Socket Programming in JAVA

Mohammed Abdalla Youssif

OUTLINE

- **BACKGROUND**
- **What Is a Socket?**
- **Ports**
- **Two essential types of sockets**
- **Network Exceptions**
- **Classes in Network programming**
- **Set up input and output streams**
- **TCP Sockets**
- **Socket Client**
- **Socket Server**
- **How do I create an input stream?**
- **How do I close sockets?**

BACKGROUND

- The communication that occurs between the client and the server must be reliable.
- ***reliable*** means That is, no data can be dropped and must arrive on the client side in the same order in which the server sent it.
- TCP provides a reliable, point-to-point communication channel that client-server applications.

What Is a Socket?

- A **socket** is one end-point of a two-way communication link between two programs running on the network.
- **Socket classes** are used to represent the connection between a client program and a server program.
- The ***java.net*** package provides two classes :
  - **Socket** --that implement the client side of the connection
  - ***ServerSocket*** implement server side of the connection, respectively.
-

What Is a Socket?

- A **socket** is bound to a port number so that the TCP layer can identify the application that data is destined to be sent to.
- Normally, a **server** runs on a **specific computer** and has a socket that is bound to a specific port number.
- The **server** just waits, listening to the socket for a client to make a connection request.
- **On the client-side:** The client knows the hostname of the machine on which the server is running and the port number on which the server is listening.

What Is a Socket?

- To make a connection request, the client tries to connect with the server on the server's machine and port.
- The client also needs to identify itself to the server so it binds to a local port number that it will use during this connection.
-

What Is a Socket?

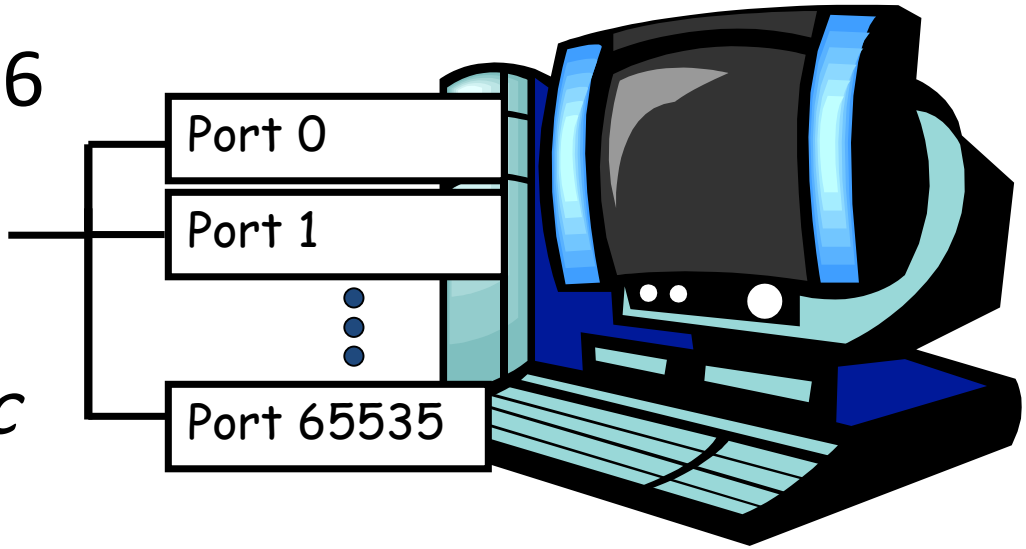
- If everything goes well, the server **accepts** the connection.
- Upon acceptance, the server gets a new socket bound to the same local port and also has its remote endpoint set to the address and port of the client.
- **On the client side**, if the connection is **accepted**, a socket is successfully created and the client can use the socket to communicate with the server.
-

What Is a Socket?

- The client and server can now communicate by writing to or reading from their sockets.
- An **endpoint** is a combination of an IP address and a port number.
- Every **TCP connection** can be uniquely identified by its two endpoints.
- That way you can have multiple connections between your host and the server.

