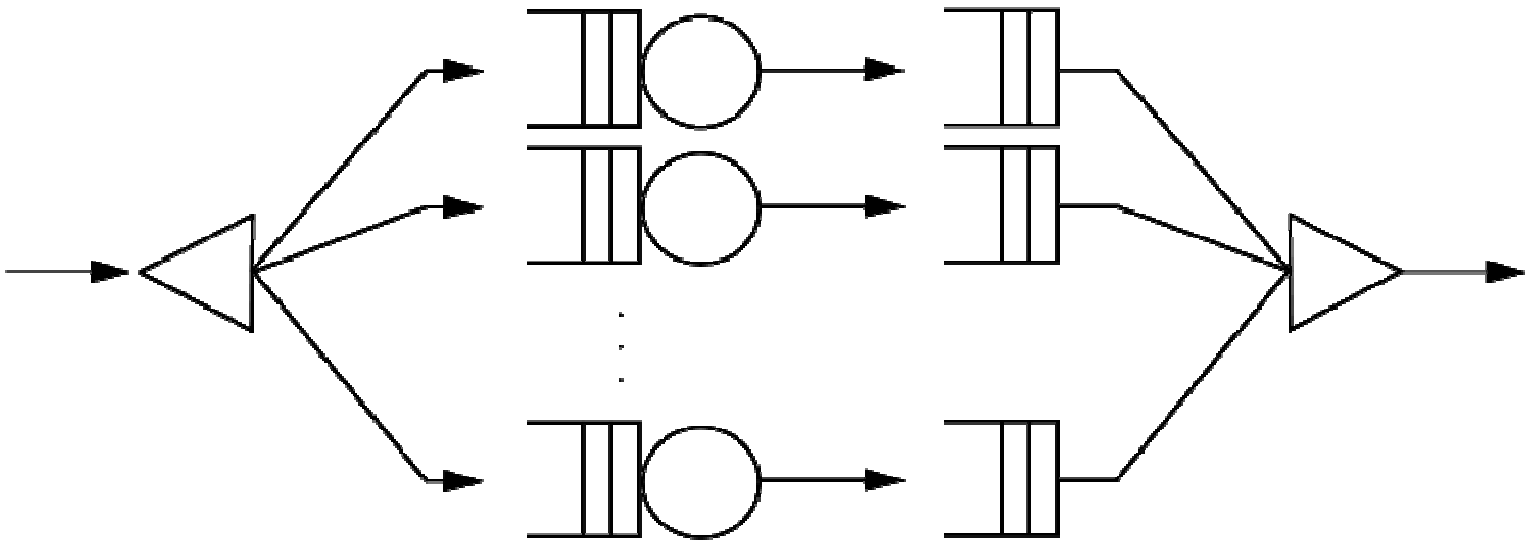
JDK 7 출시 기념 (2011.7)
JDK 7 소개
#4 Concurrenty
Fork & Join (jsr166y)

김용환

knight76.tistory.com

Knight76 at gmail.com

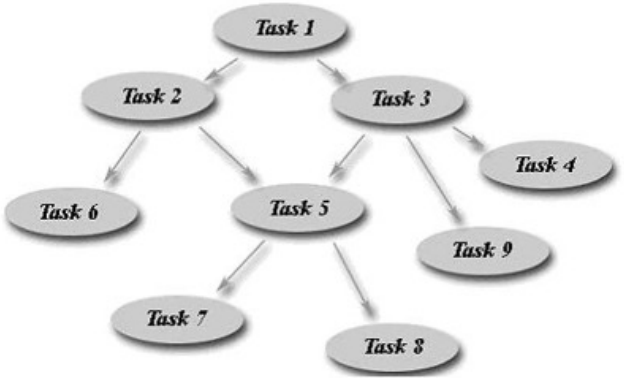
Fork & Join



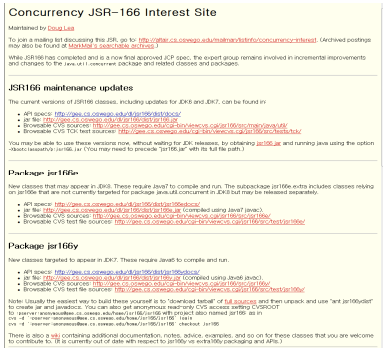
Multicore-friendly lightweight parallel framework

- JSR166y (maintenance)
- 목적
  - 놀고 있는 여러 개의 processor를 최대한 활용을 위해서 디자인됨
  - 병렬처리 필요
- MapReduce와 비슷하지만, 더 세분화할 수 있다. Work-stealing 알고리즘을 기반.

