The Java Fork-Join Pool: Structure & Functionality (Part 2)

Douglas C. Schmidt

d.schmidt@vanderbilt.edu

www.dre.vanderbilt.edu/~schmidt

Professor of Computer Science

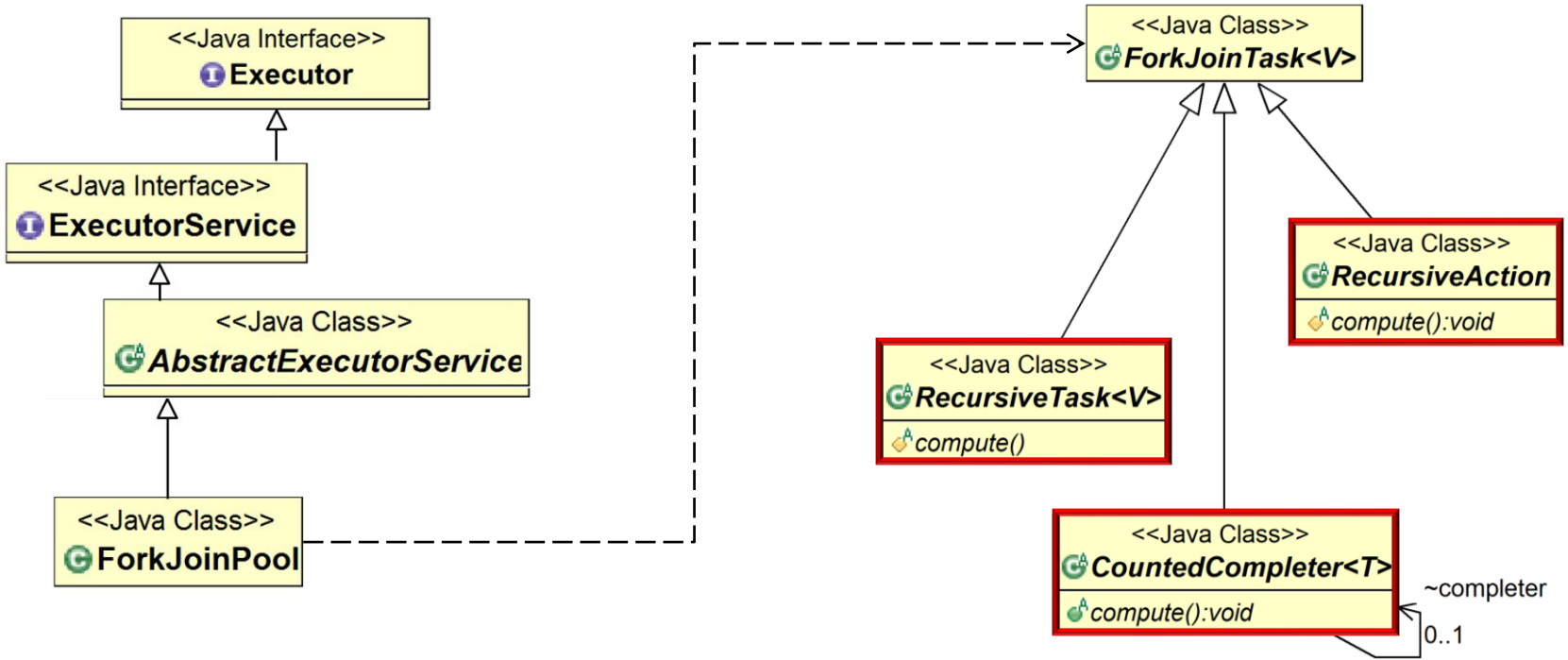
Institute for Software Integrated Systems

Vanderbilt University

Nashville, Tennessee, USA

Learning Objectives in this Part of the Lesson

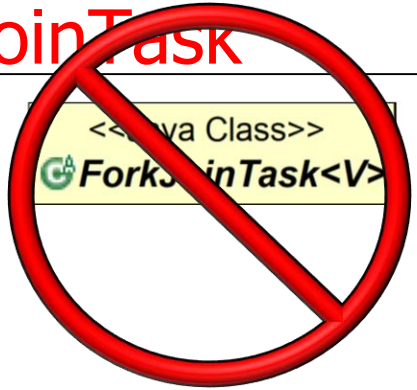
- Understand how the Java fork-join framework processes tasks in parallel
- Recognize the structure & functionality of the fork-join framework



