Maximizing Processor Core Utilization with the Java Common Fork-Join Pool

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

Professor of Computer Science

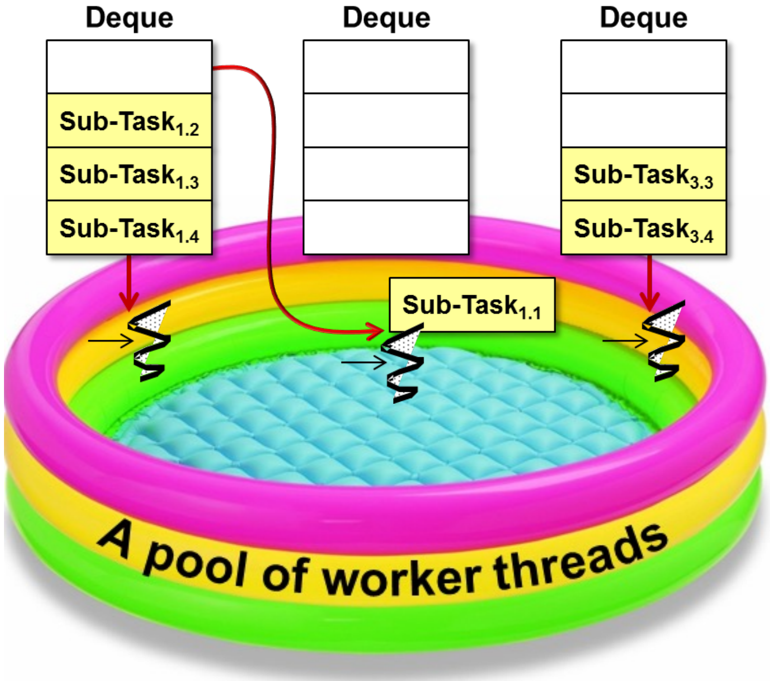
Institute for Software
Integrated Systems

Vanderbilt University
Nashville, Tennessee, USA

Learning Objectives in this Part of the Lesson

- Understand how the common fork-join pool helps to maximize processor core utilization

