

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

**Douglas C. Schmidt**

**[d.schmidt@vanderbilt.edu](mailto:d.schmidt@vanderbilt.edu)**

**[www.dre.vanderbilt.edu/~schmidt](http://www.dre.vanderbilt.edu/~schmidt)**



**Professor of Computer Science**

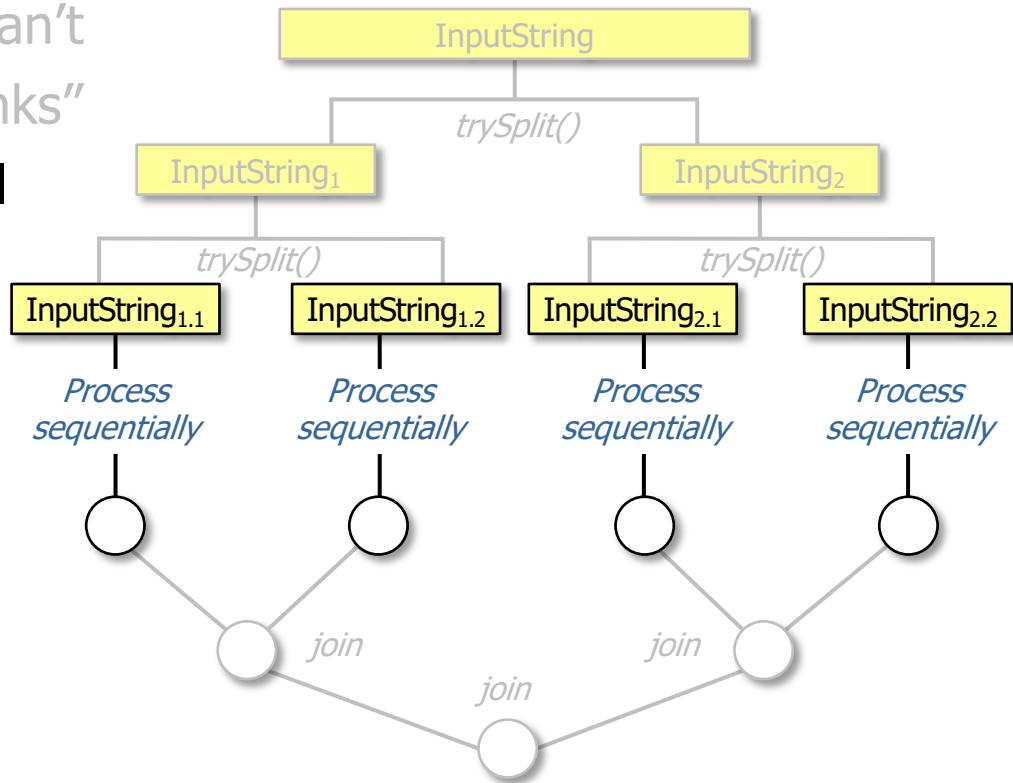
**Institute for Software  
Integrated Systems**