The Subclasses of ForkJoinTask

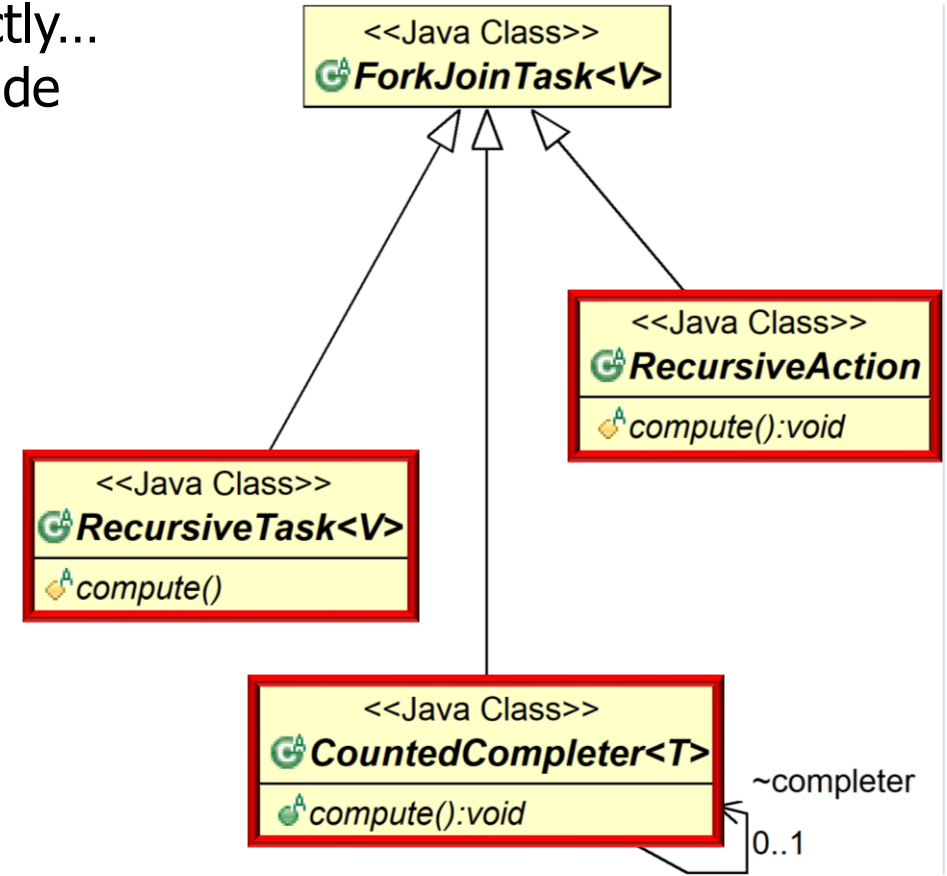
The Subclasses of ForkJoinTask

- Programs don't use ForkJoinTask directly



The Subclasses of ForkJoinTask

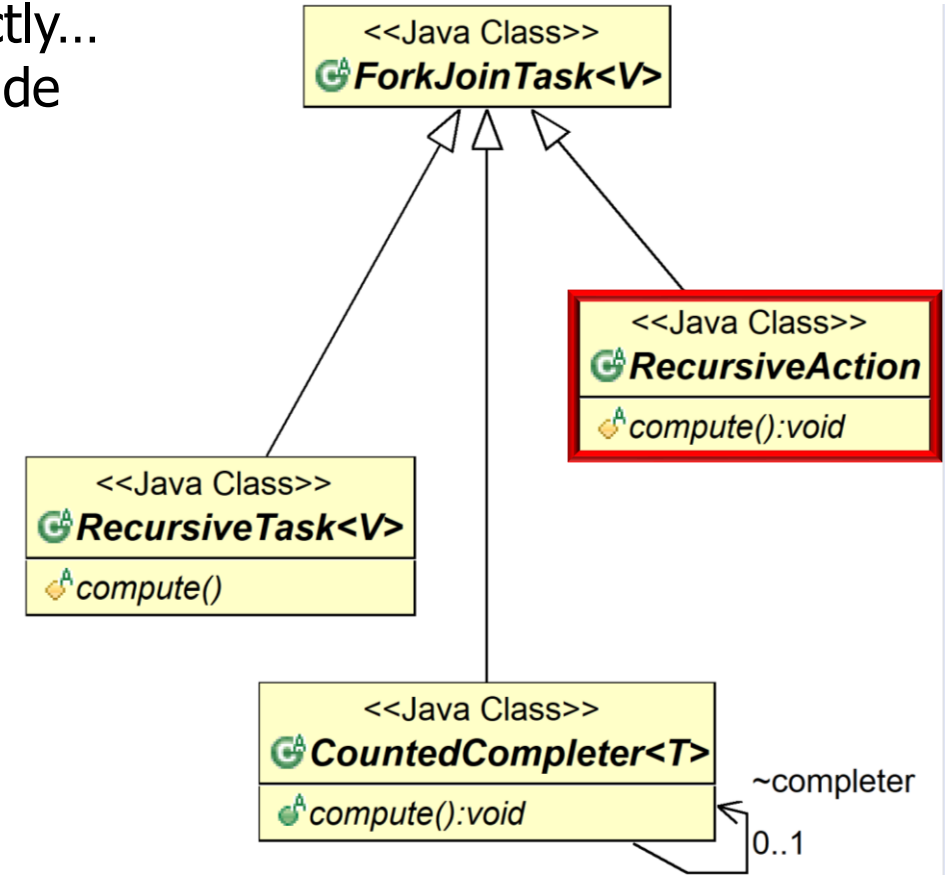
- Programs don't use ForkJoinTask directly… but instead extend a subclass & override its compute() hook method