Fork/Join



- Fork
  - **Recursively** 큰 단위의 작업을 작은 단위로 쪼갬
- Join
  - 결과를 **Recursive**하게 모음
- Doug Lee (뉴욕 주립대 교수)
  - http://gee.cs.oswego.edu/dl/papers/fj.pdf
  - http://g.oswego.edu/dl/concurrency-interest/


Concurrency JSR-166 Interest Site

Maintained by Doug Lea

To join a mailing list discussing this JSR, go to: http://altair.cs.oswego.edu/mailman/listinfo/concurrency-interest. (Archived postings may also be found at MarkMail's Searchable Archives.)

While JSR166 has completed and is a now final approved JCP spec, the expert group remains involved in incremental improvements and changes to the java.util.concurrent package and related classes and packages.

JSR166 maintenance updates

The current versions of JSR166 classes, including updates for JDK6 and JDK7, can be found in:

- API specs: http://gee.cs.oswego.edu/dl/jsr166/dist/docs/
- jar file: http://gee.cs.oswego.edu/dl/jsr166/dist/jsr166.jar
- Browsable CVS sources: http://gee.cs.oswego.edu/cgi-bin/viewcvs.cgi/jsr166/src/main/java/util/
- Browsable CVS TCK test sources: http://gee.cs.oswego.edu/cgi-bin/viewcvs.cgi/jsr166/src/test/tck/

You may be able to use these versions now, without waiting for JDK releases, by obtaining jsr166.jar and running Java using the option -Xbootclasspath/p:jsr166.jar (You may need to precede "jsr166.jar" with its full file path.)

Package jsr166e

New classes that may appear in JDK8. These require Java7 to compile and run. The subpackage jsr166e.extra includes classes relying on jsr166e that are not currently targeted for package java.util.concurrent in JDK8 but may be released separately.

- API specs: http://gee.cs.oswego.edu/dl/jsr166/dist/jsr166edocs/
- jar file: http://gee.cs.oswego.edu/dl/jsr166/dist/jsr166e.jar (compiled using Java7 javac).
- Browsable CVS sources: http://gee.cs.oswego.edu/cgi-bin/viewcvs.cgi/jsr166/src/jsr166e/
- Browsable CVS test file sources: http://gee.cs.oswego.edu/cgi-bin/viewcvs.cgi/jsr166/src/test/jsr166e/

Package jsr166y

New classes targeted to appear in JDK7. These require Java6 to compile and run.

- API specs: http://gee.cs.oswego.edu/dl/jsr166/dist/jsr166ydocs/
- jar file: http://gee.cs.oswego.edu/dl/jsr166/dist/jsr166y.jar (compiled using Java6 javac)
- Browsable CVS sources: http://gee.cs.oswego.edu/cgi-bin/viewcvs.cgi/jsr166/src/jsr166y/
- Browsable CVS test file sources: http://gee.cs.oswego.edu/cgi-bin/viewcvs.cgi/jsr166/src/test/jsr166y/

Note: Usually the easiest way to build these yourself is to "download tarball" of full sources and then unpack and use "ant jsr166dist" to create jar and javadocs. You can also get anonymous read-only CVS access setting CVSROOT to ":pserver:anoncvs@gee.cs.oswego.edu/home/jsr166/jsr166 with password also named jsr166 -as in cvs -d :pserver:anoncvs@gee.cs.oswego.edu/home/jsr166/jsr166 checkout jsr166

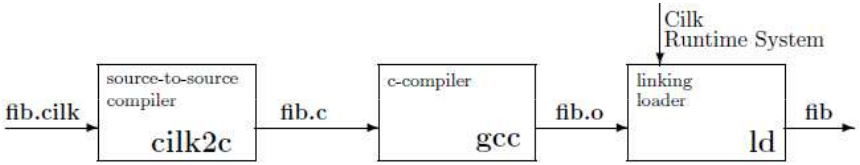
There is also a wiki containing additional documentation, notes, advice, examples, and so on for these classes that you are welcome to contribute to. (It is currently out of date with respect to jsr166y' vs extra166y packaging and APIs.)

