Networking and Client-Server Programs

Programs That Talk to Programs

- so far –
  - input from a keyboard (and maybe a file)
  - output to the console (and maybe a file)
- now –
  - the Java tools – streams and sockets
- coming up –
  - clients and servers
  - a server with several clients at once
  - designing client-server communications
  - organizing client-server code

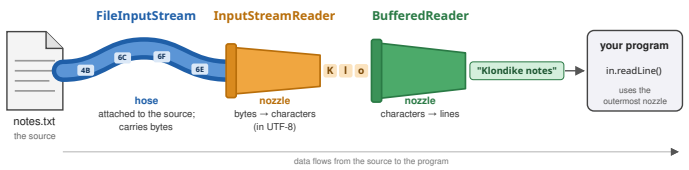
Streams

- data comes from many places and goes to many places
  - the keyboard and the screen
  - files
  - other programs over the network
- Java uses *streams* to hide the differences
  - a stream is a sequence of data flowing from a source to a destination
    - an *input stream* is read from
    - an *output stream* is read to
  - the same reading and writing code works whatever is at the other end

Working With Streams

- *start with a hose* attached to a source (input) or destination (output)
  - the constructor specifies the source/destination
- *add nozzles* as needed to convert what comes out or goes in
  - nozzle constructors take an existing stream
  - many nozzles are streams themselves – stackable
  - some classes do nozzle work without being streams themselves e.g. Scanner
- *use* the outermost nozzle to read or write
  - from a finite source such as a file, can detect the end of stream
- *close* when done (in most cases)
  - closing the outermost nozzle closes the whole chain
  - close a stream only if you created it

Hoses and Nozzles




```
BufferedReader in = new BufferedReader(          // nozzle: characters → lines
    new InputStreamReader(                       // nozzle: bytes → characters
        new FileInputStream("notes.txt"),        // hose: bytes from the file
        StandardCharsets.UTF_8));
```


Key Points – I/O

- types of streams
  - *byte streams* read and write data as raw bytes
    - Java uses the naming convention `InputStream` and `OutputStream` for byte streams
  - *character streams* read and write data as characters
    - Java uses the naming convention `Reader` and `Writer` for character streams

Key Points – I/O

- already-connected hoses (Java creates them)
  - standard input – `System.in` (type `InputStream`)
  - standard output – `System.out` (type `PrintStream`)
  - standard error – `System.err` (type `PrintStream`)
- creating hoses
  - for files: create instance of `FileInputStream`, `FileOutputStream`, `FileReader`, `FileWriter` classes
  - for URLs: `openStream()` method of the `URL` class
  - for network connections: `getInputStream()`, `getOutputStream()` methods of the `Socket` class

Key Points – I/O

- common nozzle classes
  - all `InputStream`/`OutputStream` classes provide methods to read/write single bytes and array of bytes
  - all `Reader`/`Writer` classes provide methods to read/write single characters
  - `PrintWriter` provides methods to write primitive types, `String`, and `Object`
  - `BufferedReader` provides a method to read a line of text
  - `DataInputStream`/`DataOutputStream` provide methods to read/write primitive types and `String`
  - `ObjectInputStream`/`ObjectOutputStream` provide methods to serialize objects
  - `InputStreamReader`/`OutputStreamWriter` support reading/writing characters over byte streams
- nozzle-like classes
  - `Scanner` provides `next()` and `nextX()` methods to parse character input

Questions

When would you use a `Scanner`? Aren't `Scanner` and `BufferedReader` pretty much the same?

- any `Reader` lets you read individual characters
- `BufferedReader` lets you read an entire line of text
- `Scanner` parses text into tokens
  - by default tokens consist of non-whitespace characters separated by whitespace
  - `next()` reads the next token as a `String`
  - `nextInt()`, `nextDouble()`, `nextBoolean()`, etc read the next token as an `int`, `double`, `boolean`, etc

Reading to the End of a File

- readLine() returns null at end of stream

```
BufferedReader in = null;
try {
  in = new BufferedReader(new InputStreamReader(new FileInputStream(filename),
                                                StandardCharsets.UTF_8));
  int lineNumber = 0;
  for ( ; ; ) {
    String line = in.readLine();
    if ( line == null ) {
      break;                          // end of file (end of stream)
    }
    lineNumber++;
    if ( line.contains(word) ) {
      System.out.println(lineNumber + ": " + line);
    }
  }
} catch ( FileNotFoundException e ) {
  System.out.println("can't open " + filename + ": " + e.getMessage());
} catch ( IOException e ) {
  System.out.println("problem reading " + filename + ": " + e.getMessage());
} finally {
  if ( in != null ) {
    try { in.close(); } catch ( IOException e ) {}   // closes the whole chain
  }
}
```