The Java ForkJoinPool Class

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

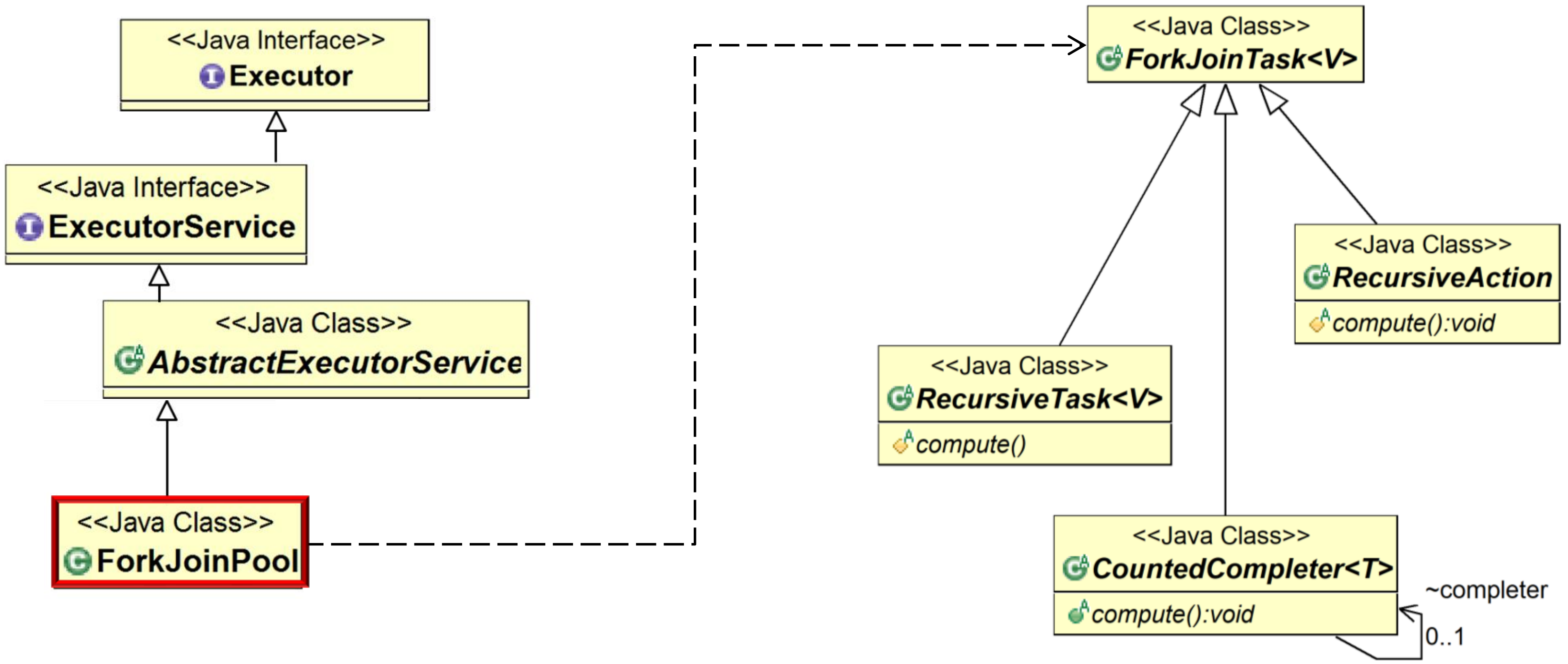
Professor of Computer Science

Institute for Software
Integrated Systems

Vanderbilt University
Nashville, Tennessee, USA

Learning Objectives in this Part of the Lesson

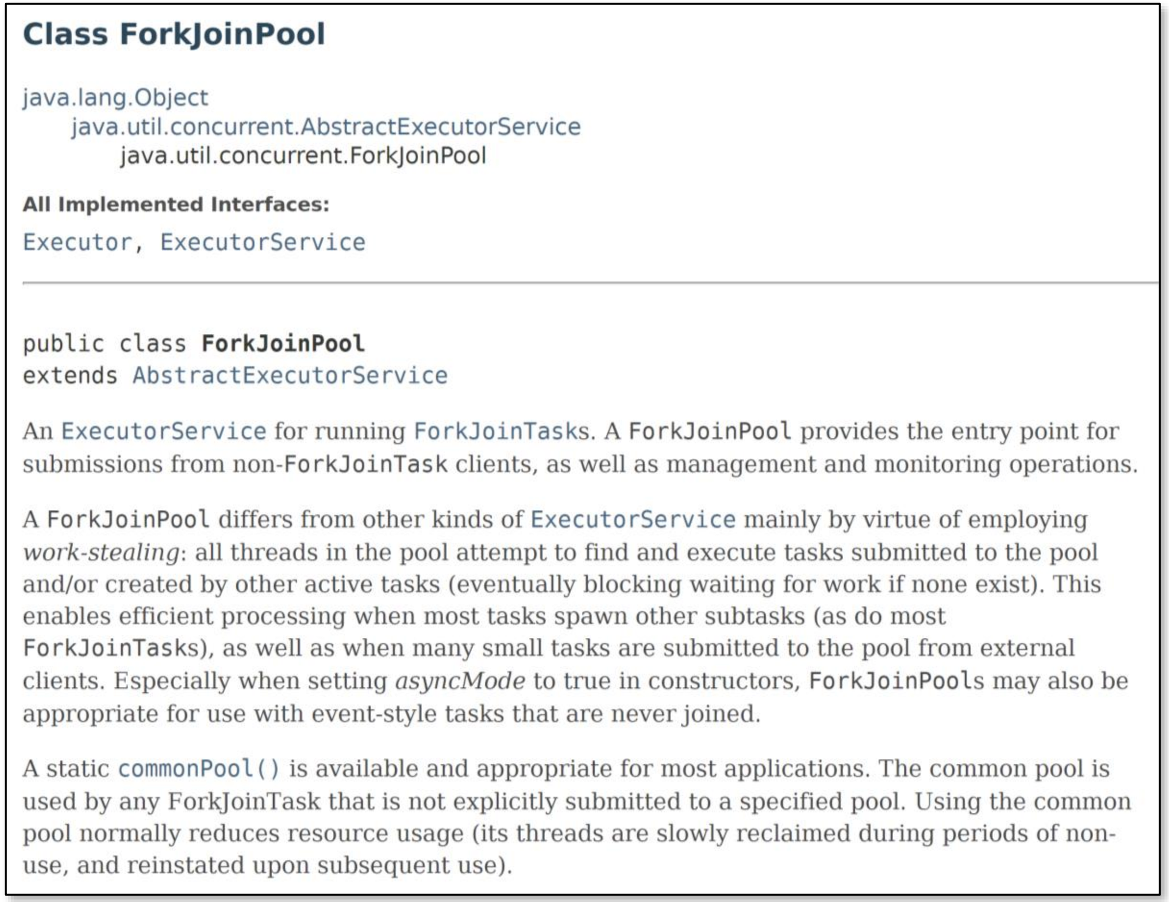
- Understand how the Java fork-join framework processes tasks in parallel
- Recognize the structure & functionality of the fork-join framework



Overview of the ForkJoinPool Class

Overview of the ForkJoinPool Class

