Slide 1

Administrivia

- Reminder: Homework 7 design due today, code (and plots) Tuesday.
- Homework 8 (tidy up all loose ends in your code, make any improvements you care to) due the day of the final.

Slide 2

I/O In Java — Overview

- Abstract view — "file" is a collection of data. Java provides methods for sequential and "random" (non-sequential) access.
- Sequential file access is via "streams" — concept that applies to other kinds of sequential I/O (stdin/stdout, sockets, etc.).
- Stream — sequential flow of data.
  - Input streams connect program with an outside "source" (stdin, file, socket, etc.). (If data is characters, use "reader" instead.)
  - Output streams connect program with outside "destination". (If data is characters, use "writer" instead.)

Slide 3

Stream I/O

- I/O in Java often requires at least two classes:
  - One that connects to the desired source/destination (file, socket, array, string, etc.).
  - One that defines interface for program (character or binary data, byte-by-byte or a line at a time, etc.)
- Short examples:

```
BufferedReader rdr =
  new BufferedReader(new FileReader("in.txt"));
String s = rdr.readLine();

PrintWriter pw =
  new PrintWriter(new FileWriter("out.txt"));
pw.println("hello, world");
```

Slide 4

I/O and Exceptions

- Many I/O methods throw "checked" exceptions — which your code must explicitly do something about. Sensible but sometimes annoying.
- First example from previous page would not compile — either declare that the method it's in throws an `IOException`, or use a "try" block, e.g.,

```
try {
  BufferedReader rdr =
    new BufferedReader(new FileReader("in.txt"));
  String s = rdr.readLine();
}
catch (FileNotFoundException e) {
  System.err.println(e); // or better error message
}
catch (IOException e) {
  System.err.println(e); // or better error message
}
```

Slide 5

Character-Based Stream I/O

- Prior to Java 1.5, typical way to parse input was to read a line at a time and use `String` methods, `Integer.parseInt`, `Double.parseDouble`, etc. `StringTokenizer`, `StreamTokenizer` also sometimes useful.
- Now, `Scanner` class may do what you need. `split()` method of `String` class may also be useful.
- For output, `PrintWriter` methods will likely be useful. Notice that Java also has (as of 1.5) a `printf`!
- (Example — "almost an editor" program(s).)

Slide 6

Binary Stream I/O

- Can also read/write binary data:
  - `DataInputStream`, `DataOutputStream` to write out primitive types.
  - `ObjectInputStream`, `ObjectOutputStream` to write out primitives, `Serializable` objects.
- Object serialization:
  - Object and all referenced objects (except `static` and `transient` variables) are turned into sequential stream of bytes.
  - Can override `readObject`, `writeObject` to control what happens more precisely.
- (Example — "silly class" and saver.)

Slide 7

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 8

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 9

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 10

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. More complex example — chat program (next time).

Slide 11

Minute Essay

- None — sign in.