Java Socket Programming

- Principles of network programming

- Socket programming
- Building your own program
- Socket Examples
  - Web and HTTP
  - DHCP
  - FTP
  - Electronic Mail
    - SMTP, POP3, IMAP
  - DNS

http://java.sun.com/j2se/1.5.0/docs/api/

Network programming

Goal: 컴퓨터 네트워크 상에서 수행되는 프로그램

(컴퓨터네트워크는 호스트와 라우터로 구성, 네트워크 프로그램은 네트워크를 통하여 정보를 교환하는 프로그램, 데이터 통신은 네트워크를 통해 데이터를 전달하는 통신시스템 (송수신자간에 의미 있는 정보의 교환 방법 및 규약), 패킷은 정보의 바이트 배열 또는 묶음, 프로토콜은 패킷 교환을 위한 상호약속 임)

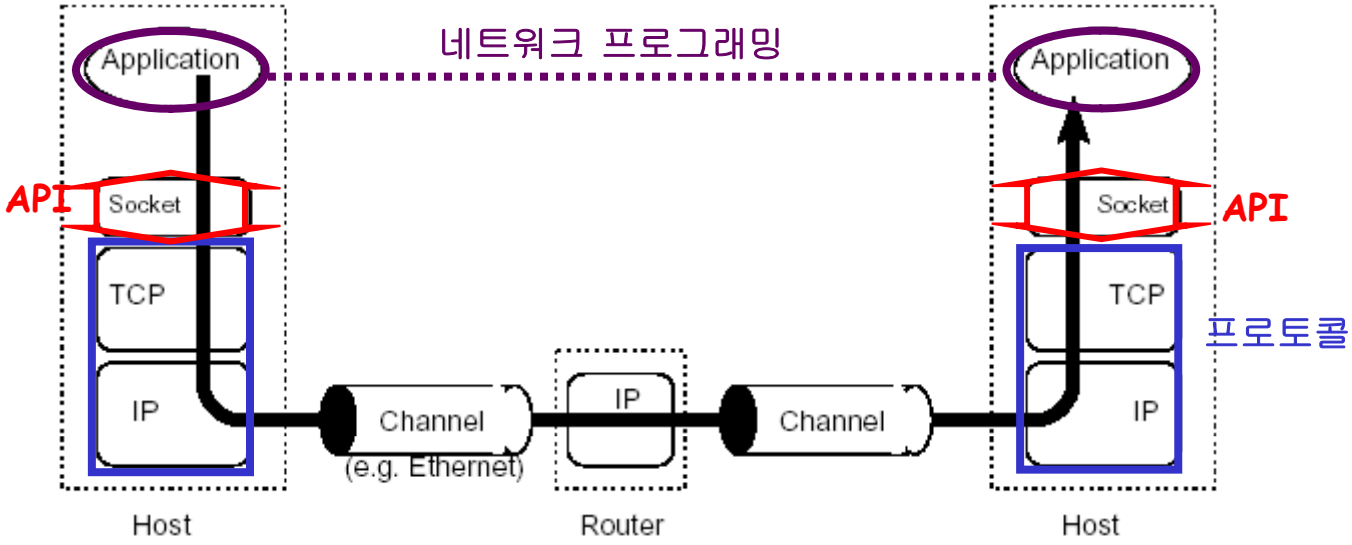
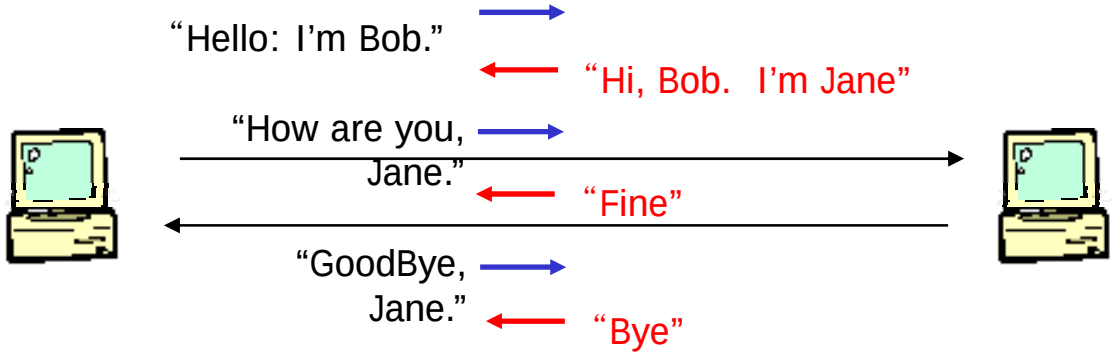


Figure 1.1: A TCP/IP Network

Network programming example: Computer Chat

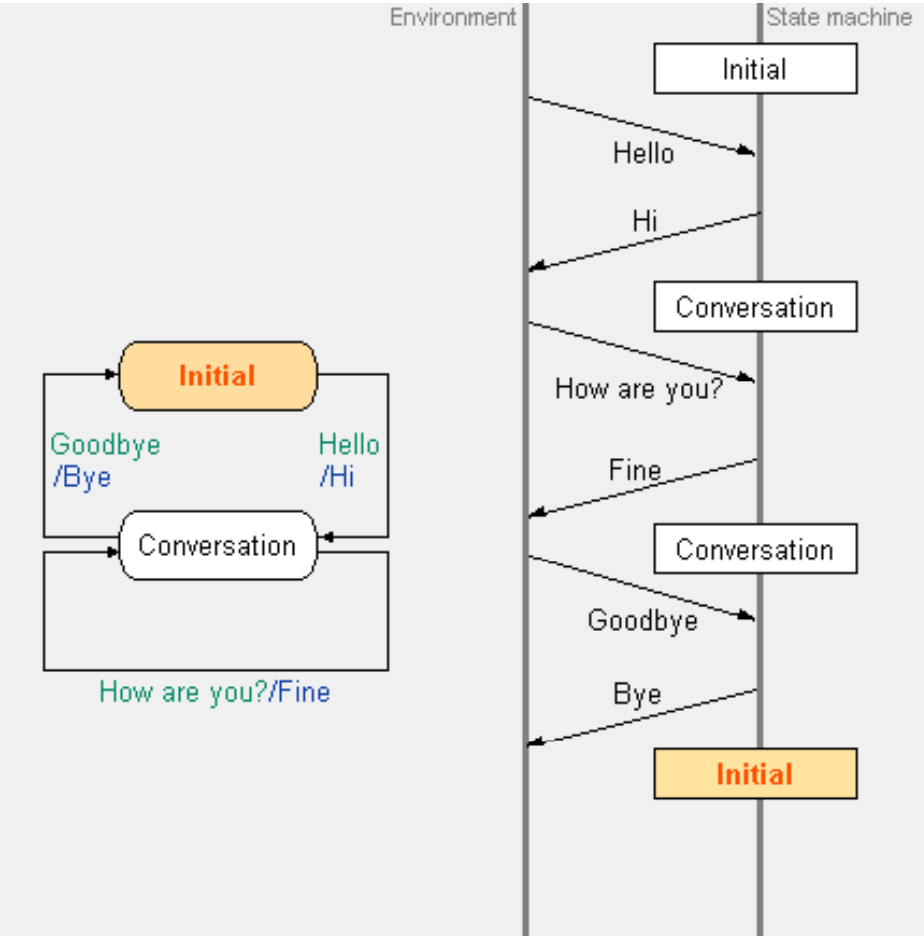
- How do we make computers talk?



- How are they interconnected?
  - Wire
  - Internet Protocol (IP) ?
  - Human Interpreter

Signal Sequence Diagrams

- Useful to represent specific exchanges of signals
  - In a signal sequence diagram, the vertical lines represent the progress of time.
  - We start at the top of the diagram and read down the vertical lines.

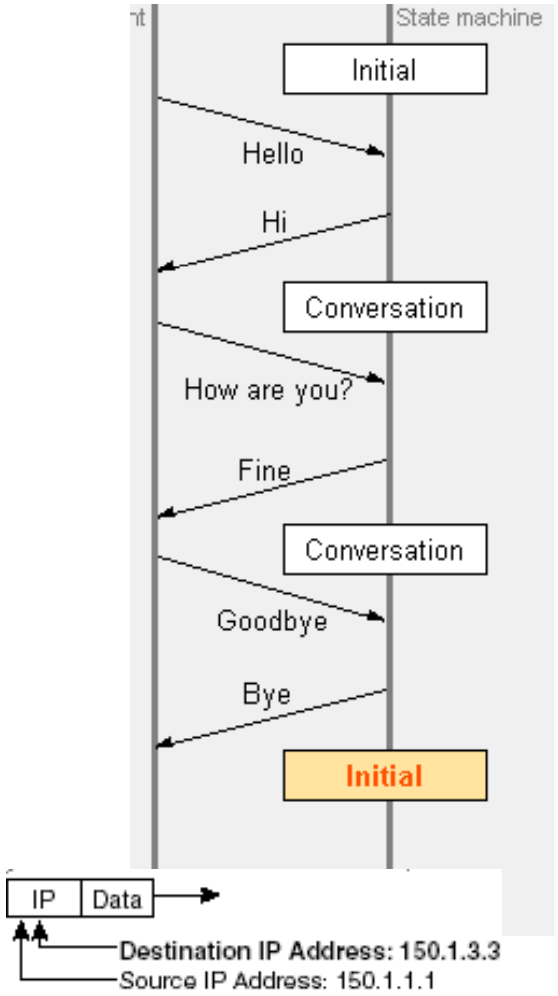


- Annotate with the name of the current state
- The signals are shown as sloping lines on a signal sequence diagram.
- The vertical displacement of the lines represents the time taken for the signals to travel between systems.
- It is not generally possible to discover the complete behaviour of a finite state machine from a signal sequence diagram.
  - A state transition table or a state transition diagram can be used to generate all the possible exchanges of signals

