CS2136: Paradigms of Computation

Class 23:
Java I/O
Intro to Java Networking

Copyright 2001, 2002, 2003 Michael J. Ciaraldi and David Finkel
Java Input / Output
Multiple Uses for I/O

- Bytes
  - Could be:
    - Bytes in general
    - Text.
  - Could be
    - Keyboard & screen (basically text)
    - Named files
    - Standard input, output, and error (could be redirected).
  - Covered today.

- Graphics screen, keyboard, and mouse.
  - Swing

- Network
  - Sockets, RMI
Java Byte Input & Output

Output is easy

- For text, just use `print()`, `println()`, `toString()`.
- For others, use `write()` and its relatives.

Java was not strong on formatting.
- Improved in JDK 1.1

Input is harder.
- You don’t know the type of data beforehand.
There were changes between Java 1.0 and 1.1. Mostly to support I18n (internationalization / internationalisation) by using Unicode. You still need many of the old ways.
Basics of Byte I/O

There are basic I/O classes (which you don’t often use), with many subclasses.

- InputStream
- OutputStream

You create streams (or have them passed in from outside).

Then you wrap them by creating other objects.
**Layering**

- Each class can wrap the one within it.
- Maintains the same interface.
  - Lets you access consistently.
Output Classes

- PrintStream
- FileOutputStream
- FileOutputStream
- DataOutputStream
- BufferedOutputStream
Output Classes: PrintStream

- Writes formatted (text) output.
- Examples: System.out and System.err
- Constructors
  - PrintStream(OutputStream out, boolean autoFlush)
    - Flush after each newline: true/false.
  - PrintStream(OutputStream out)
    - autoFlush defaults to false.
- Use print() and println().
Output Classes: FileOutputStream

z Writes raw (unformatted) output.

z Constructors
  y FileOutputStream(String name)
  y FileOutputStream(String name, boolean append)
  y FileOutputStream(File file)
  y FileOutputStream(FileDescriptor fdObj)

z Use write().
  y Argument: a byte or bytes.
Output Classes: DataOutputStream

- Writes primitives, readable by DataInputStream

- Write using:
  - write(), writeByte(), writeBytes()  
  - writeBoolean()  
  - writeDouble()  
  - etc.
Output Classes: BufferedOutputStream

- Does not call the OS for each byte.
- Good for files.
OutputStream Methods

- flush()
  - Forces a write to underlying stream.

- close()
  - Flushes, then closes the stream.
Example (adapted from IOStreamDemo.java)

```java
PrintStream out1 =
    new PrintStream(
        new BufferedOutputStream(
            new FileOutputStream("IODemo.out")))
    .println(“Hello, file system!”);
out1.close(); // Always do this when done.
```
Example: MyIOutputStreamDemo.java (part 5)

// 5. Storing & recovering data
DataOutputStream out2 =
    new DataOutputStream(
        new BufferedOutputStream(
            new FileOutputStream("Data.txt")));
out2.writeBytes("Here's the value of pi: \n");
out2.writeDouble(3.14159);
out2.close();
Input Classes

- InputStream
- FileInputStream
- BufferedInputStream
- DataInputStream
Input Classes: InputStream

- Reads raw (unformatted, binary) input.
- Example: System.in
- No Constructors
  - All abstract.

- Methods:
  - read()
  - close()
  - available()
    - # bytes you could read.
Input Classes: FileInputStream

- Reads raw (unformatted) input.
- Constructors
  - FileInputStream(String name)
  - FileInputStream(File file)
  - FileInputStream(FileDescriptor fdObj)
- Use read().
  - Argument: a byte or bytes.
Input Classes: 
BufferedInputStream

- Reads raw (unformatted, binary) input.
- Constructors:
  - BufferedInputStream(InputStream in)
  - BufferedInputStream(InputStream in, int size)
- Other methods like InputStream.
Input Classes: DataInputStream

- Reads primitives, as written by DataOutputStream

- Constructor:
  - DataInputStream(InputStream in)

- Read using:
  - read(), readByte()
  - readBoolean()
  - readDouble(), etc.
  - readLine()
    - Deprecated, but still needed.
So How Do You Read Anything Useful?

- DataInputStream and DataOutputStream are good for raw data.
- DataInputStream only works for:
  - Individual bytes.
  - Primitives where you know the exact sequence.
So How Do You Read Anything Useful? (cont.)

- You can use StreamTokenizer to parse as you read the input.
- Or, read in a line with readLine() and parse the resulting string.
  - By yourself -OR-
  - Use StringTokenizer.
Parsinig Input Lines

You need to:

- Divide the input string into tokens.
  - Token = a sequence of characters treated as a unit.
- Process each token.