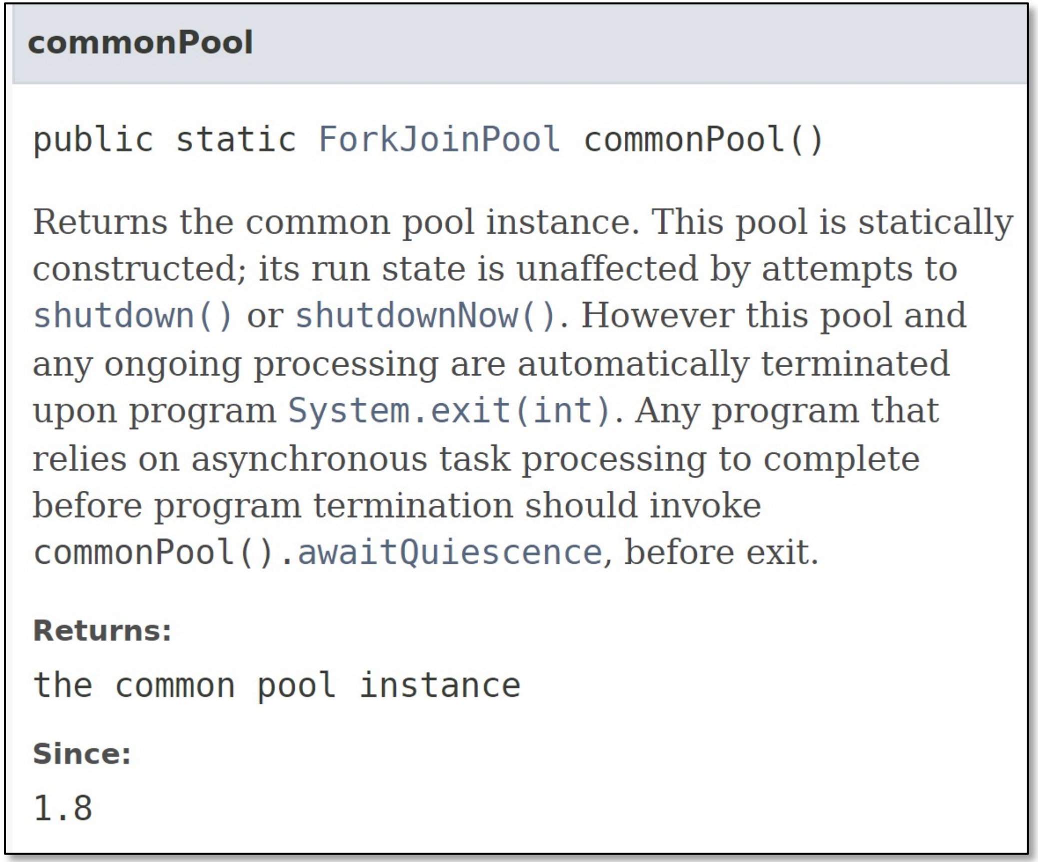
Common Fork-Join Pool



Overview of the Common Fork-Join Pool

Overview of the Common Fork-Join Pool

- A static common pool is available & appropriate for most programs



commonPool

```
public static ForkJoinPool commonPool()
```

Returns the common pool instance. This pool is statically constructed; its run state is unaffected by attempts to `shutdown()` or `shutdownNow()`. However this pool and any ongoing processing are automatically terminated upon program `System.exit(int)`. Any program that relies on asynchronous task processing to complete before program termination should invoke `commonPool().awaitQuiescence`, before exit.

Returns:

the common pool instance

Since:

1.8

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

Overview of the Common Fork-Join Pool

- A static common pool is available & appropriate for most programs
  - This pool's used by any ForkJoin Task that's not submitted to a specified pool within a process

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

Overview of the Common Fork-Join Pool

- A static common pool is available & appropriate for most programs
  - This pool's used by any ForkJoin Task that's not submitted to a specified pool within a process
  - It helps optimize resource utilization since it's aware of which cores are used globally within a process

Overview of the Common Fork-Join Pool

- A static common pool is available & appropriate for most programs
  - This pool's used by any ForkJoin Task that's not submitted to a specified pool within a process
  - It helps optimize resource utilization since it's aware of which cores are used globally within a process.
    - Goal is to maximize processor core utilization via work-stealing

See earlier lessons on "*The Java Fork-Join Pool Internals: Work Stealing*"

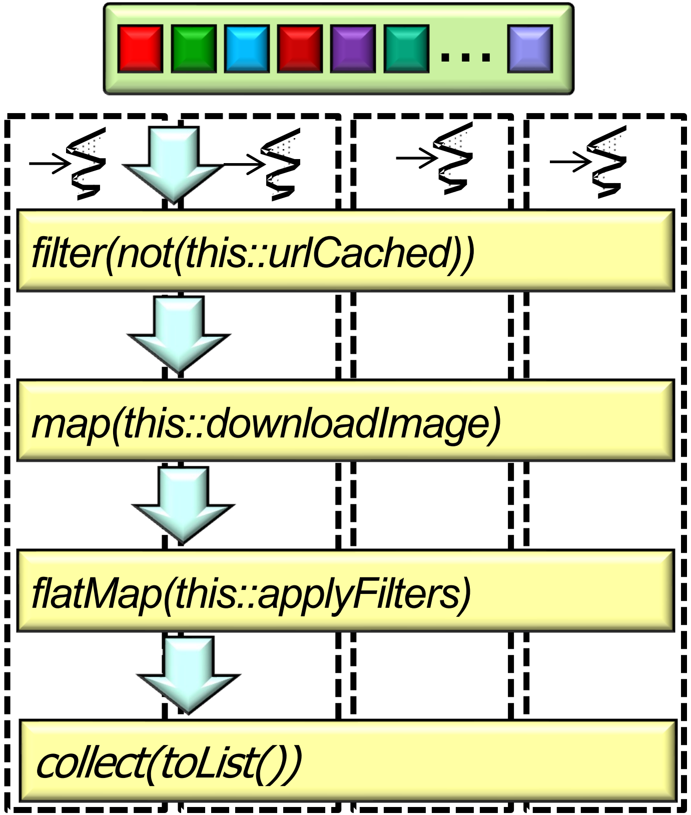
Overview of the Common Fork-Join Pool

- A static common pool is available & appropriate for most programs
  - This pool's used by any ForkJoin Task that's not submitted to a specified pool within a process
  - It helps optimize resource utilization since it's aware of which cores are used globally within a process.
    - Goal is to maximize processor core utilization via work-stealing
    - 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