Protocol Summary

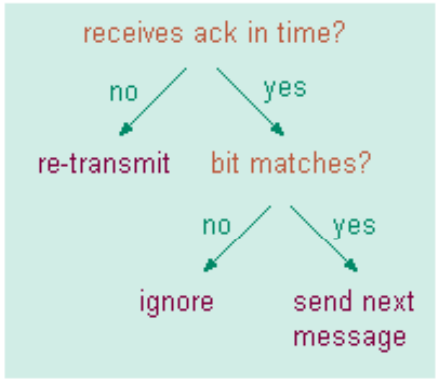
-Protocol Behaviour:Signal Sequence

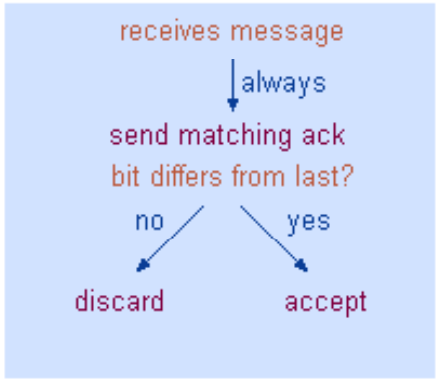
-Frame Format



Sender

Receiver

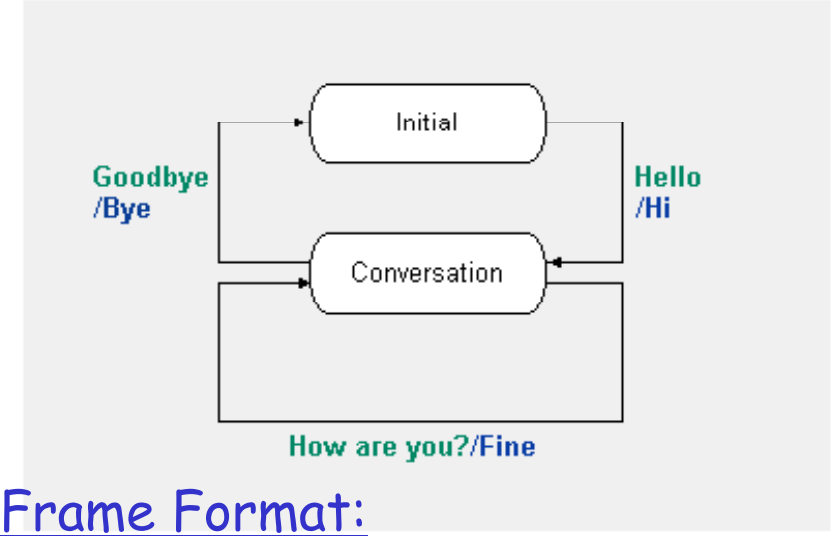




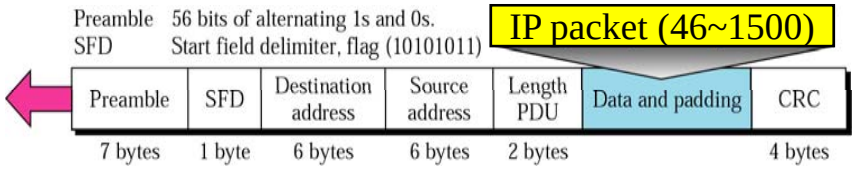
Protocol Behaviour

Signal Sequence

Input		Output	
Current State	Input signal	Next state	Output signal
Initial	Hello	Conversation	Hi
Conversation	How are you?	Conversation	Fine
Conversation	Goodbye	Initial	Bye



Frame Format:



- To describe the behaviour of a system need to determine which output signal is generated for each input signal.
- If a system always gives the same output signal for each input signal, then the relationship between input and output signals can be shown in a simple table.
- We'll start to define a state machine model for our conversation-robot.

Input	Output
Hello	Hi
How are you	Fine
Goodbye	Bye

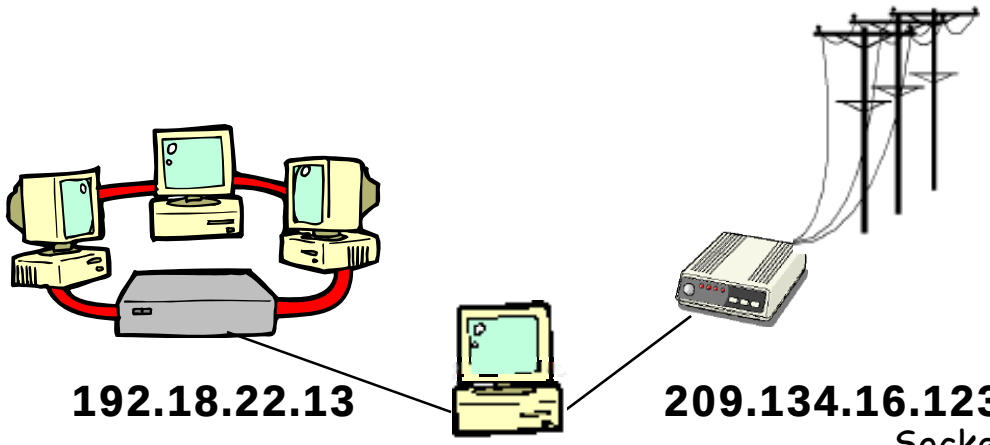
- The table now shows an appropriate output signal for each input signal, it doesn't yet define the behaviour of a normal conversation.

Internet Protocol (IP)

- Datagram (packet) protocol
- Best-effort service
  - Loss
  - Reordering
  - Duplication
  - Delay
- Host-to-host delivery

IP Address

- 32-bit identifier
- Dotted-quad: 192.118.56.25
- www.mkp.com -> 167.208.101.28
- Identifies a host interface (not a host)



Transport Protocols

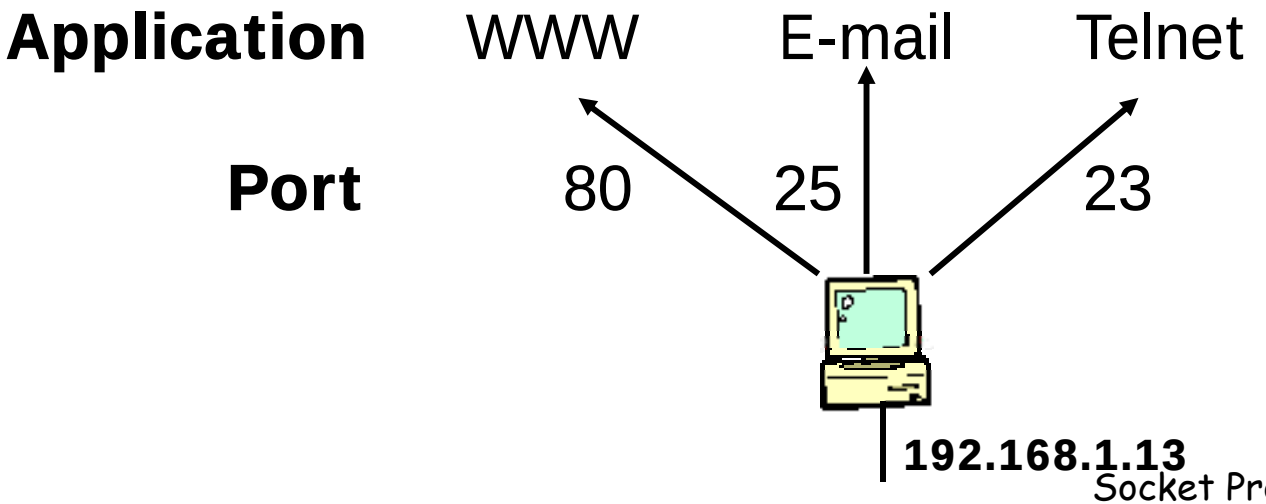
Best-effort not sufficient!

- Add services[applications or processes] on top of IP
- User Datagram Protocol (UDP)
  - Data checksum
  - Best-effort
- Transmission Control Protocol (TCP)
  - Data checksum
  - Reliable byte-stream delivery
  - Flow and congestion control

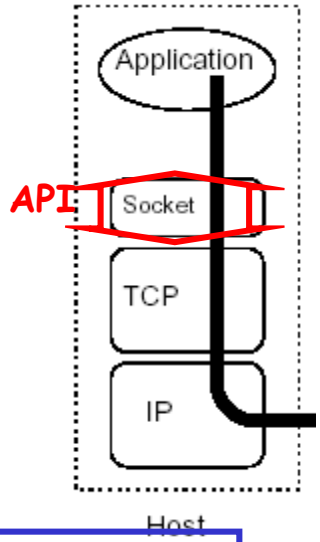
Ports (multiplexing)

Identifying the ultimate destination [process]

- IP addresses identify hosts
- Host has many applications
- Ports (16-bit identifier) indicates one of the application.



Socket



Socket: a door between application process and end-end-transport protocol (UCP or TCP)

Socket API

- client/server paradigm
- two types of transport service via socket API:
  - unreliable datagram UDP socket
  - reliable, byte stream-oriented TCP socket

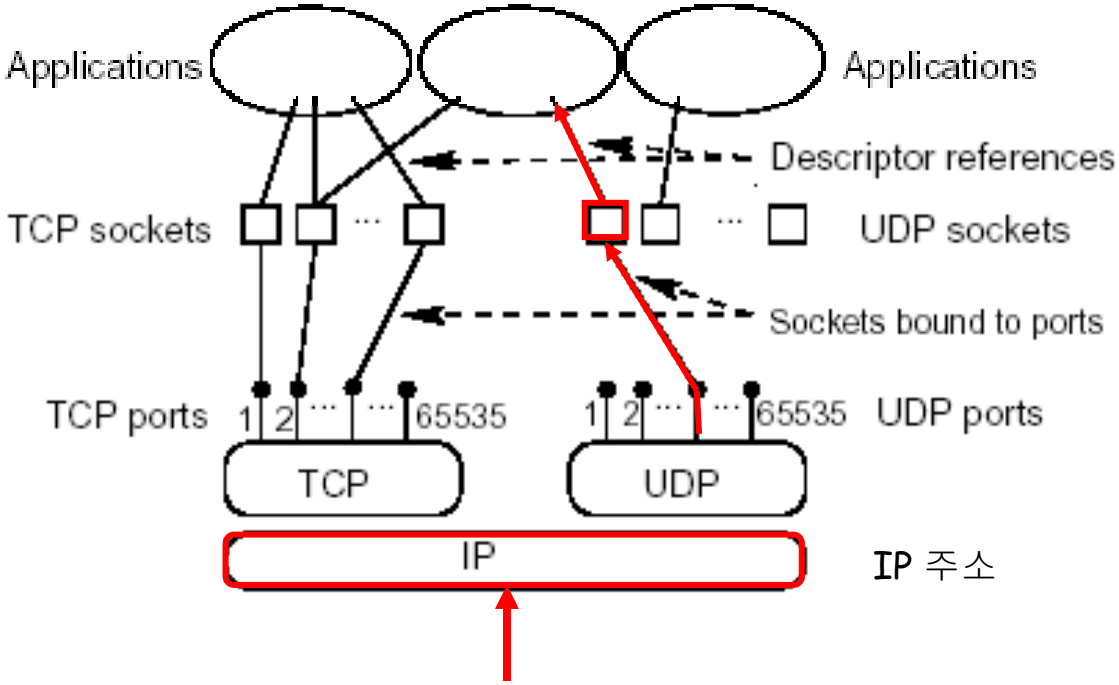
socket

Socket: 어플리케이션이 데이터를 주고받을 수 있는 추상적인 개념

a *host-local*, *application-created*, *OS-controlled* interface (a "door") into which application process can both send and receive messages to/from another application process