Map Reduce와 비교

- Map Reduce
  - Node 기반의 Cluster 기반
  - Single fork
- Fork / Join
  - 하나의 장비안에서 여러 개의 CPU를 사용하려는 하나의 JVM
  - Recursive fork
  - Work Stealing

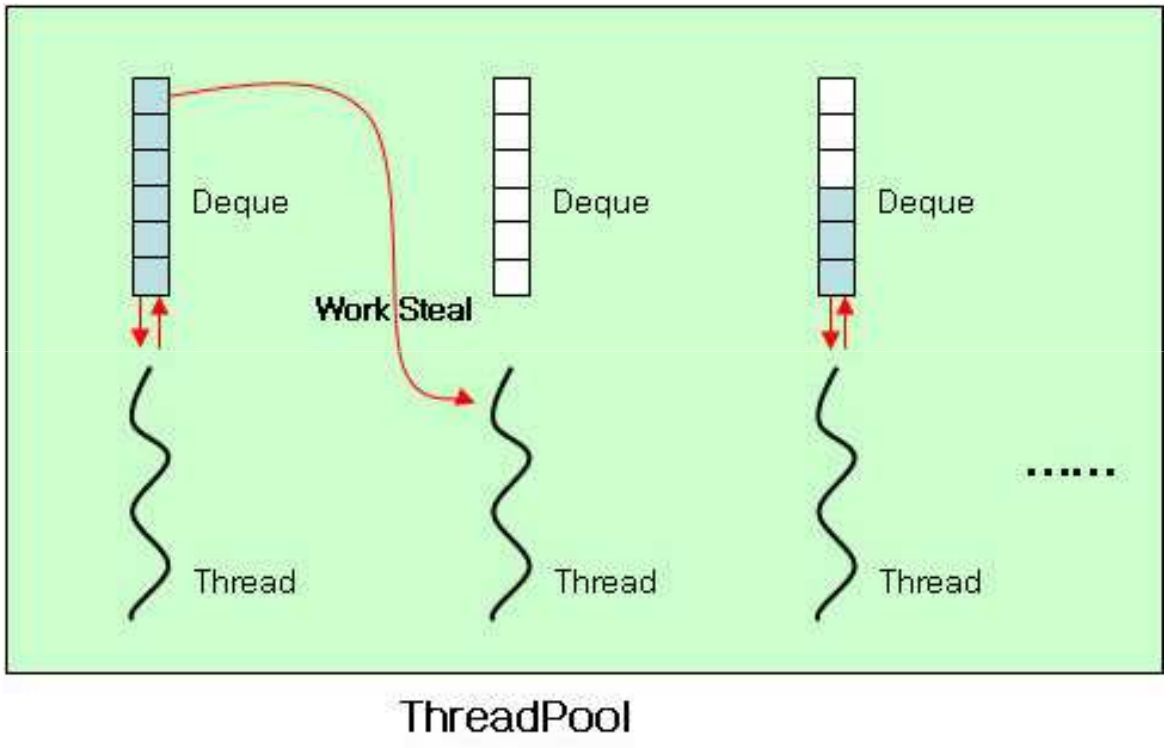
Work Stealing

- Cilk에서 개념을 채용 (논문)
  - MIT에서 만든 멀티쓰레드 프로그래밍 언어
  - 예약어 조금 + C언어 (http://supertech.csail.mit.edu/cilk/manual-5.4.6.pdf)



- 특징
  - 각 worker 는 DeQueue(double-ended queue)를 가지고 있음
  - worker가 idle일 때, 바쁜 Worker의 Queue에 쌓인 task 를 steal.
- 왜?
  - 여러 개의 processor(core)에서 동작 가능
  - 로드 밸런스 (작은 task로 나눌 수 있도록 함)

