Java 8 Parallel ImageStreamGang Example (Part 3)

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 structure & functionality of an ImageStreamGang app
- Know how Java 8 parallel streams are applied to the ImageStreamGang app
- Recognize how the Java 8 parallel stream common fork-join pool can be configured

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

Learning Objectives in this Part of the Lesson

- Understand the structure & functionality of an ImageStreamGang app
- Know how Java 8 parallel streams are applied to the ImageStreamGang app
- Recognize how the Java 8 parallel stream common fork-join pool can be configured
- Learn how a ManagedBlocker can avoid thread pool starvation and/or improve performance in ImageStreamGang

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

Configuring the Parallel Stream Common Fork-Join Pool

Configuring the Parallel Stream Common Fork-Join Pool

- Java 8 parallel streams are intentionally designed with a limited # of “knobs” to configure their behavior

See www.infoq.com/presentations/tecniques-parallelism-java

Configuring the Parallel Stream Common Fork-Join Pool

- In particular, the common fork-join pool optimizes resource utilization since it's aware of what cores are being used globally

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

Configuring the Parallel Stream Common Fork-Join Pool

- In particular, the common fork-join pool optimizes resource utilization since it's aware of what cores are being used globally
  - By default the common ForkJoinPool has one less thread than the # of cores

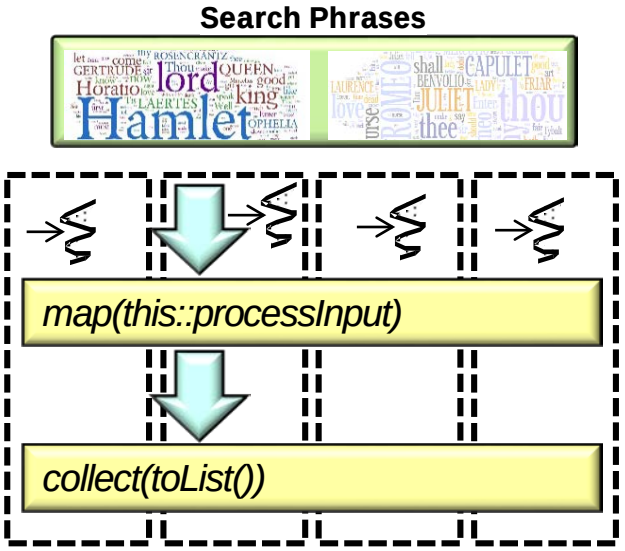
```
System.out.println
  ("The parallelism in the"
   + "common fork-join pool is "
   + ForkJoinPool
       .getCommonPoolParallelism());
```

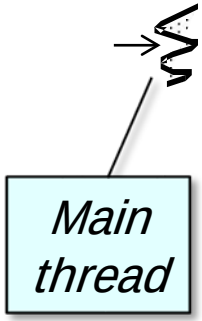
e.g., returns 7 on my quad-core hyper-threaded processor

See github.com/douglascraigschmidt/LiveLessons/tree/master/SearchStreamForkJoin/Java7

Configuring the Parallel Stream Common Fork-Join Pool

- In particular, the common fork-join pool optimizes resource utilization since it's aware of what cores are being used globally
  - By default the common ForkJoinPool has one less thread than the # of cores