Socket & Protocol

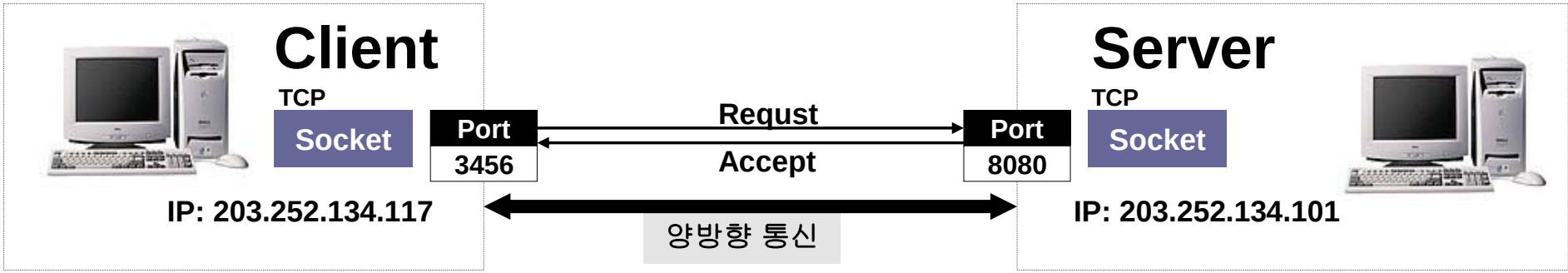
- Identified by protocol and local/remote address/port
- Applications may refer to many sockets
- Socket의 구성 = IP Address + Port 번호



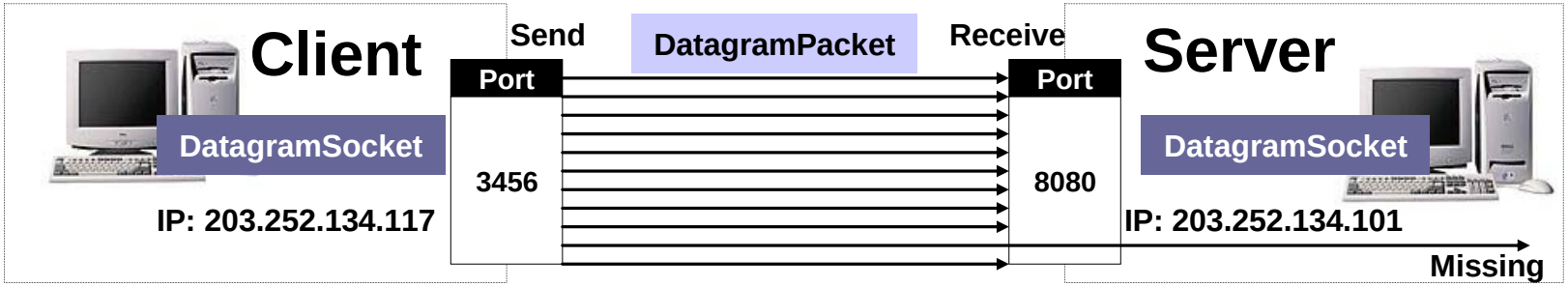
•프로토콜의 정의 :
서로 다른 시스템에 존재하는 개체(entity)간의 원활한 통신을 위해 서로 약속된 통신 규약 (예:TCP/IP,…)

Socket 통신의 개념

TCP 통신의 개념



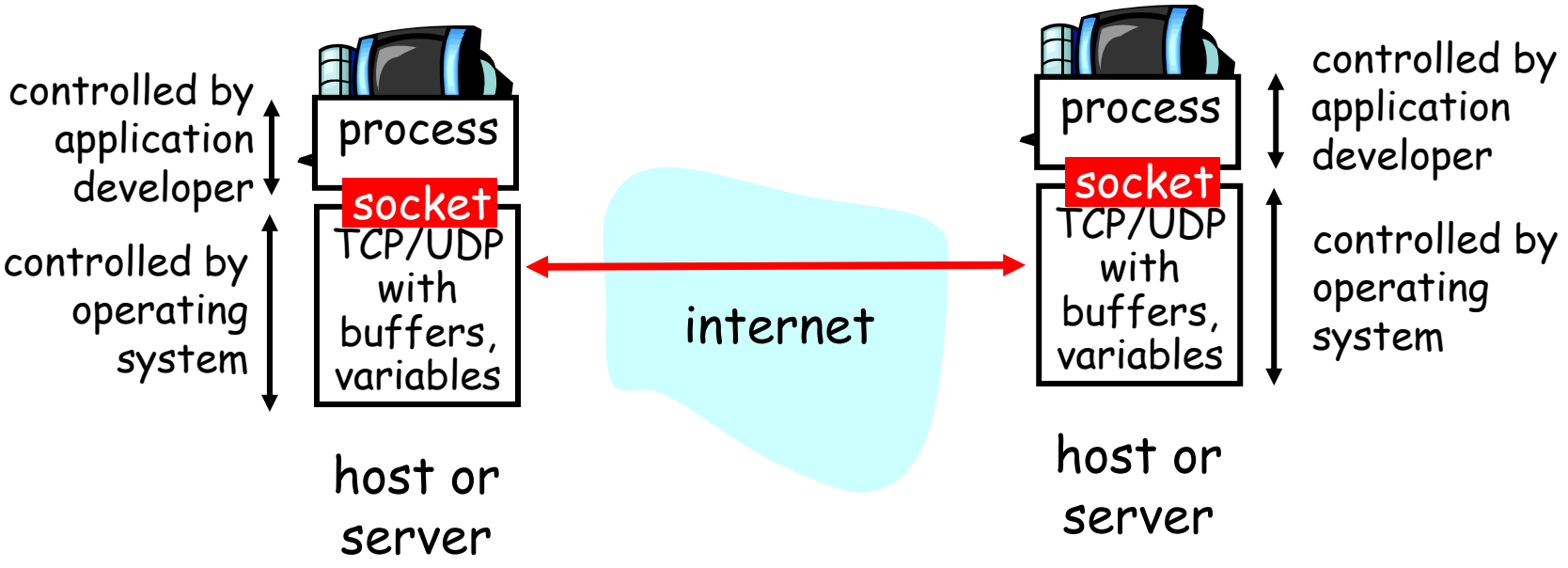
UDP 통신의 개념



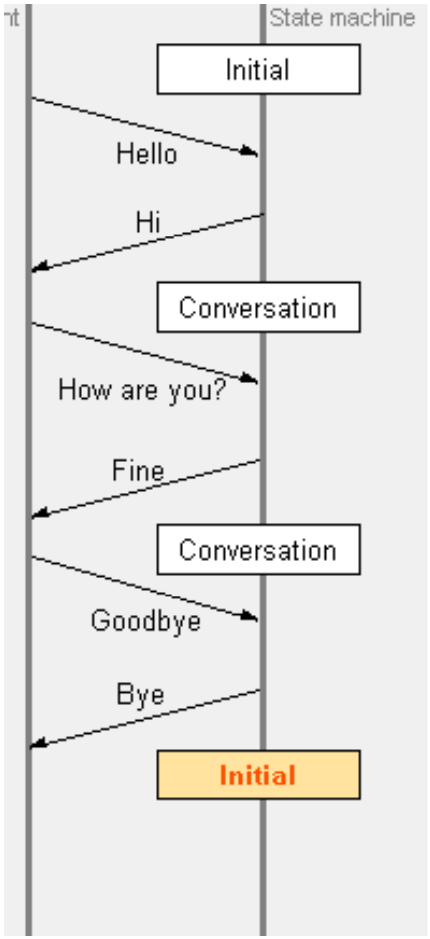
Socket-programming

Goal: learn how to build client/server application that communicate using sockets

Examples: Chatting, Mail, HTTP,...



Clients and Servers: Human



- Client: Initiates the connection

Client: Bob — Server: Jane

"Hello: I'm Bob." →

← "Hi, Bob. I'm Jane"

"How are you, Jane." →

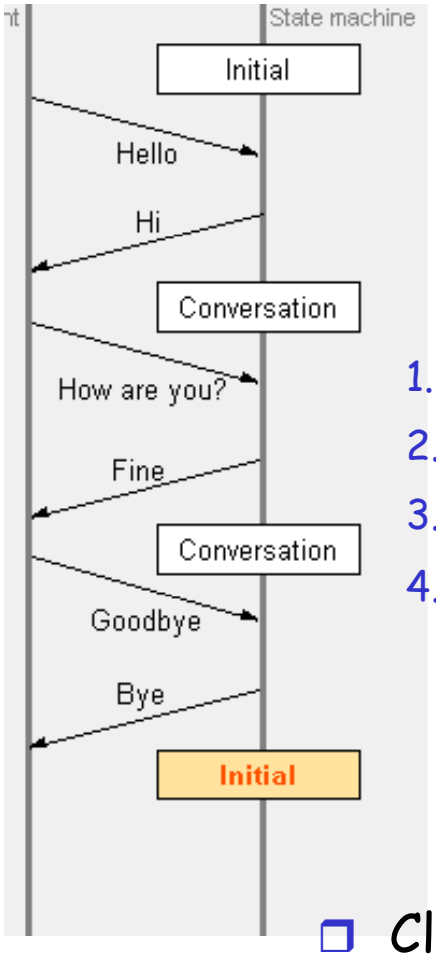
← "Fine"

"GoodBye, Jane." →

← "Bye"

