Sockets
Sockets

- Two common transport level protocols
  - UDP
  - TCP
- Both facilitate communication:
  - From a port at one IP address
  - To a port on another IP address
- The communication metaphor between the two ports is called a “socket”.
- E.g.
  - A UDP socket
    - from 128.2.25.11 port 5443
    - to 204.100.2.1 port 4568
    - Commonly noted 204.100.2.1:4568
- E.g.
  - A TCP socket
    - Between 128.2.25.22:4567
    - And 204.100.2.2:5432
Sockets

- Are a standard transport-level communication metaphor
- Both TCP and UDP use a socket abstraction

<table>
<thead>
<tr>
<th>Operation</th>
<th>Description</th>
<th>Client</th>
<th>Server</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create</td>
<td>Create a socket</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Bind</td>
<td>Associate a socket with a local IP address &amp; port</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Listen</td>
<td>Bound socket ready for connections</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Connect</td>
<td>Assign a local port and connect to remote port</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Accept</td>
<td>Accept a connection on listened-to socket</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Read, Write</td>
<td>Read and write data to the socket</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Close</td>
<td>Close the socket, terminate communication</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>
Echo Client / Server

• A simple echo client / server app.

• Client:
  – reads a line from the terminal
  – sends the line to the server via a socket
  – reads a line from the server via a socket
  – prints the line to the console

• Server:
  – reads a line from a socket
  – writes back (echoes) the line to the client via the socket
import java.net.*;
import java.io.*;

public class TCPClient {
    public static void main(String args[]) {
        Socket clientSocket = null;
        try {
            int serverPort = 7777;
            clientSocket = new Socket(args[0], serverPort);
            BufferedReader in = new BufferedReader(new InputStreamReader(clientSocket.getInputStream()));
            PrintWriter out = new PrintWriter(new BufferedWriter(new OutputStreamWriter(clientSocket.getOutputStream())));
            BufferedReader typed = new BufferedReader(new InputStreamReader(System.in));
            String m;
            while (((m = typed.readLine()) != null) {
                out.println(m);
                out.flush();
                String data = in.readLine(); // read a line of data from the stream
                System.out.println("Received: " + data);
            }
        } catch (IOException e) {
            System.out.println("IO Exception:" + e.getMessage());
        } finally {
            try {
                if (clientSocket != null) {
                    clientSocket.close();
                }
            } catch (IOException e) {
                // ignore exception on close
            }
        }
    }
}
import java.net.*;
import java.io.*;
import java.util.Scanner;

public class TCPServer {
    public static void main(String args[]) {
        Socket clientSocket = null;
        try {
            int serverPort = 7777; // the server port
            ServerSocket listenSocket = new ServerSocket(serverPort);
            clientSocket = listenSocket.accept();
            Scanner in;
            PrintWriter out;
            in = new Scanner(clientSocket.getInputStream());
            out = new PrintWriter(new BufferedWriter(new OutputStreamWriter(clientSocket.getOutputStream())));
            while (true) {
                String data = in.nextLine();
                System.out.println("Echoing: "+ data);
                out.println(data);
                out.flush();
            }
        } catch (IOException e) {
            System.out.println("IO Exception:" + e.getMessage());
        } finally {
            try {
                if (clientSocket != null) {
                    clientSocket.close();
                }
            } catch (IOException e) {
                // ignore exception on close
            }
        }
    }
}
<?
$socket = socket_create(AF_INET, SOCK_STREAM, SOL_TCP);
$result = socket_bind($socket, "localhost", 7777);

while (true):
    $result = socket_listen($socket, 3);
    $spawn = socket_accept($socket);
    $input = socket_read($spawn, 1024);
    print "Message: $input\n";
    $output = $input;
    socket_write($spawn, $output, strlen ($output));
    socket_close($spawn);
endwhile;
socket_close($socket);
?>
import socket

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.bind(('localhost', 7777))
s.listen(1)

conn, addr = s.accept()

while 1:
    data = conn.recv(20) # 20 is buffer size, typically larger
    if not data: break
    print "Echoing:", data
    conn.send(data) # echo
conn.close()
```
var net = require('net');

var server = net.createServer(function(socket) {
    socket.on('data', function(data) {
        dataReceived(socket, data);
    });
});

server.listen(7777);

function dataReceived(socket, data) {
    data = data.toString('utf8');
    console.log("Message: "+data);
    socket.write(data);
}

server.close();
```
import java.net.*;
import java.io.*;
public class EchoServerUDP{
    public static void main(String args[]){
        DatagramSocket aSocket = null;
        byte[] buffer = new byte[1000];
        try{
            aSocket = new DatagramSocket(6789);
            DatagramPacket request = new DatagramPacket(buffer, buffer.length);
            while(true){
                aSocket.receive(request);
                DatagramPacket reply = new DatagramPacket(request.getData(),
                    request.getLength(), request.getAddress(), request.getPort());
                String requestString = new String(request.getData());
                System.out.println("Echoing: "+requestString);
                aSocket.send(reply);
            }
        }catch (SocketException e){System.out.println("Socket: "+e.getMessage());
        }catch (IOException e) {System.out.println("IO: "+e.getMessage());
        }finally {if(aSocket != null) aSocket.close();
    }
}
import java.net.*;
import java.io.*;
public class EchoClientUDP{
    public static void main(String args[]){
        // args give message contents and server hostname
        DatagramSocket aSocket = null;
        try {
            InetAddress aHost = InetAddress.getByName(args[0]);
            int serverPort = 6789;
            aSocket = new DatagramSocket();
            String nextLine;
            BufferedReader typed = new BufferedReader(new InputStreamReader(System.in));
            while ((nextLine = typed.readLine()) != null) {
                byte[] m = nextLine.getBytes();
                DatagramPacket request = new DatagramPacket(m, m.length, aHost, serverPort);
                aSocket.send(request);
                byte[] buffer = new byte[1000];
                DatagramPacket reply = new DatagramPacket(buffer, buffer.length);
                aSocket.receive(reply);
                System.out.println("Reply: " + new String(reply.getData()));
            }
        }catch (SocketException e) {System.out.println("Socket: " + e.getMessage());
        }catch (IOException e){System.out.println("IO: " + e.getMessage());
        }finally {if(aSocket != null) aSocket.close();}
    }
}
**HTML5 – WebSockets**

- A new browser capability in HTML5
- Browser and server communicate via a TCP socket
  - (WebSocket is not a socket. It is a protocol over a TCP socket.)
- Allows two-way, asynchronous communication
- Starts with HTTP handshake
  - Connection is *upgraded* to WebSockets
  - Uses UTF-8 encoding
  - Each text frame
    - Starts with a 0x00 byte
    - Ends with a 0xFF byte
  - No binary data
var myWS = new WebSocket("ws://www.websocket.org");
myWS.onopen = function(evt) { alert("Connection open ..."); }; 
myWS.onmessage = function(evt) { alert("Received Message: " + evt.data); }; 
myWS.onclose = function(evt) { alert("Connection closed."); }; 
myWS.send("Hello Web Socket!"); 
myWS.close();

This is not directly interoperable with our other servers. 
This code would execute within a browser.