- ForkJoinPool implements the ExecutorService interface



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.

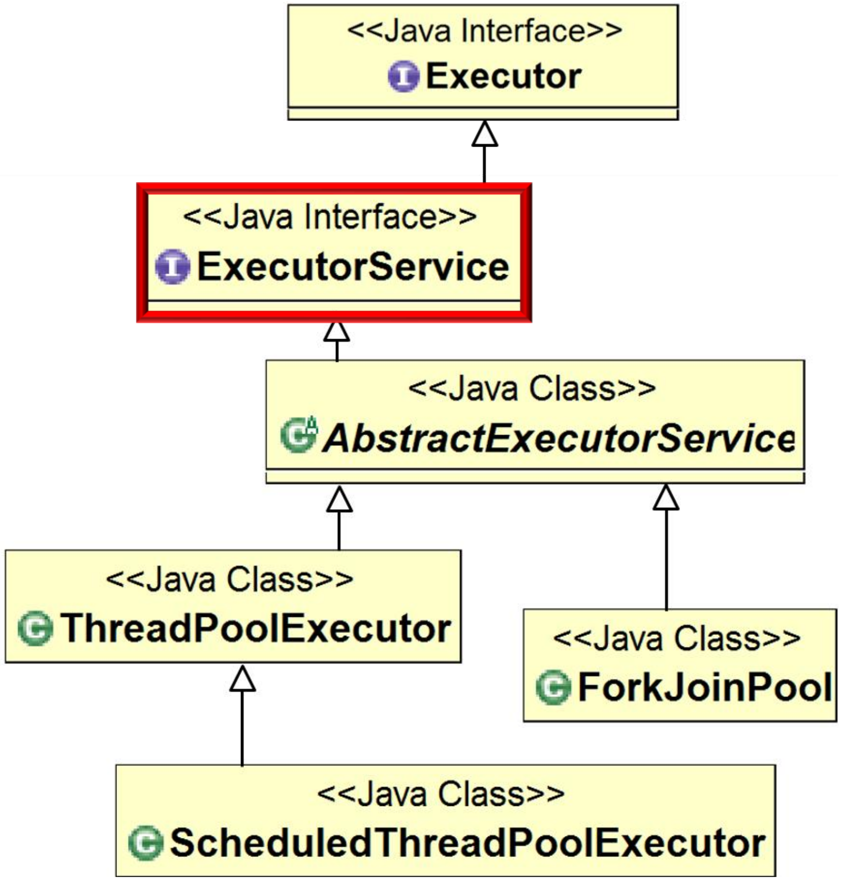
A ForkJoinPool differs from other kinds of ExecutorService mainly by virtue of employing *work-stealing*: all threads in the pool attempt to find and execute tasks submitted to the pool and/or created by other active tasks (eventually blocking waiting for work if none exist). This enables efficient processing 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).

