Slide 1

Administrivia

- Reminder: Second installment of Homework 1 (MPI program) due Friday.

Slide 2

Minute Essay From Last Lecture

- (Review question.)
- Problems include poor performance and even deadlock.

Slide 3

Synchronization in Java, Recap

- Simplest way to ensure one-at-a-time access to shared variables is to use `synchronized` keyword.
- Review `HelloSynch*.java` sample programs . . .

Slide 4

Numerical Integration Example, Revisited

- How to parallelize using Java? well, first must rewrite in Java (`NumIntSeq.java` on sample programs page).
- Now rewrite to use multiple threads (`NumIntPar.java` on sample programs page) . . .

Slide 5

Synchronization in Java, Continued

- `synchronized` methods/blocks can be used to ensure that only one thread at a time accesses some shared variable.
- For more complex synchronization problems, can use `wait` and `notify` (or `notifyAll`):

  `wait` suspends executing thread (adds to "wait set").

  `notify` wakes up one thread from the wait set. `notifyAll` wakes up all threads. Waked-up thread(s) then compete to reacquire lock and continue execution.

  Can only be done from within synchronized method/block.

  Typical idiom — loop to check condition, `wait`.
- Example — bounded buffer class (`BoundedBuffer.java` and `TestBoundedBuffer.java` on sample programs page).

Slide 6

Minute Essay

- In the bounded buffer example, what do you think would happen if we replaced `notifyAll` with `notify`? (Would the program always work, never work, sometimes work? if "sometimes", when?)

Slide 7

Minute Essay Answer

- It might not always work. `notify` wakes up a waiting process, but no control over which one. Probably would be okay for this algorithm (since it seems unlikely that both producers and consumers would be waiting), but `notifyAll` seems like easiest way to be sure of correctness. (Could this be inefficient? maybe, but as Knuth says — "premature optimization is the root of all evil.")