Overview of the Common Fork-Join Pool

- A static common pool is available & appropriate for most programs
  - This pool's used by any ForkJoin Task that's not submitted to a specified pool within a process
  - It helps optimize resource utilization since it's aware of which cores are used globally within a process.
  - This pool is also used by the Java parallel streams framework



See dzone.com/articles/common-fork-join-pool-and-streams

Overview of the Common Fork-Join Pool

- A static common pool is available & appropriate for most programs
  - This pool's used by any ForkJoin Task that's not submitted to a specified pool within a process
  - It helps optimize resource utilization since it's aware of which cores are used globally within a process.
  - This pool is also used by the Java parallel streams framework
    - & the Java completable futures framework

```
8  /page\ = Task 1
      supplyAsync
        (getStartPage())

1  Task 2
/imgNum\ = /page\ 8
  .thenApplyAsync
    (countImages(page))
  .thenApply(List::size)

26  Task 3
/imgNum\ = /page\ 8
  .thenComposeAsync
    (crawlHyperLinks
      (page))

1  Task 4  26
/imgNum1\.thenCombine(/imgNum2\,
    (imgNum1, imgNum2) ->
     Integer::sum)
```

See dzone.com/articles/common-fork-join-pool-and-streams

Overview of the Common Fork-Join Pool

- By default the common fork-join pool has one less thread than the # of cores

```
ForkJoinPool makeCommonPool() {
  ...
  parallelism = Runtime
    .getRuntime()
    .availableProcessors() - 1;
  ...
```

Sets ‘parallelism’ to three on a quad-core processor

See docs.oracle.com/javase/8/docs/api/java/lang/Runtime.html#availableProcessors

Overview of the Common Fork-Join Pool

- By default the common fork-join pool has one less thread than the # of cores

```
ForkJoinPool makeCommonPool() {
  ...
  parallelism = Runtime
    .getRuntime()
    .availableProcessors() - 1;
  ...
```

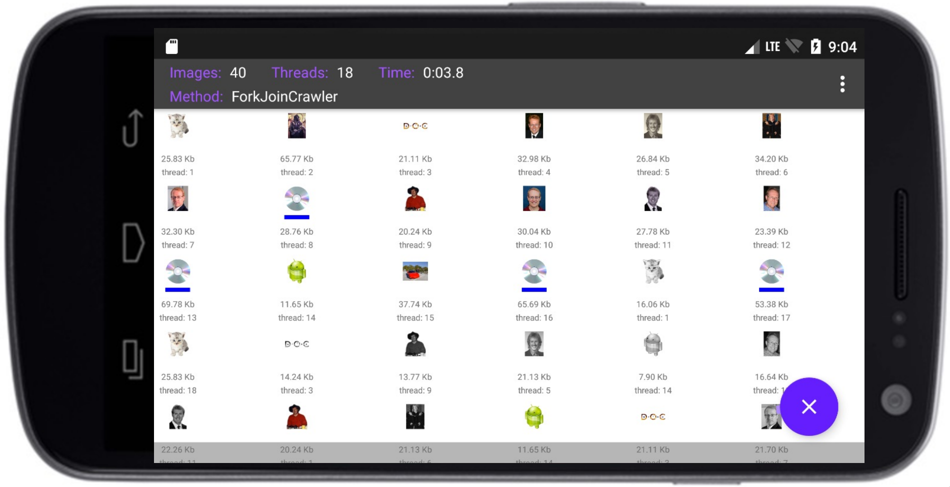
Returns three on a quad-core processor

```
System.out.println
  ("The parallelism in the"
   + "common fork-join pool is "
   + ForkJoinPool
       .getCommonPoolParallelism());
```

