**Wait() and notify()**

* mechanisms for coordination between running methods
  + these are methods defined by Object
  + must own the lock associated with the object in order to call any of these methods
    - means that the call to wait()/notify() must be within a synchronized method or statement
* wait()
  + calling thread waits until notified
* notify()
  + releases one waiting thread(as soon as lock is available)
  + causes the thread to wake up and resume execution only if the object calling it releases its lock
* notifyAll()
  + releases all waiting threads, but not at the same time
    - it just signals them to eventually wake up at some time when the lock is available
  + each resumes in turn as lock becomes available

Ex:

* a class called SynchPoint that stores a number of threads

public SynchPoint(int numExpected){

this.numExpected = numExpected;

this.numCheckIn = 0;

}

synchronized void checkIn(String prefix){

try{

System.out.println(prefix + " checking in");

numCheckedIn++;

if(numCheckedin < numExpected){

wait()

} else{

numCheckedIn = 0;

notifyAll()

}

System.out.println("All " + numExpected + "checked in");

} catch(InterruptedException e){

e.printStackTrace()

}

}

Thread.sleep(int ms)

* sleeps for ms milliseconds

System.nanoTime()

* highest resoluiton timer available in Java
* return value is long
* difference in value between two different calls represents duration in nanoseconds
  + actual resolution is courser than a nanosecond

Demonstrating Parallelism - dividing a complex task and running subtasks on multiple threads at same time

* what we've seen so far are threads in user interface, or time division on a processor
  + we make sure that tasks are divided among threads to service UI events
    - but this doesn't require a dual core processor for it to work
    - what we've relaly done is time division of the same processor, giving the illusion of multiple processors
* Generating 100 million random numbers
  + with varying number of threads from 1-16
  + how many cores does my machine have?
    - you should be able to cut a really complex problem into n pieces, you should see better and better performance ach time because each piece can potentially run simultaneously
* Runtime.getRuntime().availableProcessors()
  + gives how many cores are available in your JVM environment