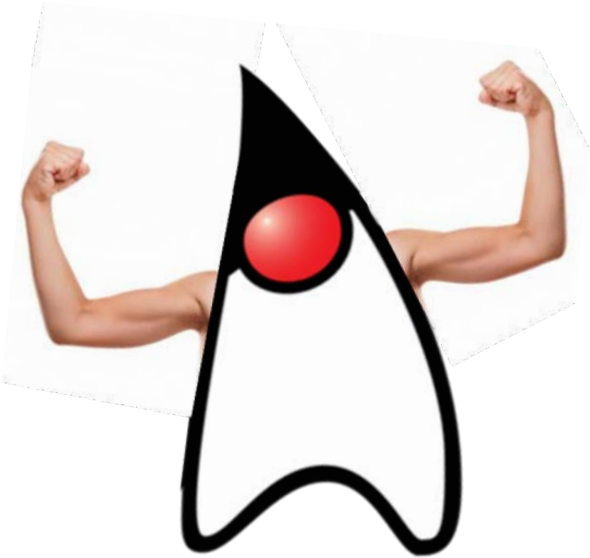


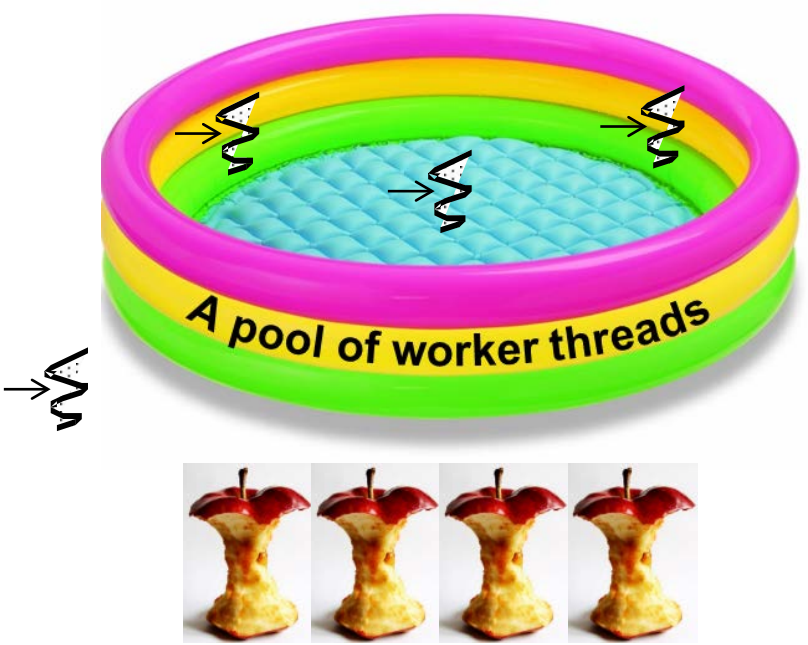


A parallel stream can thus use all cores since it also uses the main thread

Configuring the Parallel Stream Common Fork-Join Pool

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





Configuring the Parallel Stream 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 a parallel stream

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

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

Configuring the Parallel Stream Common Fork-Join Pool

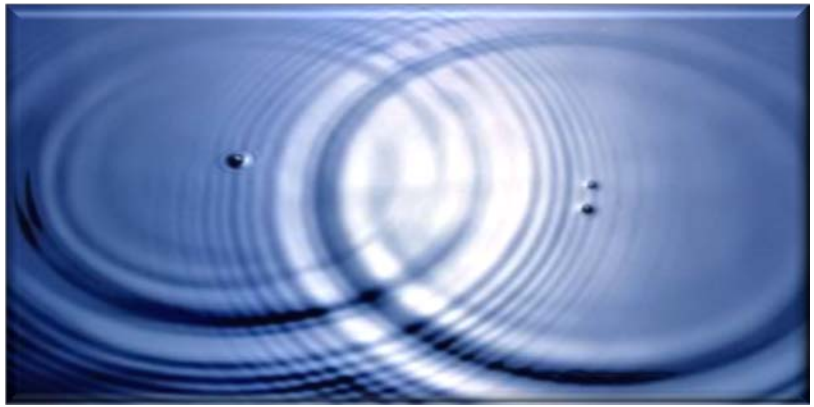
- The common pool size can be controlled programmatically

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

Configuring the Parallel Stream Common Fork-Join Pool

- The common pool size can be controlled programmatically
  - Setting this property affects all parallel streams in a process

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



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

Configuring the Parallel Stream Common Fork-Join Pool

- The common pool size can be controlled programmatically
  - Setting this property affects all parallel streams in a process
  - The ManagedBlocker class can be used to temporarily add worker threads to common fork-join pool

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

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

Configuring the Parallel Stream Common Fork-Join Pool

- The common pool size can be controlled programmatically
  - Setting this property affects all parallel streams in a process
  - The ManagedBlocker class can be used to temporarily add worker threads to common fork-join pool
    - This is useful for behaviors that block on I/O and/or synchronizers