See docs.oracle.com/javase/8/docs/api/java/util/concurrent/package-tree.html

The Subclasses of ForkJoinTask

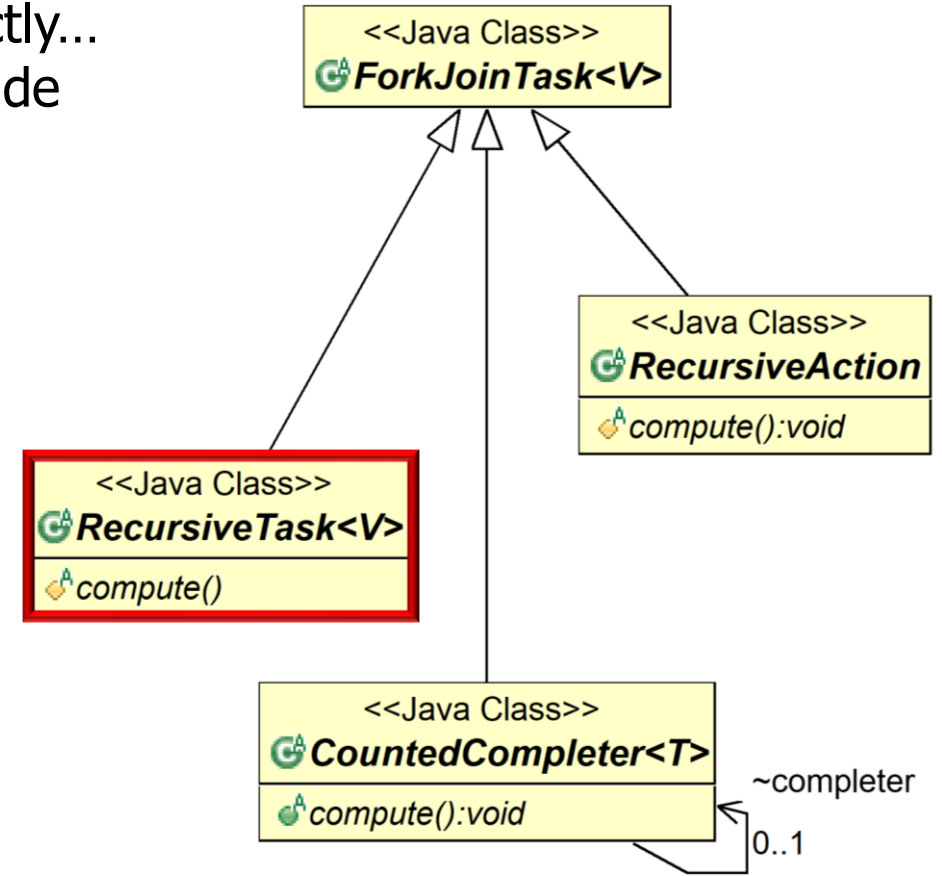
- Programs don't use ForkJoinTask directly… but instead extend a subclass & override its compute() hook method, e.g.
  - **RecursiveAction**
    - Use for computations that do not return results



