The Java ForkJoinPool Class

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

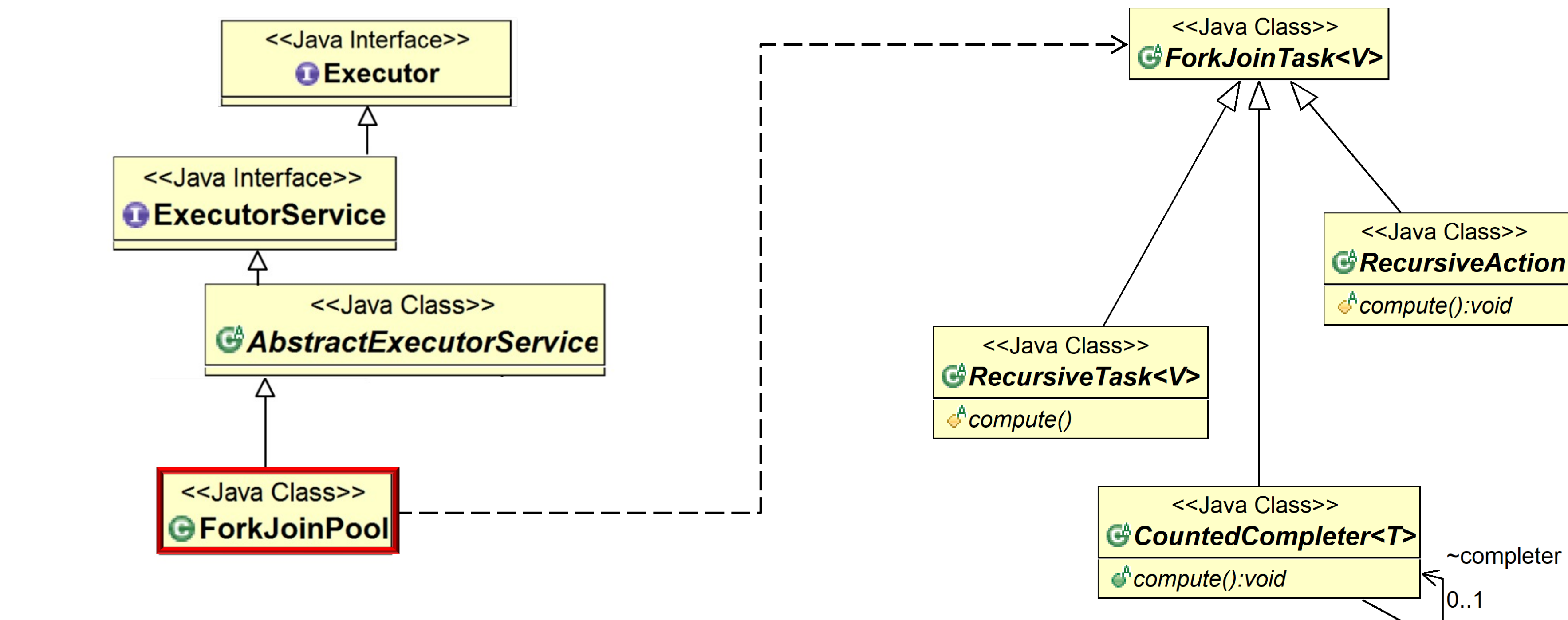
Professor of Computer Science

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

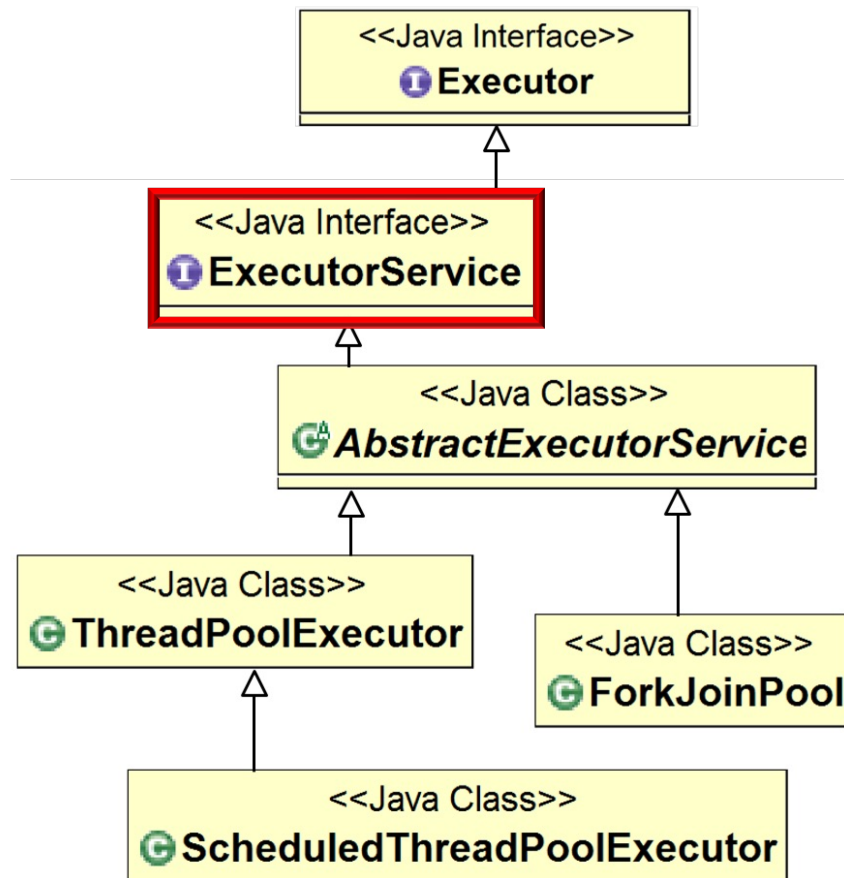
A `ForkJoinPool` differs from other kinds of `ExecutorService` mainly by virtue of employing *work-stealing*: all threads in the pool attempt to find and execute tasks submitted to the pool and/or created by other active tasks (eventually blocking waiting for work if none exist). This enables efficient processing when most tasks spawn other subtasks (as do most `ForkJoinTask`s), as well as when many small tasks are submitted to the pool from external clients. Especially when setting *asyncMode* to true in constructors, `ForkJoinPool`s 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

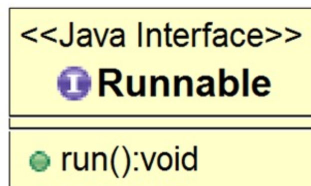
- ForkJoinPool implements the ExecutorService interface
  - This interface is the basis for Java Executor framework subclasses

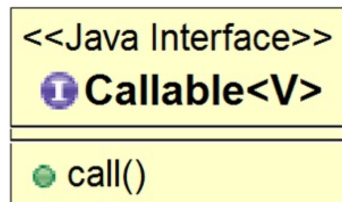


See docs.oracle.com/javase/tutorial/essential/concurrency/executors.html

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





<<Java Interface>>
Executor

<<Java Interface>>
ExecutorService

<<Java Class>>
AbstractExecutorService

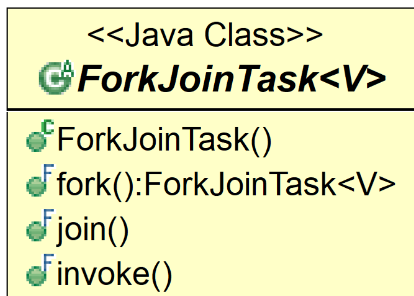
<<Java Class>>
ThreadPoolExecutor

<<Java Class>>
ForkJoinPool

<<Java Class>>
ScheduledThreadPoolExecutor

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



<<Java Interface>>
Executor

<<Java Interface>>
ExecutorService

<<Java Class>>
AbstractExecutorService

<<Java Class>>
ThreadPoolExecutor

<<Java Class>>
ForkJoinPool

<<Java Class>>
ScheduledThreadPoolExecutor

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

<<Java Interface>>
