Slide 1

Administrivia

- A word about access to multi-processor machines:

  Last year, we had seven dual-processor machines (Dwarf1 through Dwarf7) and a four-processor machine (SnowWhite).

  This year ITS bought us two new dual-processor machines (Dione01 and Dione02) and a four-processor machine (Dione00).

  Right now what's available is a mix — Dione01, Dione02, and SnowWhite. (Software on SnowWhite is somewhat back-level.)

Slide 2

Minute Essay From Last Lecture

- `synchronized` can avoid problems with shared variables. Can you think of problems this might introduce?

  "Sequentializing" everything?

  Deadlock?

  Problems if one thread throws an exception?

  Impact on public API?

  Buffers filling up?

  (When do you want to do this, when not?)

Slide 3

New Features in Java 5.0 for Multithreading

- Lots of new stuff for concurrent programming Java 5.0 (a.k.a. 1.5). Short examples — more versions of “hello world” (`Hello3.java`, `Hello4.java`, `Hello5.java` on sample programs page).

Slide 4

Synchronization in Java, Continued

- Recall — `synchronized` methods/blocks can be used to ensure that only one thread at a time accesses some shared variable. (Review sample program `HelloSynch.java`.)
- 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` on sample programs page).

Slide 5

Minute Essay

- How much of what we've talked about last time and today did you already know? (i.e., what if anything was new?)