The Java Fork-Join Pool: Overview of the Common Fork-Join Pool

Douglas C. Schmidt

d.schmidt@vanderbilt.edu

www.dre.vanderbilt.edu/~schmidt

Professor of Computer Science

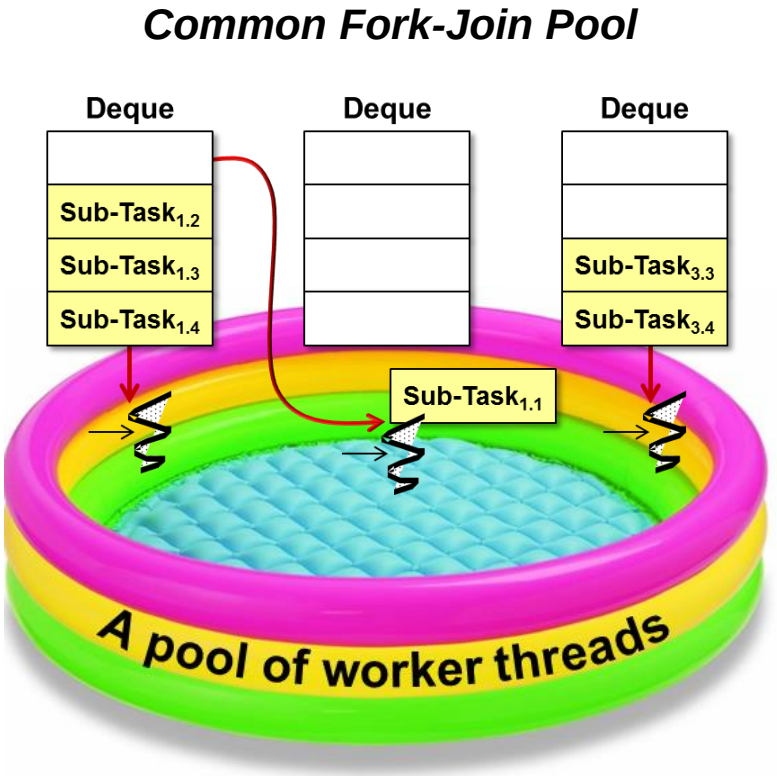
Institute for Software Integrated Systems

Vanderbilt University

Nashville, Tennessee, USA

Learning Objectives in this Part of the Lesson

- Understand the 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 what cores are being used globally within a process

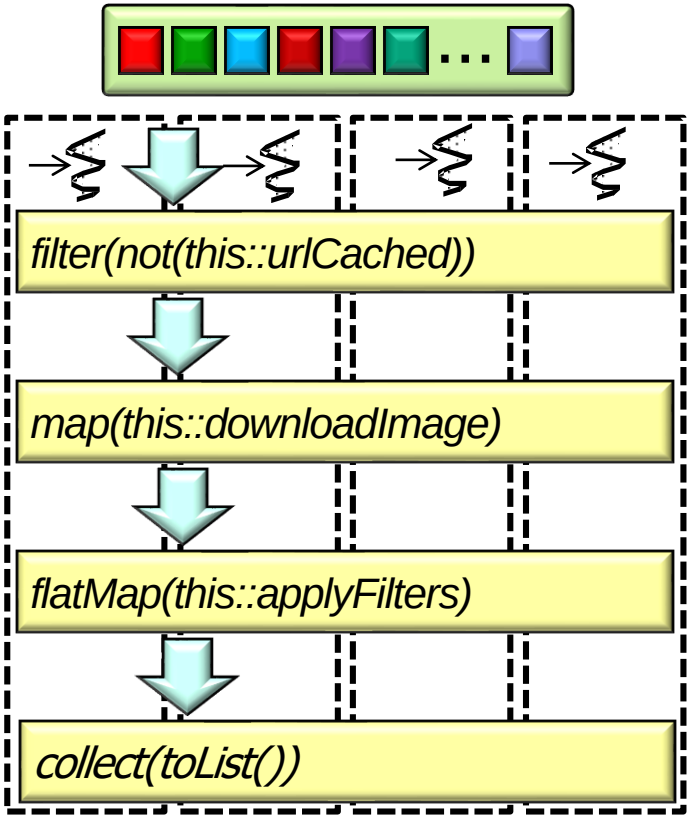
Overview of the Common Fork-Join Pool

- 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 what cores are being used globally within a process
    - 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 what cores are being 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