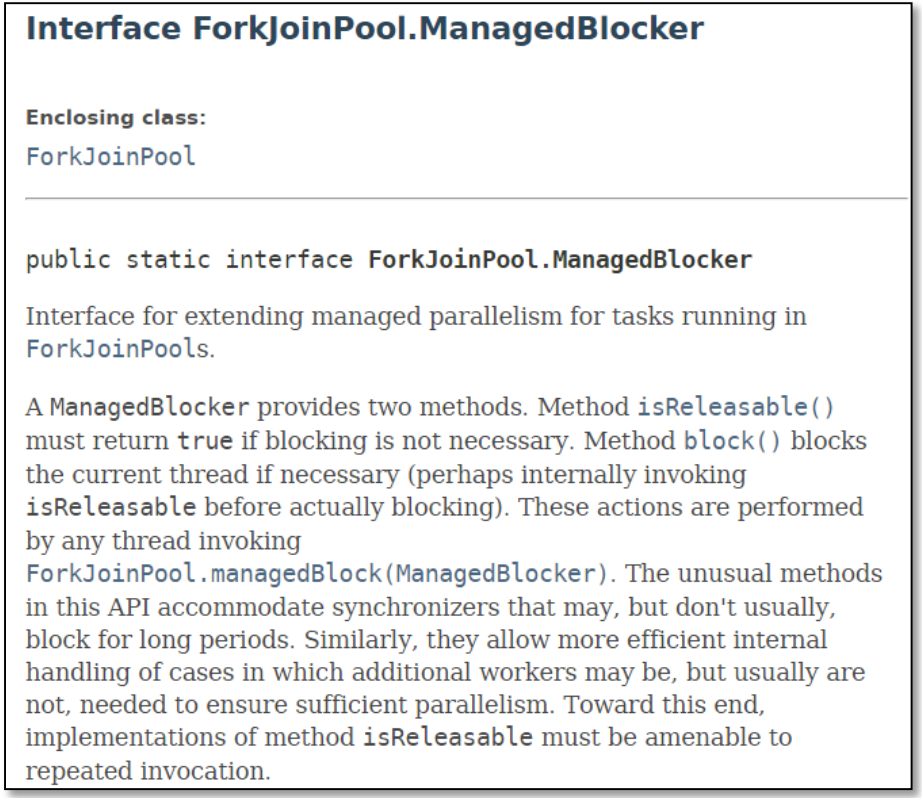
```
SupplierManagedBlocker<T> mb =
  new SupplierManagedBlocker<>
   (supplier);
...
ForkJoinPool.managedBlock(mb);
...
return mb.getResult();
```

See github.com/douglascraigschmidt/LiveLessons/tree/master/ImageStreamGang

Overview of the ManagedBlocker Interface

Overview of the ManagedBlocker Interface

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



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
  - e.g., to automatically/temporarily increase common fork/join pool size

Interface ForkJoinPool.ManagedBlocker

Enclosing class:

`ForkJoinPool`

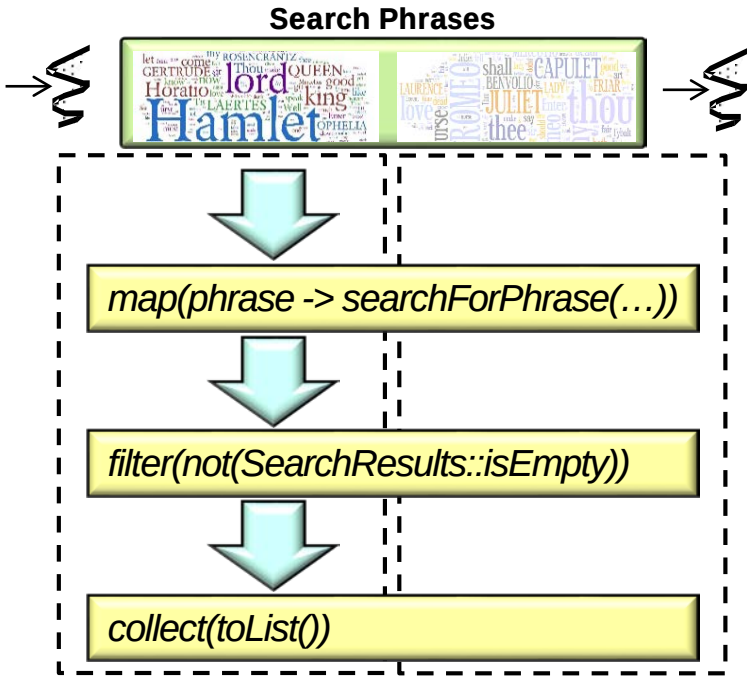
`public static interface ForkJoinPool.ManagedBlocker`

