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
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](http://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
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 Subclasses of ForkJoinTask

- ForkJoinPool enables non-ForkJoinTask clients to process ForkJoinTasks

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>void</td>
<td><strong>execute(ForkJoinTask&lt;T&gt;)</strong> – Arrange async execution</td>
</tr>
<tr>
<td>T</td>
<td><strong>invoke(ForkJoinTask&lt;T&gt;)</strong> – Performs the given task, returning its result upon completion</td>
</tr>
<tr>
<td>ForkJoinTask&lt;T&gt;</td>
<td><strong>submit(ForkJoinTask&lt;T&gt;)</strong> – Submits a ForkJoinTask for execution, returns a future</td>
</tr>
</tbody>
</table>

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

See [www.youtube.com/watch?v=sq0MX3fHkro](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
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
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
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.

Example:

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

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

See lesson on “The Java Fork-Join Pool: the ManagedBlocker Interface”
End of the Java Fork-Join Pool: Structure & Functionality (Part 2)