See github.com/douglascraigschmidt/LiveLessons/blob/master/SearchForkJoin

Overview of the Common Fork-Join Pool

- By default the common fork-join pool has one less thread than the # of cores



Images: 40 Threads: 18 Time: 0:03.8

Method: ForkJoinCrawler

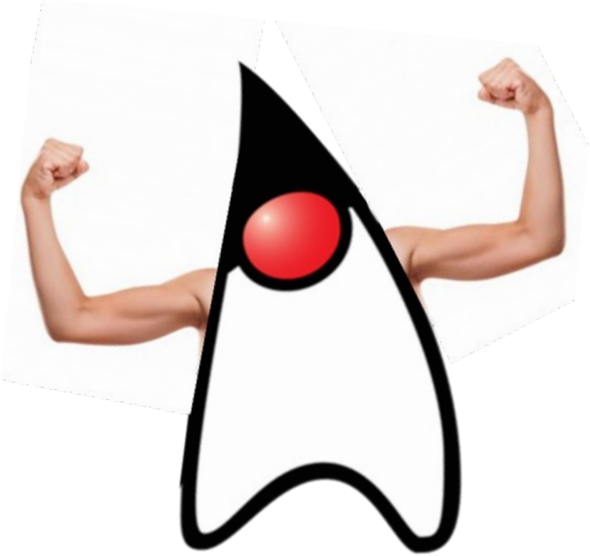
```
25.83 Kb    65.77 Kb    21.11 Kb    32.98 Kb    26.84 Kb    34.20 Kb
thread: 1   thread: 2   thread: 3   thread: 4   thread: 5   thread: 6
32.30 Kb    28.76 Kb    20.24 Kb    30.04 Kb    27.78 Kb    23.39 Kb
thread: 7   thread: 8   thread: 9   thread: 10  thread: 11  thread: 12
69.78 Kb    11.65 Kb    37.74 Kb    65.69 Kb    16.06 Kb    53.38 Kb
thread: 13  thread: 14  thread: 15  thread: 16  thread: 1   thread: 17
25.83 Kb    14.24 Kb    13.77 Kb    21.13 Kb    7.90 Kb     16.64 Kb
thread: 18  thread: 3   thread: 9   thread: 5   thread: 14  thread: 1
22.26 Kb    20.24 Kb    21.13 Kb    11.65 Kb    21.11 Kb    21.70 Kb
```

The invoking thread, e.g., the main (UI) thread, is also included in the pool

A program can therefore leverage all processor cores!