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

Overview of the ForkJoinPool Class

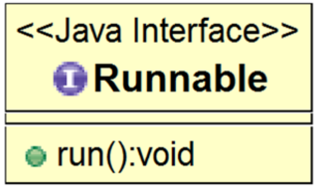
- ForkJoinPool implements the ExecutorService interface
  - This interface is the basis for Java Executor framework subclasses

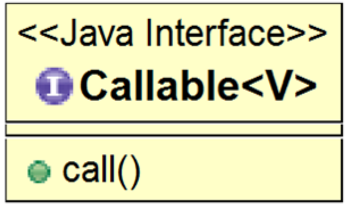


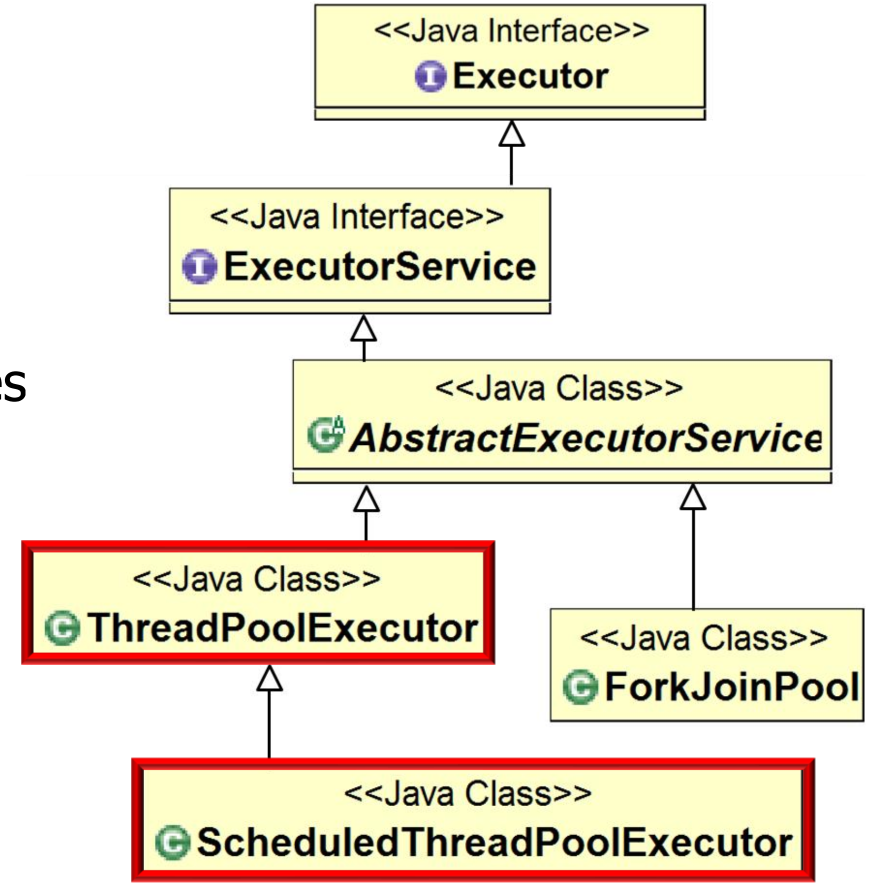
See docs.oracle.com/javase/tutorial/essential/concurrency/executors.html

Overview of the ForkJoinPool Class

- ForkJoinPool implements the ExecutorService interface
  - This interface is the basis for Java Executor framework subclasses
  - Other implementations of Executor Service execute runnables or callables