- Server: Passively waits to respond

UDP Clients and Servers Model



- Client: Initiates the connection

Client: Bob

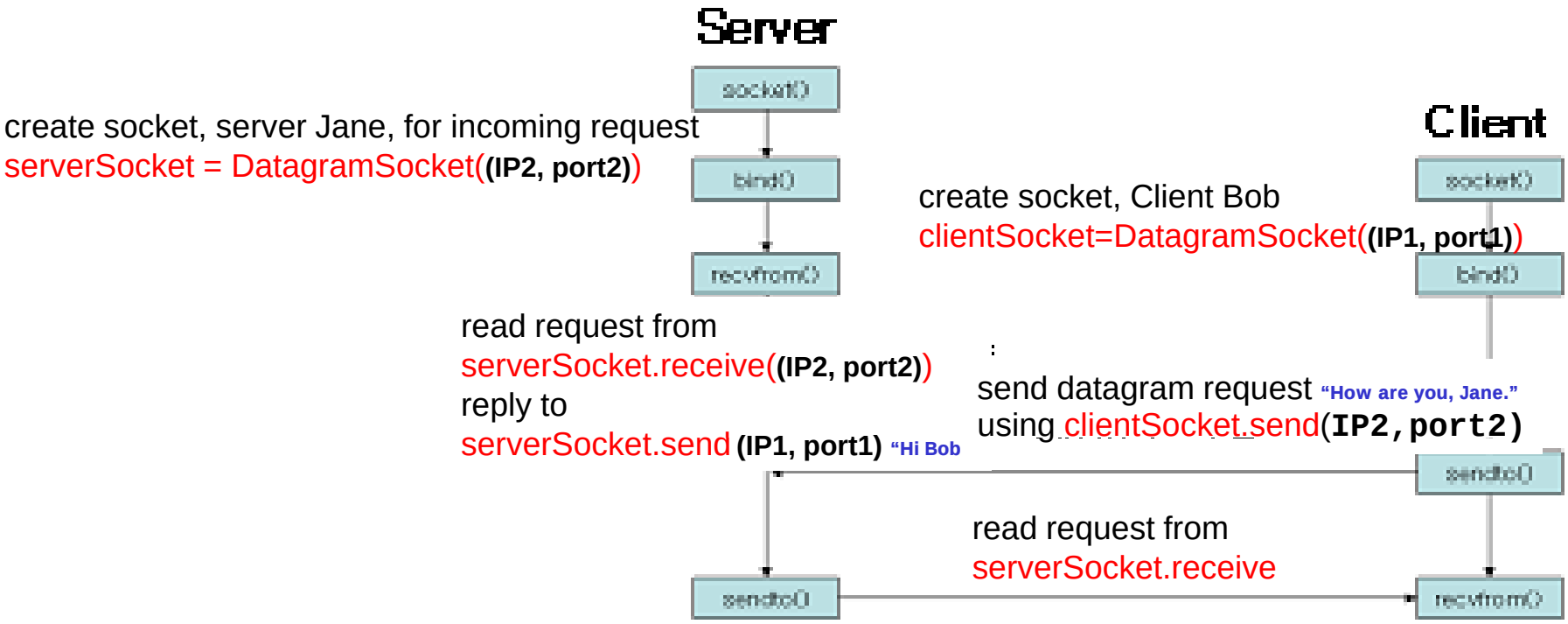
1. Create a socket
2. Establish connection
3. Communicate
4. Close the connection

Server: Jane

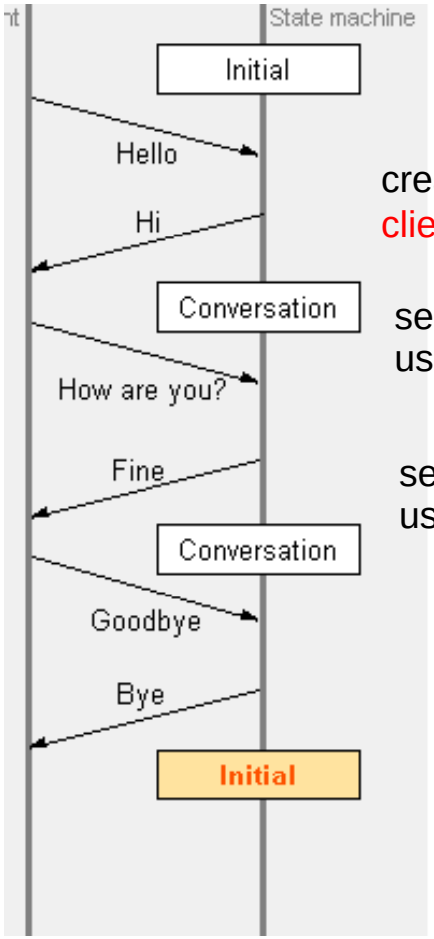
1. Create a Server socket (Waiting for Client)
2. Bind socket to a port
3. Set socket to listen
4. Close the connection
5. Repeatedly:
   a. Accept new connection
   b. Communicate
   c. Close the connection

- Client: Initiates the connection
- Server: initially waits to respond

UDP Clients and Servers API Model



Clients and Servers (UDP)



- Client: Initiates the connection

create socket, Client Bob
clientSocket=DatagramSocket(**(IP1, port1)**)

create socket, server Jane, for incoming request
serverSocket = DatagramSocket(**(IP2, port2)**)

send datagram request **"Hello, I am Bob."**
using clientSocket.send(**IP2,port2)**

read request from
serverSocket.receive(**(IP2, port2)**)
reply to
serverSocket.send **(IP1, port1)** **"Hi Bob"**

send datagram request **"How are you, Jane."**
using clientSocket.send(**IP2,port2)**

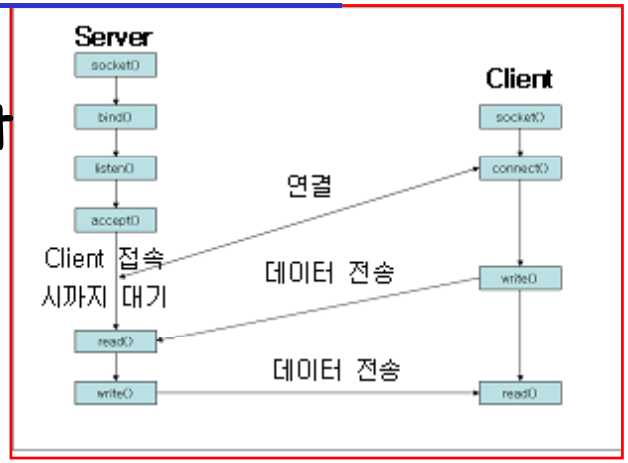
read request from
serverSocket.receive
reply to
serverSocket.send**(IP1, port1)** **"I am fine"**

send datagram request **"GoodBye, Jane."**
using clientSocket.send(**IP2,port2)**

read request
serverSocket.receive **"Bye"**
Close serverSocket **(IP2, port2)**

- Server: initially waits to respond

Client/server socket interaction: TCP



Server (running on **hostid**)

create socket,
port=**x**, for
incoming request:
welcomeSocket =
ServerSocket()

wait for incoming
connection request
connectionSocket =
welcomeSocket.accept()

read request from
connectionSocket.getInputStream()

write reply to
connectionSocket.getOutputStream()

close
connectionSocket.close()

TCP
connection setup

Client

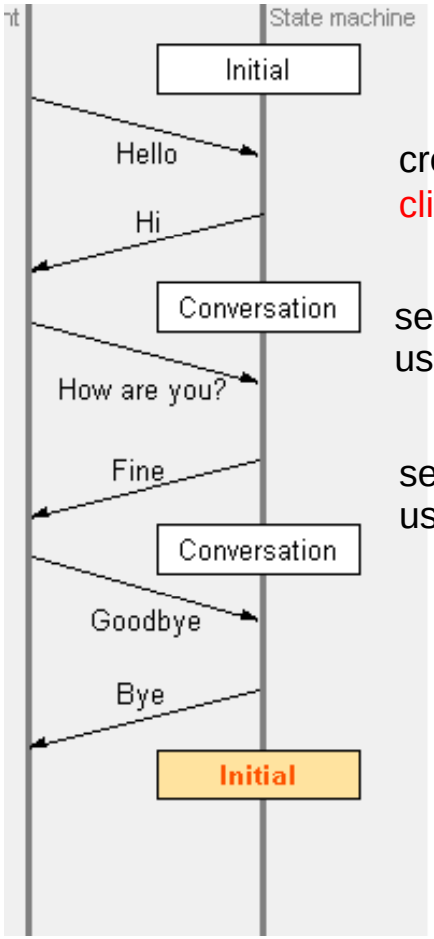
create socket,
connect to **hostid**, port=**x**
clientSocket =
Socket() + connect()

send request using
clientSocket.getOutputStream()

read reply from
clientSocket.getInputStream()

close
clientSocket.close()

Clients and Servers (TCP)



- Client: Initiates the connection

create socket, server Jane, for incoming request
serverSocket = ServerSocket(**(port2)**)
Server=serverSocket.accept();

create socket, Client Bob
clientSocket=Socket(**(IP2, port2,IP1, port1)**)

send datagram request **"Hello, I am Bob."**
using clientSocket.getOutputStream(**IP2,port2**)

read request from
Server.getInputStream(in)
reply to
Server.getOutputStream(**"Hi Bob"**)

send datagram request **"How are you, Jane."**
using clientSocket.getOutputStream(**IP2,port2**)