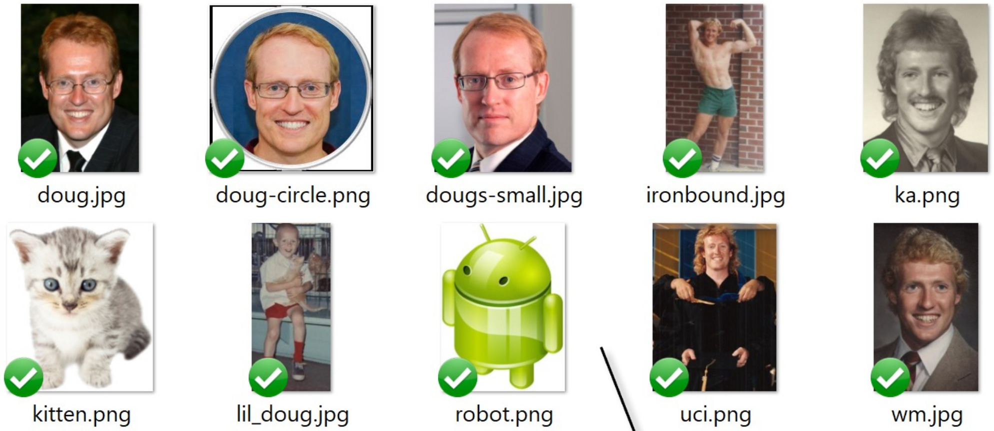
Overview of the Common Fork-Join Pool

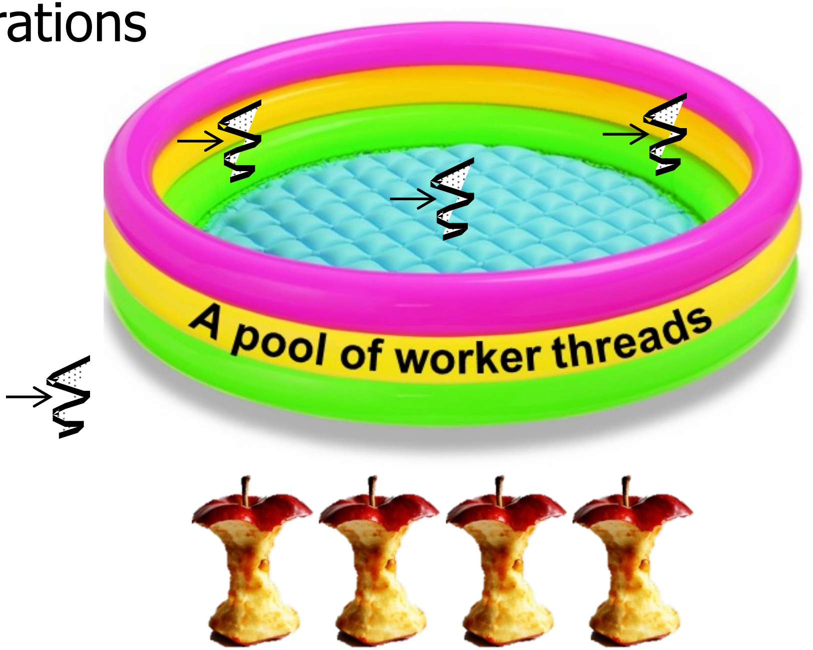
- However, the default # of threads in the fork-join pool may be inadequate



Overview of the Common Fork-Join Pool

- However, the default # of threads in the fork-join pool may be inadequate
  - e.g., problems occur when blocking operations are used in the common fork-join pool



