Slide 1

Administrivia

- Reminder: MPI for Homework 1 due today. Java program due Tuesday.

Slide 2

Java from the Command Line

- Most of you probably use Eclipse to write Java programs. You can do that for this course too, but for this course you might prefer to run them from the command line (to make it easier to supply command-line arguments if nothing else). Command to use is `java`, followed by class name and any arguments.
- You can also write them using your favorite text editor compile from the command line. Command to compile is `javac`.

Slide 3

Multithreaded Programming in Java — Recap

- Multithreaded programing in Java is based on the shared-memory model, similar to OpenMP. Support provided by a few keywords (e.g., `synchronized`) and several library classes and methods.
- Create threads like other objects, either by subclassing `Thread` or with a `Runnable` object. (Second method usually preferred as being better object-oriented design.) Also some useful classes in `java.util.concurrency` — see `Hello3.java`, `Hello4.java`, `Hello5.java` on sample programs page.

Slide 4

Shared Variables in Java

- Code executed by a thread is some object's `run` method. Access to variables is consistent with usual Java scoping — class/instance variables, parameters, etc.
- As we noted before, though, simultaneous access to shared variables can be risky, however. So . . .

Slide 5

Synchronization in Java

- Interaction among threads in Java based on “monitor” idea (Hoare (1975) and Brinch Hansen (1975)).
- Every object has implicit lock; `synchronized` keyword means “only run this when you have the relevant lock” — if another thread has the lock, wait. Can be used to ensure one-at-a-time access to critical variables.

  “Relevant lock”? For synchronized methods, lock for object (instance methods) or class (static methods). For synchronized blocks, you specify the object.

  Example — `HelloSynch*.java` on sample programs page.
- `wait` and `notify` methods allow more interesting kinds of coordination. But first . . .

Slide 6

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, based on same strategy we used for OpenMP — split loop iterations among threads, give each its own copy of work variables, compute sum based on “reduction” idea. Some things must be done more explicitly in Java. See `NumIntPar.java` on sample programs page.

Slide 7

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`, `TestBoundedBuffer.java` on sample programs page).

Slide 8

Minute Essay

- `synchronized` has benefits — avoiding multiple threads changing a shared variable at the same time. What risks/disadvantages can you imagine that it might have?

Minute Essay Answer

- Among the possible problems are the possibility of deadlock, and performance that's worse than it needs to be.