CHAPTER FIVE

Network Programming

Outline

- Introduction
- Connecting to a server
- Implementing Servers in Java
- Sockets, ports, URIs

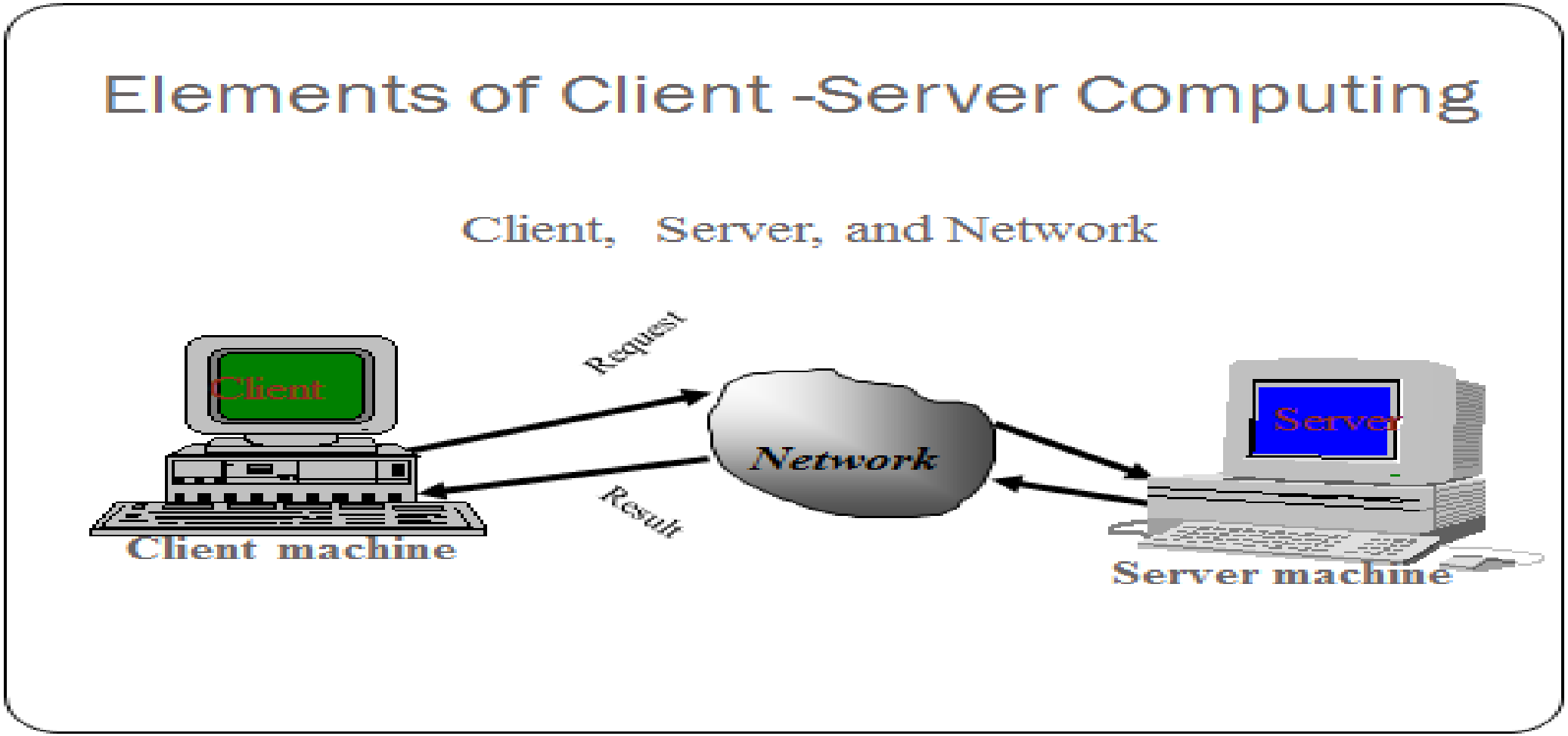
Introduction

- Internet and WWW have emerged as global media for communication everywhere and changed the way we conduct science, engineering, and commerce.
- They are also changing the way we learn, live, enjoy , communicate, interact, engage, etc.
- The Internet is all about connecting machines together.
- One of the most exciting aspects of Java is that it incorporates an easy-to-use, cross-platform model for network communications that makes it possible to learn network programming without years of study.