- By default the common ForkJoinPool has one less thread than the # of cores

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

e.g., returns 4 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 ForkJoinPool has one less thread than the # of cores

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

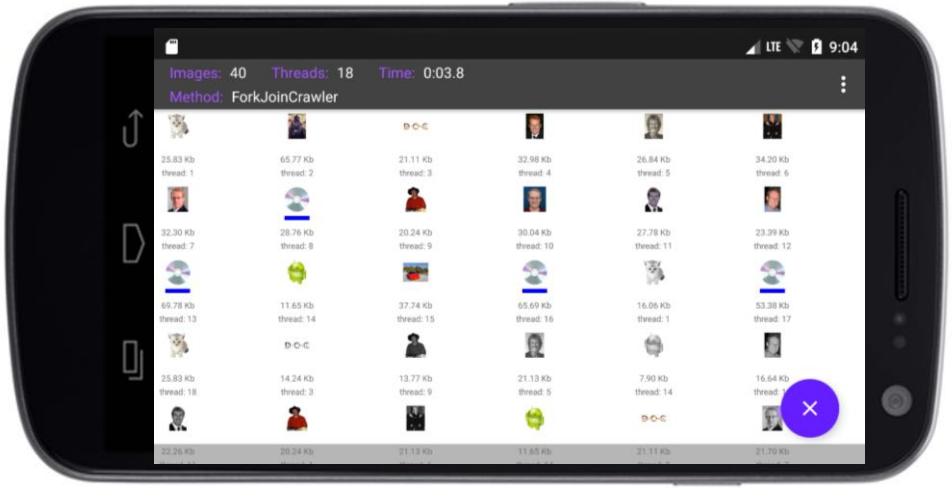
e.g., returns 3 on a quad-core processor

```
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 ForkJoinPool has one less thread than the # of cores

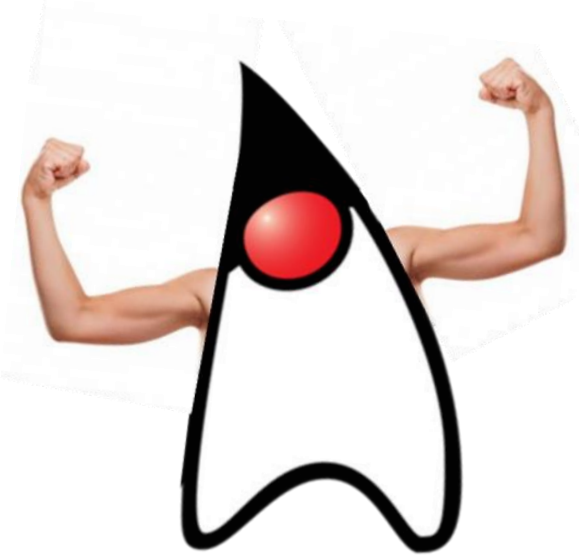


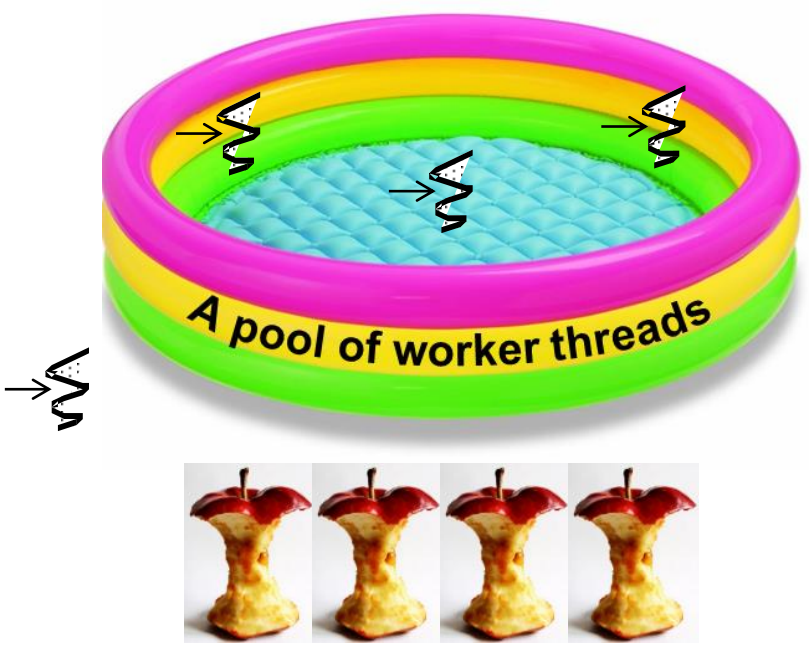
Invoking thread

A program can leverage all cores since it uses the invoking thread, e.g., main thread