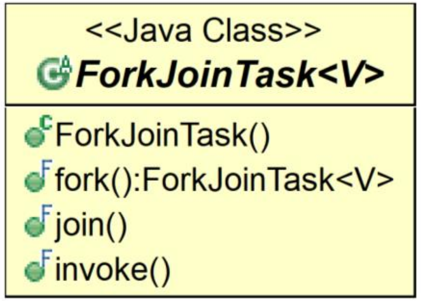


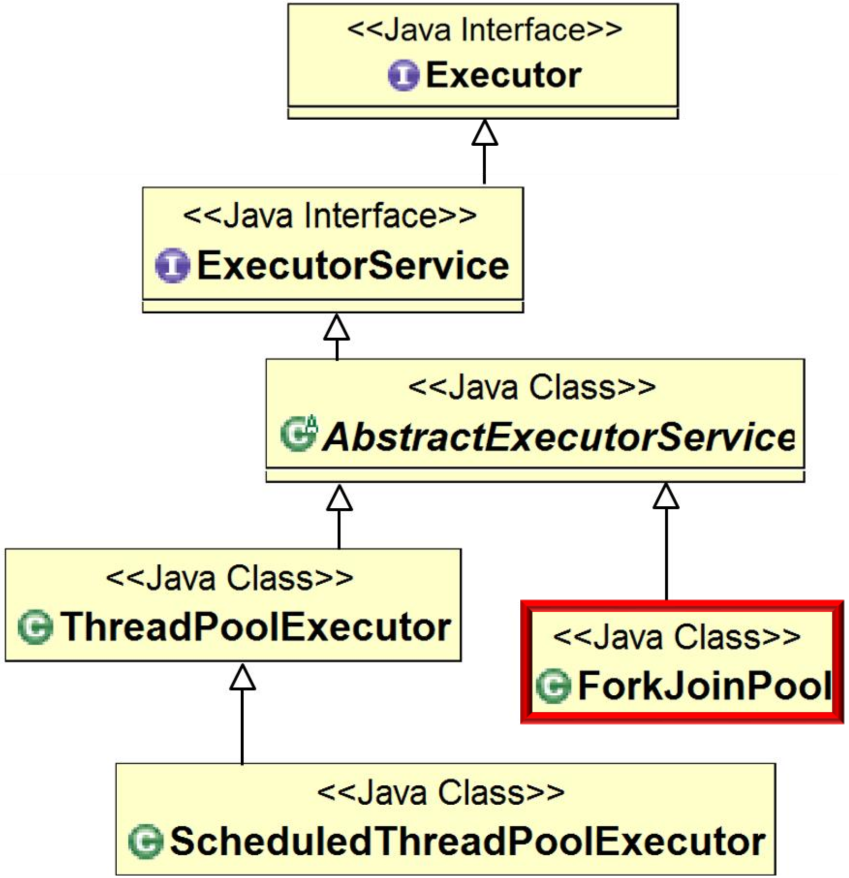




Overview of the ForkJoinPool Class

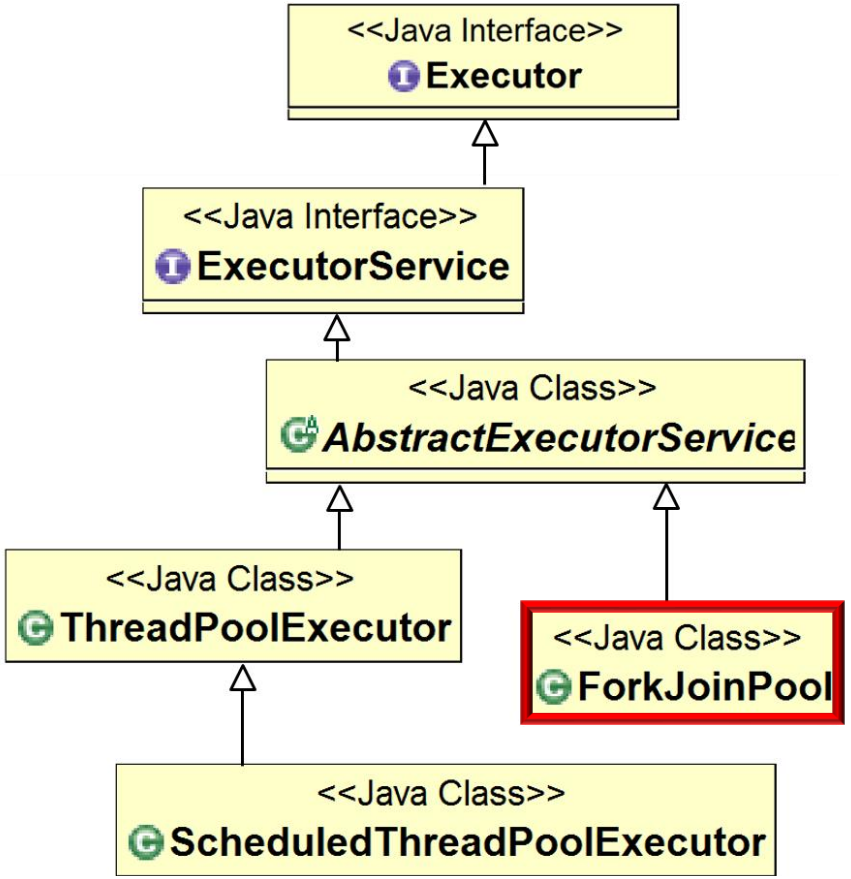
- ForkJoinPool implements the ExecutorService interface
  - This interface is the basis for Java Executor framework subclasses
  - Other implementations of Executor Service execute runnables or callables
  - In contrast, the ForkJoinPool executes ForkJoinTasks





Overview of the ForkJoinPool Class

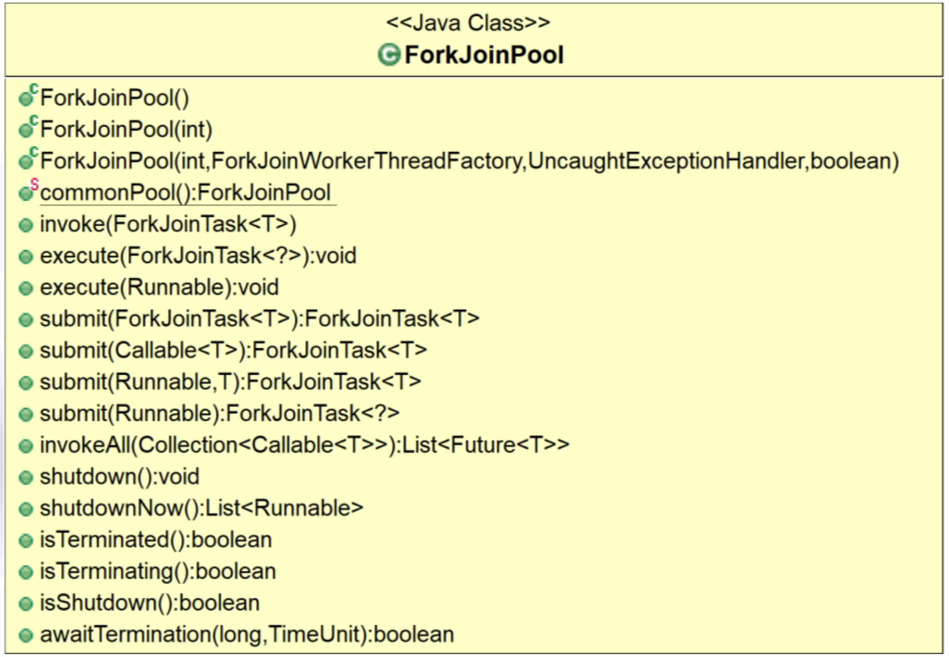
- ForkJoinPool implements the ExecutorService interface
  - This interface is the basis for Java Executor framework subclasses
  - Other implementations of Executor Service execute runnables or callables
  - In contrast, the ForkJoinPool executes ForkJoinTasks