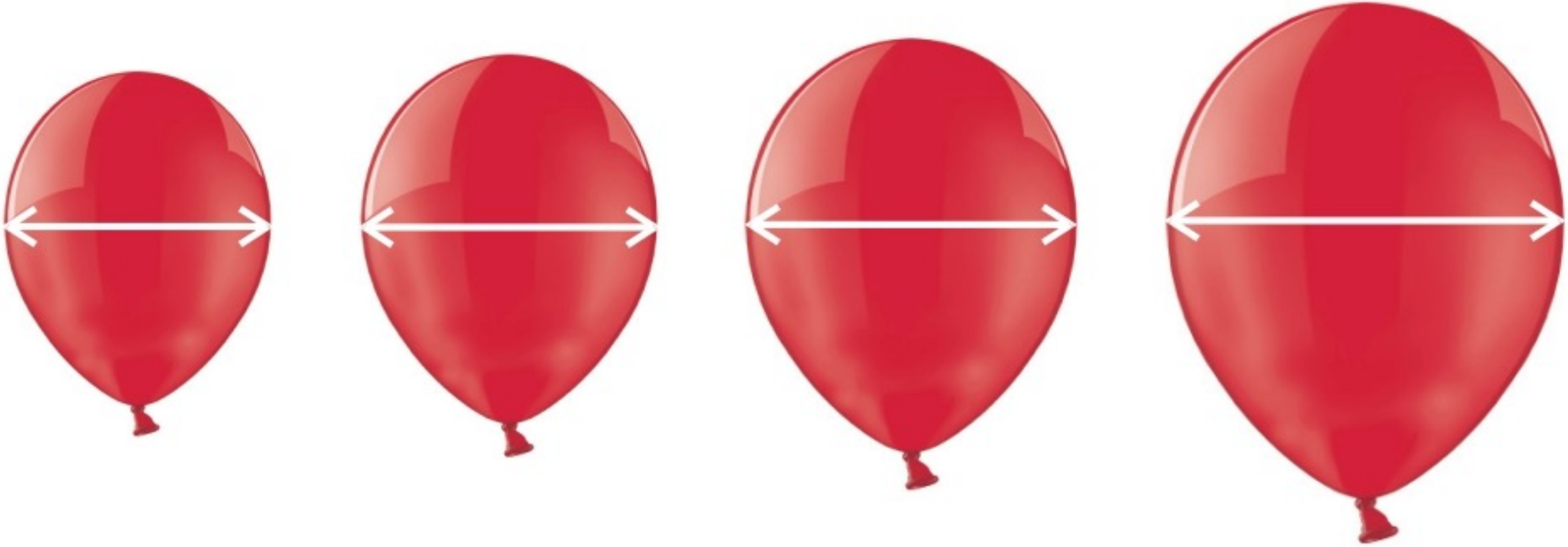


e.g., downloading more images than # of cores

These problems may range from underutilization of processor cores to deadlock..

Overview of the Common Fork-Join Pool

- The common pool size can thus be expanded & contracted programmatically



Overview of the Common Fork-Join Pool

- The common pool size can thus be expanded & contracted programmatically
  - By modifying a system property

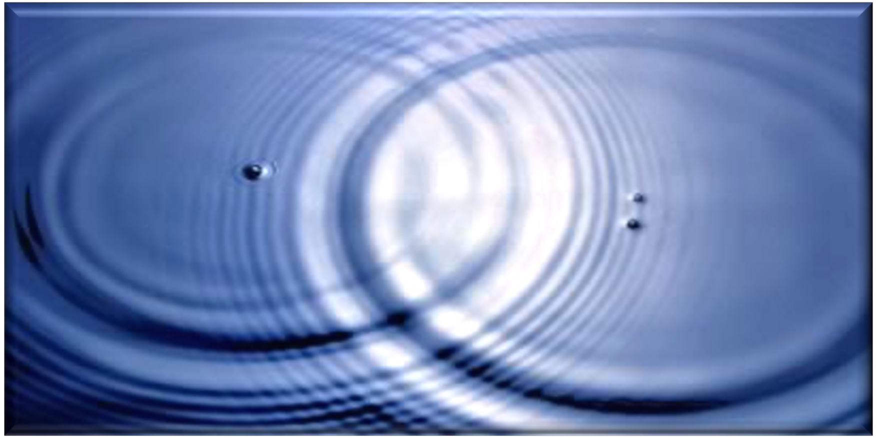
```
String desiredThreads = "10";
System.setProperty
   ("java.util.concurrent." +
    "ForkJoinPool.common." +
    "parallelism",
    desiredThreads);
```

It's hard to estimate the total # of threads to set in the common fork-join pool

Overview of the Common Fork-Join Pool

- The common pool size can thus be expanded & contracted programmatically
  - By modifying a system property
    - Modifying this property affects all common fork-join usage in a process!

```
String desiredThreads = "10";
System.setProperty
   ("java.util.concurrent." +
    "ForkJoinPool.common." +
    "parallelism",
    desiredThreads);
```



Overview of the Common Fork-Join Pool

- The common pool size can thus be expanded & contracted programmatically
  - By modifying a system property
    - Modifying this property affects all common fork-join usage in a process!
    - This property can be changed only before the common fork-join pool is initialized
      - It's initialized "on-demand" the first time it's used

