The Java Fork-Join Pool: the ManagedBlocker Interface

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
- Recognize how the ManagedBlocker interface helps avoid starvation & improve performance

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 ManagedBlocker Interface

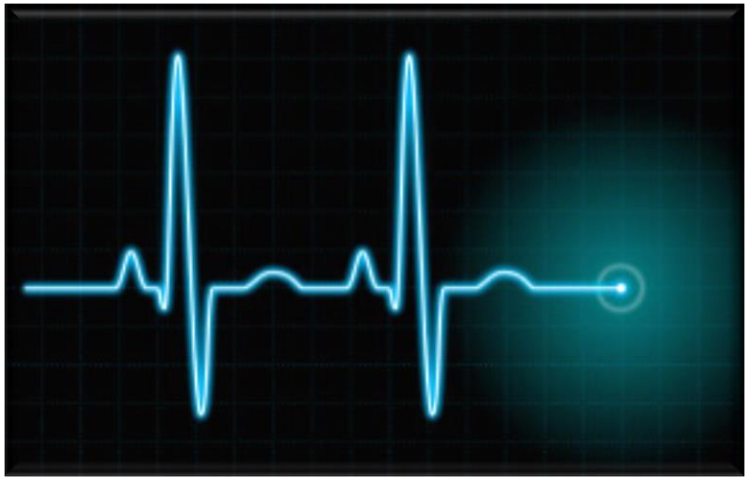
Overview of the ManagedBlocker Interface

- The Java fork-join framework is largely designed for tasks that "run to completion" without blocking

See en.wikipedia.org/wiki/Run_to_completion_scheduling

Overview of the ManagedBlocker Interface

- ManagedBlocker handles cases where more worker threads may be needed to ensure liveness/responsiveness for blocking operations



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 ManagedBlocker Interface

- ManagedBlocker handles cases where more worker threads may be needed to ensure liveness/responsiveness for blocking operations
  - e.g., to automatically/temporarily increase common fork/join pool size

Interface ForkJoinPool.ManagedBlocker

Enclosing class:

`ForkJoinPool`

`public static interface ForkJoinPool.ManagedBlocker`

Interface for extending managed parallelism for tasks running in `ForkJoinPools`.

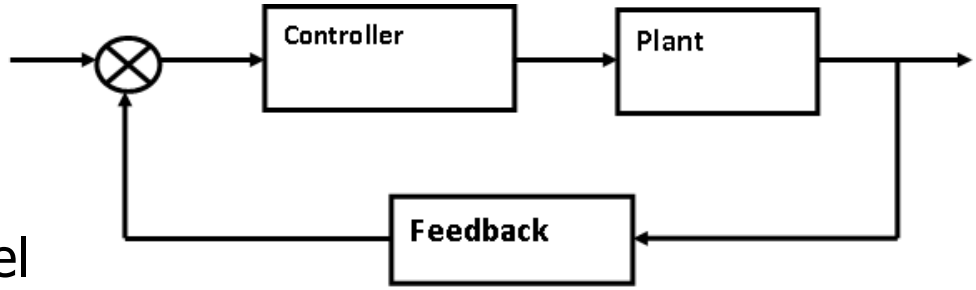
A `ManagedBlocker` provides two methods. Method `isReleasable()` must return `true` if blocking is not necessary. Method `block()` blocks the current thread if necessary (perhaps internally invoking `isReleasable` before actually blocking). These actions are performed by any thread invoking `ForkJoinPool.managedBlock(ManagedBlocker)`. 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.

Overview of the ManagedBlocker Interface

- ForkJoinPool reclaims threads during periods of non-use & reinstates them on later use

Overview of the ManagedBlocker Interface

- ForkJoinPool reclaims threads during periods of non-use & reinstates them on later use
  - It also tries to create or activate threads to ensure the target level of parallelism is met



Overview of the ManagedBlocker Interface

- ManagedBlocker defines two methods


```
interface ManagedBlocker {
  boolean isReleasable();

  boolean block();
}
```

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

Overview of the ManagedBlocker Interface

- ManagedBlocker defines two methods
  - Returns true if blocking is unnecessary

```
interface ManagedBlocker {
  boolean isReleasable();

  boolean block();
}
```