**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 of data in parallel via the common fork-join pool

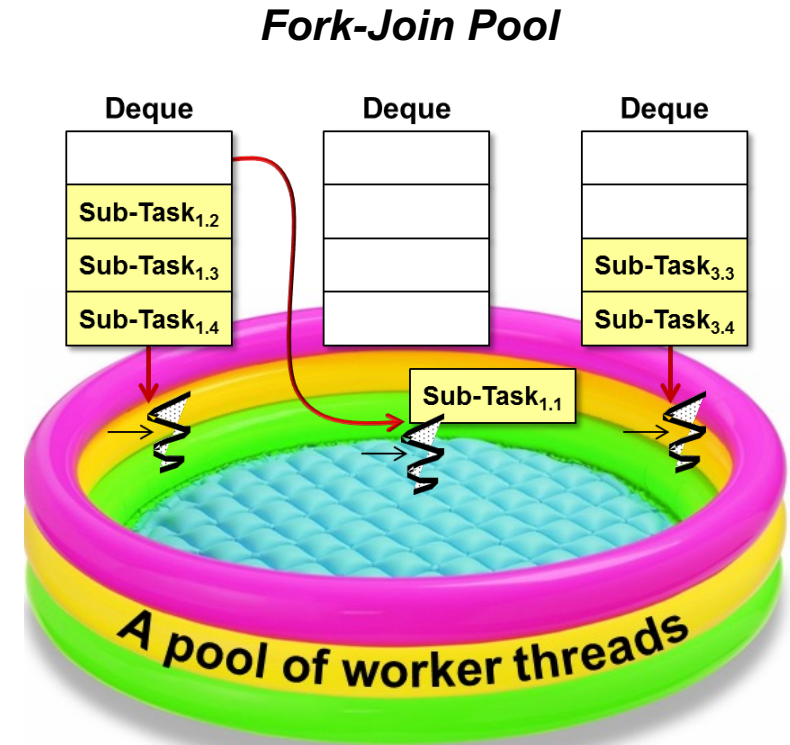


---

# 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



# 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).

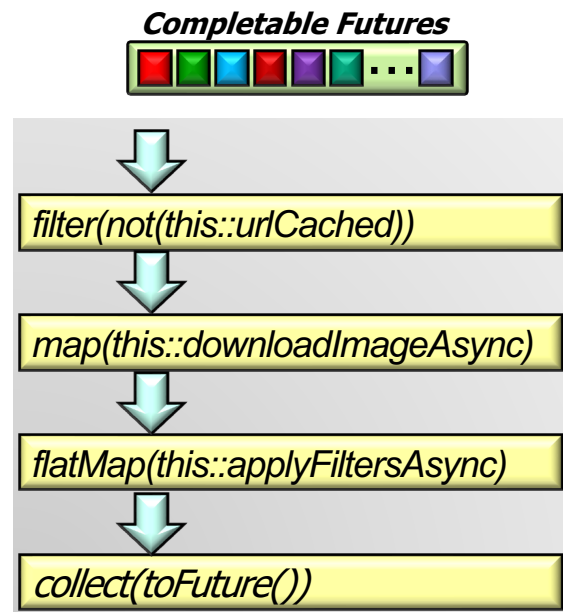
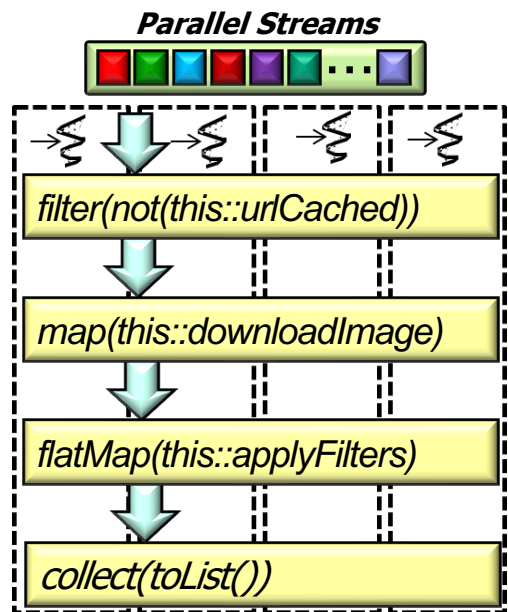
For applications that require separate or custom pools, a `ForkJoinPool` may be constructed with a given target parallelism level; by default, equal to the number of available processors. The pool attempts to maintain enough active (or available) threads by dynamically adding, suspending, or resuming 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](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/ForkJoinPool.html)