Recognizing tokens:

- Might be separated by delimiters.
- Might fit some pattern.
StringTokenizer

- Recognizes tokens by delimiters

- Constructors:
  - StringTokenizer(String str)
  - StringTokenizer(String str, String delim)
  - StringTokenizer(String str, String delim, boolean returnTokens)

- Defaults:
  - delim = " 	

- returnTokens = false (i.e. delimiters do not count as tokens).
StringTokenizer II

- Must set up for each string.

Methods:
- boolean hasMoreTokens() a.k.a. hasMoreElements()
- int countTokens()
- String nextToken()
- String nextToken(String delim)
  - updates delim
- Object nextElement()
  - Same as nextToken except for return type.
Now What?

- Need to parse the token.
- Methods for various types (all take a String):
  - parseFloat(), parseDouble()
  - parseByte(), parseShort, parseInt(), parseLong
- Throw NumberFormatException
Fixed Format I/O

z Input:
1) Read the line.
2) Chop into substrings.
3) Parse each substring.

z Output:
1) Use NumberFormat to convert numbers into formatted Strings.
2) Concatenate Strings.
3) Output the final String.
Java Networking
Java Networking

- Java can open sockets to other machines.
- Java has higher-level functions.
  - HTTP
  - HTML
- Access
  - Applications can connect to any machine.
  - Applets can only connect to the machine they downloaded from.
Internet Addressing

- Every host on the Internet has one or more numeric IP addresses.
  - Written as 130.215.24.65
  - Internally, a 32-bit integer (currently).
- A host can have one or more names.
  - ccc.wpi.edu
- Domain Name System (DNS)
  - On Unix: nslookup
Java Internet Addressing

z Class InetAddress
  y Holds an Internet (IP) address.
  y Internal structure not specified.
Static Methods in Class InetAddress

**z** `getByName()`

- `public static InetAddress getByName(String host)` throws `UnknownHostException`

- Host can be:
  - Name, e.g. “ccc.wpi.edu”
  - Address, e.g. “130.215.24.65”

- Returns a new `InetAddress` object.

**z** `getLocalHost()`

- `public static InetAddress getLocalHost()` throws `UnknownHostException`
Regular Methods in Class InetAddress

- `String getHostName()`
  - Returns, e.g. “ccc.wpi.edu”

- `String getHostAddress()`
  - Returns, e.g. “130.215.24.65”
import java.net.*;
public class WhoAmI {
    public static void main(String[] args) throws Exception {
        if(args.length != 1) {
            System.err.println("Usage: WhoAmI " + "MachineName");
            System.exit(1);
        }
        InetAddress a = InetAddress.getByName(args[0]);
        System.out.println(a);
    }
}
Connecting

- Each host has “ports” to connect to.
- Standard servers are waiting on “well-known ports”.
  - 7 = echo
  - 13 = daytime
  - 19 = chargen
  - 23 = telnet
  - 25 = SMTP
  - 80 = HTTP

- Socket = connection to port.
Connecting Methods

z Socket(InetAddress address, int port) throws IOException
   y Returns a Socket.

z void close()

z InputStream getInputStream() throws IOException

z OutputStream getOutputStream() throws IOException
More Socket Methods

- int getPort()
- int getLocalPort()
- InetAddress getInetAddress()
- InetAddress getLocalAddress()
Example:

MyDaytimeClient.java

String host; // Host name
int port; // Port number to connect to
int DAYTIME_PORT = 13;

if (args.length < 1) host = "localhost";
    else host = args[0];
if (args.length < 2) port = DAYTIME_PORT;
    else port = Integer.parseInt(args[1]);
InetAddress addr = InetAddress.getByName(host);
System.out.println("Connecting to " + addr);
Socket socket = new Socket(addr, port);
try {
    System.out.println("socket = " + socket);
    BufferedReader in =
        new BufferedReader(
            new InputStreamReader(
                socket.getInputStream()));
    // read and print input
    for(int i = 0; i < 2; i++) {
        String str = in.readLine(); System.out.println(str);
    }
} finally {} 
System.out.println("closing..."); socket.close();
Input and Output: MyEchoClient.java

```java
PrintWriter out =
    new PrintWriter(new BufferedWriter(
        new OutputStreamWriter(socket.getOutputStream())),true);
try {
    Thread.currentThread().sleep(1000);
} catch(InterruptedException e) {};
// read and print input
String str;
for(int i = 0; i < 10; i++) {
    out.println("Howdy " + i);
    str = in.readLine(); if (str != null) System.out.println(str);
}
```
Next Time

- Future of Java
- Wrapup: Programming paradigms