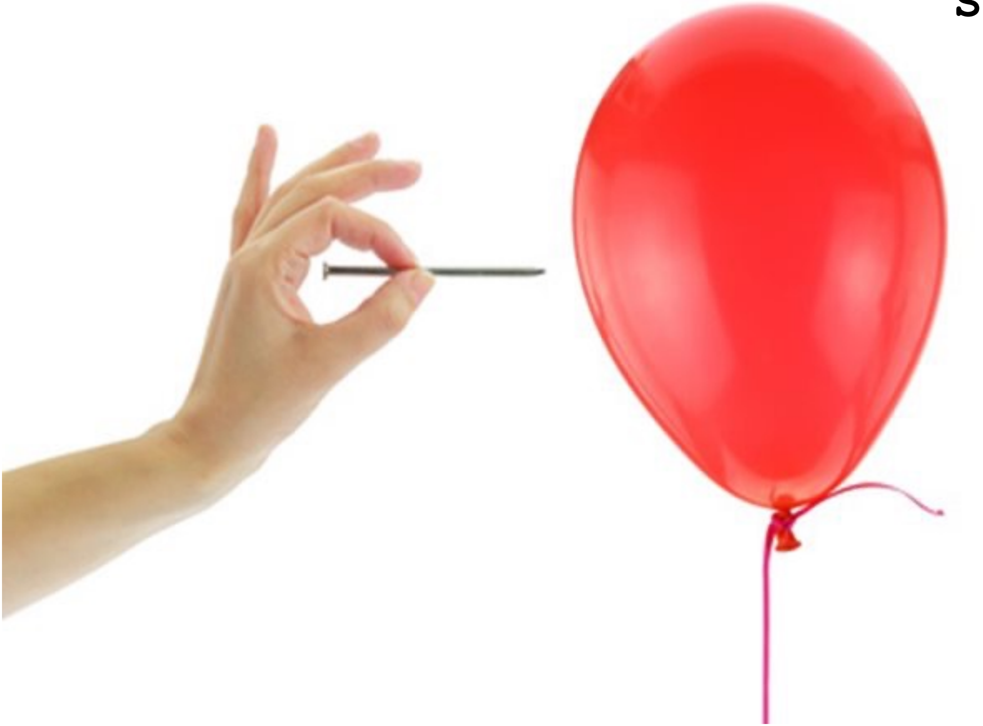
```
String desiredThreads = "10";
System.setProperty
   ("java.util.concurrent." +
    "ForkJoinPool.common." +
    "parallelism",
    desiredThreads);
```

See en.wikipedia.org/wiki/Lazy_initialization

Overview of the Common Fork-Join Pool

- The common pool size can thus be expanded & contracted programmatically
  - By modifying a system property

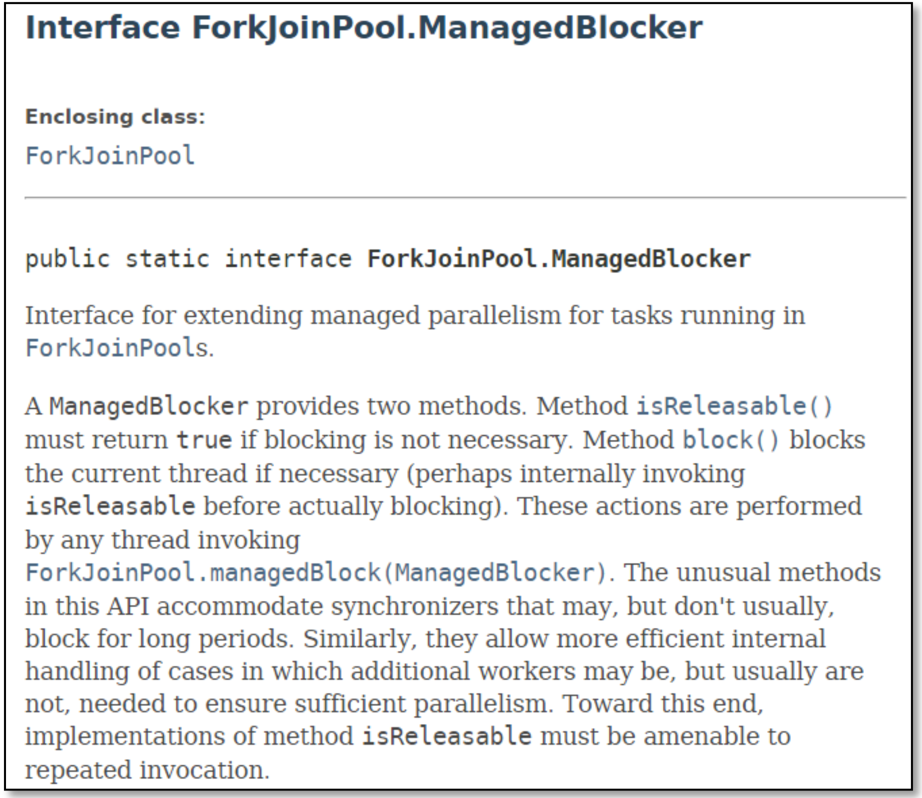
```
String desiredThreads = "10";
System.setProperty
   ("java.util.concurrent." +
    "ForkJoinPool.common." +
    "parallelism",
    desiredThreads);
```



