Java Concurrency Utilities

http://java.sun.com/j2se/1.5.0/docs/guide/concurrency/

- **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

Java Concurrency: Semaphores

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

- **Semaphores**
  - Acquiring and releasing resources
  - Acquire(), tryAcquire(), acquireUninterruptibly(), release(), availablePermits(), availablePermits(),drainPermits(), getQueuedThreads(), getQueueLength() hasQueuedThreads(), isFair(), reducePermits()

Java Concurrency: Barriers

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

- 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()

Java Concurrency: Latches

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

- 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()

Java Concurrency: Exchangers

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

- 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)

Java Concurrency: Locks

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

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

Java Concurrency: Queues

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

- Collection to hold elements prior to processing
- Implementing classes
  - AbstractQueue, ArrayBlockingQueue, ConcurrentLinkedQueue, DelayQueue, LinkedBlockingQueue, LinkedList, PriorityBlockingQueue, PriorityQueue
- 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

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

- Wait for the queue to be non-empty
- wait for spaced when storing an element
- Implementing Classes
  - ArrayBlockingQueue, DelayQueue, LinkedBlockingQueue, PriorityBlocking Queue, SynchronousQueue
- Methods
  - add(E o), drainTo(Collection<? Super E> c), offer(E o), poll(long timeout, TimeUnit unit), put(E o), remainingCapacity(), take()

Java Concurrency: Atomic Variables

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

- Support for lock-free thread-safe programming on single variables – extend ***volatile***
  - boolean compareAndSet(expectedValue, updateValue)
- AtomicBoolean, AtomicInteger, AtomicIntegerArray, AtomicIntegerFieldUpdater, AtomicLong, AtomicLongArray, AtomicLongFieldUpdater, AtomicMarkableReference, AtomicReference, AtomicReferenceArray, AtomicReferenceFieldUpdater, AtomicStampedReference