Socket Programming in Java

CIS 421 Web-based Java Programming

Socket Programming

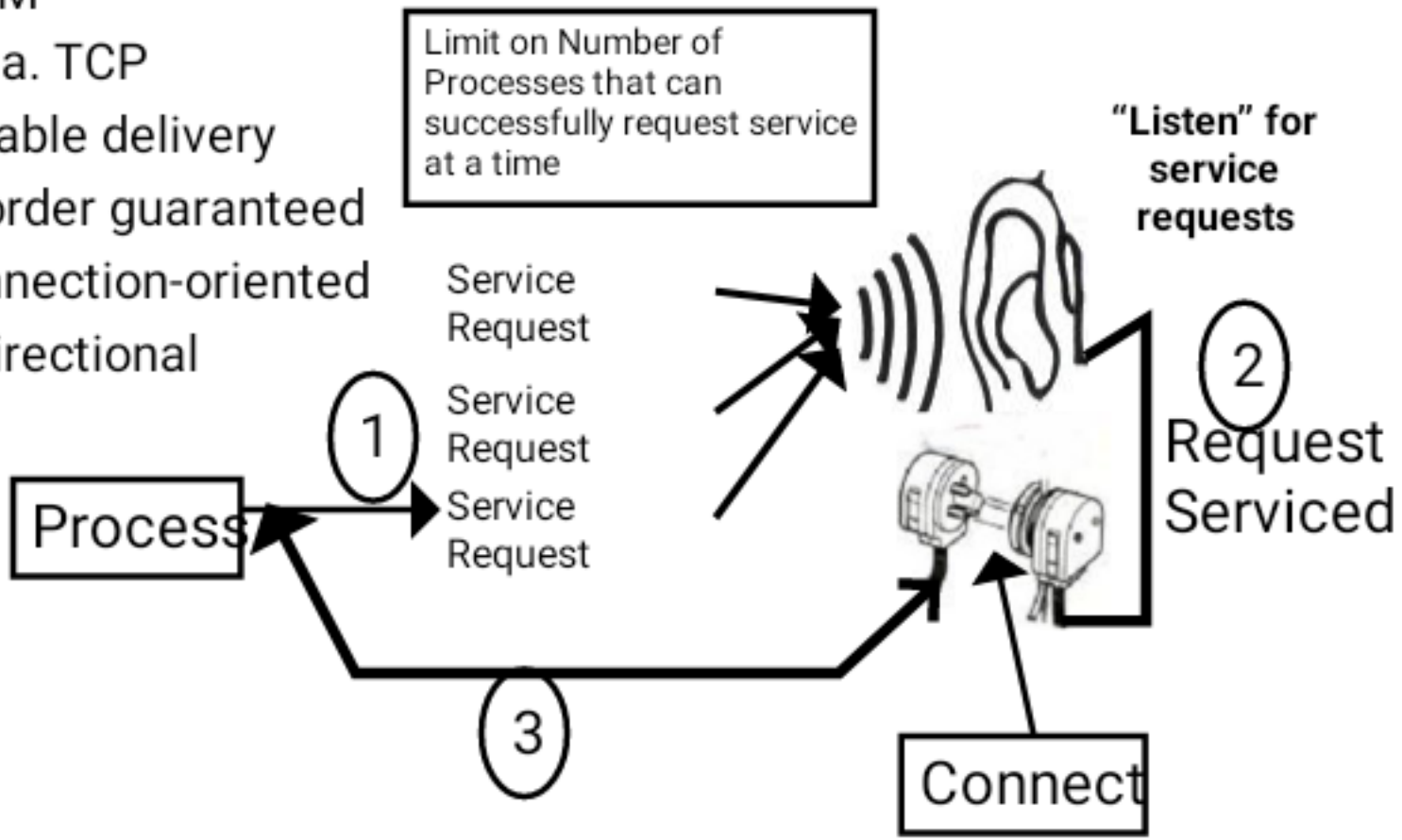
- What is a socket?
- Using sockets
  - Types (Protocols)
  - Associated functions
  - Styles

What is a socket?

- An interface between application and network
  - The application creates a socket
  - The socket *type* dictates the style of communication
    - reliable vs. best effort
    - connection-oriented vs. connectionless
- Once configured, the application can
  - pass data to the socket for network transmission
  - receive data from the socket (transmitted through the network by some other host)

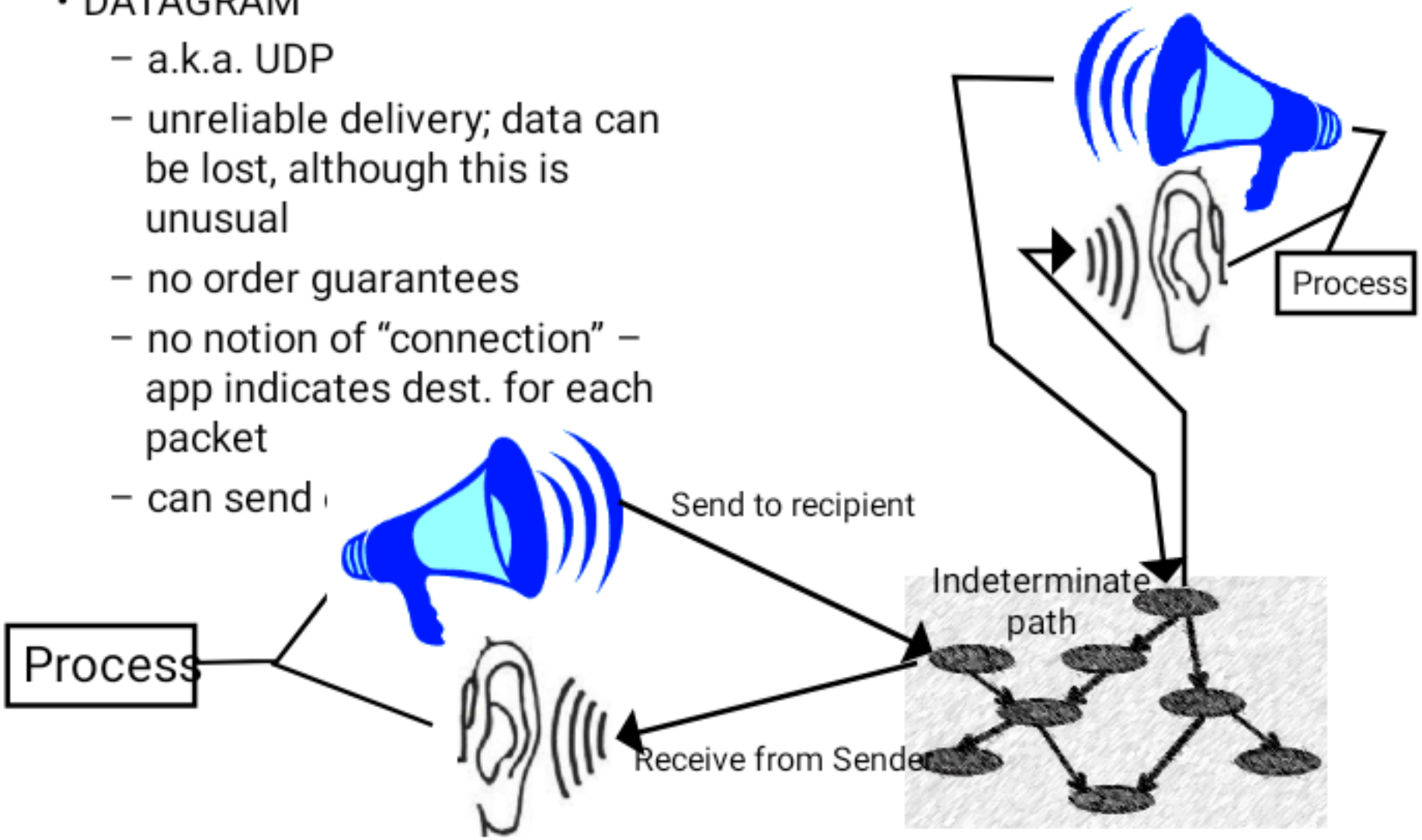
Two essential types of sockets

- STREAM
  - a.k.a. TCP
  - reliable delivery
  - in-order guaranteed
  - connection-oriented
  - bidirectional



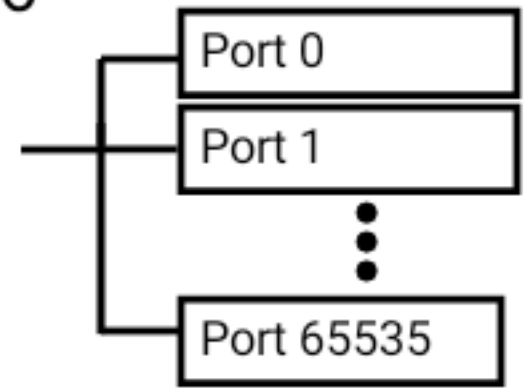
Two essential types of sockets

- DATAGRAM
  - a.k.a. UDP
  - unreliable delivery; data can be lost, although this is unusual
  - no order guarantees
  - no notion of "connection" – app indicates dest. for each packet
  - can send



Ports

- Each host has 65,536 ports
- Some ports are *reserved for specific apps*
  - 20,21: FTP
  - 23: Telnet
  - 80: HTTP



r A socket provides an interface to send data to/from the network through a port

Objectives

- The InetAddress Class
- Using sockets
  - TCP sockets
  - Datagram Sockets

Classes in java.net

- The core package java.net contains a number of classes that allow programmers to carry out network programming
  - ContentHandler
  - DatagramPacket
  - DatagramSocket
  - DatagramSocketImplHttpURLConnection
  - InetAddress
  - MulticastSocket
  - ServerSocket
  - Socket
  - SocketImpl
  - URL
  - URLConnection
  - URLEncoder
  - URLStreamHandler

Exceptions in Java

- BindException
- ConnectException
- MalformedURLException
- NoRouteToHostException
- ProtocolException
- SocketException
- UnknownHostException
- UnknownServiceException

The InetAddress Class

- Handles Internet addresses both as host names and as IP addresses
- Static Method getByName returns the IP address of a specified host name as an InetAddress object
- Methods for address/name conversion:

```
public static InetAddress   getByName(String host) throws UnknownHostException
public static InetAddress[] getAllByName(String host) throws UnknownHostException
public static InetAddress   getLocalHost() throws UnknownHostException

public boolean isMulticastAddress()
public String  getHostName()
public byte[]  getAddress()
public String  getHostAddress()
public int     hashCode()
public boolean equals(Object obj)
public String  toString()
```

Find an IP Address: IPFinder.java

```
// File: IPFinder.java
// Get the IP address of a host

import java.net.*;
import java.io.*;
import javax.swing.*;


public class IPFinder
{
 public static void main(String[] args) throws IOException
 { String host;
  host = JOptionPane.showInputDialog("Please input the server's name");
  try
  {InetAddress address = InetAddress.getByName(host);
    JOptionPane.showMessageDialog(null,"IP address: " + address.toString());

  }
      catch (UnknownHostException e)
      {JOptionPane.showMessageDialog(null,"Could not find " + host);
  }
 }
}
```

Retrieving the current machine's address

```
import java.net.*;

public class LocalIP
{
    public static void main(String[] args)
     {
 try
 {
                InetAddress address =
InetAddress.getLocalHost();
         System.out.println (address);
        }
        catch (UnknownHostException e)
 {
                System.out.println("Could not find local
address!");
        }
```

The Java.net.Socket Class

- Connection is accomplished via construction.
  - Each Socket object is associated with exactly one remote host.
    - To connect to a different host, you must create a new Socket object.

public Socket(String host, int port) throws UnknownHostException, IOException

- connect to specified host/port

public Socket(InetAddress address, int port) throws IOException

- connect to specied IP address/port

public Socket(String host, int port, InetAddress localAddress, int localPort) throws IOException

- connect to specified host/port and bind to specified local address/port

public Socket(InetAddress address, int port, InetAddress localAddress, int localPort) throws IOException

- connect to specified IP address/port and bind to specified local address/port

- Sending and receiving data is accomplished with output and input streams. There are methods to get an input stream for a socket and an output stream for the socket.

public InputStream getInputStream() throws IOException
public OutputStream getOutputStream() throws IOException

- To close a socket:

public void close() throws IOException

The Java.net.ServerSocket Class

- The *java.net.ServerSocket* class represents a server socket. It is constructed on a particular port. Then it calls accept() to listen for incoming connections.
  - accept() blocks until a connection is detected.
  - Then accept() returns a java.net.Socket object that is used to perform the actual communication with the client.
    - the "plug"
  - backlog is the maximum size of the queue of connection requests

```
public ServerSocket(int port) throws IOException
public ServerSocket(int port, int backlog) throws IOException
public ServerSocket(int port, int backlog, InetAddress bindAddr) throws IOException

public Socket accept() throws IOException
public void close() throws IOException
```