What Is a Network?

- A network is a collection of devices that share a common communication protocol and a common communication medium (such as network cables, dial-up connections, and wireless links).
- The term **Network programming** refers to writing programs that execute across multiple devices (computers) in which the devices are all connected to each other using a **network**.

Client-Server Computing



Java Networking Terminology

- **IP address:** It is a unique number assigned to a node of a network e.g. 192.168.0.1 . It is composed of octets that range from 0 to 255. It is a logical address that can be changed.
- **Protocol** :It is a set of rules basically that is followed for communication. For example: TCP, FTP, Telnet, SMTP, POP etc.
- **Port Number:** It is used to uniquely identify different applications.
  - It acts as a communication endpoint between applications.
  - The port number is associated with the IP address for communication between two applications.
- **MAC (Media Access Control):-** Address is a unique identifier of NIC (Network Interface Controller). A network node can have multiple NIC but each with unique MAC.

…Java Networking Terminology

-) Connection-oriented and connection-less protocol
  - In **connection-oriented protocol**, acknowledgement is sent by the receiver. So it is reliable but slow. The example of connection-oriented protocol is TCP.
  - But, in **connection-less protocol**, acknowledgement is not sent by the receiver. So it is not reliable but fast. The example of connection-less protocol is UDP.
- **Socket**: It is an endpoint between two way communication.

Socket Programming with TCP

- **TCP (Transmission Control Protocol)**
  - A connection-oriented protocol.
  - Allows reliable communication between two applications.
  - Usually used over the Internet with IP as TCP/IP
  - Resembles making a telephone call
    - The person placing the telephone call – client
    - The person waiting for a call – server
- The **java.net.ServerSocket** and **java.net.Socket** classes are the only two classes you will probably ever need to create a TCP/IP connection between two computers.

…with TCP

- For a secure connection, **SSLServerSocket** and **SSLSocket** classes in the javax.net.ssl package are used.
- Guarantees that data sent from one end of the connection actually gets to the other end and in the same order it was sent. Otherwise, an error is reported.
- The Hypertext Transfer Protocol (HTTP), File Transfer Protocol (FTP), and Telnet are all examples of applications that require a reliable communication channel.

Sockets and Java Socket Classes

- A program can read from a socket or write to a socket as simply as reading from a file or writing to a file.
- A socket is bound to a port number so that the TCP layer can identify the application that data destined to be sent.
- Java's **.net** package provides two classes:
  - Socket – for implementing a client
  - ServerSocket – for implementing a server

Socket Communication

- A server (program) runs on a specific computer and has a socket that is bound to a specific port.
- The server waits and listens to the socket for a client to make a connection request.