Executor

<<Java Interface>>
ExecutorService

<<Java Class>>
AbstractExecutorService

<<Java Class>>
ThreadPoolExecutor

<<Java Class>>
ForkJoinPool

<<Java Class>>
ScheduledThreadPoolExecutor

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

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

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!

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

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"

ThreadPoolExecutor	
ThreadPoolExecutor(int, int, long, TimeUnit, BlockingQueue<Ru	
ThreadPoolExecutor(int, int, long, TimeUnit, BlockingQueue<Ru	
ThreadPoolExecutor(int, int, long, TimeUnit, BlockingQueue<Ru	
ThreadPoolExecutor(int, int, long, TimeUnit, BlockingQueue<Ru	
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

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

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

ThreadPoolExecutor	
ThreadPoolExecutor(int, int, long, TimeUnit, BlockingQueue<Ru	
ThreadPoolExecutor(int, int, long, TimeUnit, BlockingQueue<Ru	
ThreadPoolExecutor(int, int, long, TimeUnit, BlockingQueue<Ru	
ThreadPoolExecutor(int, int, long, TimeUnit, BlockingQueue<Ru	
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

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

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 goal was to enable programmers to maximally customize instances of ThreadPoolExecutor

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<Ru	
ThreadPoolExecutor(int, int, long, TimeUnit, BlockingQueue<Ru	
ThreadPoolExecutor(int, int, long, TimeUnit, BlockingQueue<Ru	
ThreadPoolExecutor(int, int, long, TimeUnit, BlockingQueue<Ru	
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

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