Ports

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



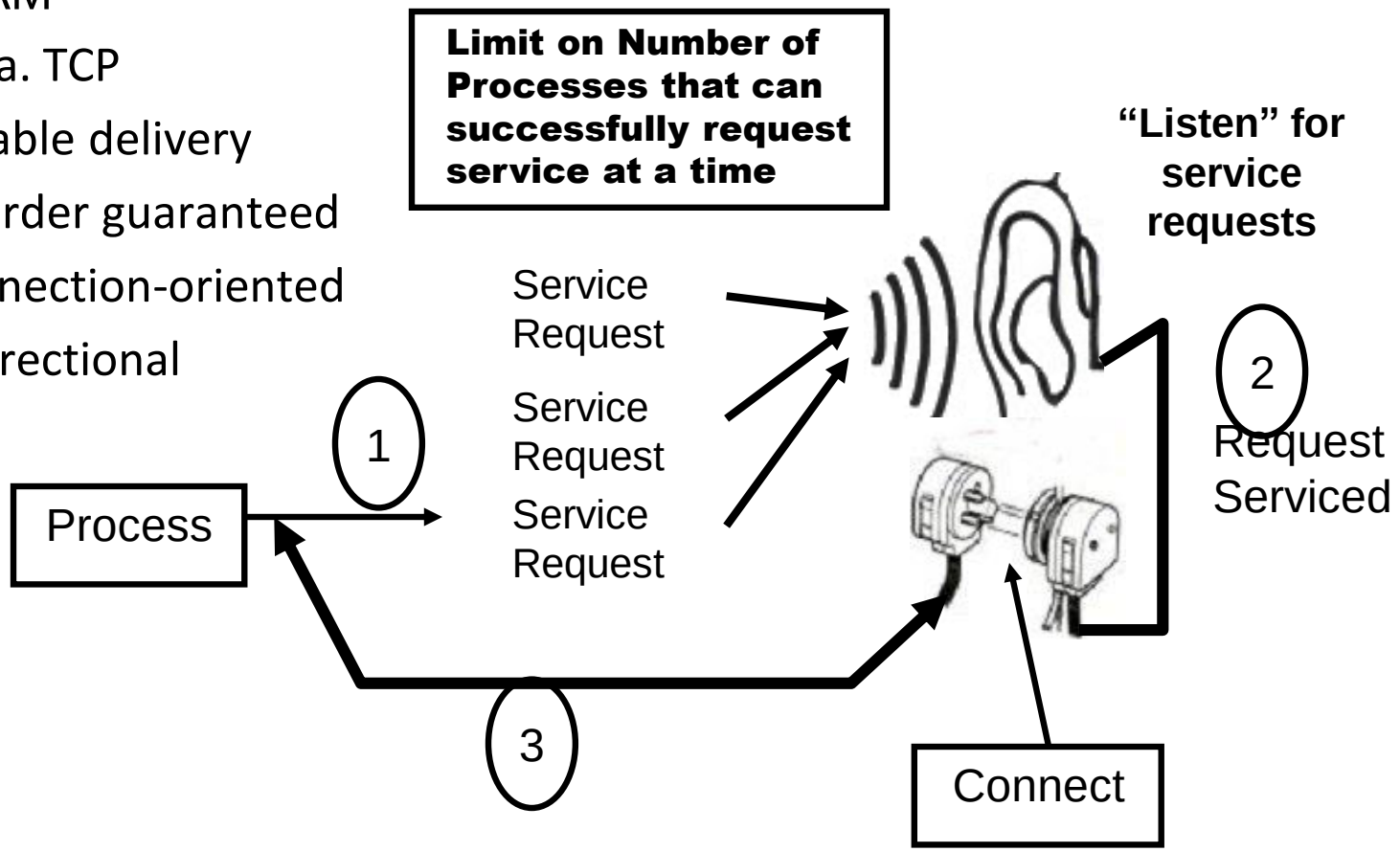
- A socket provides an interface to send data to/from the network through a port

What is a Port? A Port Number?