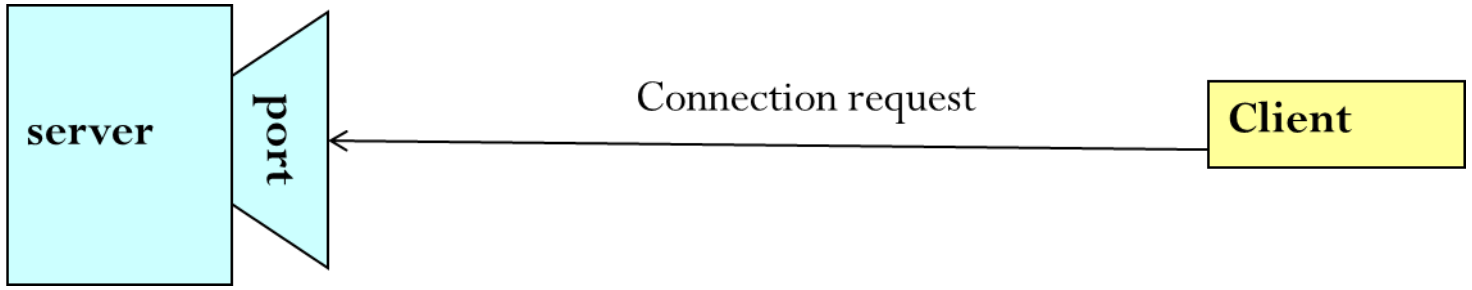
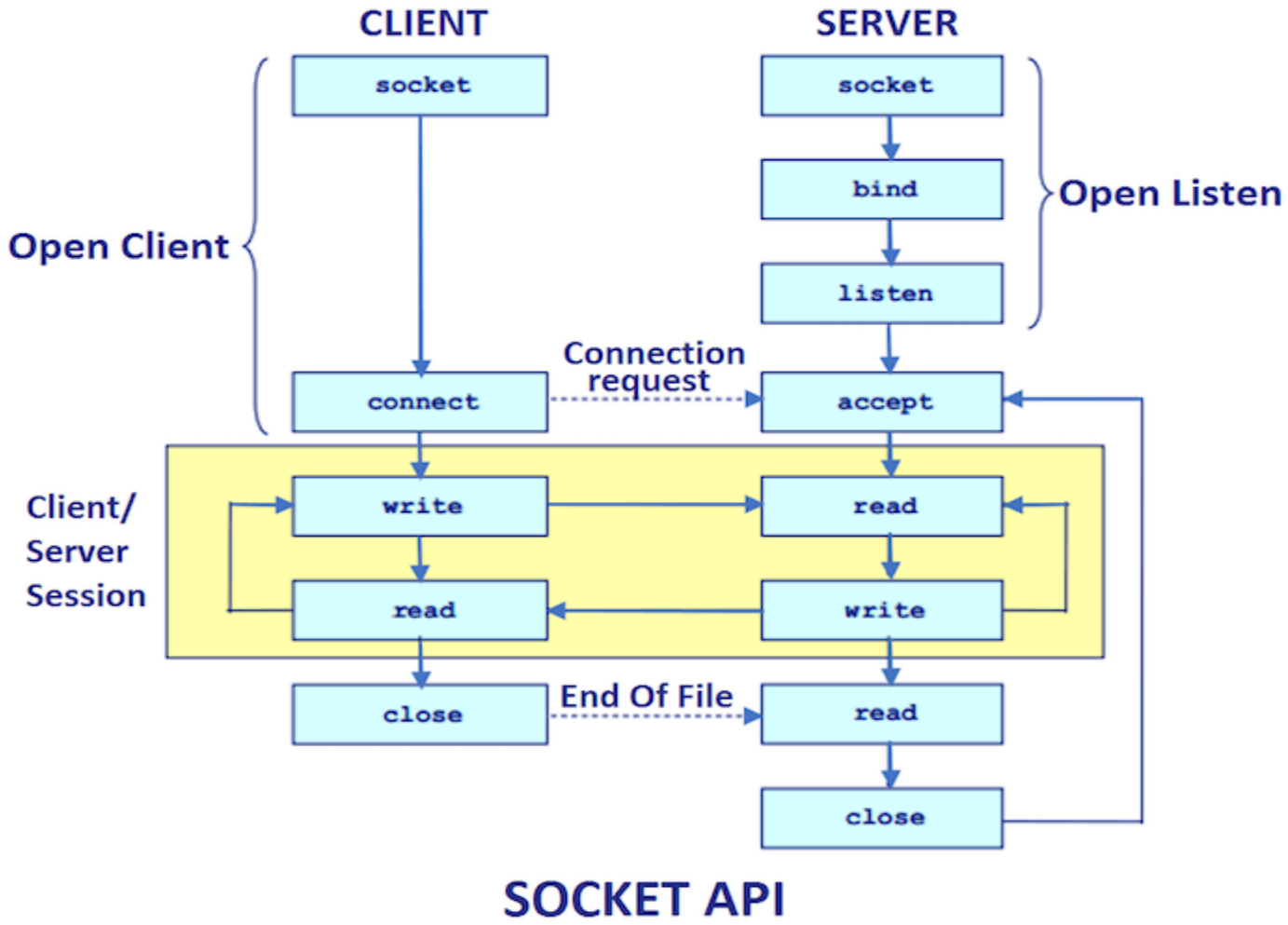