It can also execute runnables & callables, but that's not its main purpose

Overview of the ForkJoinPool Class

- There are (intentionally) few “knobs” that can control a ForkJoinPool

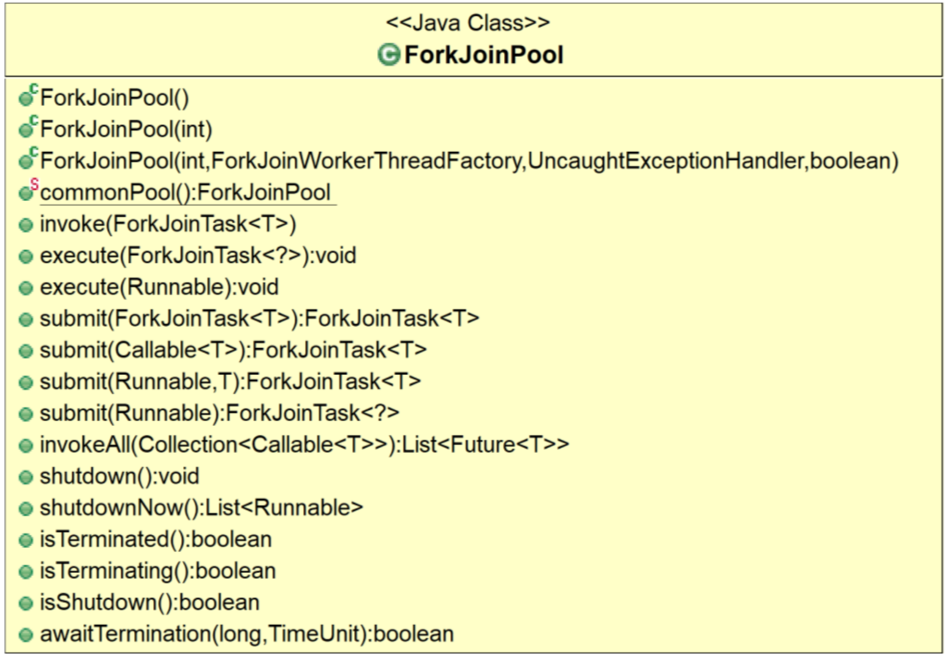


```
<<Java Class>>
ForkJoinPool

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
```

Overview of the ForkJoinPool Class

- There are (intentionally) few "knobs" that can control a ForkJoinPool
  - The design goal was to make the ForkJoinPool implementation so clever that programmers can't improve on its default behavior!



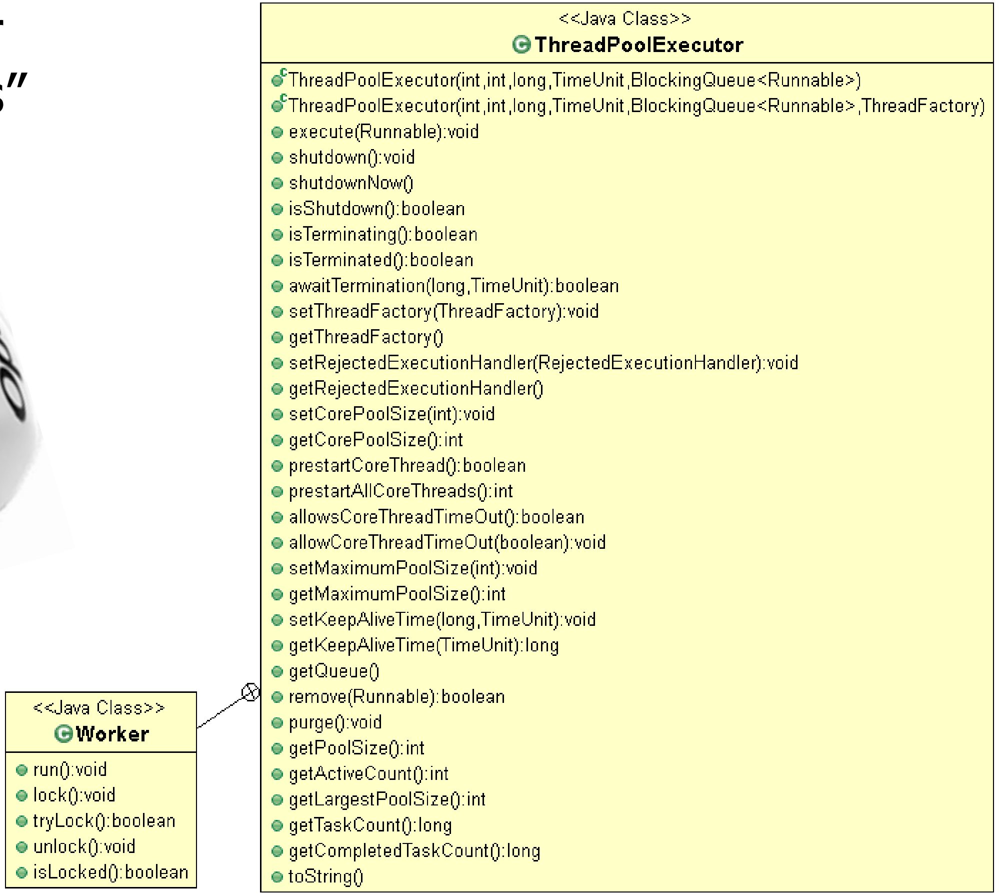
<<Java Class>>
ForkJoinPool
ForkJoinPool()
ForkJoinPool(int)
ForkJoinPool(int,ForkJoinWorkerThreadFactory,UncaughtExceptionHandler,boolean)
commonPool():ForkJoinPool
invoke(ForkJoinTask<T>)
execute(ForkJoinTask<?>):void
execute(Runnable):void
submit(ForkJoinTask<T>):ForkJoinTask<T>
submit(Callable<T>):ForkJoinTask<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