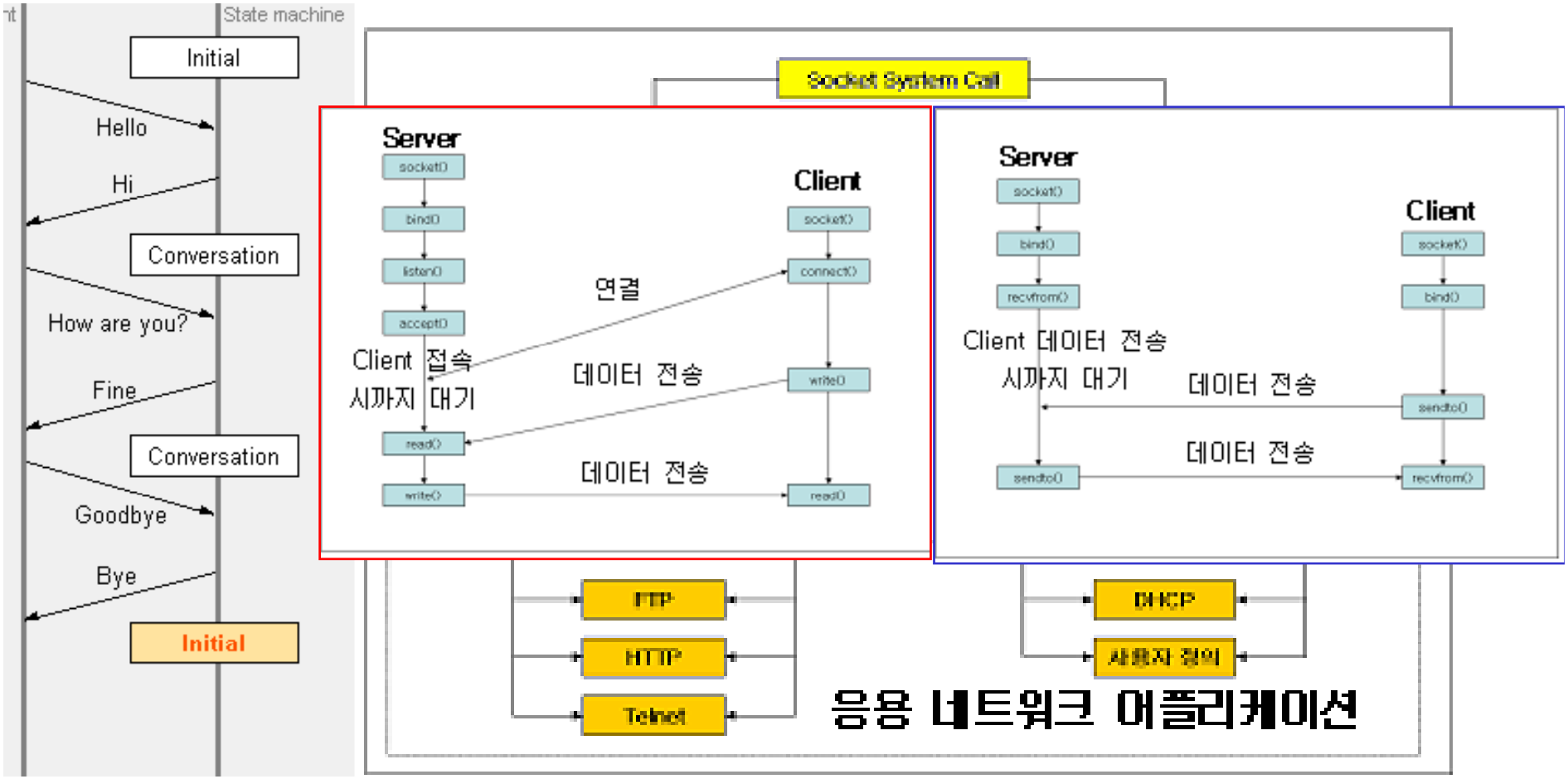
read request from
Server.getInputStream(in)
reply to
Server.getOutputStream(**" I am fine"**)

send datagram request **"GoodBye, Jane."**
using clientSocket.getOutputStream(**IP2,port2**)

read request
Server.getInputStream(in= **"Bye"**)
Close serverSocket **(IP2, port2)**

- Server: initially waits to respond

Clients and Servers Model (UDP/TCP)



Socket programming *with UDP*

UDP: no "connection" between client and server

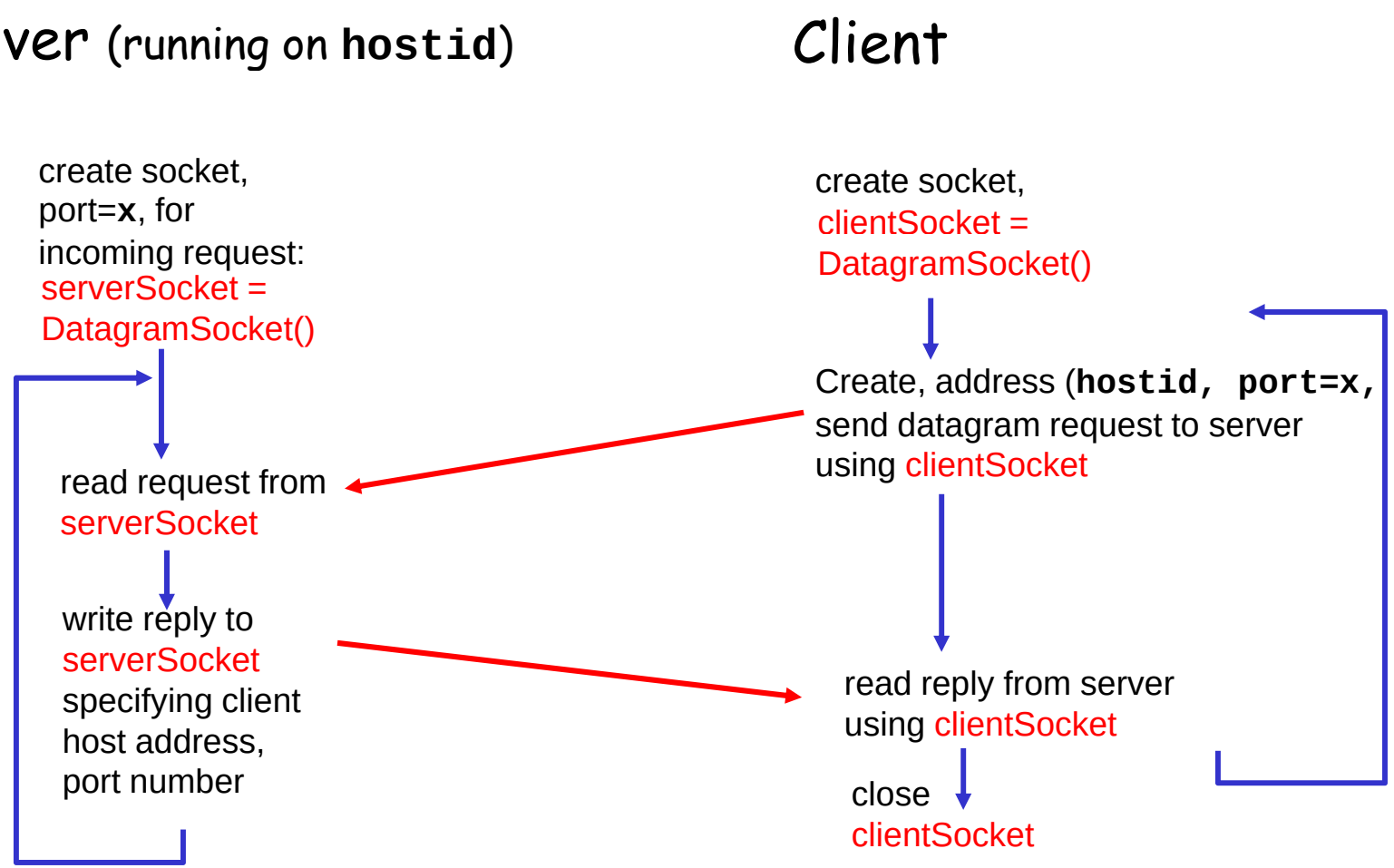
- no handshaking (call setup)
- sender explicitly attaches IP address and port of destination to each packet
- server must extract IP address, port of sender from received packet

UDP: transmitted data may be received out of order, or lost

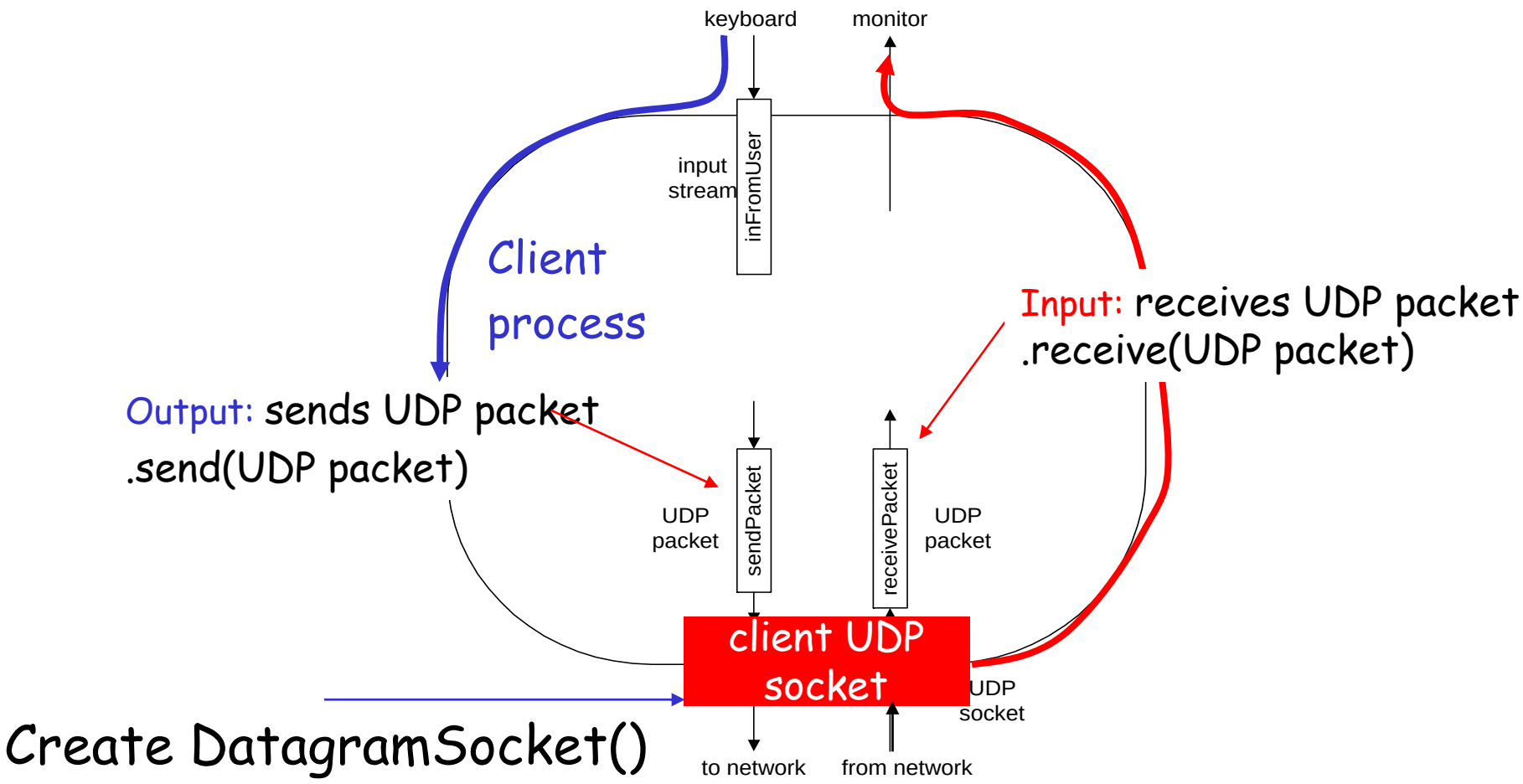
application viewpoint

UDP provides unreliable transfer of groups of bytes ("datagrams") between client and server

