Slide 1

Administrivia

- Homework 7 code due Thursday (deadline extended).
- Homework 8 will be due the day of the final (May 10).

Slide 2

Networking in Java — Sockets

- Last time we talked a little about Java programs communicating over networks using sockets.
- Simple example — “silly class saver” example. More complex example — chat program. See updated notes for last time.

Slide 3

Networking in Java — RMI

- Motivation — for client/server applications, can be annoying to have to design your own protocol.
- Instead, idea is to define “remote objects” that can be treated (at program level) like any other objects — invoke methods.
- Typical use in client/server program:
  - Server creates some remote objects and “registers” them.
  - Clients look up server’s remote objects and invoke their methods.
  - Both sides can pass around references to other remote objects.
- Dynamic code loading possible too.

Slide 4

Networking in Java — RMI, Quick How-To

- Define a class for remote objects:
  - Define interface that extends `Remote`
  - Define class that implements that interface, extends a Java “remote object” class. Can also include other methods, only available locally.
  - Write code using classes — if using as remote object, reference interface; otherwise can reference class.
- Compile and execute:
  - Compile as usual. (Prior to Java 1.5, an extra step was required to generate “stubs” to be used in communicating with remote objects as remote objects.
  - Make classes network-accessible.
  - Start `rmiregistry`.
  - Run server and clients as usual.

Slide 5

Networking in Java — RMI

- Example — revised chat program. Design is somewhat more elaborate than absolutely necessary, in an attempt to be modular and flexible:
  - Common interface `ChatParty` for remote objects for both client and server, with subinterfaces `ChatClient` and `ChatServer`, and classes implementing all of these.
  - Interface `ChatClientUI` for non-remote local UI for clients, with two implementations.
- Need for multithreading in server goes away — all handled by RMI under the hood (though we still need to be careful about possible concurrent access to variables — experiment suggests RMI may use multiple threads). In client UI, however, we still need separate threads to get input from the user and listen for messages from the server.

Slide 6

Threads in Java, Revisited

- Earlier in the semester we talked a little bit about multithreading in Java. Basic functionality — starting up new threads, coordinating actions of different threads — has been part of Java from the beginning, but more because it's a nice model than for performance reasons.
- With multicore machines becoming mainstream, though, using threads to improve performance is becoming more important. Much, much useful functionality in `java.util.concurrent`.

Slide 7

Example — Numerical Integration

- Compute π by integrating $\int_0^1 \frac{4}{1+x^2}\,dx$.
- Do this numerically by approximating area under curve by many small rectangles, computing their area, adding results.
- Sequential program fairly straightforward — loop to compute and sum areas of rectangles.
- How to divide up work to make use of multiple processors? divide up iterations of loop among processors, have each compute a partial sum, then combine results.
- In Java, we can do this by creating and starting multiple threads. Old way is to explicitly create and start threads. New way uses classes from `java.util.concurrent`. Look at example code (to be on Web later) . . .

Slide 8

Minute Essay

- Tell me about the status of your game — which assignments are done, how close you are to what you originally envisioned.