EMERGING TECHNOLOGIES
FOR THE ENTERPRISE CONFERENCE
"Engineering Concurrent Library Components"
Doug Lea

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

Overview of the ForkJoinPool Class

- In contrast, the ThreadPoolExecutor framework has many control "knobs"


```
<<Java Class>>
ThreadPoolExecutor
ThreadPoolExecutor(int,int,long,TimeUnit,BlockingQueue<Runnable>)
ThreadPoolExecutor(int,int,long,TimeUnit,BlockingQueue<Runnable>,ThreadFactory)
execute(Runnable):void
shutdown():void
shutdownNow()
isShutdown():boolean
isTerminating():boolean
isTerminated():boolean
awaitTermination(long,TimeUnit):boolean
setThreadFactory(ThreadFactory):void
getThreadFactory()
setRejectedExecutionHandler(RejectedExecutionHandler):void
getRejectedExecutionHandler()
setCorePoolSize(int):void
getCorePoolSize():int
prestartCoreThread():boolean
prestartAllCoreThreads():int
allowsCoreThreadTimeOut():boolean
allowCoreThreadTimeOut(boolean):void
setMaximumPoolSize(int):void
getMaximumPoolSize():int
setKeepAliveTime(long,TimeUnit):void
getKeepAliveTime(TimeUnit):long
getQueue()
remove(Runnable):boolean
purge():void
getPoolSize():int
getActiveCount():int
getLargestPoolSize():int
getTaskCount():long
getCompletedTaskCount():long
toString()
```

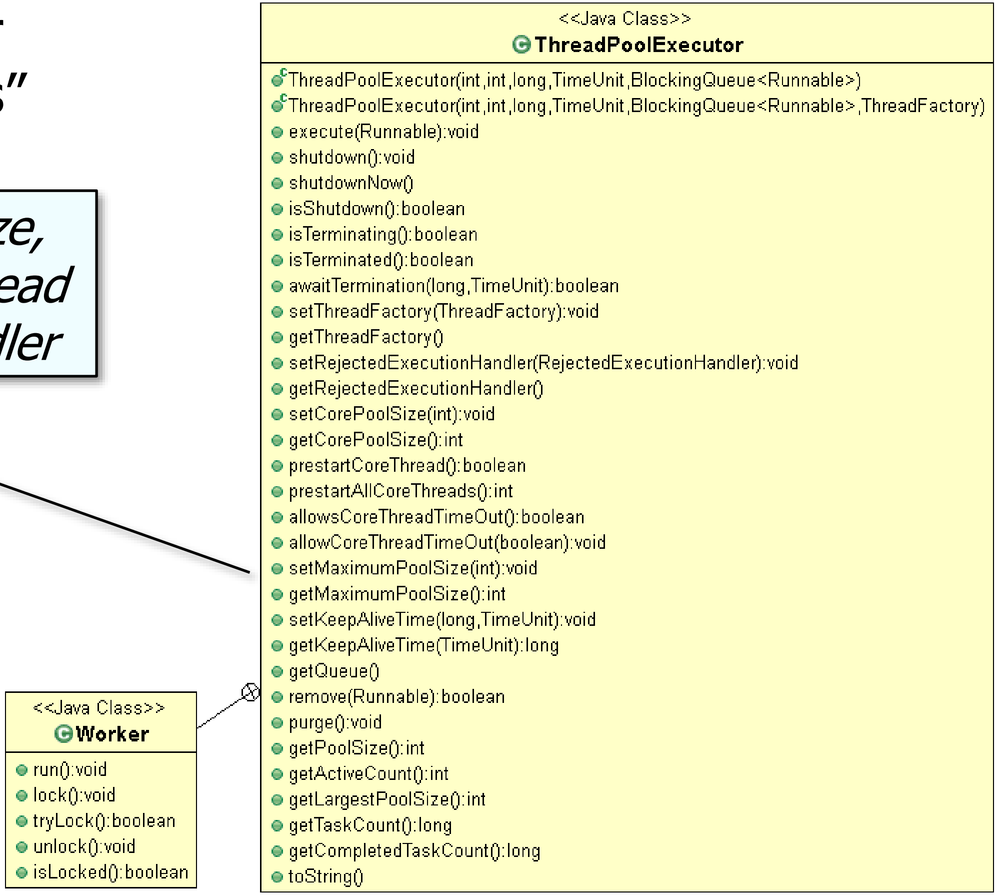
```
<<Java Class>>
Worker
run():void
lock():void
tryLock():boolean
unlock():void
isLocked():boolean
```

