Multithreading

* Thread: single sequential flow of control within a program
  + Single threaded program can handle one task at any time
* Multitasking allows a single processor to run several concurrent threads
* Creating threads in java
  + Extend Java.lang.Thread
    - Run() methods must be overridden
    - Run() is called when execution of the thread begins
    - A Thread terminates when run() returns
    - Start() method invokes the run() method.
  + Implement hava.lang.Runnable
    - If already inheriting another class
    - Single method
      * Public void run()
* States
  + New: A new thread begins its life cycle in the new state. It remains in this state until the program starts the thread. Once started it is a born thread.
  + Runnable: after a newly born thread is started, the threat becomes a runnable. Considered to be executing its task
  + Waiting: when a thread is waiting for another thread to run its task
  + Dead: done
* A thread becomes not runnable when one of these events occurs
  + Its sleep method is invoked
  + the thread calls the wait method
  + file io
* The static yield() method
  + The current thread is willing to yield its current use of a processor
* Sleep(int miliseconds)
* isAliive()
  + method used to find out the state of a thread.
  + Returns true if the thread is in the ready, blocked, or running
  + False if the thread is new, not started, not finished
* Stop()
  + Stops it
  + Should assign null to a thread variable instead
* Suspsend(), resume()
* Thread Scgeduking
  + Most OS use time slicing for threads of equal priority
  + Preemptive scheduling
    - When a thread of higher priority enters the running state, it preempts the current level
  + Starvation
    - Higher priority threads can postpone)possibly forever) the execution of lower priority threads.
* Thread pools
  + Starting a new thread for each task could limit throughout and cause poor performance
  + A thread pool is ideal to manage the number of tasks executing concurrently
  + Executor interface for executing Runnable object in a thread pool
* Thread Synchronization
  + A shared resource may be corrupted if it is accessed simultaneously by multiple threads
  + A synchronized method acquires a lock before it executes
  + Instance method: the lock is on the object for which it was invoked
  + Static method: the lock is on the class.
  + Lock is released when thread is finished executing.
* Synchrinizing statement
  + A synchronized block can be used to acquire a lock on any object
  + Synchronized(expr)

{

}

or

public synchronized void myMethod()

* A lock is an instance of the Lock interface
* Wait(), Notify(), notifyAll()
  + All must be called in synchronized block of code
* Wait()
  + Lets the thread wait until some condition occurs
* Notify()
  + Resume normal execution
* NotifyAll()
  + Resume all threads normal execution.
* Sometimes two or more threads need to acquire the locks on several shared objects
  + This could cause deadlock, in which each thread has the lock on one of the objects and is waiting for the other to release their lock.
    - Preventable with resource ordering

The classes in the Java collections are not thread safe

* + Can cause corruption
  + Java offers synchronization wrappers for collections.
* Invoking synchronizedCollection(Collection C0) returns a new Collection object in which all methods are synced.