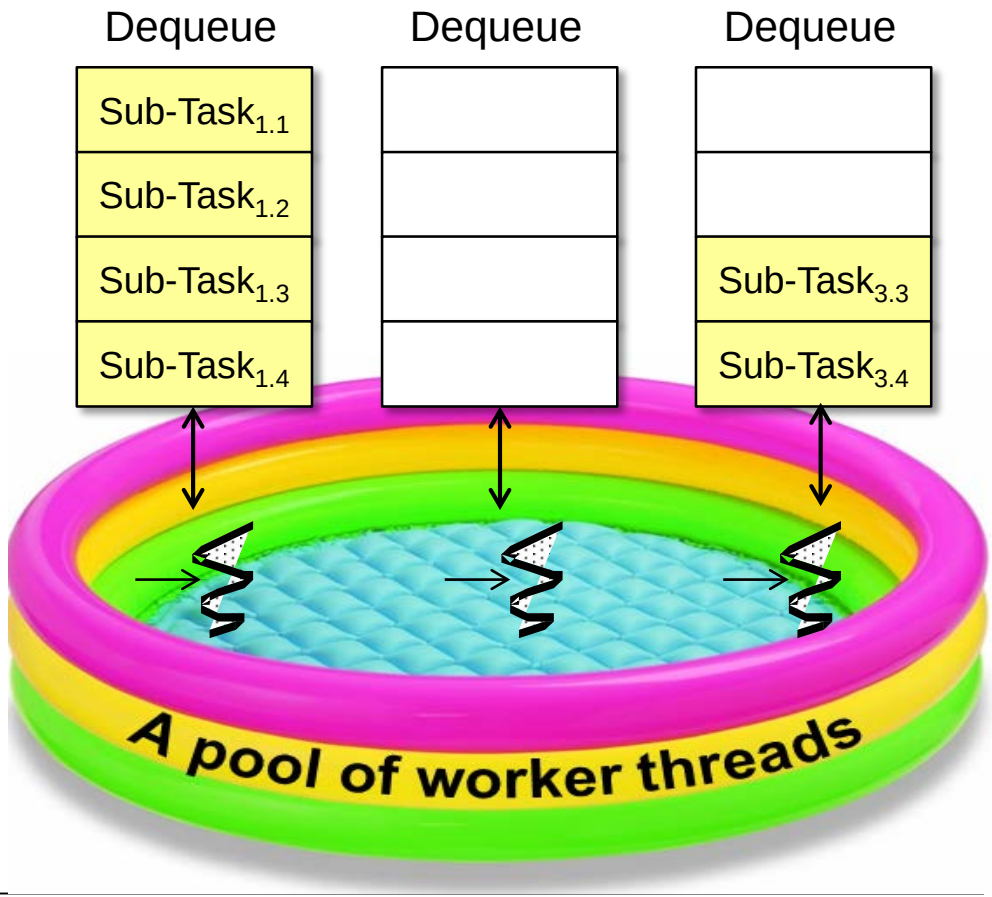
Interface for extending managed parallelism for tasks running in `ForkJoinPools`.