- Port numbers are used to identify services on a host
- – Port numbers can be:
  - well-known
  - dynamic or private
  - Clients usually use dynamic ports

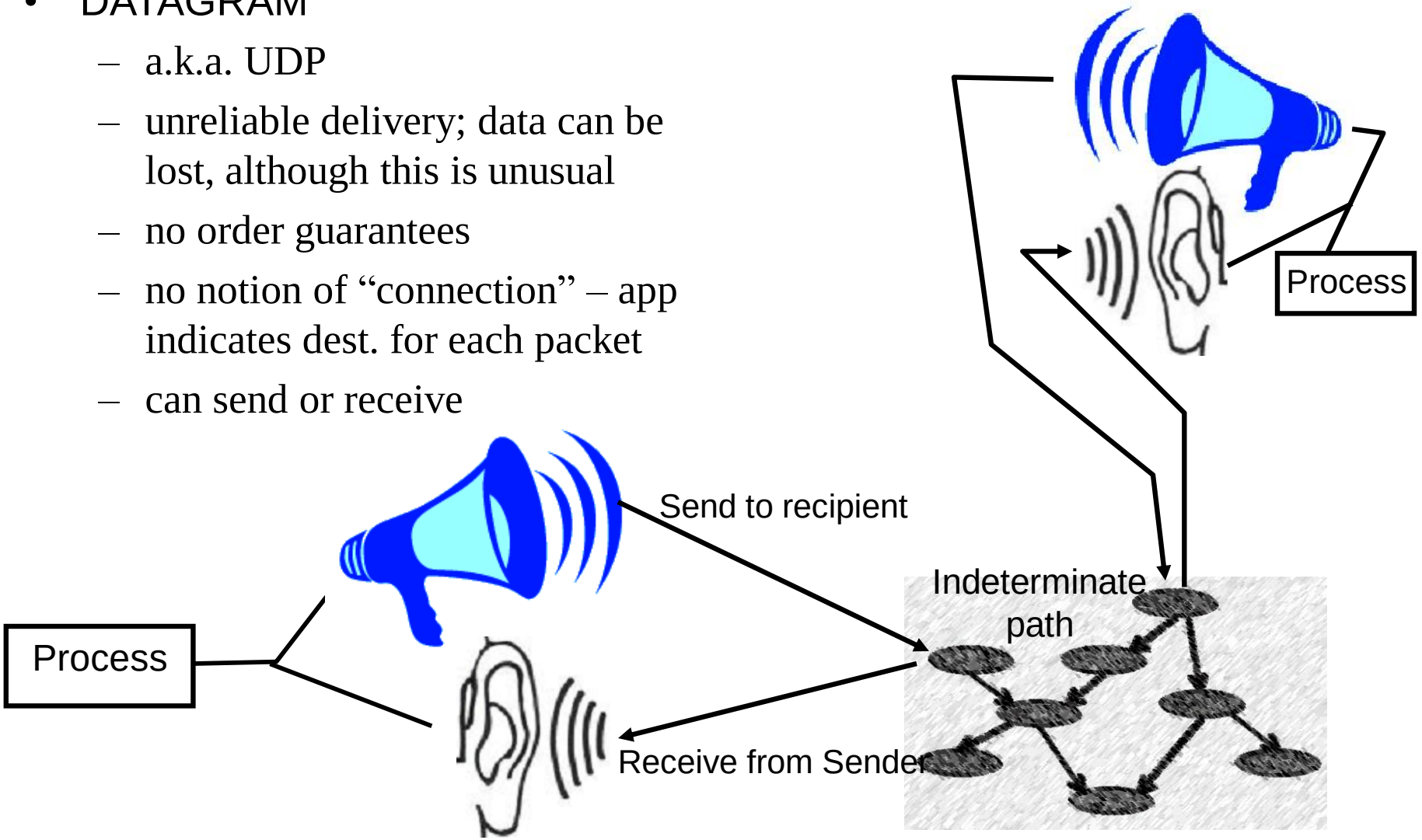
Two essential types of sockets

- 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 or receive



TCP Vs. UDP Socket programming

- Sockets are a protocol independent method of creating a connection between processes. Sockets can be either
- **Connection based or Connectionless:** Is a connection established before communication or does each packet describe the destination?
- **Packet based or Streams based:** Are there message boundaries or is it one stream?
- **Reliable or Unreliable:** Can messages be lost, duplicated, reordered, or corrupted?

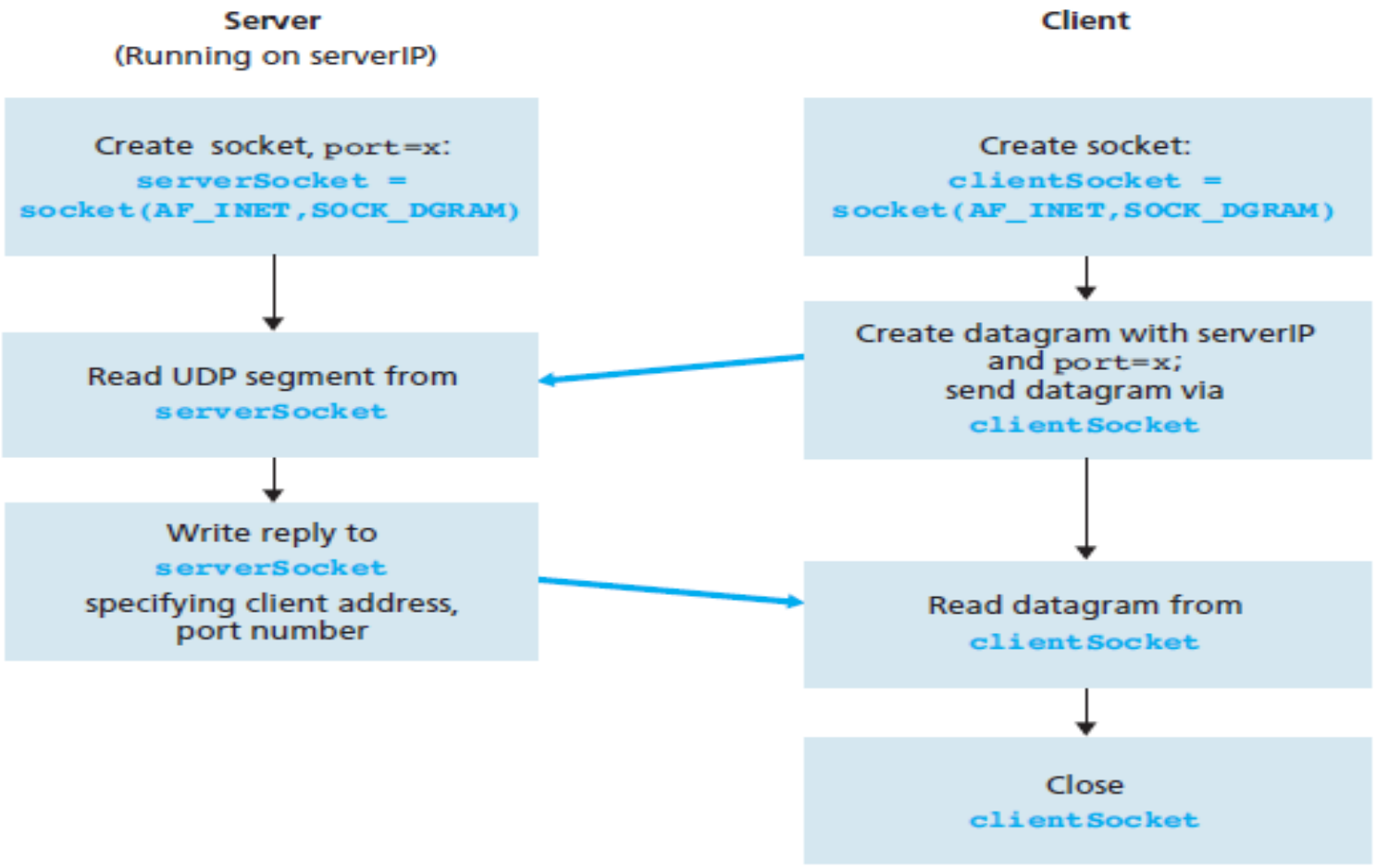
TCP Vs. UDP Socket programming

- **SOCK_STREAM**
  - TCP
  - connection-oriented
  - reliable delivery
  - in-order guaranteed
  - bidirectional

Two essential types of sockets

- **SOCK_DGRAM**
  - UDP
  - no notion of “connection” – app
  - indicates dest. for each packet
  - unreliable delivery
  - no order guarantees
  - can send or receive

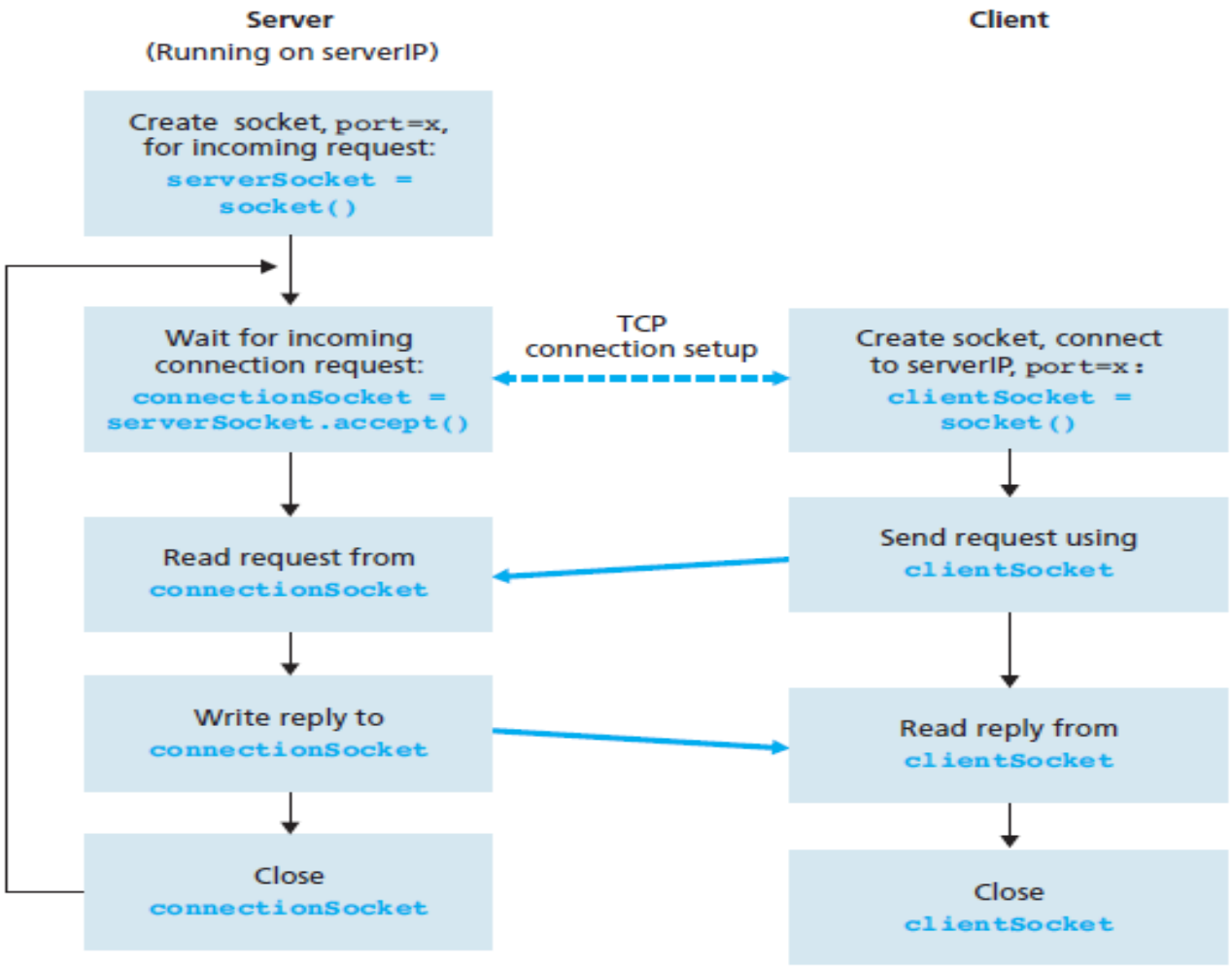
UDP Connection



TCP

- Unlike UDP, TCP is a connection-oriented protocol. This means that before the client and server can start to send data to each other, they first need to handshake and establish a TCP connection.
- When creating the **TCP connection**, we associate with it **the client socket address (IP address and port number)** and the server socket address (IP address and port number).
- With the **TCP connection established**, when one side wants to send data to the other side, it just drops the data into the TCP connection via its socket. This is different from UDP, for which the server must attach a destination address to the packet before dropping it into the socket.

TCP Connection



Network Exceptions in Java

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

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

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()
```

The Java.net.Socket Class

- Connection is accomplished via construction.
  - Each Socket object is associated with exactly one remote host.
- 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.

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
```

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:

1. Establish a connection to the server
   Socket link =
   new Socket(<server>,<port>);
2. Set up input and output streams
3. Send and receive data
4. Close the connection

Socket Client

```
Socket MyClient;
MyClient = new Socket("Machine name",
PortNumber);
```

- Where Machine name is the machine you are trying to open a connection to, and PortNumber is the port (a number) on which the server you are trying to connect to is running.

TCP Sockets

Example: **SocketServer.java**

SERVER:

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

Socket Server

- If you are programming a server, then this is how you open a socket:

```
ServerSocket MyService;
try {
   MyServerice = new ServerSocket(PortNumber);
    }
    catch (IOException e) {
       System.out.println(e);
    }
```

Socket Server

- When implementing a server you also need to create a socket object from the ***ServerSocket*** in order to listen for and accept connections from clients.

```
Socket clientSocket = null;
try {
    serviceSocket = MyService.accept();
     }
catch (IOException e) {
    System.out.println(e);
}
```

How do I create an input stream?

- On the client side, you can use the **DataInputStream** class to create an input stream to receive response from the server:

```
DataInputStream input;
try {
    input = new DataInputStream(MyClient.getInputStream());
}
catch (IOException e) {
    System.out.println(e);
}
```

How do I close sockets?

- You should always close the output and input stream before you close the socket.
- **On the client side:**

```
try {
        output.close();
        input.close();
    MyClient.close();
}
catch (IOException e) {
    System.out.println(e);
}
```

How do I close sockets?

- **On the server side:**

```
try {
    output.close();
    input.close();
    serviceSocket.close();
    MyService.close();
}
catch (IOException e) {
    System.out.println(e);
}
```

Questions

Project

- Create new message system to employs.
- System has the ability to broadcast the message.
- System has the ability to multicast the message

REFERNCES

1. **Comparison between TCP sockets and UDP Sockets**
   http://www.cyberciti.biz/faq/key-differences-between-tcp-and-udp-protocols/