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

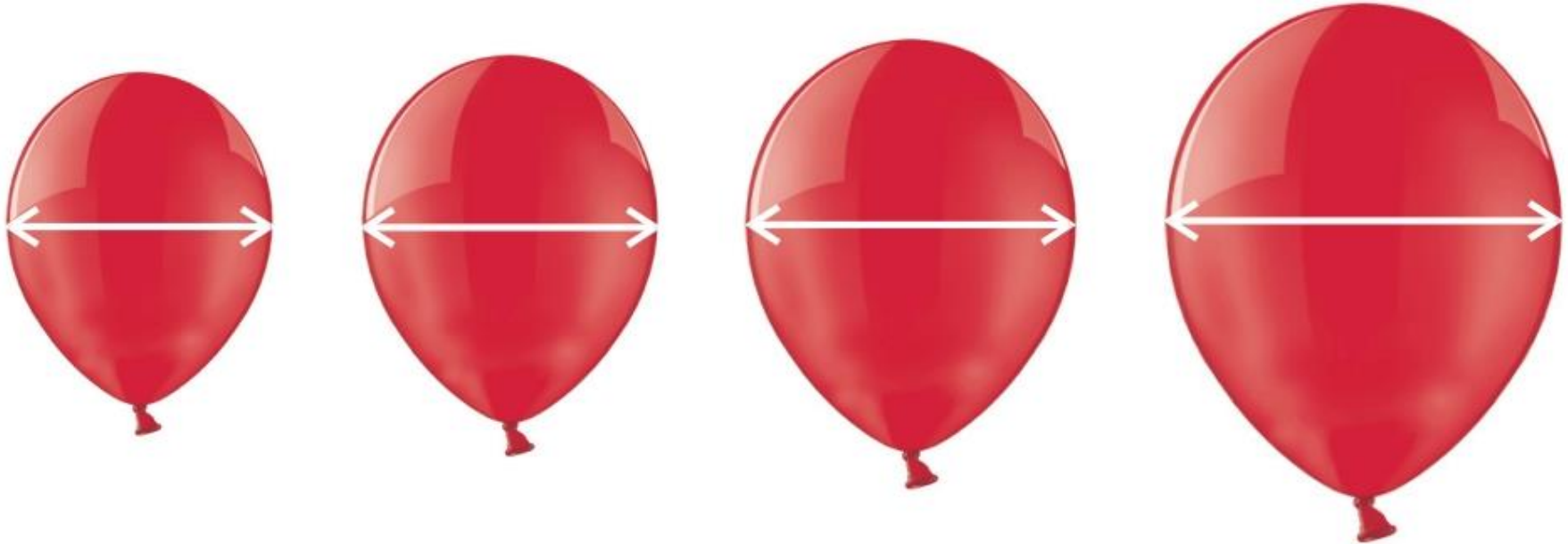
- However, the default # of threads in the fork-join pool may be inadequate
  - e.g., problems occur when blocking operations are used in the common fork-join pool

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

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

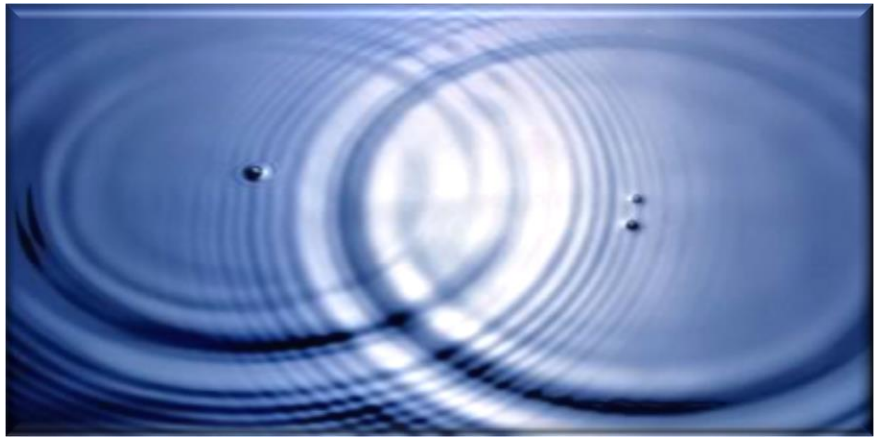
It's hard to estimate the total # of threads to set in the common fork-join pool

Overview of the Common Fork-Join Pool

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

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