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

- ManagedBlocker can be used for both conventional fork-join & parallel stream use-cases



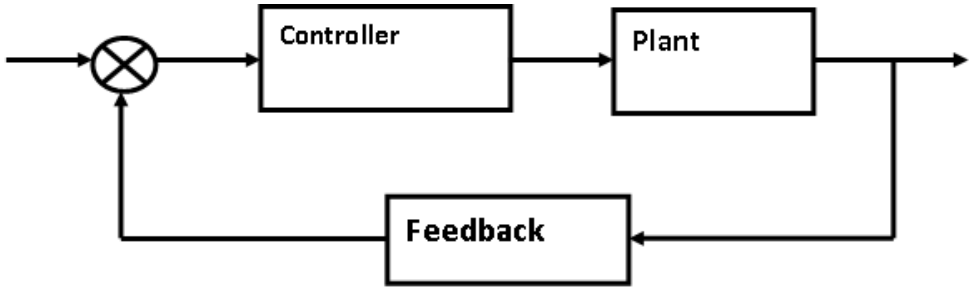


Overview of the ManagedBlocker Interface

- ManagedBlocker can be used for both conventional fork-join & parallel stream use-cases
  - ForkJoinPool reclaims threads during periods of non-use & reinstates them on later use

Overview of the ManagedBlocker Interface

- ManagedBlocker can be used for both conventional fork-join & parallel stream use-cases
  - ForkJoinPool reclaims threads during periods of non-use & reinstates them on later use
  - ForkJoinPool also tries to create or activate threads to ensure the target parallelism level is met



Overview of the ManagedBlocker Interface

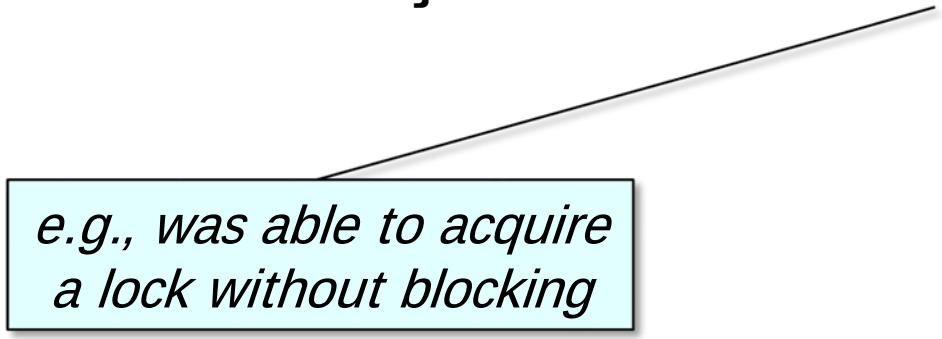
- ManagedBlocker defines two methods

```
interface ManagedBlocker {
  boolean block();

  boolean isReleasable();
}
```

Overview of the ManagedBlocker Interface

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

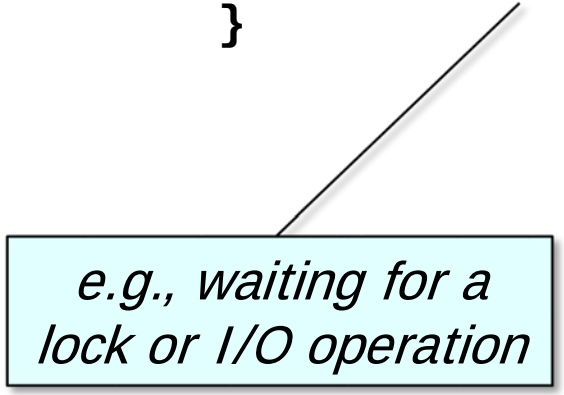
```
interface ManagedBlocker {
  boolean isReleasable();

  boolean block();
}
```



Overview of the ManagedBlocker Interface

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

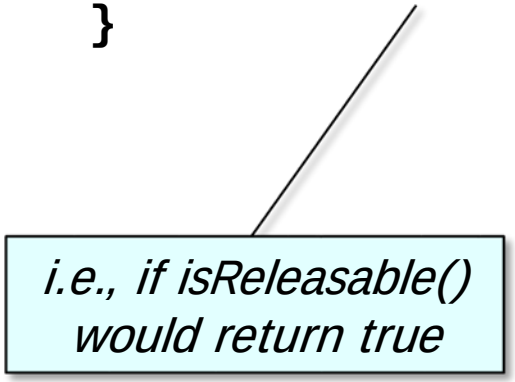
```
interface ManagedBlocker {
  boolean isReleasable();

  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();
}
```