Work Stealing

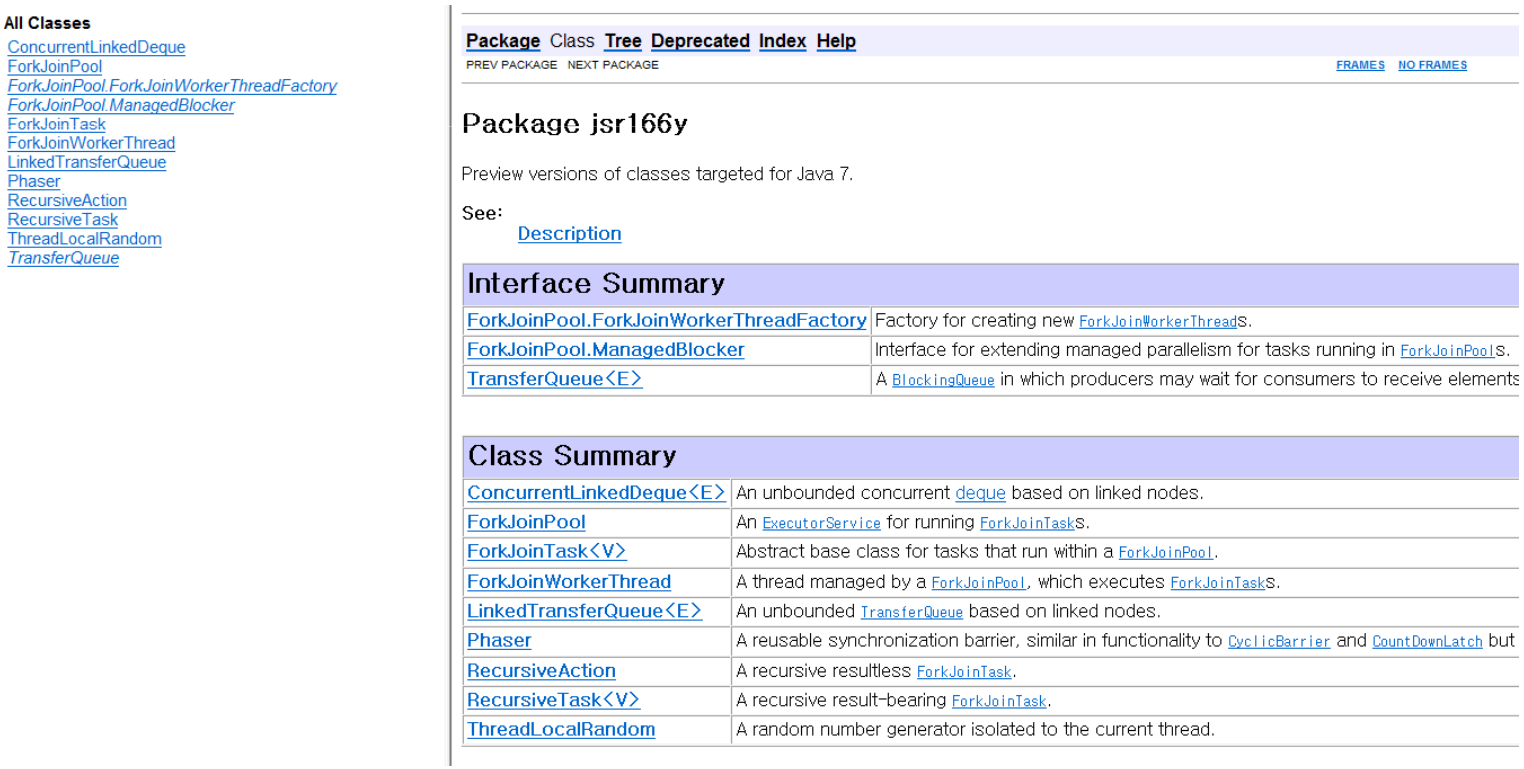


ForkJoinPool 클래스에 구현되어 있음

출처 : http://karlsenchoi.blogspot.com/2011/02/threadpool-work-stealing.html

jsr166y

- 추가된 API
  - http://gee.cs.oswego.edu/dl/jsr166/dist/jsr166ydocs/



All Classes

ConcurrentLinkedDeque
ForkJoinPool
ForkJoinPool.ForkJoinWorkerThreadFactory
ForkJoinPool.ManagedBlocker
ForkJoinTask
ForkJoinWorkerThread
LinkedTransferQueue
Phaser
RecursiveAction
RecursiveTask
ThreadLocalRandom
TransferQueue

Package Class **Tree** **Deprecated** **Index** **Help**

PREV PACKAGE NEXT PACKAGE FRAMES NO FRAMES

Package jsr166y

Preview versions of classes targeted for Java 7.

