

Overview of the Java Fork-Join Framework's 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 how the common fork-join pool helps to maximize processor core utilization
- 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.

Learning Objectives in this Part of the Lesson

- Understand how the common fork-join pool helps to maximize processor core utilization
- Recognize how the ManagedBlocker interface helps avoid starvation & improve performance
 - This mechanism isn't limited to the Java common fork-join pool



Overview of the Managed Blocker Mechanism

Overview of the ManagedBlocker Mechanism

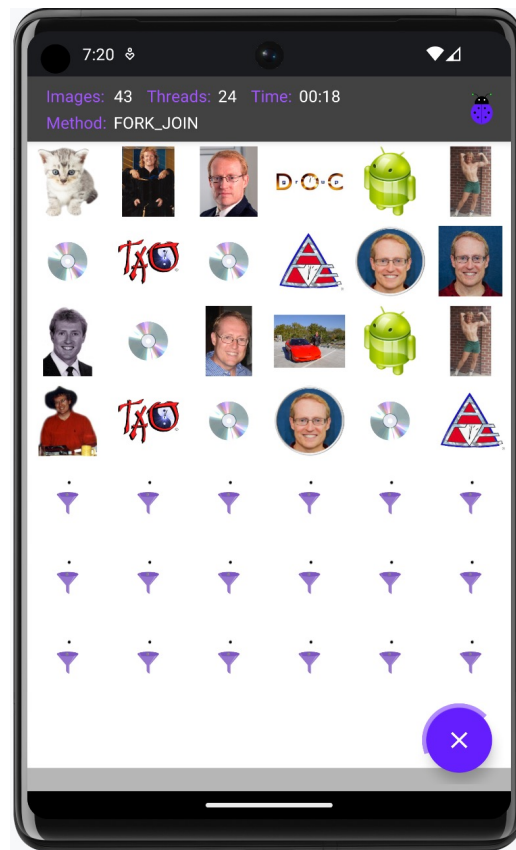
- The Java fork-join framework is largely designed for tasks that “run to completion” without blocking



See en.wikipedia.org/wiki/Run_to_completion_scheduling

Overview of the ManagedBlocker Mechanism

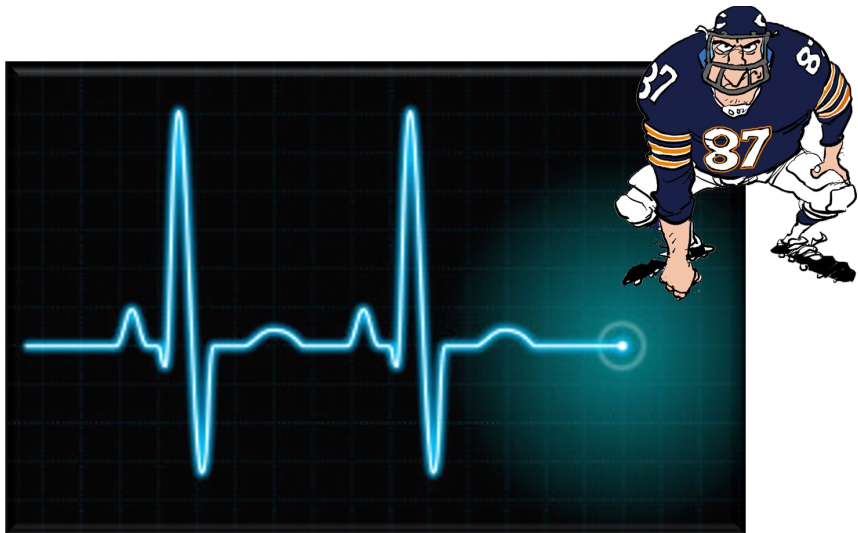
- The Java fork-join framework is largely designed for tasks that “run to completion” without blocking
 - However, many apps perform blocking operations
 - e.g., for I/O, synchronizers, bounded-buffer queues, etc.



See www.geeksforgeeks.org/blocking-methods-in-java

Overview of the ManagedBlocker Mechanism

- The ManagedBlocker mechanism is designed to handle cases where more worker threads may be needed to ensure liveness/responsiveness for blocking operations in a ForkJoinPool



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 Mechanism

- The ManagedBlocker mechanism is designed to handle cases where more worker threads may be needed to ensure liveness/responsiveness for blocking operations in a ForkJoinPool
- 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 Mechanism

