

Slide 2



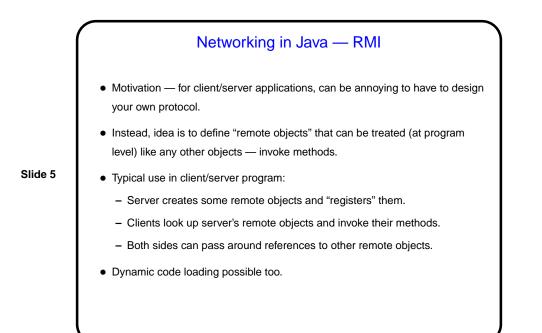
- In client/server programming, program must define "protocol" for clients and server to communicate. For chat program, fairly simple:
- Interaction starts with client sending identifying information and server responding with list of participants.
- Interaction continues with client sending messages 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.

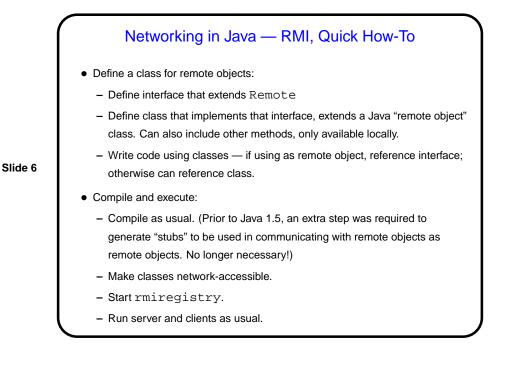
## Client/Server Programming with Sockets — Chat Example, Continued

- 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 3

Slide 4







- 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.

Slide 7

- 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.