See:
Description

Interface Summary

ForkJoinPool.ForkJoinWorkerThreadFactory	Factory for creating new ForkJoinWorkerThreads.
ForkJoinPool.ManagedBlocker	Interface for extending managed parallelism for tasks running in ForkJoinPools.
TransferQueue<E>	A BlockingQueue in which producers may wait for consumers to receive elements

Class Summary

ConcurrentLinkedDeque<E>	An unbounded concurrent deque based on linked nodes.
ForkJoinPool	An ExecutorService for running ForkJoinTasks.
ForkJoinTask<V>	Abstract base class for tasks that run within a ForkJoinPool.
ForkJoinWorkerThread	A thread managed by a ForkJoinPool, which executes ForkJoinTasks.
LinkedTransferQueue<E>	An unbounded TransferQueue based on linked nodes.
Phaser	A reusable synchronization barrier, similar in functionality to CyclicBarrier and CountDownLatch but
RecursiveAction	A recursive resultless ForkJoinTask.
RecursiveTask<V>	A recursive result-bearing ForkJoinTask.
ThreadLocalRandom	A random number generator isolated to the current thread.

Pseudo code

```
if (my portion of the work is small enough)
    do the work directly
else
    split my work into two pieces
    invoke the two pieces and wait for the results
```

Java pseudo code

```
class OOOTask extends RecursiveTask {

    @Override
    public void compute() {
        // small enough
        if (problem.size < THRESHOLD) {
            // do directly
            problem.solve();
        } else {
            // split
            Task worker1 = new OOOTask(new Problem(problem.size - 1));
            Task worker2 = new OOOTask(new Problem(problem.size - 2));

            // invoke
            invokeAll(worker1, worker2);
        }
    }
}
```

예제

- 피보나치 계산
  - Fib(7) 계산
  - CPU는 4

FibonacciProblem

```
class FibonacciProblem {
        public int n;

        public FibonacciProblem(int n) {
                this.n = n;
        }

        public long solve() {
                return fibonacci(n);
        }

        private long fibonacci(int n) {
                System.out.println("Thread: " + Thread.currentThread().getName()
                                + " calculates " + n);
                if (n <= 1) {
                        return n;
                } else {
                        return fibonacci(n - 1) + fibonacci(n - 2);
                }
        }
}
```

FibonacciTask

```
@SuppressWarnings("serial")
class FibonacciTask extends RecursiveTask<Long> {
        private static final int THRESHOLD = 5;
        private FibonacciProblem problem;
        public long result;

        public FibonacciTask(FibonacciProblem problem) {
                this.problem = problem;
        }

        @Override
        public Long compute() {
                if (problem.n < THRESHOLD) {
                        result = problem.solve();
                } else {
                        FibonacciTask worker1 = new FibonacciTask(
                                new FibonacciProblem(problem.n - 1));
                        FibonacciTask worker2 = new FibonacciTask(
                                new FibonacciProblem(problem.n - 2));
                        worker1.fork();
                        result = worker2.compute() + worker1.join();
                }
                return result;
        }
}
```

Main

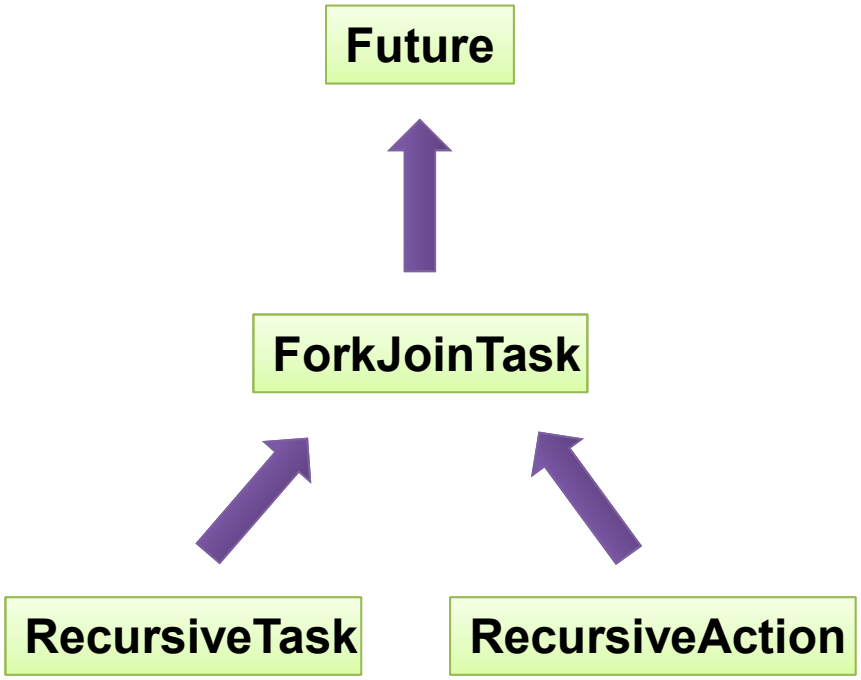
```
public static void main(String[] args) throws Exception {
    System.out.println("Number of processors: " +
                               Runtime.getRuntime().availableProcessors());
    int n = 7;

    StopWatch stopWatch = new StopWatch();
    FibonacciProblem bigProblem = new FibonacciProblem(n);

    FibonacciTask task = new FibonacciTask(bigProblem);
    ForkJoinPool pool = new ForkJoinPool();
    pool.invoke(task);

    long result = task.result;
    System.out.println("Computed Result: " + result);

    stopWatch.stop();
    System.out.println("Elapsed Time: " + stopWatch.getTotalTimeMillis());
    System.out.println("Steal Count : " + pool.getStealCount());
}
```