Overview of the ManagedBlocker Interface

- 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

Overview of the ManagedBlocker Interface

- 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

Overview of the ManagedBlocker Interface

- 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());
        ...
  }
  ...
```

Unless there are already enough live threads, create or re-activate a spare thread to compensate for blocked joiners until they unblock

Overview of the ManagedBlocker Interface

- 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

Overview of the ManagedBlocker Interface

- Here is a ManagedBlocker based on a ReentrantLock (from Java docs)

```
class ManagedLocker implements ManagedBlocker {
  final ReentrantLock mLock;
  boolean mHasLock = false;

  ManagedLocker(ReentrantLock lock) { mLock = lock; }

  public boolean isReleasable()
  { return mHasLock || (mHasLock = mLock.tryLock()); }

  public boolean block() {
    if (!mHasLock)
      mLock.lock();
    return true;
  }
}
```

Handles a blocking synchronizer

Overview of the ManagedBlocker Interface

- Here is a ManagedBlocker based on a ReentrantLock (from Java docs)

```
class ManagedLocker implements ManagedBlocker {
  final ReentrantLock mLock;
  boolean mHasLock = false;

  ManagedLocker(ReentrantLock lock) { mLock = lock; }

  public boolean isReleasable()
  { return mHasLock || (mHasLock = mLock.tryLock()); }

  public boolean block() {
    if (!mHasLock)
      mLock.lock();
    return true;
  }
}
```

Constructor stores the lock

Overview of the ManagedBlocker Interface

- Here is a ManagedBlocker based on a ReentrantLock (from Java docs)

```
class ManagedLocker implements ManagedBlocker {
  final ReentrantLock mLock;
  boolean mHasLock = false;

  ManagedLocker(ReentrantLock lock) { mLock = lock; }

  public boolean isReleasable()
  { return mHasLock || (mHasLock = mLock.tryLock()); }

  public boolean block() {
    if (!mHasLock)
      mLock.lock();
    return true;
  }
}
```

Tries to acquire the lock (non-blocking)

Overview of the ManagedBlocker Interface

- Here is a ManagedBlocker based on a ReentrantLock (from Java docs)

```
class ManagedLocker implements ManagedBlocker {
  final ReentrantLock mLock;
  boolean mHasLock = false;

  ManagedLocker(ReentrantLock lock) { mLock = lock; }

  public boolean isReleasable()
  { return mHasLock || (mHasLock = mLock.tryLock()); }

  public boolean block() {
    if (!mHasLock)
      mLock.lock();
    return true;
  }
}
```

Performs a blocking lock operation

Overview of BlockedTask Class in ImageStreamGang

Overview of BlockingTask Class in ImageStreamGang

- BlockingTask integrates blocking Suppliers with the common fork/join pool

```
public class BlockingTask {
  ...
  public static<T> T callInManagedBlocker(Supplier(T> supplier){



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

See app/src/main/java/livelessons/imagestreamgang/utils/BlockingTask.java

Overview of BlockingTask Class in ImageStreamGang

- BlockingTask integrates blocking Suppliers with the common fork/join pool

```
public class BlockingTask {
  ...
  public static<T> T callInManagedBlocker(Supplier(T> supplier){

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

Enables the use of blocking Suppliers with the common Java fork/join thread pool

See stackoverflow.com/q/37512662 for pros & cons of this approach

Overview of BlockingTask Class in ImageStreamGang

- BlockingTask integrates blocking Suppliers with the common fork/join pool

```
public class BlockingTask {
  ...
  public static<T> T callInManagedBlocker(Supplier(T> supplier){