Client/server socket interaction: UDP



Example: Java Echo client (UDP)



Example: Java client (UDP)

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

class UDPClient {
    public static void main(String args[]) throws Exception
    {

      BufferedReader inFromUser =
        new BufferedReader(new InputStreamReader(System.in));

      DatagramSocket clientSocket = new DatagramSocket();

      InetAddress IPAddress = InetAddress.getByName("hostname");

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

      String sentence = inFromUser.readLine();

      sendData = sentence.getBytes();
```

Create input stream

Create client socket

Translate hostname to IP address using DNS

Example: Java client (UDP), cont.

Create datagram with data-to-send, length, IP addr, port

Send datagram to server

Read datagram from server

```
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();
}

}
```

Example: Java Echo server (UDP)

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

class UDPServer {
  public static void main(String args[]) throws Exception
    {
      DatagramSocket serverSocket = new DatagramSocket(9876);

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

      while(true)
        {
          DatagramPacket receivePacket =
            new DatagramPacket(receiveData, receiveData.length);
          serverSocket.receive(receivePacket);
```

Create datagram socket at port 9876

Create space for received datagram

Receive datagram

Example: Java server (UDP), cont

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

InetAddress IPAddress = receivePacket.getAddress();

int port = receivePacket.getPort();

                String capitalizedSentence = sentence.toUpperCase();

sendData = capitalizedSentence.getBytes();

DatagramPacket sendPacket =
   new DatagramPacket(sendData, sendData.length, IPAddress,
                      port);

serverSocket.send(sendPacket);
    }
  }
}
```

Get IP addr port #, of sender

Create datagram to send to client

Write out datagram to socket

End of while loop, loop back and wait for another datagram

Socket programming *with TCP*

Client must contact server

- server process must first be running
- server must have created socket (door) that welcomes client's contact

Client contacts server by:

- creating client-local TCP socket
- specifying IP address, port number of server process
- When client creates socket: client TCP establishes connection to server TCP

- When contacted by client, server TCP creates new socket for server process to communicate with client
  - allows server to talk with multiple clients
  - source port numbers used to distinguish clients (more in Chap 3)

application viewpoint

TCP provides reliable, in-order transfer of bytes ("pipe") between client and server

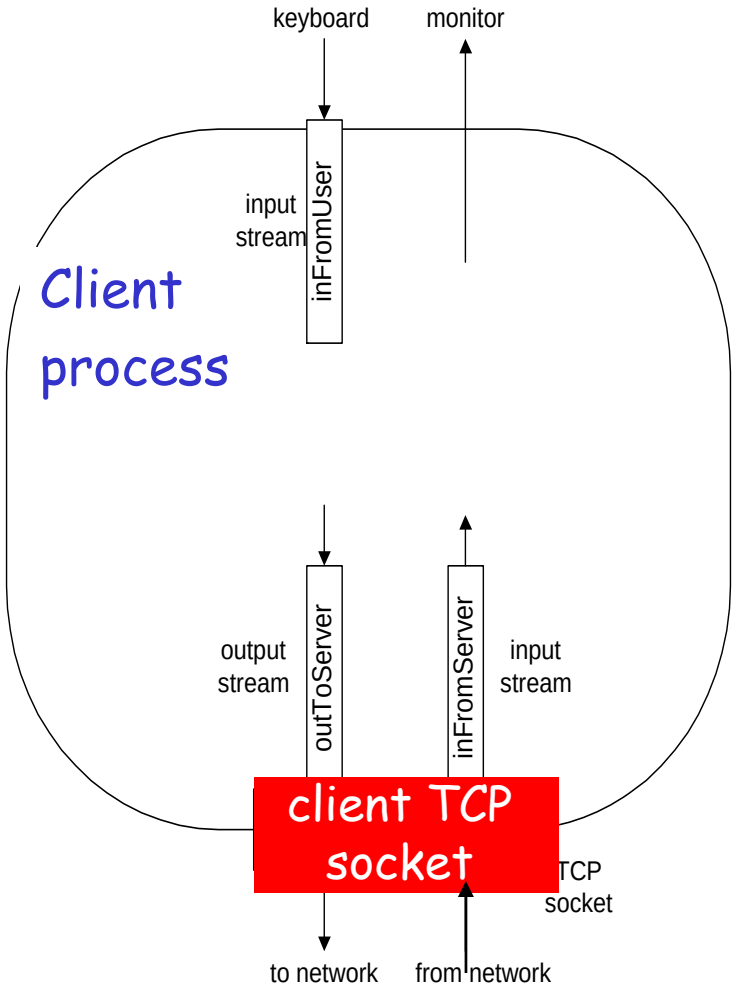
Stream jargon

- A stream is a sequence of characters that flow into or out of a process.
- An input stream is attached to some input source for the process, eg, keyboard or socket.
- An output stream is attached to an output source, eg, monitor or socket.

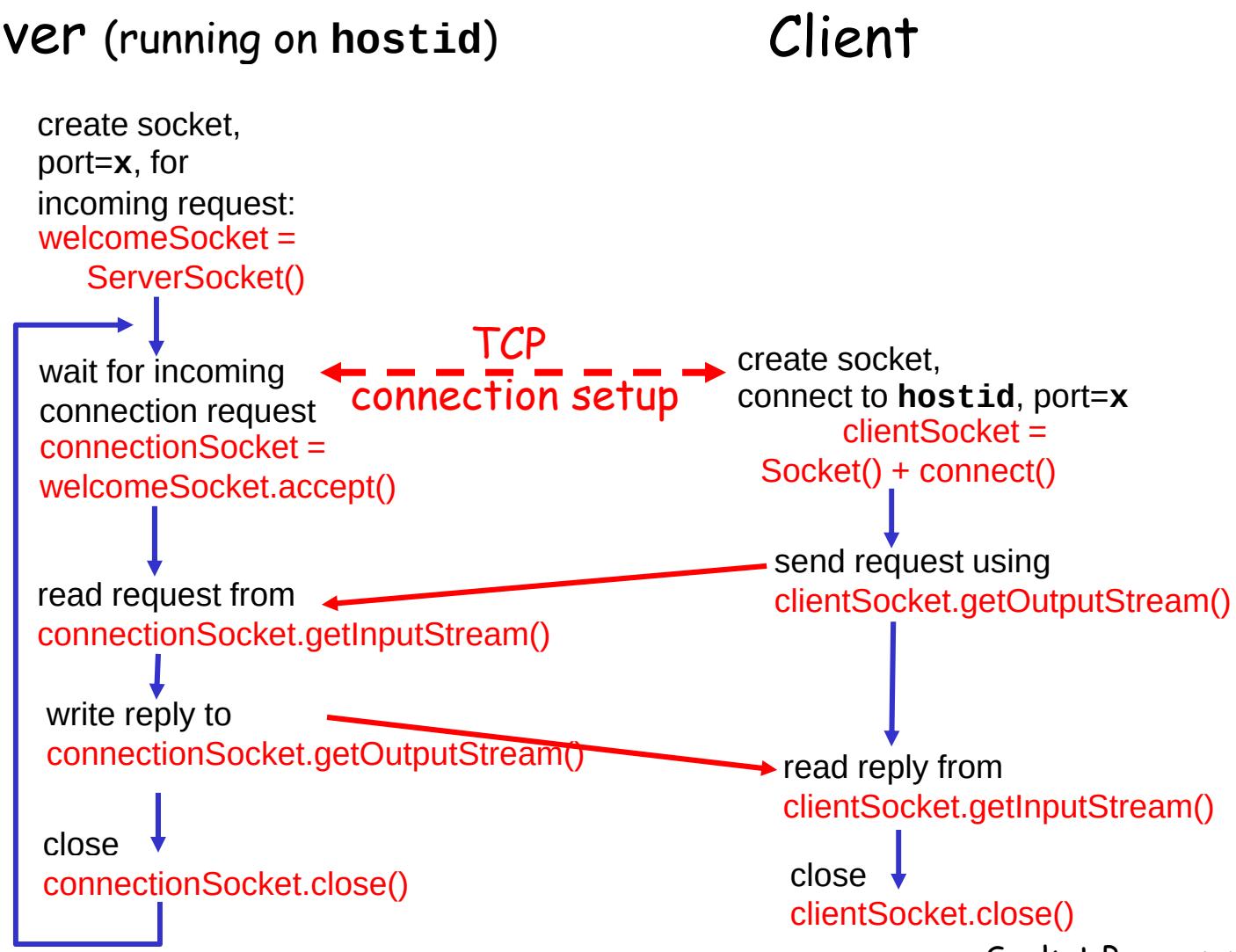
Socket programming with TCP

Example client-server app:

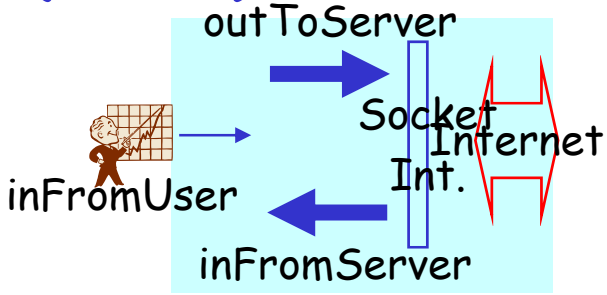
1) client reads line from standard input (**inFromUser** stream) , sends to server via socket (**outToServer** stream)
2) server reads line from socket
3) server converts line to uppercase, sends back to client
4) client reads, prints modified line from socket (**inFromServer** stream)



Client/server socket interaction: TCP



Example: Java Echo client (TCP)




```
import java.io.*;
import java.net.*;
class TCPClient {

    public static void main(String argv[]) throws Exception
    {
        String sentence;
        String modifiedSentence;

        BufferedReader inFromUser =
          new BufferedReader(new InputStreamReader(System.in));

        Socket clientSocket = new Socket("hostname", 6789);