실행 결과

```
No of processors: 4
Thread: ForkJoinPool-1-worker-1 calculates 3
Thread: ForkJoinPool-1-worker-2 calculates 4
Thread: ForkJoinPool-1-worker-3 calculates 4
Thread: ForkJoinPool-1-worker-3 calculates 3
Thread: ForkJoinPool-1-worker-3 calculates 2
Thread: ForkJoinPool-1-worker-3 calculates 1
Thread: ForkJoinPool-1-worker-4 calculates 3
Thread: ForkJoinPool-1-worker-4 calculates 2
Thread: ForkJoinPool-1-worker-4 calculates 1
Thread: ForkJoinPool-1-worker-2 calculates 3
Thread: ForkJoinPool-1-worker-2 calculates 2
Thread: ForkJoinPool-1-worker-1 calculates 2
Thread: ForkJoinPool-1-worker-1 calculates 1
Thread: ForkJoinPool-1-worker-2 calculates 1
Thread: ForkJoinPool-1-worker-4 calculates 0
Thread: ForkJoinPool-1-worker-3 calculates 0
Thread: ForkJoinPool-1-worker-3 calculates 1
```

```
Thread: ForkJoinPool-1-worker-4 calculates 1
Thread: ForkJoinPool-1-worker-2 calculates 0
Thread: ForkJoinPool-1-worker-1 calculates 0
Thread: ForkJoinPool-1-worker-2 calculates 1
Thread: ForkJoinPool-1-worker-3 calculates 2
Thread: ForkJoinPool-1-worker-2 calculates 2
Thread: ForkJoinPool-1-worker-1 calculates 1
Thread: ForkJoinPool-1-worker-2 calculates 1
Thread: ForkJoinPool-1-worker-3 calculates 1
Thread: ForkJoinPool-1-worker-3 calculates 0
Thread: ForkJoinPool-1-worker-2 calculates 0
Thread: ForkJoinPool-1-worker-4 calculates 4
Thread: ForkJoinPool-1-worker-4 calculates 3
Thread: ForkJoinPool-1-worker-4 calculates 2
Thread: ForkJoinPool-1-worker-4 calculates 1
Thread: ForkJoinPool-1-worker-4 calculates 0
Thread: ForkJoinPool-1-worker-4 calculates 1
Thread: ForkJoinPool-1-worker-4 calculates 2
Thread: ForkJoinPool-1-worker-4 calculates 1
Thread: ForkJoinPool-1-worker-4 calculates 0
Computed Result: 13
Elapsed Time: 4
Steal Count : 4
```

RecursiveTask & RecursiveAction



API (RecursiveAction)

