

Networking

Slides based partly on sun's java socket tutorial:

`java.sun.com/docs/books/tutorial`

Overview

- Networking basics
- Client/Server Model
- Telnet
- Sockets
- URLConnection Class

Networking Basics

Computers communicating over the internet use either:

- Transmission Control Protocol (TCP)
 - Like a phone call
 - Guarantees information arrives, and in correct order
 - Used e.g. for http
- User Datagram Protocol (UDP)
 - Like posting a series of letters
 - Sends a series of information *packets* called *datagrams*
 - No guarantee of delivery, or delivery order
 - Less overhead than TCP
 - Used e.g. for audio and video

Clients, Servers and Addresses

- Servers serve data to clients
- Clients request data from servers
- One computer can host many servers
 - E.g. web server and email server and...
- Computers identified by a 32-bit IP address
 - Can also be specified by text, e.g. www.cs.bris.ac.uk
= 137.222.102.8
- Each server binds to a port on its computer
 - Ports identified by a 16-bit number
 - Ports 0-1023 are reserved; your apps should not bind to them

Client/server connection

- Server started on a machine
- Binds to specified port
- Waits and listens for a client to connect
- Client contacts server
 - New port is agreed for further use by this client
 - Original port still available for new clients
- Client requests data, server serves
- Disconnection
- Server waits for next client
 - Often threads are used to allow server to connect to multiple clients simultaneously

Telnet Client and Echo Server

- Widely available application
 - Useful for testing servers
 - Sometimes replaced by SSH (secure shell)
- Try this at unix prompt:

```
telnet www.cs.bris.ac.uk 7
```
- Connects to port 7 on machine above
- Port 7 is the *echo server*
 - Echo server just repeats strings you send it
 - Found on most machines. Already running.
 - Press ctrl-] then enter to exit echo server
- Type “quit” to exit telnet

Sockets

- A low-level approach to client/server communications
- Implemented on top of (hides) TCP
- Client and server each have a socket representing the connection
- `java.net` provides `Socket` and `ServerSocket` classes
- Sockets look like files: streams of data

EchoClient.java

- Simple client using a socket to connect to Echo server
 - Duplicates telnet connection to echo server
- Reads text from standard input, writes it to socket
- Echo server echoes text back through socket
- EchoClient reads text from socket and prints it
- Ctrl-c to quit

Generic Process

- Open a socket
- Open input and output streams of socket
- Read and write to streams using server's protocol
- Close the streams
- Close the socket

```
import java.io.*;
import java.net.*;

public class EchoClient {
    public static void main(String[] args) throws IOException
    {
        final int ECHO = 7; // echo server on port 7
        Socket echoSocket = null;
        PrintWriter out = null;
        BufferedReader in = null;
        try {
            echoSocket = new Socket("www.cs.bris.ac.uk", ECHO);
            out = new PrintWriter(echoSocket.getOutputStream(),
                true);
            in = new BufferedReader(new InputStreamReader(
                echoSocket.getInputStream()));
        } catch (UnknownHostException e) {
            System.err.println("Don't know about host");
            System.exit(1);
        } catch (IOException e) {
            System.err.println("Couldn't get I/O");
            System.exit(1);
        }
    }
}
```

```
BufferedReader stdIn = new BufferedReader(  
    new InputStreamReader(System.in));  
String userInput;  
while ((userInput = stdIn.readLine()) != null) {  
    out.println(userInput);  
    System.out.println("echo: " + in.readLine());  
}  
out.close();  
in.close();  
stdIn.close();  
echoSocket.close();  
}  
}
```

Client/Server Example

3 classes:

- **BankServer**
 - Sets up ServerSocket
 - ServerSocket generates Sockets
 - Passes Sockets to BankService
- **BankService**
 - Communicates with client
 - Gives instructions to Bank
 - Threaded version would have many BankServices and sockets, each pair in a thread
- **Bank**
 - Knows about bank accounts

Server

- **Wait for client on a certain port:**

```
ServerSocket server = new  
ServerSocket(8888);  
Socket s = server.accept();
```

- **Use reader and writer to exchange text with client**

```
BufferedReader in = new BufferedReader(new  
    InputStreamReader(s.getInputStream()));  
PrintWriter out = new  
PrintWriter(s.getOutputStream());  
String line = in.readLine();
```

- **Use StringTokenizer to parse client's commands**

Application Protocol

Client Request	Server Response
BALANCE <i>acc</i>	Balance for account <i>acc</i>
DEPOSIT <i>acc sum</i>	New balance
WITHDRAW <i>acc sum</i>	New balance
QUIT	None

Connecting to Bank Server

- Start the server:

```
java BankServer
```

- Start the client in another window:

```
telnet localhost 8888
```

- localhost means “on this machine”
 - Need not be in same dir. as BankServer
- Type commands following protocol
 - Note: no error checking!
- Ctrl-C to shut server down when done

URLConnection Class

- Sockets fairly low-level
- URLConnection hides sockets
- Useful for HTTP and FTP

Construct URL object (Uniform Resource Locator)

- Create URLConnection using openConnection
- Call getInputStream
- Wrap it up in a BufferedReader and use ReadLine

URL Connection Example

```
URL u = new URL("
    http://www.cs.bris.ac.uk/index.html");
URLConnection connection = u.openConnection();
InputStream in = connection.getInputStream();
BufferedReader in = new BufferedReader(new
    InputStreamReader(in));
boolean done = false
while (!done)
{
    String input = reader.readLine();
    if (input == null) done = true;
    else { do something with input }
}
```