

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.
- Sample programs page updated (slightly more elaborate version of the binary search tree from last time, I/O and networking examples).

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.)

## Stream I/O

Slide 3

- 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");
```

## I/O and Exceptions

Slide 4

- 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  
}
```

### Character-Based Stream I/O

Slide 5

- 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).)

### Binary Stream I/O

Slide 6

- 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.)

## Networking Basics

Slide 7

- 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 IP or name.
  - A process, identified by “port number” (16 bits). 0 — 1023 are “well-known ports”, others available for applications.

## Networking Basics — TCP and UDP

Slide 8

- 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”.

## Networking in Java

Slide 9

- 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).
- (To be continued . . .)

## Networking in Java — Sockets

Slide 10

- 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).

## Minute Essay

- None — quiz.

Slide 11