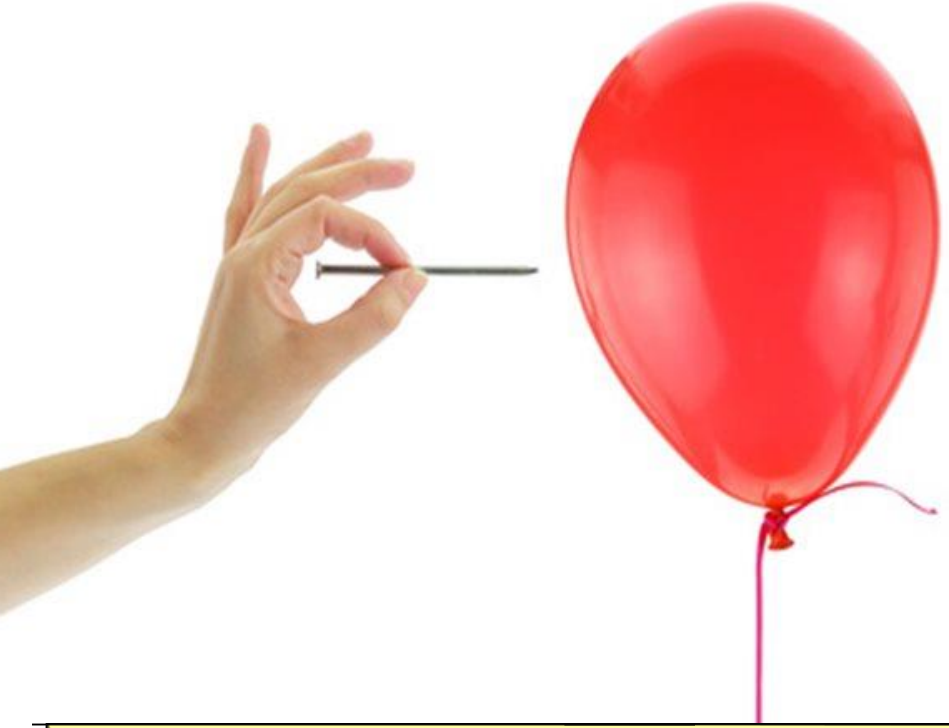
Modifying this property affects all common fork-join usage in a process!



Overview of the Common Fork-Join Pool

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

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



It's thus necessary to be able to automatically increasing fork/join pool size

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.

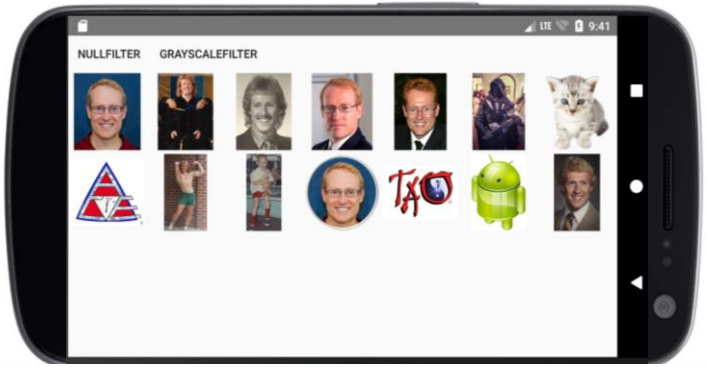
See docs.oracle.com/javase/8/docs/api/java/util/concurrent/ForkJoinPool.ManagedBlocker.html

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

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 block on I/O and/or synchronizers



