Slide 1

Administrivia

- Reminder: Homework 7 design due today, code Tuesday.
- (If you're behind on turning in homework: The late policy in the syllabus says normally no credit for homework more than a week late. I will give up to 50% credit, as long as you get it in by the last day of class.)

Slide 2

Networking Basics

- Inter-computer communication based on layered approach and "protocols":
  - Application level — HTTP, FTP, telnet, SMTP, POP, IMAP, NTP, etc., etc.
  - Transport level — TCP (Transmission Control Protocol), UDP (User Datagram Protocol).
  - Network level — IP (Internet Protocol — addressing, routing of packets).
  - Link level — device drivers, etc.
- Messages are routed to
  - A machine ("host"), identified by IPA or name.
  - A process, identified by "port number" (16 bits). 0 — 1023 are "well-known ports", others available for applications.

Slide 3

Networking Basics — TCP and UDP

- UDP — independent messages, no guarantees about reliability or message order — analogous to (snailmail) letter.
- TCP — point-to-point channel, guarantees reliability and message order — analogous to phone call. Endpoints called "sockets".

Slide 4

Networking in Java

- Classes for communicating at application level — e.g., `URL` ("show URL" example).
- Classes for communicating at network level:
  - TCP — `Socket`, `ServerSocket`.
  - UDP — `Datagram*`.
- RMI (Remote Method Invocation).

Slide 5

Networking in Java — Sockets

- Client/server model:
  - Server sets up "server socket" specifying port number, then waits to accept connections. Connection generates socket.
  - Client connects to server by giving name/IPA and port number — generates a socket.
  - On each side, get input/output streams for socket. Program must define protocol for the two sides to communicate.
- Simple example in binary-I/O program from last week. More complex example — chat program.

Slide 6

Client/Server Programming with Sockets — Chat Example

- In client/server programming, program must define "protocol" for clients and server to communicate. For chat program, fairly simple:
- Interaction starts with client sending identifying information and server responding with list of participants.
- Interaction continues with client sending messages to server, which broadcasts them (to other clients), and accepting broadcast messages from server.
- Interaction ends when client sends "done" message to server, which broadcasts this information to other clients.

Slide 7

Client/Server Programming with Sockets — Chat Example, Continued

- Code is fairly simple — classes for client and server, plus inner class for server to keep track of clients. Only tricky bits are related to concurrency . . .
- Server needs to be able to communicate with multiple clients asynchronously (i.e., no way to know which one will send a message next). One way to deal with this — start a new thread for each client. Must then be sure these threads don't concurrently modify shared data (here, list of clients).
- Client needs to be able to present GUI and also listen for messages broadcast by server. Less coding here since GUI runs in its own thread automagically, so we can use the main thread to listen for message from server. Only complication is that anything in this thread that needs to change the GUI must use `SwingUtilities.invokeLater` to be sure changes happen in event dispatch thread.

Slide 8

Minute Essay

- None — quiz.