# Processing Chunks in Parallel via the Common Fork-Join Pool

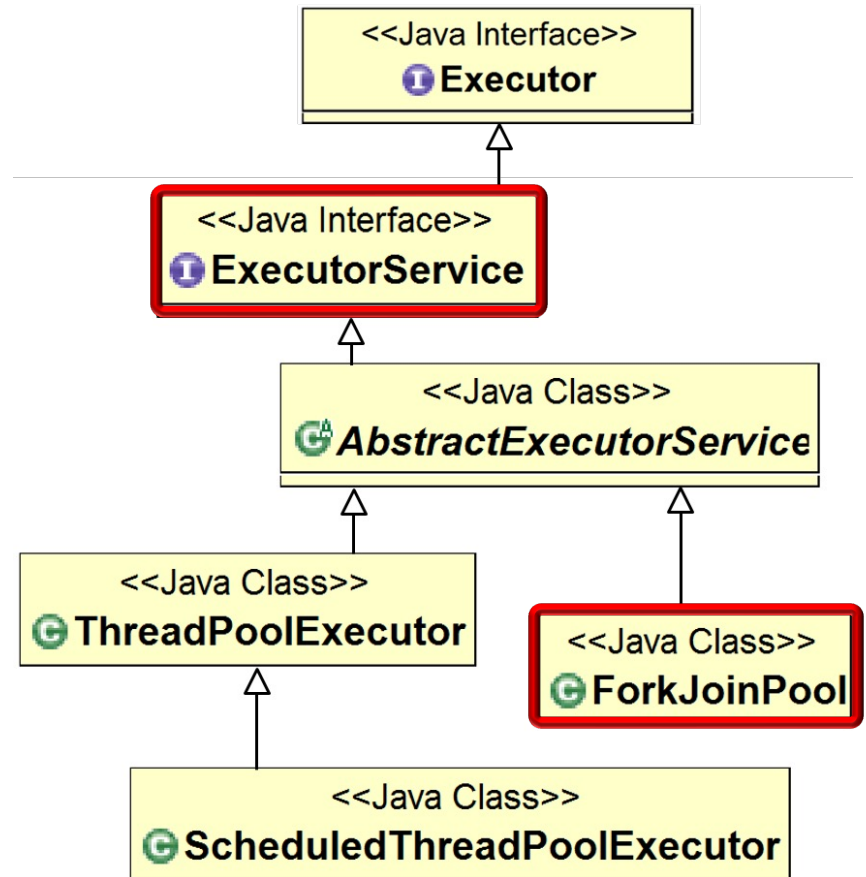
- A fork-join pool provides a high performance, fine-grained task execution framework for Java data parallelism
- It provides a parallel computing engine for many higher-level frameworks



See [www.infoq.com/interviews/doug-lea-fork-join](http://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](https://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](https://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:

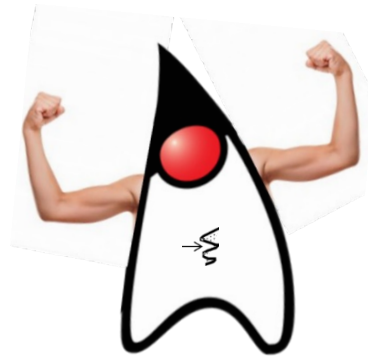
```
CountedCompleter, RecursiveAction,
RecursiveTask
```

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

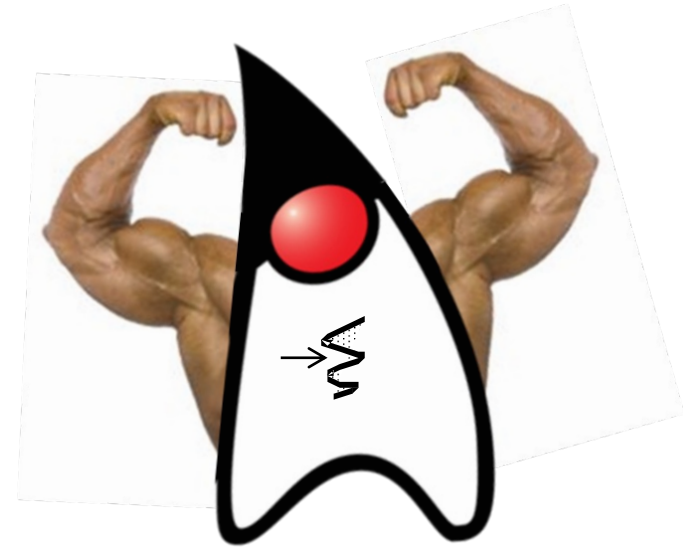
Abstract base class for tasks that run within a ForkJoinPool. A ForkJoinTask is a thread-like entity that is much lighter weight than a normal thread. Huge numbers of tasks and subtasks may be hosted by a small number of actual threads in a ForkJoinPool, at the price of some usage limitations.