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

The Subclasses of ForkJoinTask

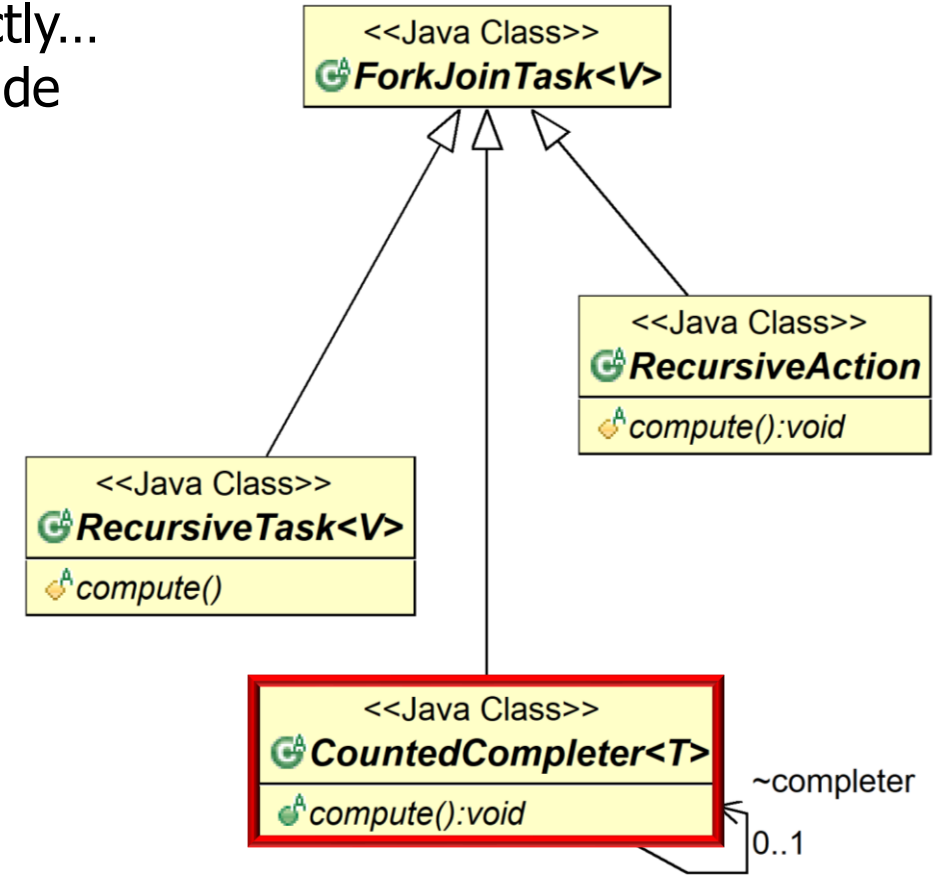
- Programs don't use ForkJoinTask directly… but instead extend a subclass & override its compute() hook method, e.g.
  - RecursiveAction
  - **RecursiveTask**
    - Use for computations that do return results