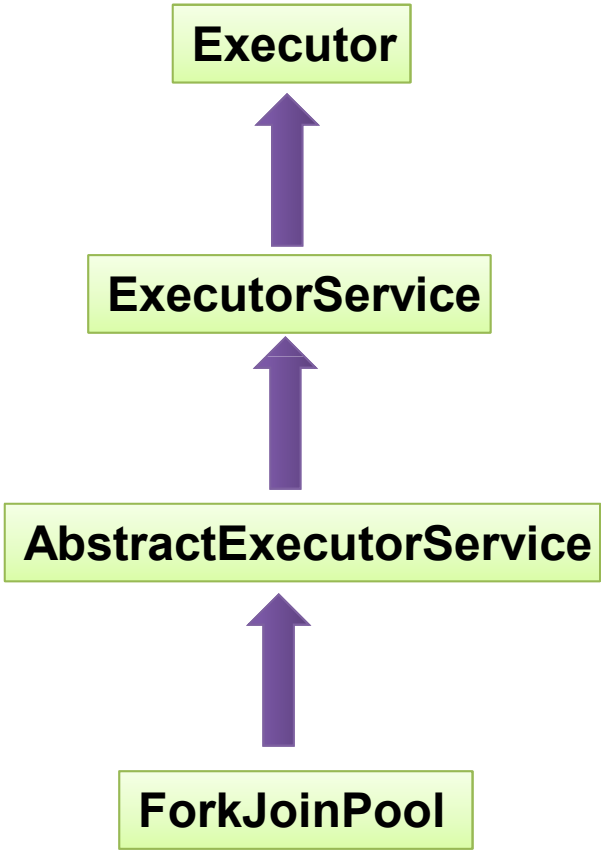
```
public abstract class RecursiveAction<V> extends ForkJoinTask<V> {
    private static final long serialVersionUID = 5232453952276485270L;

    protected abstract V compute();

    public final void getRawResult() {
        return null;
    }

    protected final void setRawResult(V value) {
    }

    protected final boolean exec() {
        compute();
        return true;
    }
}
```

API (RecursiveTask)

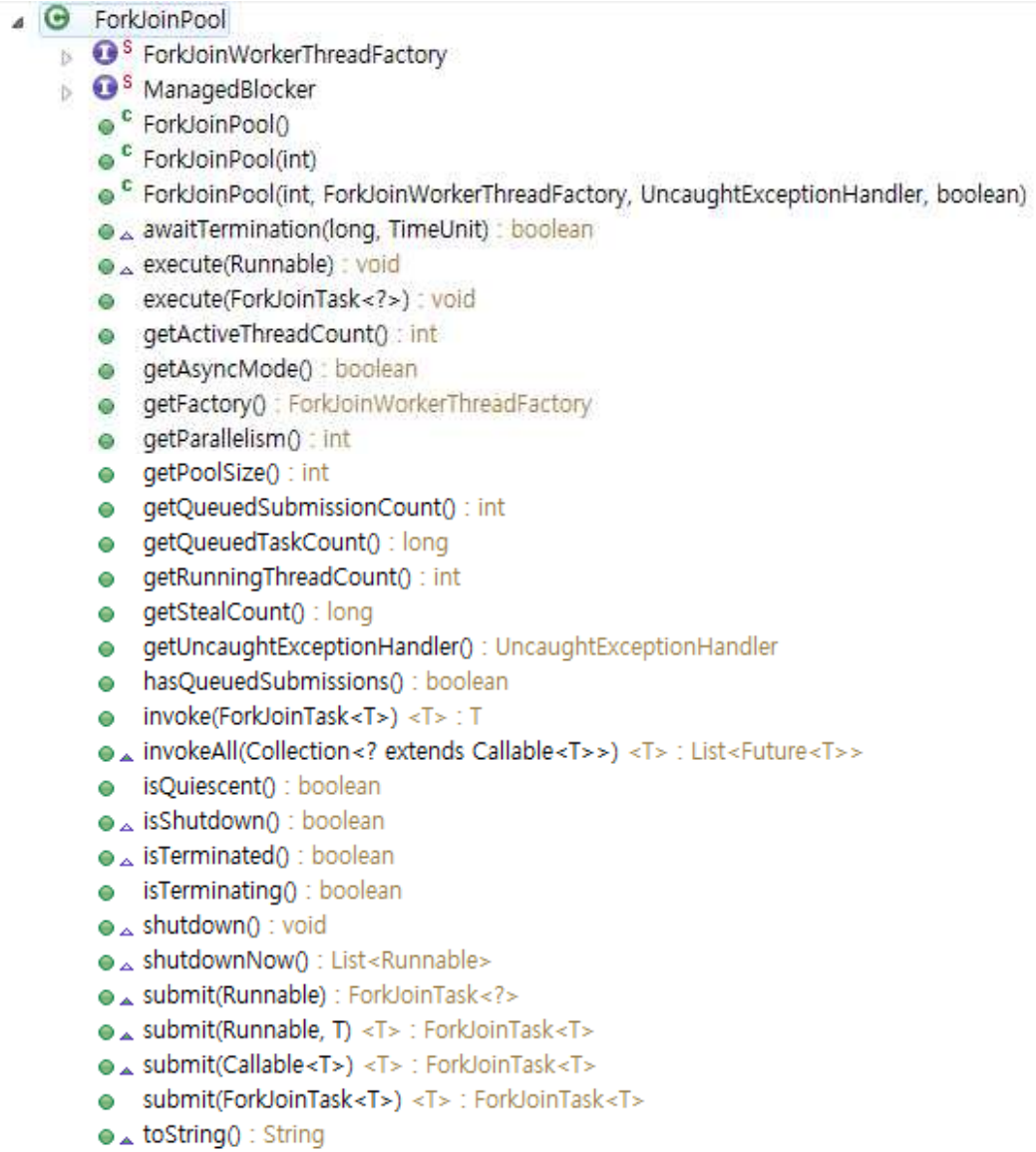
```
public abstract class RecursiveTask extends ForkJoinTask<Void> {
    private static final long serialVersionUID = 5232453952276485270L;

    V result;
    protected abstract V compute();

    public final V getRawResult() {
        return result;
    }

    protected final void setRawResult(V value) {
        result = value;
    }

    protected final boolean exec() {
        result = compute();
        return true;
    }
}
```