Fig. Client making connection request to server

…Socket Communication



Socket Operations

- There are four fundamental operations a socket performs. These are:
  - Connect to a remote machine
  - Send data
  - Receive data
  - Close the connection
- The **java.net.Socket** class allows you to perform all four fundamental socket operations
- **Socket** class represents the socket that both the client and server use to communicate with each other.
- Connection is accomplished through the constructors.
- Each Socket object is associated with exactly one remote host.
- To connect to a different host, you must create a new Socket object.

Socket constructors

- Socket(String host, int port) throws UnknownHostException, IOException
- Socket(InetAddress address, int port) throws IOException
- Socket(String host, int port, InetAddress localAddress, int localPort) throws IOException
- Socket(InetAddress address, int port, InetAddress localAddress, int localPort) throws IOException
- Sending and receiving data is accomplished with output and input streams.
  - InputStream getInputStream() throws IOException :
    - ✓ Returns the input stream of the socket.
    - ✓ The input stream is connected to the output stream of the remote socket.
  - OutputStream getOutputStream() throws IOException:
    - ✓ Returns the output stream of the socket.
    - ✓ The output stream is connected to the input stream of the remote socket
- There's a method to close a socket.
  - void close() throws IOException

Socket methods

- There are methods to return information about the socket:
  - InetAddress getInetAddress()
  - InetAddress getLocalAddress()
  - int getPort() //Returns the port the socket is bound to on the remote machine.
  - int getLocalPort() //Returns the port the socket is bound to on the local machine.
  - public void connect(SocketAddress host, int timeout)
  - String toString()

ServerSocket methods

1. public int getLocalPort(): Returns the port that the server socket is listening on.
   - This method is useful if you passed in 0 as the port number in a constructor and let the server find a port for you.
2. public Socket accept() throws IOException: Waits for an incoming client.
   - This method blocks until either a client connects to the server on the specified port or the socket times out, assuming that the time-out value has been set using the setSoTimeout() method. Otherwise, this method blocks indefinitely

… methods

3. public void setSoTimeout(int timeout) Sets the time-out value for how long the server socket waits for a client during the accept().

4. public void bind(SocketAddress host, int backlog): Binds the socket to the specified server and port in the SocketAddress object. Use this method if you instantiated the ServerSocket using the no-argument constructor.

Establishing a Simple Server Using Stream Sockets

- Five steps to create a simple stream server in Java:

1. Open the Server Socket: Each client connection handled with a Socket object.
   - Server blocks until client connects.

   ```
   ServerSocket server = new ServerSocket( PORT );
   ```
2. Wait for the Client Request.
   - Socket client = server.accept();
3. Create I/O streams for communicating to the client
   - DataInputStream is = new DataInputStream(client.getInputStream());
   - DataOutputStream os = new DataOutputStream(client.getOutputStream());
4. Perform communication with client Receive from client:
   - String line = is.readLine();
   - Send to client: os.writeBytes("Hello\n");
5. Close socket:
   - client.close();

Establishing a Simple Client Using Stream Sockets

- Four steps to create a simple stream client in Java:

1. Create a Socket Object: Obtains Socket's InputStream and OutputStream.
   - Socket client = new Socket(server, port_id);
2. Create I/O streams for communicating with the server.
   - is = new DataInputStream(client.getInputStream());
   - os = new DataOutputStream(client.getOutputStream());
3. Perform I/O or communication with the server:
   - Receive data from the server: String line = is.readLine();
   - Send data to the server: os.writeBytes("Hello\n");
4. Close the socket when done:

   client.close();