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

The Subclasses of ForkJoinTask

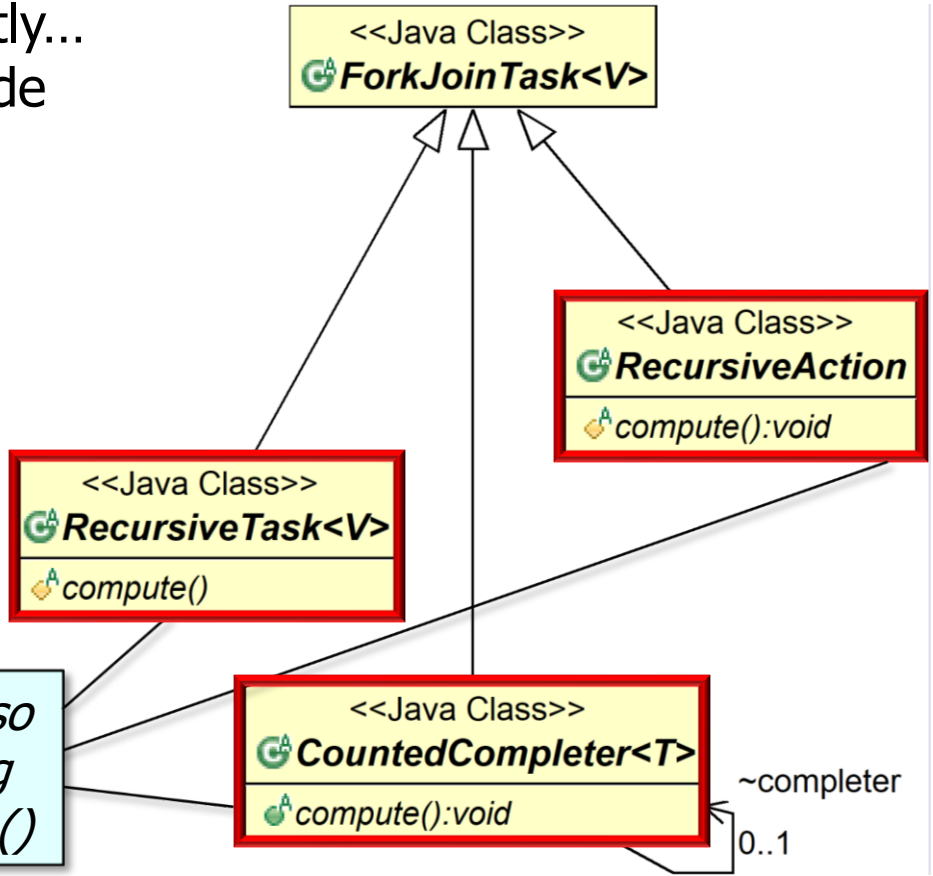
- Programs don't use ForkJoinTask directly… but instead extend a subclass & override its compute() hook method, e.g.
  - RecursiveAction
  - RecursiveTask
  - **CountedCompleter**
    - Used for computations in which completed actions trigger other actions



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

The Subclasses of ForkJoinTask

- Programs don't use ForkJoinTask directly… but instead extend a subclass & override its compute() hook method, e.g.
  - **RecursiveAction**
  - **RecursiveTask**
  - **CountedCompleter**



These classes aren't functional interfaces, so they must be subclassed rather than using lambda expressions to implement compute()

The Java 8 parallel streams framework provides a functional API to the ForkJoinPool