        DataOutputStream outToServer =
          new DataOutputStream(clientSocket.getOutputStream());
```

Create input stream

Create client socket & connect to server

Create output stream attached to socket

Example: Java client (TCP), cont.

Create input stream attached to socket

```
BufferedReader inFromServer =
  new BufferedReader(new
  InputStreamReader(clientSocket.getInputStream()));
```

String 형태의 data만을 읽기 위한 방법

Byte -> String 형태로 변환 방법

```
sentence = inFromUser.readLine();
```

Send line to server

```
outToServer.writeBytes(sentence + '\n');
```

Read line from server

```
modifiedSentence = inFromServer.readLine();

System.out.println("FROM SERVER: " + modifiedSentence);

clientSocket.close();

    }
}
```

Example: Java Echo server (TCP)

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

class TCPServer {

  public static void main(String argv[]) throws Exception
   {
     String clientSentence;
     String capitalizedSentence;

     ServerSocket welcomeSocket = new ServerSocket(6789);

     while(true) {

       Socket connectionSocket = welcomeSocket.accept();

      BufferedReader inFromClient =
        new BufferedReader(new
        InputStreamReader(connectionSocket.getInputStream()));
```

Create welcoming socket at port 6789

Wait, on welcoming socket for contact by client

Create input stream, attached to socket

Example: Java server (TCP), cont

Create output stream, attached to socket

```
DataOutputStream  outToClient =
  new DataOutputStream(connectionSocket.getOutputStream());
```

Read in line from socket

```
clientSentence = inFromClient.readLine();

capitalizedSentence = clientSentence.toUpperCase() + '\n';
```

Write out line to socket

```
outToClient.writeBytes(capitalizedSentence);
            }
        }
    }
```

End of while loop, loop back and wait for another client connection

클래스 BufferedReader

Read text from a character-input stream, buffering characters so as to provide for the efficient reading of characters, arrays, and lines. For example
BufferedReader in = new BufferedReader(new FileReader("foo.in")); will buffer the input from the specified file. Without buffering, each invocation of read() or readLine() could cause bytes to be read from the file, converted into characters, and then returned, which can be very inefficient. Programs that use DataInputStreams for textual input can be localized by replacing each DataInputStream with an appropriate BufferedReader.

Constructor Summary
BufferedReader(Reader in) Create a buffering character-input stream that uses a default-sized input buffer.
BufferedReader(Reader in, int sz) Create a buffering character-input stream that uses an input buffer of the specified size.

Method Summary	
void	close() Close the stream.
void	mark(int readAheadLimit) Mark the present position in the stream.
boolean	markSupported() Tell whether this stream supports the mark() operation, which it does.
int	read() Read a single character.
int	read(char[] cbuf, int off, int len) Read characters into a portion of an array.
String	readLine() Read a line of text.
boolean	ready() Tell whether this stream is ready to be read.
void	reset() Reset the stream to the most recent mark.
long	skip(long n) Skip characters.

Application Layer Protocol

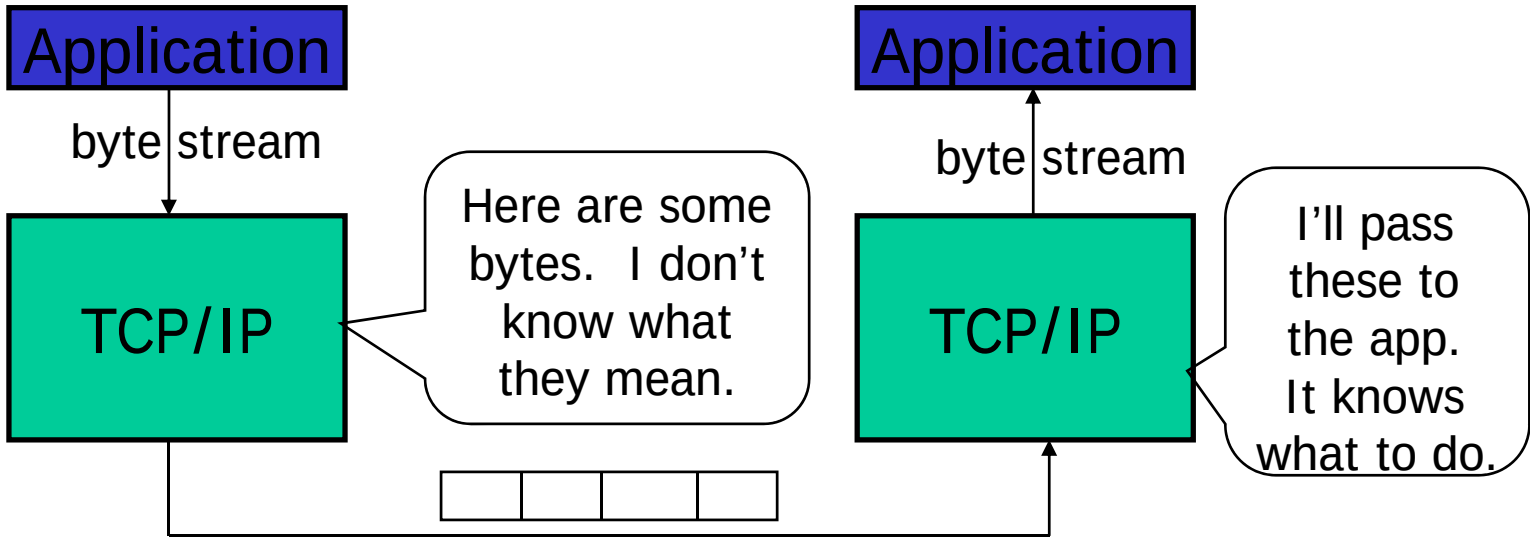
- Encode information in bytes
- Sender and receiver must agree on semantics
- Data encoding
  - Primitive types: strings, integers, and etc.
  - Composed types: message with fields

Constructing Messages

...beyond simple strings

TCP/IP Byte Transport

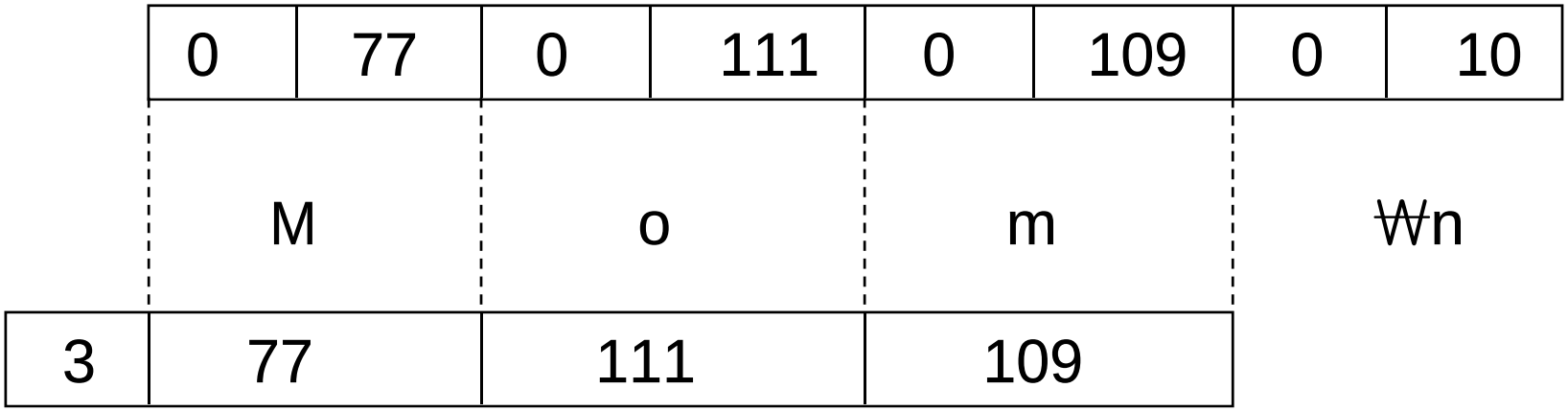
- TCP/IP protocols transports bytes



- Application protocol provides semantics

Primitive Types

- String
  - Character encoding: ASCII, Unicode, UTF
  - Delimit: length vs. termination character



Primitive Types

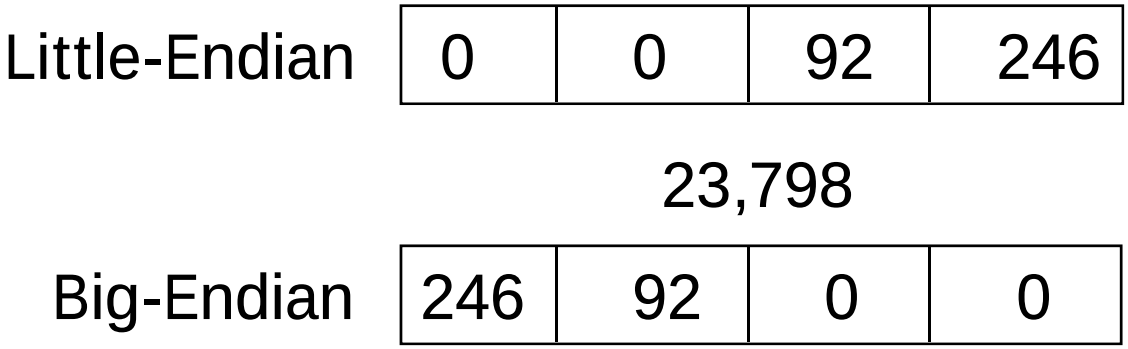
- Integer
  - Strings of character encoded decimal digits

49	55	57	57	56	55	48	10
'1'	'7'	'9'	'9'	'8'	'7'	'0'	₩n

- Advantage: 1. Human readable
  2. Arbitrary size
- Disadvantage: 1. Inefficient
  2. Arithmetic manipulation

Primitive Types

- Integer
  - Native representation



4-byte two's-complement integer

  - Network byte order (Big-Endian)
    - Use for multi-byte, binary data exchange
    - htonl(), htons(), ntohl(), ntohs()

Message Composition

- Message composed of fields
  - Fixed-length fields

integer	short	short

  - Variable-length fields

M	i	k	e		1	2	\n

Java Classes

...beyond simple Class

JAVA Socket-programming (java.net)

Goal: Java에서는 Network API 개발을 위한 java.net 클래스를 제공한다. 이 Socket 관련 class을 이용하여 임의의 서버와 클라이언트, 다중 캐스팅 서버 등을 개발함. 단, 자바는 네트워크 프로그래밍의 장점을 개발하기 쉽도록 API가 잘 발달되어 있음.