Example of Java Socket Programming (Read-Write both side)

File: MyServer.java

```
import java.net.*;
import java.io.*;
class MyServer{
public static void main(String args[])throws Exception{
ServerSocket ss=new ServerSocket(3333);
Socket s=ss.accept();
DataInputStream din=new DataInputStream(s.getInputStream());
DataOutputStream dout=new DataOutputStream(s.getOutputStream());
BufferedReader br=new BufferedReader(new InputStreamReader(System.in));

String str="",str2="";
while(!str.equals("stop")){
str=din.readUTF();
System.out.println("client says: "+str);
str2=br.readLine();
dout.writeUTF(str2);
dout.flush();
}
din.close();
s.close();
ss.close();
}}
```

File: MyClient.java

```
import java.net.*;
import java.io.*;
class MyClient{
public static void main(String args[])throws Exception{
Socket s=new Socket("localhost",3333);
DataInputStream din=new DataInputStream(s.getInputStream());
DataOutputStream dout=new DataOutputStream(s.getOutputStream());
BufferedReader br=new BufferedReader(new InputStreamReader(System.in));

String str="",str2="";
while(!str.equals("stop")){
str=br.readLine();
dout.writeUTF(str);
dout.flush();
str2=din.readUTF();
System.out.println("Server says: "+str2);
}

dout.close();
s.close();
}}
```

Socket Programming with UDP

- **UDP (User Datagram Protocol):** A connectionless protocol. because the packets have no relationship to each other and because there is no state maintained.
  - Allows packets of data (datagram) to be transmitted between applications.
  - Resembles mailing someone a letter
    - The datagram packet is like a letter, where a client sends a datagram to a server without actually connecting to the server.
    - This makes UDP an unreliable protocol.

…with UDP

- UDP does not guarantee that packets will be received in the order they were sent or that they will even be delivered at all.
  - The java.net.DatagramPacket class –to send datagram packets
  - The java.net.DatagramSocket class – to receive datagram packets
- Sender does not wait for acknowledgements
- Arrival order is not guaranteed
- Arrival is not guaranteed.
- So why use UDP if it unreliable? Two reasons: speed and overhead.
- USED when speed is essential, even in cost of reliability

e.g. Streaming Media, Games, Internet Telephony, etc.

…with UDP

- Datagram sockets transmit individual packets of information.
- This is typically not appropriate for use by everyday programmers because the transmission protocol is UDP (User Datagram Protocol).
- With UDP, packets can be lost or duplicated.
- Significant extra programming is required on the programmer's part to deal with these problems.
- UDP is most appropriate for network applications that do not require the error checking and reliability of TCP.

…with UDP

- Under UDP there is no “connection” between the server and the client.
- There is no “handshaking”.
- The sender explicitly attaches the IP address and port of the destination to each packet.
- The server must extract the IP address and port of the sender from the received packet.
- From an application viewpoint, UDP provides unreliable transfer of groups of bytes (“datagrams”) between client and server.

Steps to Receive a Datagram packet - UDP

1. Create an array of bytes large enough to hold the data of the incoming packet.

```
byte[] buffer = new byte[1024];
```

2. A DatagramPacket object is instantiated using the array of bytes.

```
DatagramPacket packet =new DatagramPacket(buffer, buffer.length);
```

3. A DatagramSocket is instantiated, and it is specified which port (and specific localhost address, if necessary) on the localhost the socket will bind to.

```
int port = 1234;
DatagramSocket socket = new DatagramSocket(port);
```

4. The receive() method of the DatagramSocket class is invoked, passing in the DatagramPacket object. This causes the thread to block until a datagram packet is received or a time out occurs.

```
// Block on receive()
socket.receive(packet);
// Find out where packet came from so we can reply to the same host/port
InetAddress remoteHost = packet.getAddress();
int remotePort = packet.getPort();
// Extract the packet data
byte[] data = packet.getData();
```

Steps to Send a Datagram packet - UDP

1. Create an array of bytes large enough to hold the data of the packet to be sent, and fill the array with the data.

   byte[] buffer = new byte[1024];

2. Create a new DatagramPacket object that contains the array of bytes, as well as the server name and port number of the recipient.

   int port = 1234;

   InetAddress host =InetAddress.getByName("www.mysite.com");
   DatagramPacket packet =new DatagramPacket(buffer, buffer.length, host, port);

3. A DatagramSocket is instantiated, and it is specified which port (and specific localhost address, if necessary) on the localhost the socket will bind to.

   DatagramSocket socket = new DatagramSocket();

4. The send() method of the DatagramSocket class is invoked, passing in the DatagramPacket object.

Example: Java server using UDP

```
import java.io.*;
import java.net.*;
class UDPServer {
  public static void main(String args[]) throws Exception
            {
                        //Create datagram socket on port 9876
                        DatagramSocket serverSocket = new DatagramSocket(9876);

                        byte[] sendData = new byte[1024];
                        byte[] receiveData = new byte[1024];

                        while (true)
                        {
                                    //create space for the received datagram
                                    DatagramPacket receivePacket = new
                        DatagramPacket(receiveData,
            receiveData.length);
                                    //receive the datagram
                                    serverSocket.receive(receivePacket);

                                    String sentence = new String(receivePacket.getData());
```

…Example: Java server using UDP

```
                    //get IP address and port number of sender
                        InetAddress IPAddress = receivePacket.getAddress();
                        int port = receivePacket.getPort();
                        String capitalizedSentence =sentence.toUpperCase();
                        sendData = capitalizedSentence.getBytes();
                        //create datagram to send to client
                        DatagramPacket sendPacket = new
                     DatagramPacket(sendData, sendData.length, IPAddress, port);
                        //write out the datagram to the socket
                        serverSocket.send(sendPacket);
                    } //end while loop
            }
}
```

Example: Java client using UDP

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

class UDPClient {
   public static void main(String args[]) throws Exception
             {
                        //Create input stream
                        BufferedReader inFromUser = new BufferedReader(new
                        InputStreamReader(System.in));
                        //Create client socket
                        DatagramSocket clientSocket = new DatagramSocket();
                        //Translate hostname to IP address using DNS
                        InetAddress IPAddress = InetAddress.getByName("localhost");

                        byte[] sendData = new byte[1024];
                        byte[] receiveData = new byte[1024];