The Subclasses of ForkJoinTask

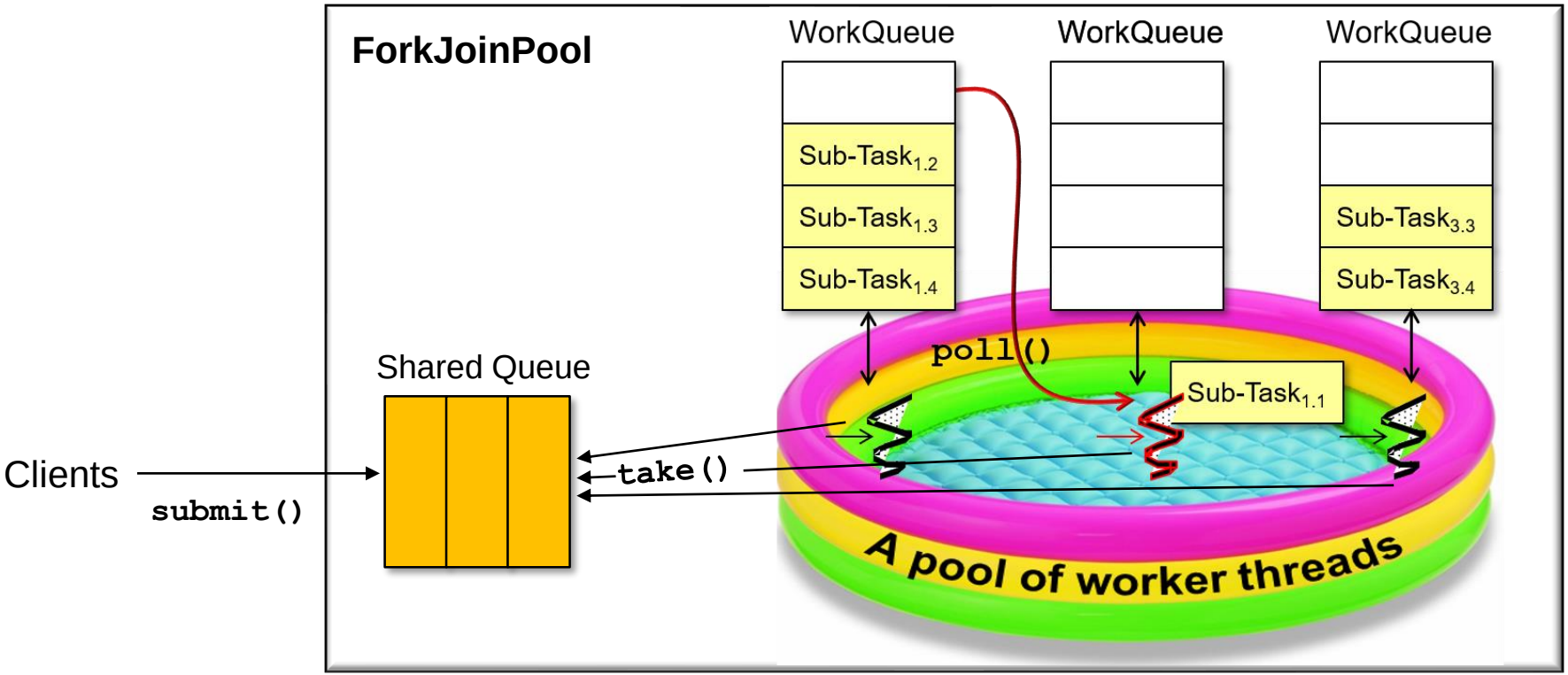
- ForkJoinPool enables non-ForkJoinTask clients to process ForkJoinTasks

void	execute(ForkJoinTask<T>) – Arrange async execution
T	invoke(ForkJoinTask<T>) – Performs the given task, returning its result upon completion
ForkJoinTask <T>	submit(ForkJoinTask) – Submits a ForkJoinTask for execution, returns a future

We'll discuss these methods later in part 3 of this lesson

The Subclasses of ForkJoinTask

- Clients insert new tasks onto a fork-join pool's shared queued, which feeds "work-stealing" queues managed by worker threads



See en.wikipedia.org/wiki/Work_stealing

The Subclasses of ForkJoinTask

- Clients insert new tasks onto a fork-join pool's shared queued, which feeds "work-stealing" queues managed by worker threads
  - The goal of "work-stealing" is to maximize processor core utilization