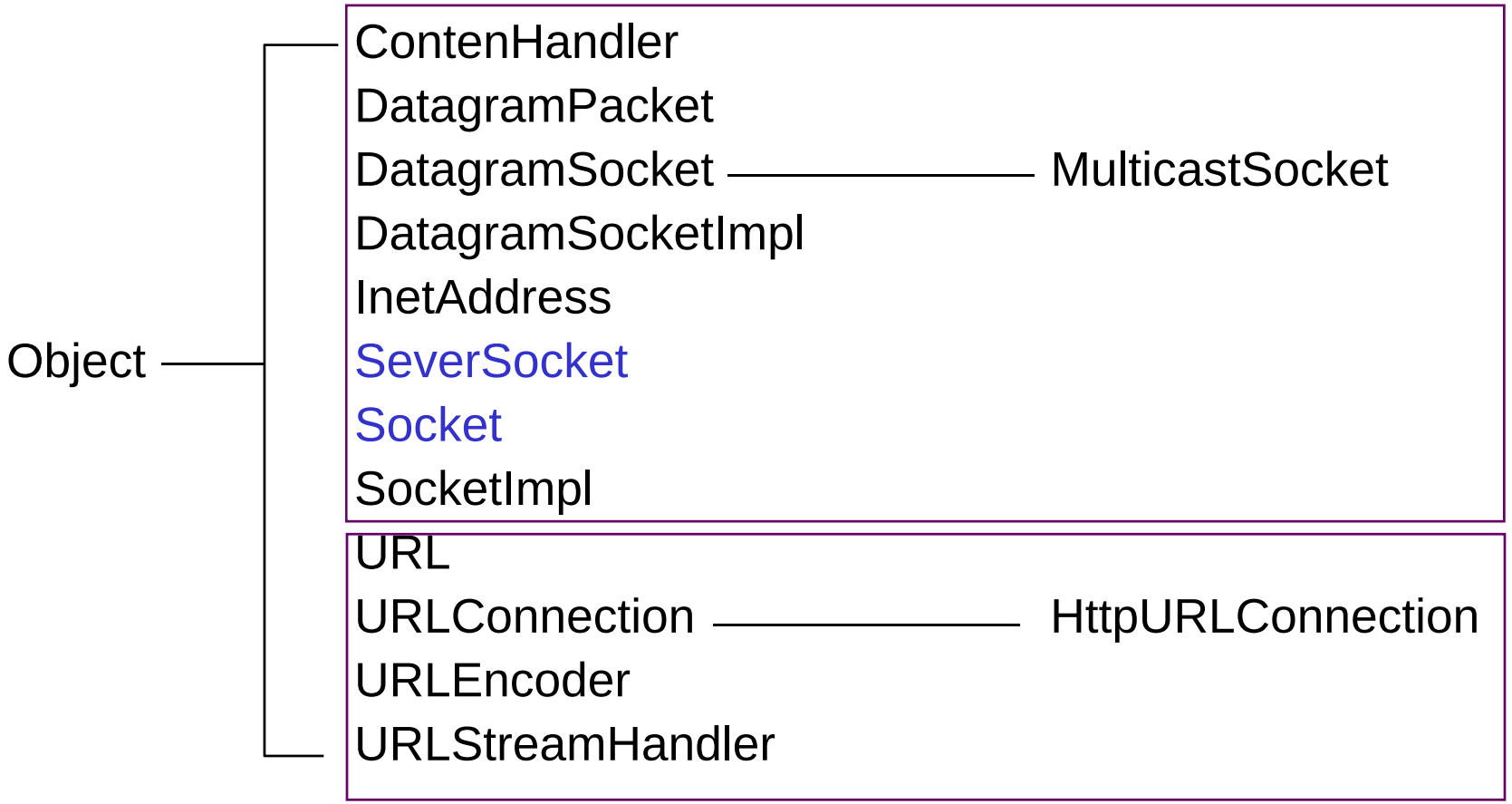
A low level API: Addresses (networking ID), Sockets, Interfaces

A High level API: URI, URL, Connections (connection to the resource pointed to by URLs)

인터페이스	클래스		예외클래스
ContentHandlerFactory	**InetAddress**	URL	BindException
DatagramSocketImplFactory	**DatagramSocket**	URLClassLoader	ConnectionException
FileNameMap	**DatagramPacket**	URLConnection	MalformedURLException
SocketImplFactory	DatagramSocketImpl	URLDecpder	NoRouteToHostException
SocketOptions	MulticastSocket	URLEncoder	ProtocolException
URLStreamHandlerFactory	NetPermission	URLStreamHandler	SocketException
	Authenticator	HttpURLConnection	UnknownHostException
	ServerSocket	JarURLConnection	UnknownServiceException
	Socket	ContentHandler	
	SocketImpl		
	SocketPermission		

네트워킹의 개요와 java.net 패키지

- Java.net 패키지



Low Level API

- The InetAddress class is the abstraction representing an IP (Internet Protocol) address,
  - Addresses are used throughout the java.net APIs as either host identifiers, or socket endpoint identifier.
- Sockets are means to establish a communication link between machines over the network. The java.net package provides 4 kinds of Sockets:
  - Socket is a TCP client API, and will typically be used to connect (java.net.Socket.connect(SocketAddress)) to a remote host.
  - ServerSocket is a TCP server API, and will typically accept (java.net.ServerSocket.accept) connections from client sockets.
  - DatagramSocket is a UDP endpoint API and is used to send, and receive, java.net.DatagramPackets.
  - MulticastSocket is a subclass of the DatagramSocket used when dealing with multicast groups.
- The NetworkInterface class provides APIs to browse and query all the networking interfaces (e.g. ethernet connection or PPP endpoint) of the local machine. It is through that class that you can check if any of the local interfaces is configured to support IPv6.

High Level API allow for easy access to resources on the network

- URI is the class representing a Universal Resource Identifier, as specified in RFC 2396. As the name indicates, this is just an Identifier and doesn't provide directly the means to access the resource.
- URL is the class representing a Universal Resource Locator, which is both an older concept for URIs and a mean to access the resources.
- URLConnection is created from a URL and is the communication link used to access the resource pointed by the URL. This abstract class will delegate most of the work to the underlying protocol handlers like http or ftp.
- HttpURLConnection is a subclass of URLConnection and provides some additional functionalities specific to the HTTP protocol.
- The recommended usage is to use URI to identify resources, then convert it into a URL when it is time to access the resource. From that URL, you can either get the URLConnection for fine control, or get directly the InputStream
  - URI uri = new URI("http://java.sun.com/");
    URL url = uri.toURL();
    InputStream in = url.openStream();

Class InetAddress

- This class represents an Internet Protocol (IP) address
  - Unicast (an identifier for a single interface)
  - Multicast (an identifier for a set of interfaces)

The textual representation of an IP address is address family specific.

The InetAddress class provides methods to resolve host names to their IP addresses and vise versa.

Host name-to-IP address *resolution* is accomplished through the use of a combination of local machine configuration information and network naming services such as the Domain Name System (DNS) and Network Information Service(NIS).

The InetAddress class has a cache to store successful as well as unsuccessful host name resolutions. The positive caching is there to guard against DNS spoofing attacks; while the negative caching is used to improve performance.

InetAddress Methods

Method Summary

byte[]	getAddress() Returns the raw IP address of this InetAddress object. 4byte 주소를 얻음
InetAddress	getLocalAddress() Gets the local address to which the socket is bound.
string	getHostName() Gets the host name for this IP address. 도메인 네임을 얻음
static InetAddress	getByAddress(byte[] addr) Returns an InetAddress object given the raw IP address .
static InetAddress[]	getAllByName(String host) Given the name of a host, returns an array of its IP addresses, based on the configured name service on the system.
string	getHostAddress() Returns the IP address string in textual presentation. Dotted decimal 주소를 얻음
static InetAddress[]	getLocalHost() Returns the local host.
boolean	isMulticastAddress() Utility routine to check if the InetAddress is an IP multicast address.
string	toString() Converts this IP address to a String.
static InetAddress	getByAddress(String host, byte[] addr) Create an InetAddress based on the provided host name and IP address No name service is checked for the validity of the address.

Class DatagramSocket

- DatagramSocket is a UDP endpoint API and is used to send, and receive, java.net.DatagramPackets.

This class represents a socket for sending and receiving datagram packets.

A datagram socket is the sending or receiving point for a packet delivery service. Each packet sent or received on a datagram socket is individually addressed and routed. Multiple packets sent from one machine to another may be routed differently, and may arrive in any order.

UDP broadcasts sends are always enabled on a DatagramSocket. In order to receive broadcast packets a DatagramSocket should be bound to the wildcard address. In some implementations, broadcast packets may also be received when a DatagramSocket is bound to a more specific address.

Example: DatagramSocket s = new DatagramSocket(null); s.bind(new InetSocketAddress(8888)); Which is equivalent to: DatagramSocket s = new DatagramSocket(8888); Both cases will create a DatagramSocket able to receive broadcasts on UDP port 8888.

DatagramSocket Constructor

- Constructs a datagram socket and binds it to any available port on the local host machine. The socket will be bound to the wildcard address, an IP address chosen by the kernel..

Constructor Summary

	DatagramSocket() Constructs a datagram socket and binds it to any available port on the local host machine.
protected	DatagramSocket(DatagramSocketImpl impl) Creates an unbound datagram socket with the specified DatagramSocketImpl.
	DatagramSocket(int port) Constructs a datagram socket and binds it to the specified port on the local host machine.
	DatagramSocket(int port, InetAddress laddr) Creates a datagram socket, bound to the specified local address.
	DatagramSocket(SocketAddress bindaddr) Creates a datagram socket, bound to the specified local socket address.

DatagramSocket Methods

- Constructs a datagram socket and binds it to any available port on the local host machine. The socket will be bound to the wildcard address, an IP address chosen by the kernel..

InetAddress	getInetAddress() Returns the address to which this socket is connected.
InetAddress	getLocalAddress() Gets the local address to which the socket is bound.
int	getLocalPort() Returns the port number on the local host to which this socket is bound.
void	receive(DatagramPacket p) Receives a datagram packet from this socket.
void	send(DatagramPacket p) Sends a datagram packet from this socket.
void	setBroadcast(boolean on) