Slide 1

Administrivia

- Reminder: Homework 7 code due today.
- Homework 8 (final version of game) due day of final. Description on Web.
- We need some "not accepted past" dates. How about next Tuesday (11:59pm) for Homework 1 through 6, Friday for Homework 7, and day/time of exam for Homework 8?
- Final is May 9 at 8:30am. Review sheet on Web. No formal review session, but feel free to ask questions, in person or by e-mail.
- Information about office hours next week coming by e-mail soon.

Slide 2

More Administrivia

- "What about our grades?" You will get information by e-mail as soon as I have it.
- Recall(?) weights from syllabus:
  - 50 points class participation (attendance).
  - 30 points quiz scores (lowest dropped).
  - 300 points exams (100 midterm, 200 final).
  - 480 points homework.
- Late penalties will be reduced for most homeworks — say 5% per day with a maximum of 25%.

Slide 3

Networking in Java — Review/Recap

- Many library classes to support networking.
- Some provide application-level support — e.g., `URL`. Others provide lower-level support — e.g., `Socket`.
- Many work by supplying input/output streams, which can be used in the same way as other input/output streams (except that extra caution may be needed to make sure output isn't buffered longer than it needs to be).

Slide 4

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 (as shown, briefly, earlier). More complex example — chat program.

Slide 5

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 6

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 7

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 8

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. No longer necessary!)
  - Make classes network-accessible.
  - Start `rmiregistry`.
  - Run server and clients as usual.

Slide 9

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 10

Course Recap — What Did We Do?

- Java basics.
- Object-oriented programming — polymorphism, inheritance, etc. Not stressed much in class, but game is a good example of a non-trivial o-o design.
- Basic ADTs — stacks, queues, trees (sorted and heaps); different implementations (arrays versus dynamic data structures using references).
- Recursion review.
- Tour of the Java libraries — GUIs, graphics, I/O; a very little about threads and networking.
- A fairly large programming project involving using someone else's code.
- To get a sense of what you learned — compare what you knew in January to what you know now.

Slide 11

Minute Essay

- How did the course compare to your expectations/goals? Did you learn what you hoped to learn?