TCP Sockets

Example: **SocketThrdServer.java**
SERVER:
Create a ServerSocket object

- *ServerSocket servSocket = new ServerSocket(1234);*
  Put the server into a waiting state
- *Socket link = servSocket.accept();*
  Set up input and output streams
  use thread to serve this client via *link*
  Send and receive data
- *out.println(awaiting data…);*
- *String input = in.readLine();*
  Close the connection
- *link.close()*

Set up input and output streams

- Once a socket has connected you send data to the server via an output stream. You receive data from the server via an input stream.
  Methods *getInputStream* and *getOutputStream* of class *Socket:*

```
BufferedReader in =
new BufferedReader(
    new InputStreamReader(link.getInputStream()));
PrintWriter out =
new PrintWriter(link.getOutputStream(),true);
```

TCP Sockets

Example: **SocketClient.java**
CLIENT:
Establish a connection to the server

- *Socket link =*
- *new Socket(<server>,<port>);*
  Set up input and output streams
  Send and receive data
  Close the connection

The UDP classes

- 2 classes:

  java.net.DatagramSocket class

  is a connection to a port that does the sending and receiving. Unlike TCP sockets, there is no distinction between a UDP socket and a UDP server socket. Also unlike TCP sockets, a DatagramSocket can send to multiple, different addresses.The address to which data goes is stored in the packet, not in the socket.

  - *public DatagramSocket() throws SocketException*
    public DatagramSocket(int port) throws SocketException
    public DatagramSocket(int port, InetAddress laddr) throws SocketException

    java.net.DatagramPacket class

    is a wrapper for an array of bytes from which data will be sent or into which data will be received. It also contains the address and port to which the packet will be sent.

  - *public DatagramPacket(byte[] data, int length)*

public DatagramPacket(byte[] data, int length, InetAddress host, int port)

  - No distinction between server and client sockets

Datagram Sockets

Example: **UDPListener.java**
SERVER:
Create a DatagramSocket object

- *DatagramSocket dgramSocket =*
- *new DatagramSocket(1234);*
  Create a buffer for incoming datagrams
- *byte[] buffer = new byte[256];*
  Create a *DatagramPacket* object for the incoming datagram
  DatagramPacket inPacket =
  new DatagramPacket(buffer, buffer.length);
  Accept an incoming datagram
- *dgramSocket.receive(inPacket)*

Datagram Sockets

SERVER:
Accept the sender's address and port from the packet

- *InetAddress clientAddress = inPacket.getAddress();*
- *int clientPort = inPacket.getPort();*
  Retrieve the data from the buffer
- *string message =*
- *new String(inPacket.getData(), 0, inPacket.getLength());*
  Create the response datagram
  DatagramPacket outPacket =
  new DatagramPacket(
  response.getBytes(), response.length(),
  clientAddress, clientPort);
  Send the response datagram
- *dgramSocket.send(outPacket)*
- Close the *DatagramSocket: dgram.close();*

Datagram Sockets

Example: **UDPTalk.java**
CLIENT:
Create a DatagramSocket object

- *DatagramSocket dgramSocket = new DatagramSocket;*
  Create the outgoing datagram
- *DatagramPacket outPacket =*
  new DatagramPacket(message.getBytes(), message.length(),host, port);
  Send the datagram message
- *dgramSocket.send(outPacket)*
  Create a buffer for incoming datagrams
- *byte[] buffer = new byte[256];*

Datagram Sockets

CLIENT:

Create a *DatagramPacket* object for the incoming datagram

DatagramPacket inPacket =

new DatagramPacket(buffer, buffer.length);

Accept an incoming datagram

- *dgramSocket.receive(inPacket)*

  Retrieve the data from the buffer

- *string response = new String(inPacket.getData(), 0, inPacket.getLength());*

  Close the *DatagramSocket:*

- *dgram.close();*

Handling Data

- Data arrives/is sent as byte array
  - To send int
    - Convert to string (construct String from it)
    - use getBytes() to convert to byte[] and send
  - Receive int
    - Convert byte[] to String
    - use Integer.ParseInt() to convert to Integer