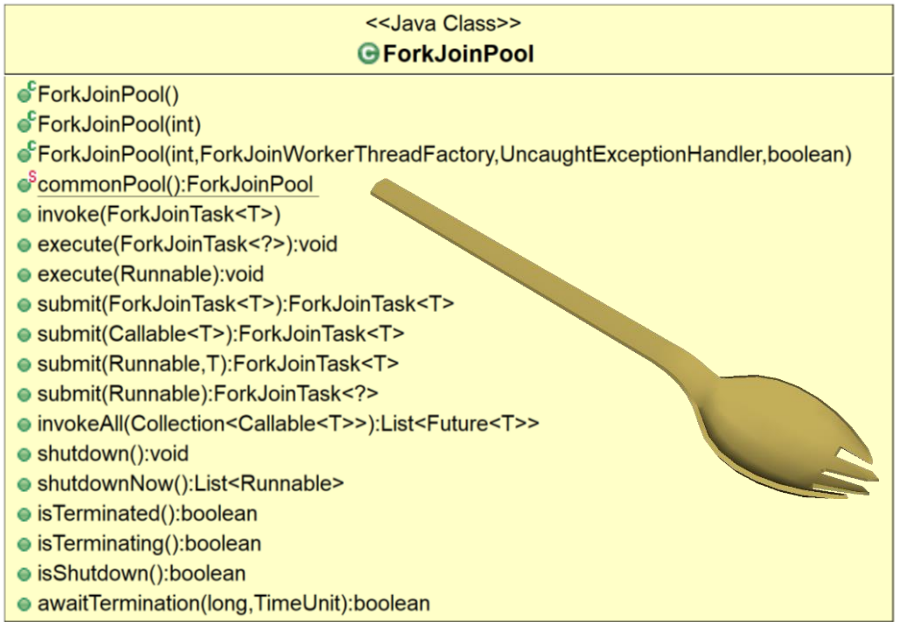
Processing Chunks in Parallel via the Common Fork-Join Pool

- There are few "knobs" to control this (or any) fork-join pool

See www.infoq.com/presentations/tecniques-parallelism-java

Processing Chunks in Parallel via the Common Fork-Join Pool

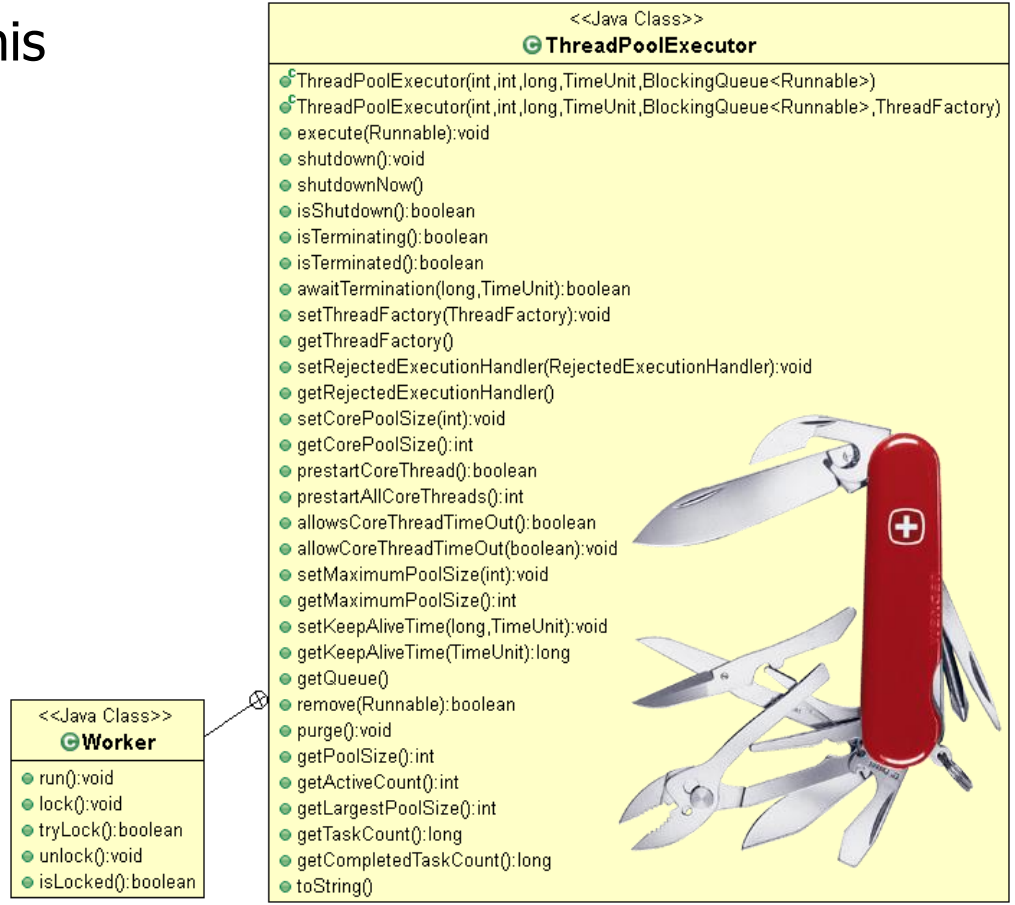
- There are few "knobs" to control this (or any) fork-join pool
  - This simplicity is intentional..



See www.youtube.com/watch?v=sq0MX3fHkro

Processing Chunks in Parallel via the Common Fork-Join Pool

- There are few "knobs" to control this (or any) fork-join pool
  - This simplicity is intentional..
  - Contrast ForkJoinPool with ThreadPoolExecutor



Processing Chunks in Parallel via the Common Fork-Join Pool

- There are few "knobs" to control this (or any) fork-join pool
  - This simplicity is intentional..
  - Contrast ForkJoinPool with ThreadPoolExecutor
  - However, the size of the common fork-join pool *can* be configured

```
System.setProperty
   ("java.util.concurrent"
    + ".ForkJoinPool.common"
    + ".parallelism",
    "8");
```

Set desired # of threads

Interface ForkJoinPool.ManagedBlocker

Enclosing class:
ForkJoinPool

public static interface **ForkJoinPool.ManagedBlocker**

Interface for extending managed parallelism for tasks running in ForkJoinPools.

See upcoming lesson on "*Java Parallel Stream Internals: Configuration*"

End of Understand Java Parallel Streams Internals: Parallel Processing w/the Common Fork-Join Pool