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

Overview of the ForkJoinPool Class

- In contrast, the ThreadPoolExecutor framework has many control "knobs"

e.g., corePool size, maxPool size, workQueue, keepAliveTime, thread Factory, rejectedExecutionHandler

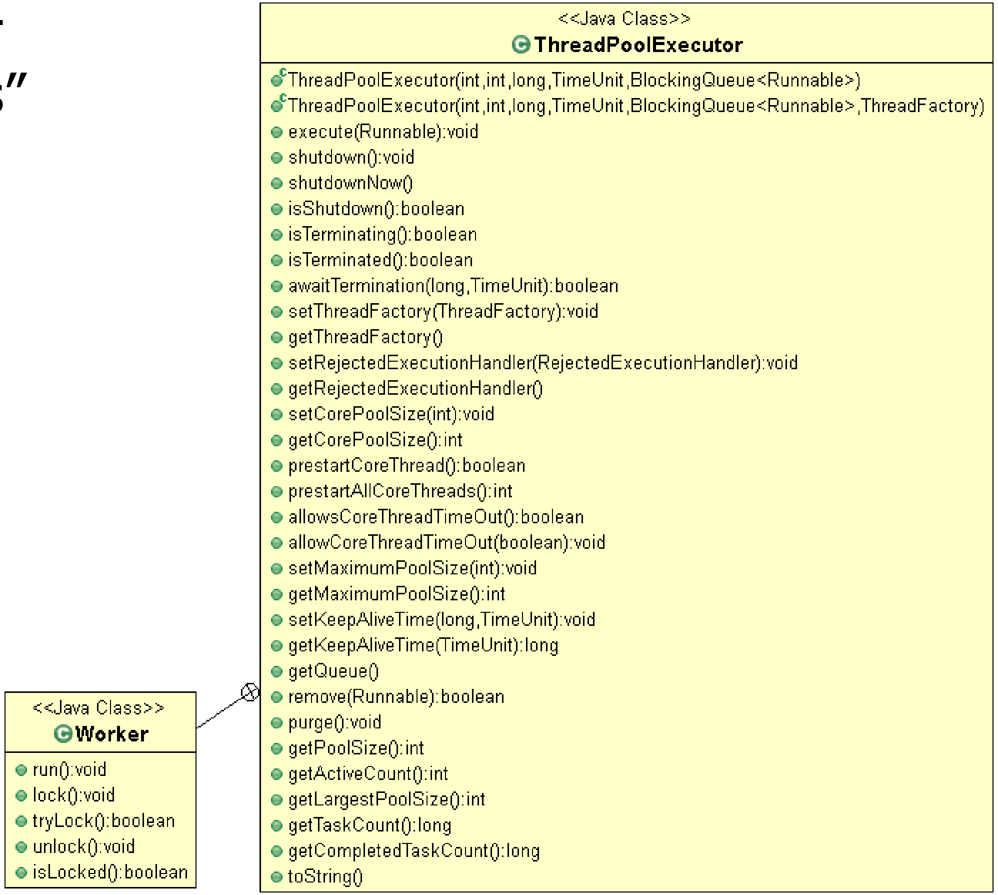


<<Java Class>>
ThreadPoolExecutor
ThreadPoolExecutor(int,int,long,TimeUnit,BlockingQueue<Runnable>)
ThreadPoolExecutor(int,int,long,TimeUnit,BlockingQueue<Runnable>,ThreadFactory)
execute(Runnable):void
shutdown():void
shutdownNow()
isShutdown():boolean
isTerminating():boolean
isTerminated():boolean
awaitTermination(long,TimeUnit):boolean
setThreadFactory(ThreadFactory):void
getThreadFactory()
setRejectedExecutionHandler(RejectedExecutionHandler):void
getRejectedExecutionHandler()
setCorePoolSize(int):void
getCorePoolSize():int
prestartCoreThread():boolean
prestartAllCoreThreads():int
allowsCoreThreadTimeOut():boolean
allowCoreThreadTimeOut(boolean):void
setMaximumPoolSize(int):void
getMaximumPoolSize():int
setKeepAliveTime(long,TimeUnit):void
getKeepAliveTime(TimeUnit):long
getQueue()
remove(Runnable):boolean
purge():void
getPoolSize():int
getActiveCount():int
getLargestPoolSize():int
getTaskCount():long
getCompletedTaskCount():long
toString()
<<Java Class>>
Worker
run():void
lock():void
tryLock():boolean
unlock():void
isLocked():boolean