                        String sentence = inFromUser.readLine();
                        sendData = sentence.getBytes();
```

…Example: Java client using UDP

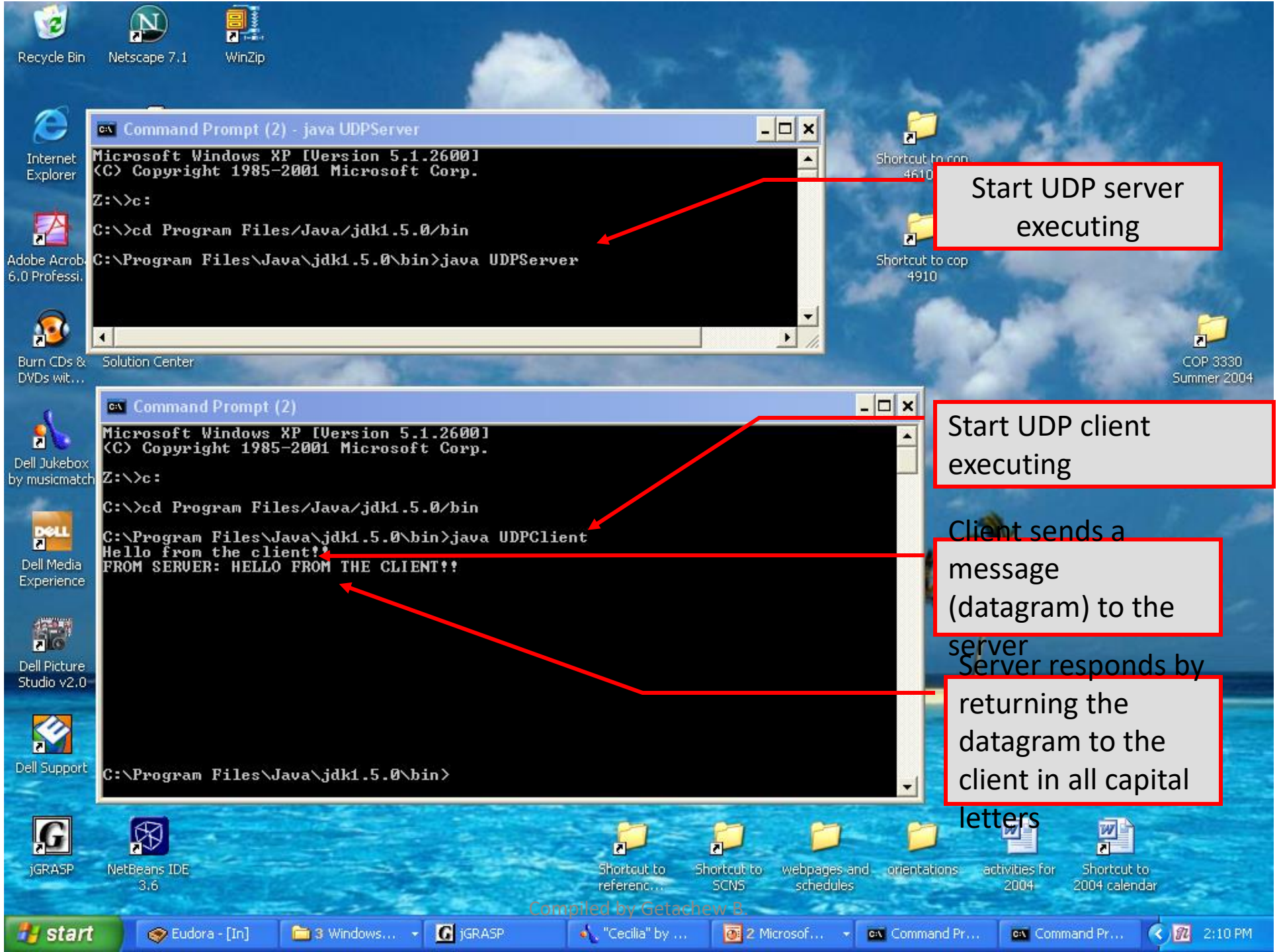
```
                DatagramPacket sendPacket = new DatagramPacket(sendData,
                                sendData.length, IPAddress, 9876);
                clientSocket.send(sendPacket);

                DatagramPacket receivePacket = new DatagramPacket(receiveData,
                                receiveData.length);

                clientSocket.receive(receivePacket);

                String modifiedSentence = new String(receivePacket.getData());

                System.out.println("FROM SERVER: " + modifiedSentence);
                clientSocket.close();
        }
}
```


Command Prompt (2) - java UDPServer
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.
Z:\>c:
C:\>cd Program Files/Java/jdk1.5.0/bin
C:\Program Files\Java\jdk1.5.0\bin>java UDPServer
Start UDP server executing
Command Prompt (2)
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.
Z:\>c:
C:\>cd Program Files/Java/jdk1.5.0/bin
C:\Program Files\Java\jdk1.5.0\bin>java UDPClient
Hello from the client!!
FROM SERVER: HELLO FROM THE CLIENT!!
C:\Program Files\Java\jdk1.5.0\bin>
Start UDP client executing
Client sends a message (datagram) to the server
Server responds by returning the datagram to the client in all capital letters

The end