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

The Subclasses of ForkJoinTask

- There are (intentionally) few "knobs" that can control a fork-join pool

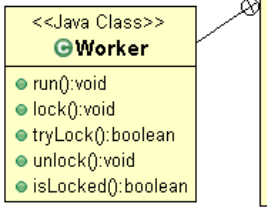
<<Java Class>>
ForkJoinPool

- ForkJoinPool()
- ForkJoinPool(int)
- ForkJoinPool(int,ForkJoinWorkerThreadFactory,UncaughtExceptionHandler,boolean)
- commonPool():ForkJoinPool
- invoke(ForkJoinTask<T>)
- execute(ForkJoinTask<?>):void
- execute(Runnable):void
- submit(ForkJoinTask<T>):ForkJoinTask<T>
- submit(Callable<T>):ForkJoinTask<T>
- submit(Runnable,T):ForkJoinTask<T>
- submit(Runnable):ForkJoinTask<?>
- invokeAll(Collection<Callable<T>>):List<Future<T>>
- shutdown():void
- shutdownNow():List<Runnable>
- isTerminated():boolean
- isTerminating():boolean
- isShutdown():boolean
- awaitTermination(long,TimeUnit):boolean

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

The Subclasses of ForkJoinTask

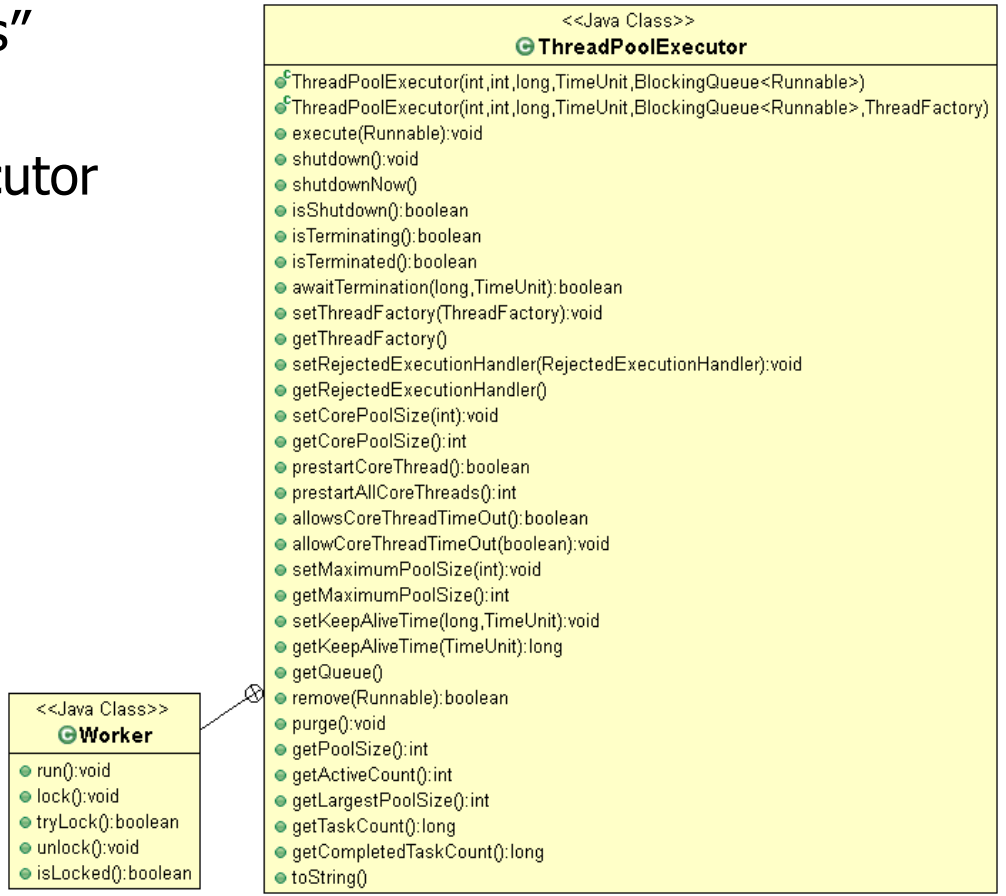
- There are (intentionally) few "knobs" that can control a fork-join pool
  - Contrast with the ThreadPoolExecutor framework