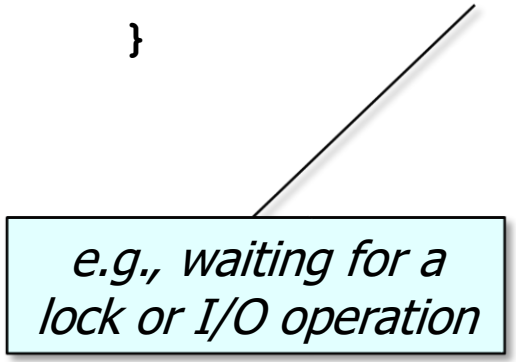
e.g., was able to acquire a lock or a message without blocking

Overview of the ManagedBlocker Interface

- ManagedBlocker defines two methods
  - Returns true if blocking is unnecessary
  - Possibly blocks the calling thread

```
interface ManagedBlocker {
  boolean isReleasable();

  boolean block();
}
```

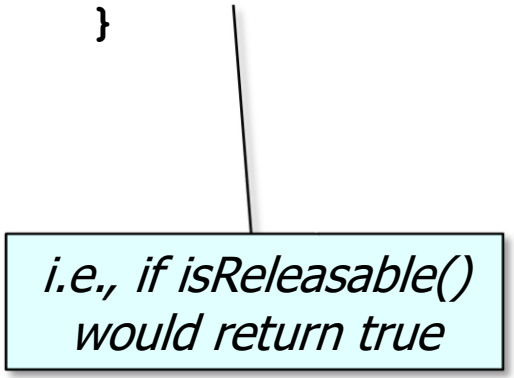


Overview of the ManagedBlocker Interface

- ManagedBlocker defines two methods
  - Returns true if blocking is unnecessary
  - Possibly blocks the calling thread
    - Returns true if no additional blocking is necessary

```
interface ManagedBlocker {
  boolean isReleasable();

  boolean block();
}
```



How the Java Fork-Join Pool Applies ManagedBlocker

How the Java Fork-Join Pool Applies ManagedBlocker

- The ForkJoinPool class uses a ManagedBlocker internally

```
class ForkJoinPool extends AbstractExecutorService {
  ...
  static void managedBlock(ManagedBlocker blocker) {
    ...
    while (!blocker.isReleasable()) {
      if (p.tryCompensate(p.ctl)) {
        ...
        do {}
        while (!blocker.isReleasable()
               && !blocker.block());
        ...
}
...
```

See openjdk/7-b147/java/util/concurrent/ForkJoinPool.java

How the Java Fork-Join Pool Applies ManagedBlocker

- The ForkJoinPool class uses a ManagedBlocker internally

```
class ForkJoinPool extends AbstractExecutorService {
  ...
  static void managedBlock(ManagedBlocker blocker) {
    ...
    while (!blocker.isReleasable()) {
      if (p.tryCompensate(p.ctl)) {
        ...
        do {}
        while (!blocker.isReleasable()
               && !blocker.block());
        ...
}
...
```

This method activates a spare thread to ensure sufficient parallelism while calling thread is blocked

See openjdk/7-b147/java/util/concurrent/ForkJoinPool.java

How the Java Fork-Join Pool Applies ManagedBlocker

- The ForkJoinPool class uses a ManagedBlocker internally

```
class ForkJoinPool extends AbstractExecutorService {
  ...
  static void managedBlock(ManagedBlocker blocker) {
    ...
    while (!blocker.isReleasable()) {
      if (p.tryCompensate(p.ctl)) {
        ...
        do {}
        while (!blocker.isReleasable()
               && !blocker.block());
        ...
  }
  ...
```

If there aren't enough live threads, create or re-activate a spare thread to compensate for blocked joiners 'til they unblock

How the Java Fork-Join Pool Applies ManagedBlocker

- The ForkJoinPool class uses a ManagedBlocker internally

```
class ForkJoinPool extends AbstractExecutorService {
  ...
  static void managedBlock(ManagedBlocker blocker) {
    ...
    while (!blocker.isReleasable()) {
      if (p.tryCompensate(p.ctl)) {
        ...
        do {}
        while (!blocker.isReleasable()
               && !blocker.block());
        ...
}
...
```

Potentially block the calling thread

End of the Java Fork-Join Pool: the Managed Blocker Interface