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

Overview of the Common Fork-Join Pool

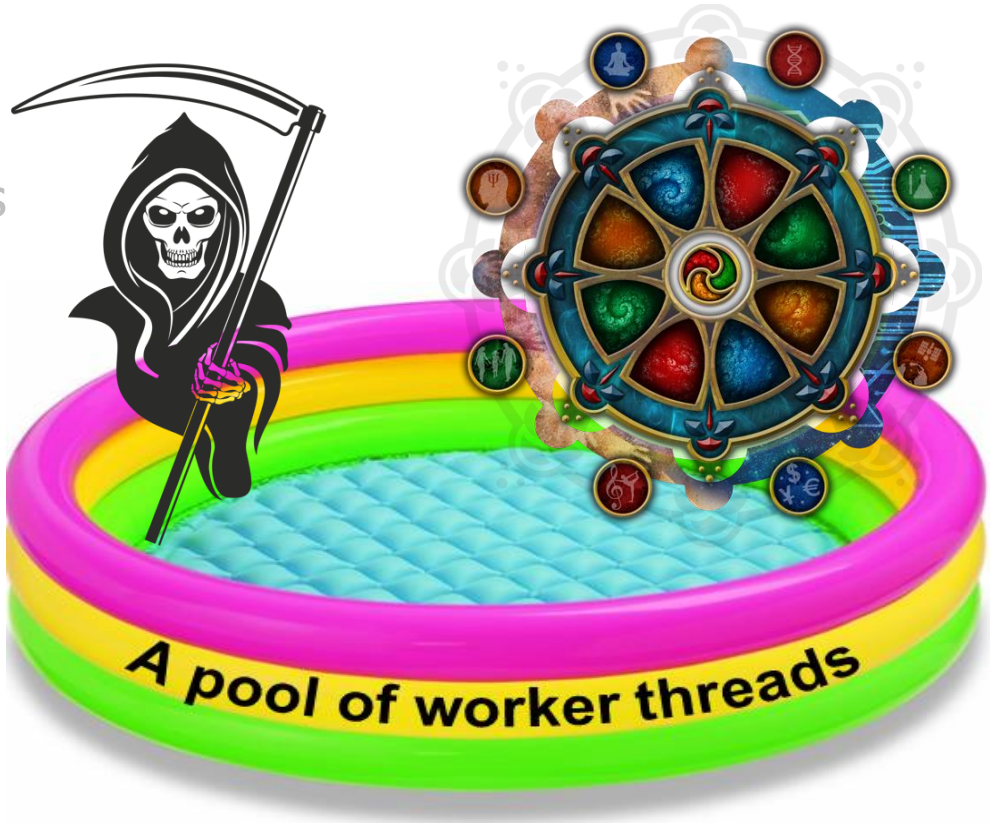
- The common pool size can thus be expanded & contracted programmatically
  - By modifying a system property
  - By using a ManagedBlocker
    - Temporarily add worker threads to the common fork-join pool
    - Useful when tasks block on I/O and/or synchronizers

```
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 block on I/O and/or synchronizers
    - ForkJoinPool reclaims threads during periods of non-use & reinstates them on later use



End of the Java Fork-Join Pool Framework: Common Fork-Join Pool