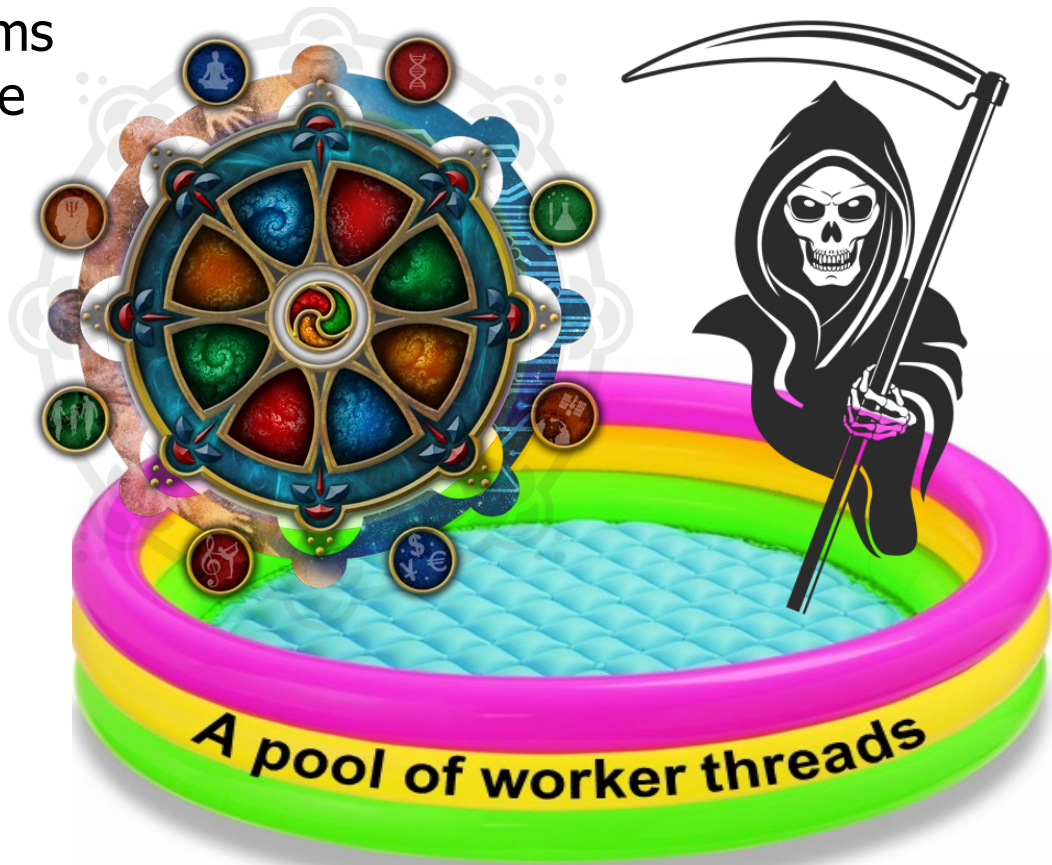
- The ManagedBlocker mechanism can be used with any instance of Java fork-join pool

```
public static void managedBlock
(ManagedBlocker blocker) ... {
    Thread t; ForkJoinPool p;
    if ((t = Thread.currentThread())
        instanceof
        ForkJoinWorkerThread &&
        (p = ((ForkJoinWorkerThread) t)
            .pool) != null)
        p.compensatedBlock
        (blocker);
    else
        unmanagedBlock (blocker);
}
```

This method checks if the current thread is a ForkJoinWorkerThread & if so, it blocks on that thread's associated ForkJoinPool instance

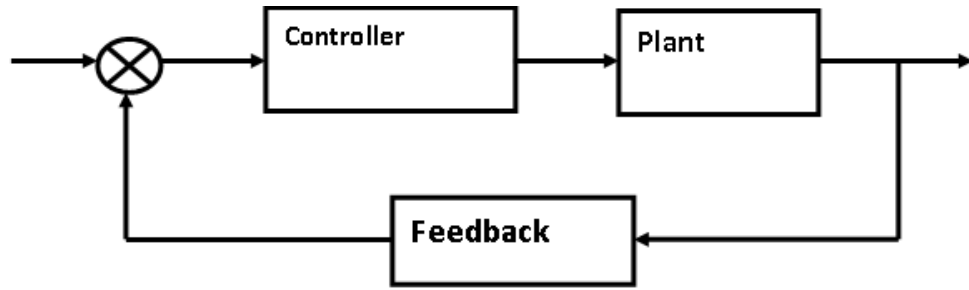
Overview of the ManagedBlocker Mechanism

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



Overview of the ManagedBlocker Mechanism

- The common 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

Overview of the ManagedBlocker Interface

- ManagedBlocker defines two methods



```
interface ManagedBlocker {  
    boolean isReleasable();  
  
    boolean block();  
}
```

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();  
}
```



e.g., waiting for a lock or I/O operation

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();  
}
```



*i.e., if isReleasable()
would return true*

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(pctl)) {
                ...
                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(pctl)) {
                ...
                do {}
                while (!blocker.isReleasable()
                    && !blocker.block());
            }
            ...
        }
        ...
    }
    ...
}
```

Implements the ExecutorService interface

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.cntl)) {  
                ...  
                do {}  
                while (!blocker.isReleasable()  
                    && !blocker.block());  
                ...  
            }  
        }  
        ...  
    }  
    ...  
}
```

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

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.cnt1)) {  
                ...  
                do {}  
                while (!blocker.isReleasable()  
                    && !blocker.block());  
                ...  
            }  
            ...  
        }  
        ...  
    }  
    ...  
}
```

*Interface for extending
managed parallelism for tasks
running in ForkJoinPools*

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.ct1)) {
                ...
                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.cntl)) {  
                ...  
                do {}  
                while (!blocker.isReleasable()  
                    && !blocker.block());  
                ...  
            }  
            ...  
        }  
        ...  
    }  
    ...  
}
```

First attempt to acquire the resource without blocking

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.ctrl)) {
                ...
                do {}
                while (!blocker.isReleasable()
                    && !blocker.block());
                ...
            }
        }
        ...
    }
    ...
}
```

May block the calling thread

End of Overview of the Java Fork-Join Framework's ManagedBlocker Interface