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

Create a helper object to encapsulate the supplier

Overview of BlockingTask Class in ImageStreamGang

- BlockingTask integrates blocking Suppliers with the common fork/join pool

```
public class BlockingTask {
  ...
  public static<T> T callInManagedBlocker(Supplier(T> supplier){



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

Submit managedBlock to common ForkJoin thread pool

Overview of BlockingTask Class in ImageStreamGang

- BlockingTask integrates blocking Suppliers with the common fork/join pool

```
public class BlockingTask {
  ...
  public static<T> T callInManagedBlocker(Supplier(T> supplier){



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

Return the result of the blocking call

Overview of BlockingTask Class in ImageStreamGang

- BlockingTask integrates blocking Suppliers with the common fork/join pool

```
public class BlockingTask {
  ...

  private static class SupplierManagedBlocker<T>
                       implements ForkJoinPool.ManagedBlocker {
    /* The blocking supplier. */
    private final Supplier<T> mSupplier;

    /* Keeps track of whether blocking supplier is done. */
    private boolean mDone = false;

    /* Constructor initializes the field. */
    private SupplierManagedBlocker(final Supplier supplier)
    { mSupplier = supplier; } ...
```

Blocking Supplier can work with common fork/join pool

Overview of BlockingTask Class in ImageStreamGang

- BlockingTask integrates blocking Suppliers with the common fork/join pool

```
public class BlockingTask {
   ...

  private static class SupplierManagedBlocker<T>
                  implements ForkJoinPool.ManagedBlocker {
    ...
    public boolean block()
    { mSupplier.get(); mDone = true; return true; }

    public boolean isReleasable()
    { return mDone; }

    public T getResult() { return mResult; }
}
```

Calls the blocking Supplier's get() method

Overview of BlockingTask Class in ImageStreamGang

- BlockingTask integrates blocking Suppliers with the common fork/join pool

```
public class BlockingTask {
   ...

   private static class SupplierManagedBlocker<T>
                  implements ForkJoinPool.ManagedBlocker {
     ...
     public boolean block()
     { mSupplier.get(); mDone = true; return true; }

     public boolean isReleasable()
     { return mDone; }

     public T getResult() { return mResult; }
}
```

True if blocking supplier has finished, else false

Overview of BlockingTask Class in ImageStreamGang

- BlockingTask integrates blocking Suppliers with the common fork/join pool

```
public class BlockingTask {
   ...

   private static class SupplierManagedBlocker<T>
                  implements ForkJoinPool.ManagedBlocker {
     ...
     public boolean block()
     { mSupplier.get(); mDone = true; return true; }

     public boolean isReleasable()
     { return mDone; }

     public T getResult() { return mResult; }
}
```

Returns the supplier's result

Overview of BlockingTask Class in ImageStreamGang

- The ImageStreamParallel app uses BlockingTask in blockingDownload() to ensure there are enough threads in the common thread pool

```
Image blockingDownload(URL url) {
  return BlockingTask
    .callInManagedBlocker
      (()-> downloadImage(url));
}
```

Overview of BlockingTask Class in ImageStreamGang

- The ImageStreamParallel app uses BlockingTask in blockingDownload() to ensure there are enough threads in the common thread pool

```
Image blockingDownload(URL url) {
  return BlockingTask
    .callInManagedBlocker
      (()-> downloadImage(url));
}
```

Transform a URL to an Image by downloading each image via its URL

Overview of BlockingTask Class in ImageStreamGang

- The ImageStreamParallel app uses BlockingTask in blockingDownload() to ensure there are enough threads in the common thread pool

```
Image blockingDownload(URL url) {
   return BlockingTask
      .callInManagedBlocker
        (()-> downloadImage(url));
}
```

This call ensures the common fork/join thread pool is expanded to handle the blocking image download

Overview of BlockingTask Class in ImageStreamGang

- The ImageStreamParallel app uses BlockingTask in blockingDownload() to ensure there are enough threads in the common thread pool

```
Image blockingDownload(URL url) {
  return BlockingTask
    .callInManagedBlocker
      (()-> downloadImage(url));
}
```

Extra threads in the common fork-join pool are automatically terminated later

End of Java 8 Parallel ImageStreamGang Example (Part 3)