# Processing Chunks in Parallel via the Common Fork-Join Pool

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



*ForkJoinTask*

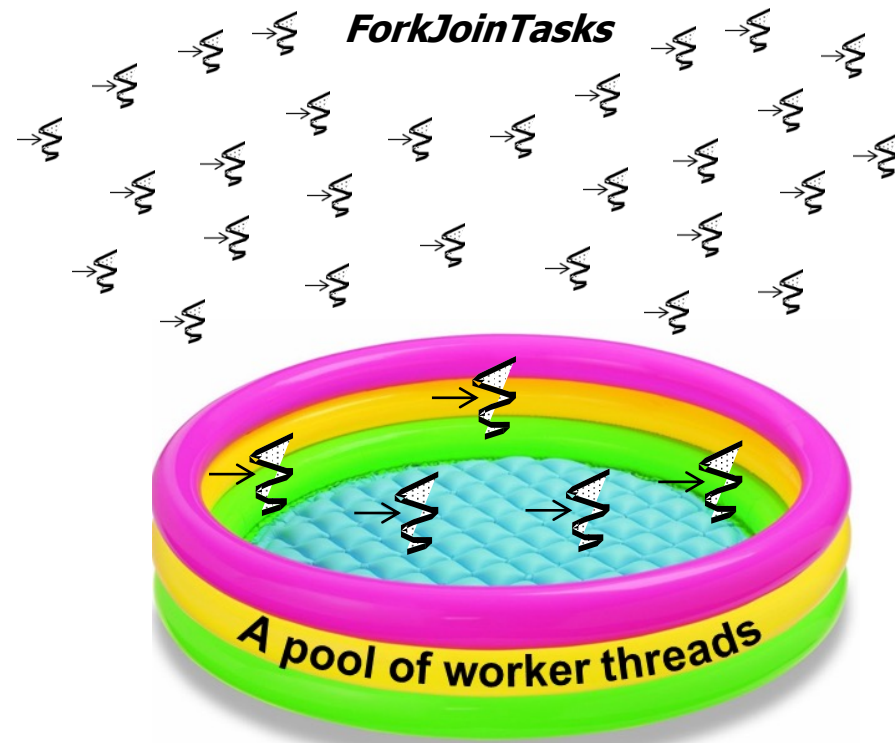


*Thread*

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

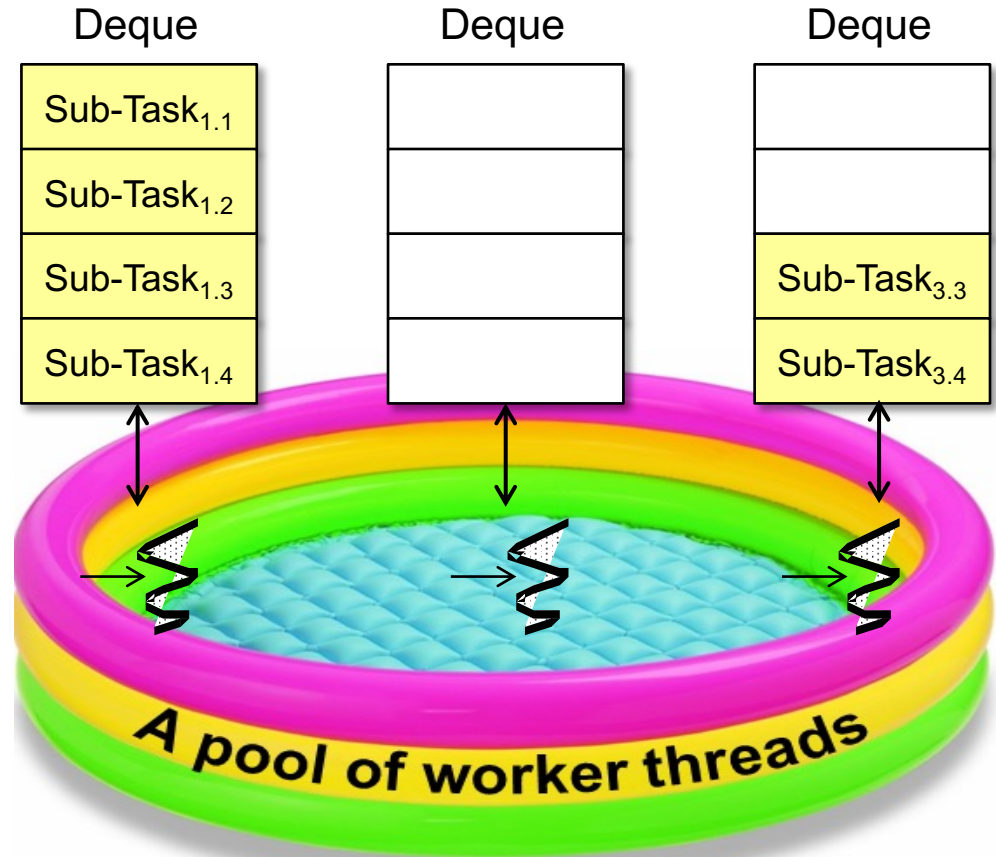
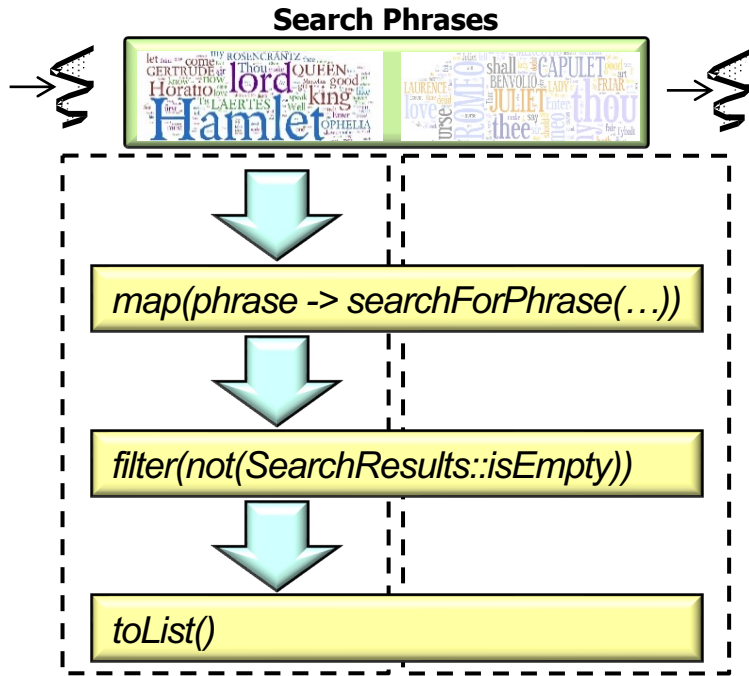
# Processing Chunks in Parallel via the Common Fork-Join Pool

- 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



# 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](https://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](https://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

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



See [dzone.com/articles/common-fork-join-pool-and-streams](https://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](https://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](http://blog.tsia.com/blog/local-or-global-resource-management-which-model-is-better)

# Processing Chunks in Parallel via the Common Fork-Join Pool

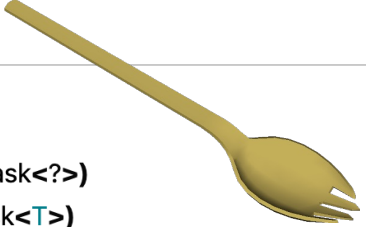
- There are few “knobs” to control this (or any) fork-join pool



See [www.infoq.com/presentations/tecniques-parallelism-java](http://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..