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

The Subclasses of ForkJoinTask

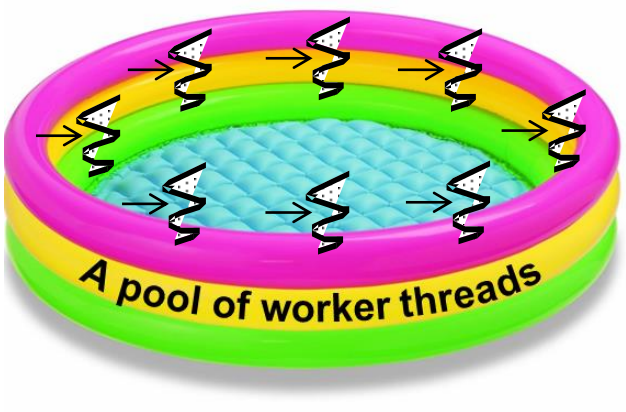
- There are (intentionally) few "knobs" that can control a fork-join pool
  - Contrast with the ThreadPoolExecutor framework, e.g.
    - corePool size
    - maxPool size
    - workQueue
    - keepAliveTime
    - threadFactory
    - rejectedExecutionHandler



See dzone.com/articles/a-deep-dive-into-the-java-executor-service

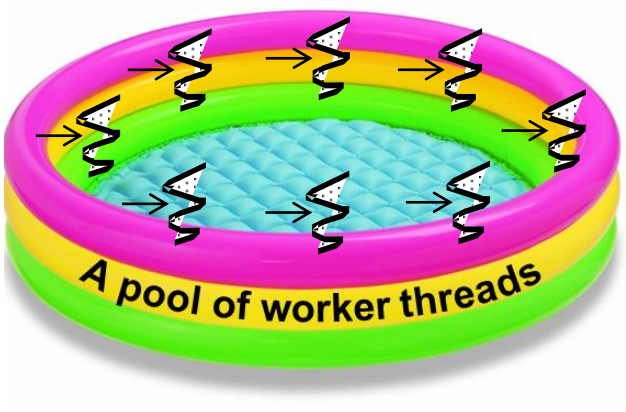
The Subclasses of ForkJoinTask

- There are (intentionally) few "knobs" that can control a fork-join pool
  - Contrast with the ThreadPoolExecutor framework
  - However, you *can* configure the size of the common fork-join pool



The Subclasses of ForkJoinTask

- There are (intentionally) few "knobs" that can control a fork-join pool
  - Contrast with the ThreadPoolExecutor framework
  - However, you *can* configure the size of the common fork-join pool



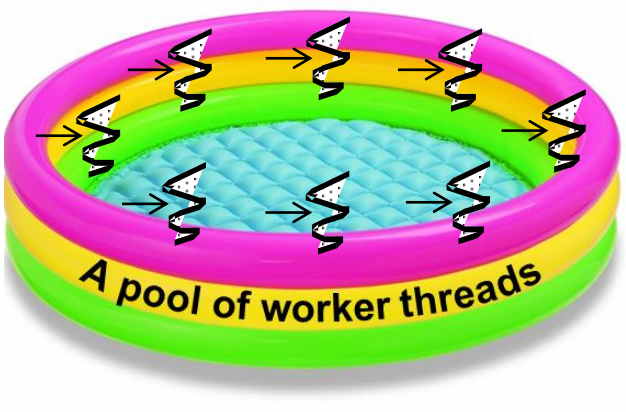

```
System.setProperty
        ("java.util.concurrent"
         + ".ForkJoinPool.common"
         + ".parallelism", 8);
```

# of desired threads

See lesson on "*The Java Fork-Join Pool: Overview of the Common Fork-Join Pool*"

The Subclasses of ForkJoinTask

- There are (intentionally) few "knobs" that can control a fork-join pool
  - Contrast with the ThreadPoolExecutor framework
  - However, you *can* configure the size of the common fork-join pool



Interface ForkJoinPool.ManagedBlocker

Enclosing class:

ForkJoinPool

```
public static interface ForkJoinPool.ManagedBlocker
```

Interface for extending managed parallelism for tasks running in ForkJoinPools.

See lesson on "*The Java Fork-Join Pool: the ManagedBlocker Interface*"

End of the Java Fork-Join Pool: Structure & Functionality (Part 2)