Java Concurrency Utilities

• Why use them?
  • Reduced programming effort
  • Increased performance
  • Increased reliability
  • Improved maintainability
  • Increased productivity

• Synchronizing Mechanisms
  • Task scheduling framework
  • Concurrent collections
  • Atomic variables
  • Synchronizers
  • Locks
  • Nanosecond-granularity timing
Java Concurrency: Task Scheduling

http://java.sun.com/j2se/1.5.0/docs/api/java/util/concurrent/Executor.html

- **Executor**
  - An object that executes submitted Runnable tasks
  - `Void execute(Runnable command)`
  - Throws `RejectedExecutionException` and `NullPointerException`
  - `Executor executor = anExecutor`
  - `executor.execute(new RunnableTask1());`
  - `executor.execute(new RunnableTask2());`

- Single background thread
- Thread pool
- Saturation policy
Semaphores

- Acquiring and releasing resources
- $\text{Acquire}()$, $\text{tryAcquire}()$, $\text{acquireUninterruptibly}()$, $\text{release}()$, $\text{availablePermits}()$, $\text{availablePermits}()$, $\text{drainPermits}()$, $\text{getQueuedThreads}()$, $\text{getQueueLength}()$, $\text{hasQueuedThreads}()$, $\text{isFair}()$, $\text{reducePermits}()$

http://java.sun.com/j2se/1.5.0/docs/api/java/util/concurrent/Semaphore.html
Java Concurrency: Barriers

- Barriers
  - Many-times used as a cyclical barrier
    - All threads wait until all threads have “checked in” and then all threads are released
  - await(), await(long timeout, TimeUnit unit), getNumberWaiting(), getParties(), isBroken(), reset()

http://java.sun.com/j2se/1.5.0/docs/api/java/util/concurrent/CyclicBarrier.html
Java Concurrency: Latches

CountDownLatch
- Blocks until current count == 0,
  - Then releases all waiting threads
- Many-times used as a cyclical
- await()
- await(long timeout, TimeUnit unit)
- countdown()
- getCount()

http://java.sun.com/j2se/1.5.0/docs/api/java/util/concurrent/CountDownLatch.html
Java Concurrency: Exchangers

Synchronization point for two threads to exchange objects
- Example: neighboring nodes in a network or neighboring regions of a simulation space
- Exchange(V, x)
- Exchange(V x, long timeout, TimeUnit unit)

http://java.sun.com/j2se/1.5.0/docs/api/java/util/concurrent/Exchanger.html
Java Concurrency: Locks

- Locking and waiting for specific conditions
  - Condition
  - Lock
  - ReadWriteLock
  - AbstractQueuedSynchronizer
  - LockSupport

http://java.sun.com/j2se/1.5.0/docs/api/java/util/concurrent/locks/Condition.html
Java Concurrency: Queues

- Collection to hold elements prior to processing
- Implementing classes
  - AbstractQueue, ArrayBlockingQueue, ConcurrentLinkedQueue, DelayQueue, LinkedBlockingQueue, LinkedList, PriorityBlockingQueue
- Methods
  - `element()`, `offer(E o)`, `peak()`, `poll()`, `remove()`
  - Methods inherited from `java.util.Collection`
    - Add, `addAll`, `clear`, `contains`, `containsAll`, `equals`, `hashCode`, `isEmpty`, `iterator`, `remove`, `removeAll`, `retainAll`, `size`, `toArray`
Java Concurrency: BlockingQueue

- Wait for the queue to be non-empty
- Wait for spaced when storing an element
- Implementing Classes
  - ArrayBlockingQueue, DelayQueue, LinkedBlockingQueue, PriorityBlockingQueue, SynchronousQueue
- Methods
  - `add(E o)`, `drainTo(Collection<? Super E> c)`, `offer(E o)`, `poll(long timeout, TimeUnit unit)`, `put(E o)`, `remainingCapacity()`, `take()`
Support for lock-free thread-safe programming on single variables – extend **volatile**

- boolean compareAndSet(expectedValue, updateValue)


[http://java.sun.com/j2se/1.5.0/docs/api/java/util/concurrent/atomic/package-summary.html#package_description](http://java.sun.com/j2se/1.5.0/docs/api/java/util/concurrent/atomic/package-summary.html#package_description)