ForkJoinPool	
m	ForkJoinPool()
m	ForkJoinPool(int)
m	commonPool() ForkJoinPool
m	execute(Runnable) void
m	execute(ForkJoinTask<?>) void
m	invoke(ForkJoinTask<T>) T
m	invokeAll(Collection<Callable<T>>) List<Future<T>>
m	invokeAll(Collection<Callable<T>>, long, TimeUnit) List<Future<T>>
m	invokeAny(Collection<Callable<T>>, long, TimeUnit) T
m	invokeAny(Collection<Callable<T>>) T
m	submit(Runnable, T) ForkJoinTask<T>
m	submit(ForkJoinTask<T>) ForkJoinTask<T>

See [docs.oracle.com/javase/8/docs/api/java/util/concurrent/ForkJoinPool.html](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/ForkJoinPool.html)

# 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..

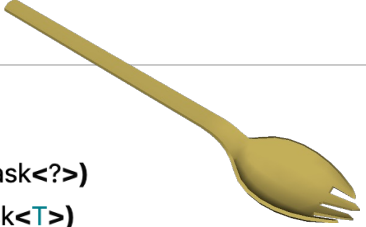


**EMERGING TECHNOLOGIES**  
FOR THE ENTERPRISE CONFERENCE

"Engineering Concurrent Library Components"

**Doug Lea**

ForkJoinPool	
m	ForkJoinPool()
m	ForkJoinPool(int)
m	commonPool() ForkJoinPool
m	execute(Runnable) void
m	execute(ForkJoinTask<?>) void
m	invoke(ForkJoinTask<T>) T
m	invokeAll(Collection<Callable<T>>) List<Future<T>>
m	invokeAll(Collection<Callable<T>>, long, TimeUnit) List<Future<T>>
m	invokeAny(Collection<Callable<T>>, long, TimeUnit) T
m	invokeAny(Collection<Callable<T>>) T
m	submit(Runnable, T) ForkJoinTask<T>
m	submit(ForkJoinTask<T>) ForkJoinTask<T>



See [www.youtube.com/watch?v=sq0MX3fHkro](http://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, e.g.
  - corePoolSize
  - maximumPoolSize
  - keepAliveTime
  - workQueue
  - threadFactory



Worker	
Worker(Runnable)	
interruptIfStarted()	void
isHeldExclusively()	boolean
isLocked()	boolean
lock()	void
run()	void
tryAcquire(int)	boolean
tryLock()	boolean
tryRelease(int)	boolean
unlock()	void

ThreadPoolExecutor	
ThreadPoolExecutor(int, int, long, TimeUnit, BlockingQueue<Runnable>)	
ThreadPoolExecutor(int, int, long, TimeUnit, BlockingQueue<Runnable>, ThreadFactory)	
ThreadPoolExecutor(int, int, long, TimeUnit, BlockingQueue<Runnable>, ThreadFactory, int)	
ThreadPoolExecutor(int, int, long, TimeUnit, BlockingQueue<Runnable>, ThreadFactory, int, long)	
afterExecute(Runnable, Throwable)	void
allowCoreThreadTimeOut(boolean)	void
allowsCoreThreadTimeOut()	boolean
beforeExecute(Thread, Runnable)	void
ensurePrestart()	void
execute(Runnable)	void
getCorePoolSize()	int
getKeepAliveTime(TimeUnit)	long
getMaximumPoolSize()	int
getQueue()	BlockingQueue<Runnable>
getThreadFactory()	ThreadFactory
prestartAllCoreThreads()	int
prestartCoreThread()	boolean
purge()	void
remove(Runnable)	boolean
setCorePoolSize(int)	void
setKeepAliveTime(long, TimeUnit)	void
setMaximumPoolSize(int)	void
setThreadFactory(ThreadFactory)	void

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