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

Overview of the ForkJoinPool Class

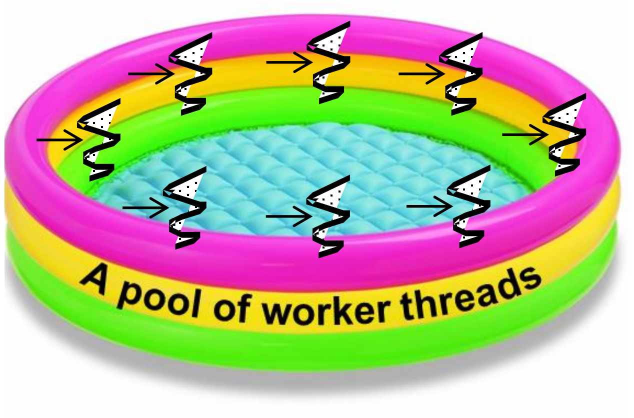
- In contrast, the ThreadPoolExecutor framework has many control "knobs"
  - The design goal was to enable programmers to maximally customize ThreadPoolExecutor


```
<<Java Class>>
ThreadPoolExecutor
ThreadPoolExecutor(int,int,long,TimeUnit,BlockingQueue<Runnable>)
ThreadPoolExecutor(int,int,long,TimeUnit,BlockingQueue<Runnable>,ThreadFactory)
execute(Runnable):void
shutdown():void
shutdownNow()
isShutdown():boolean
isTerminating():boolean
isTerminated():boolean
awaitTermination(long,TimeUnit):boolean
setThreadFactory(ThreadFactory):void
getThreadFactory()
setRejectedExecutionHandler(RejectedExecutionHandler):void
getRejectedExecutionHandler()
setCorePoolSize(int):void
getCorePoolSize():int
prestartCoreThread():boolean
prestartAllCoreThreads():int
allowsCoreThreadTimeOut():boolean
allowCoreThreadTimeOut(boolean):void
setMaximumPoolSize(int):void
getMaximumPoolSize():int
setKeepAliveTime(long,TimeUnit):void
getKeepAliveTime(TimeUnit):long
getQueue()
remove(Runnable):boolean
purge():void
getPoolSize():int
getActiveCount():int
getLargestPoolSize():int
getTaskCount():long
getCompletedTaskCount():long
toString()
```

```
<<Java Class>>
Worker
run():void
lock():void
tryLock():boolean
unlock():void
isLocked():boolean
```

Overview of the ForkJoinPool Class

- However, you *can* configure the size of the common fork-join pool

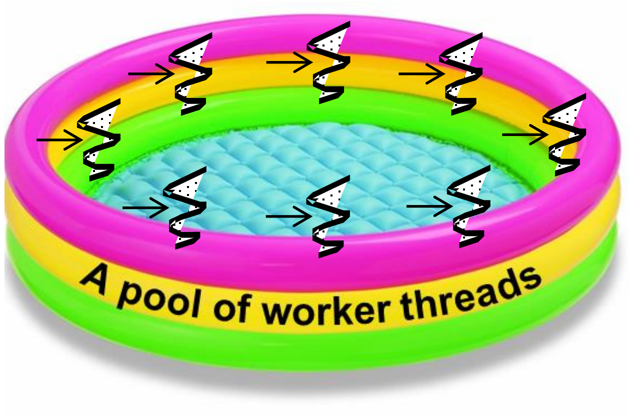


Overview of the ForkJoinPool Class

- However, you *can* configure the size of the common fork-join pool

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

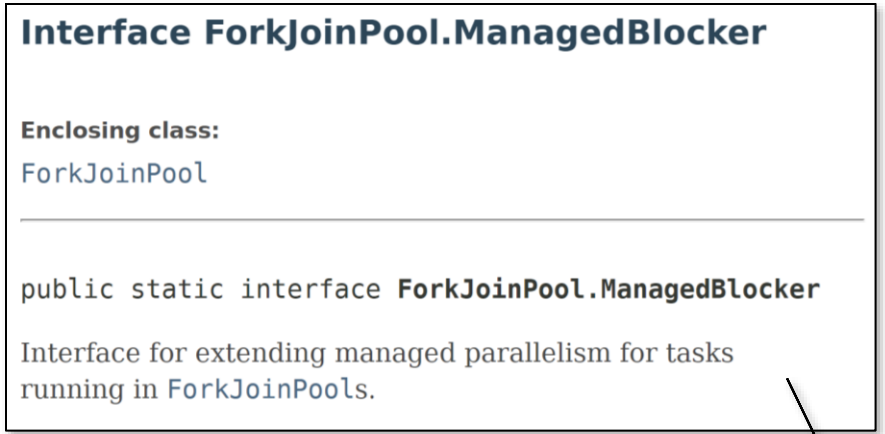
Explicitly set the desired # of threads



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

Overview of the ForkJoinPool Class

- However, you *can* configure the size of the common fork-join pool

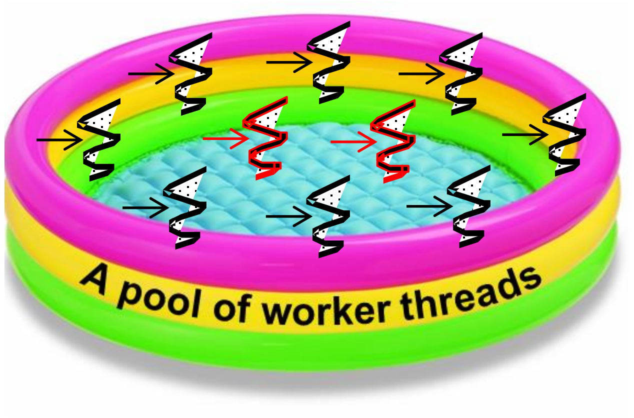


Interface ForkJoinPool.ManagedBlocker

Enclosing class:
ForkJoinPool

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

Interface for extending managed parallelism for tasks running in ForkJoinPools.



Dynamically adjust the # of threads

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

End of the Java ForkJoinPool Class