Another approach is thus needed to increase the fork/join pool size automatically

Overview of the Common Fork-Join Pool

- The common pool size can thus be expanded & contracted programmatically
  - By modifying a system property
  - By using a ManagedBlocker



Interface ForkJoinPool.ManagedBlocker

Enclosing class:

ForkJoinPool

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

Interface for extending managed parallelism for tasks running in ForkJoinPools.

A ManagedBlocker provides two methods. Method isReleasable() must return true if blocking is not necessary. Method block() blocks the current thread if necessary (perhaps internally invoking isReleasable before actually blocking). These actions are performed by any thread invoking ForkJoinPool.managedBlock(ManagedBlocker). The unusual methods in this API accommodate synchronizers that may, but don't usually, block for long periods. Similarly, they allow more efficient internal handling of cases in which additional workers may be, but usually are not, needed to ensure sufficient parallelism. Toward this end, implementations of method isReleasable must be amenable to repeated invocation.

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

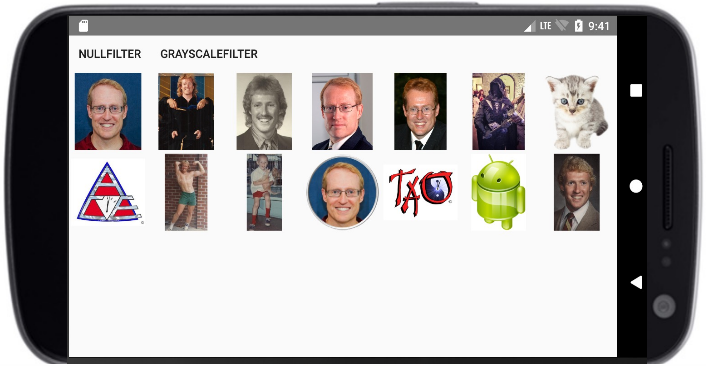
Overview of the Common Fork-Join Pool

- The common pool size can thus be expanded & contracted programmatically
  - By modifying a system property
  - By using a ManagedBlocker
    - Temporarily add worker threads to the common fork-join pool

A pool of worker threads

Overview of the Common Fork-Join Pool

- The common pool size can thus be expanded & contracted programmatically
  - By modifying a system property
  - By using a ManagedBlocker
    - Temporarily add worker threads to the common fork-join pool
    - Useful when tasks wait on I/O, synchronizers, or blocking queues



A pool of worker threads

ManageBlockers can only be used with the common fork-join pool..

Overview of the Common Fork-Join Pool

- The common pool size can thus be expanded & contracted programmatically
  - By modifying a system property
  - By using a ManagedBlocker
    - Temporarily add worker threads to the common fork-join pool
    - Useful when tasks wait on I/O, synchronizers, or blocking queues
    - It’s helpful to encapsulate the ManagedBlocker mechanism

```
SupplierManagedBlocker<T> mb =
  new SupplierManagedBlocker<>
   (supplier);
...
ForkJoinPool.managedBlock(mb);
return mb.getResult();
```

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

Overview of the Common Fork-Join Pool

- The common pool size can thus be expanded & contracted programmatically
  - By modifying a system property
  - By using a ManagedBlocker
    - Temporarily add worker threads to the common fork-join pool
    - Useful when tasks wait on I/O, synchronizers, or blocking queues
    - It’s helpful to encapsulate the ManagedBlocker mechanism
    - The common ForkJoinPool reclaims threads during periods of non-use & reinstates them on later use

End of Maximizing Processor Core Utilization with the Java Common Fork-Join Pool