API (ForkJoinTask)

API (ForkJoinPool)



ForkJoinPool



기타

ThreadLocalRandom

- 멀티쓰레드상에서 Random 은 contention과 overhead가 존재. Random 내부에서 thread 를 isolation을 함
- ForkJoinPool에서 Random 사용시 이슈가 속도가 저하되기 때문에 이를 해결할 수 있는 클래스

```
long f = ThreadLocalRandom.current().nextLong();
System.out.println("1 : " + f);
f = ThreadLocalRandom.current().nextLong();
System.out.println("2 : " + f);
```

```
1 : 4232237
2 : 76795643120952621 2
```

Phaser

- 특징
  - 쓰레드를 동시 시작/종료시킬 수 있도록 함
  - CyclicBarrier와 CountDownLatch 클래스를 보다 유연함
- 좋은 자료
  - http://download.oracle.com/javase/7/docs/api/java/util/concurrent/Phaser.html
  - http://olegignatenko.livejournal.com/16771.html

Phaser 예제

```
public static void main(String[] args) {
    runDevelopment(Arrays.asList(new Developer(), new Developer(),
                        new Developer(), new Developer()));
}

private static void runDevelopment(Iterable<Developer> team) {
        final Phaser manager = new Phaser(1); //"1" to register self
        System.out.println("mgr: assign all developers, then start coding");

        for (final Developer developer : team) {
            final String dev = developer.toString();
            System.out.println("mgr: assigns a new unarrived " + dev + " to the project");
            manager.register();
            new Thread() {
                public void run() {
                    System.out.println("mgr: " + dev + ", please await all developers");
                    manager.arriveAndAwaitAdvance(); // await all creation
                    System.out.println("mgr: " + dev + ", OK to start coding");
                    developer.run();
                }
            }.start();
        }
        System.out.println("mgr: all assigned, start coding after all arrive");
        manager.arriveAndDeregister();
    }

class Developer implements Runnable {
    private final static AtomicInteger idSource = new AtomicInteger();
    private final int id = idSource.incrementAndGet();
    public void run() {     System.out.println(toString() + ": coding");     }
    public String toString() { return "developer #" + id; }
}
```

Phaser 결과

```
mgr: assign all developers, then start coding
mgr: assigns a new unarrived developer #1 to the project
mgr: assigns a new unarrived developer #2 to the project
mgr: developer #1, please await all developers
mgr: assigns a new unarrived developer #3 to the project
mgr: assigns a new unarrived developer #4 to the project
mgr: developer #3, please await all developers
mgr: all assigned, start coding after all arrive
mgr: developer #4, please await all developers
mgr: developer #2, please await all developers
mgr: developer #2, OK to start coding
developer #2: coding
mgr: developer #1, OK to start coding
developer #1: coding
mgr: developer #3, OK to start coding
developer #3: coding
mgr: developer #4, OK to start coding
developer #4: coding
```

To be continued #5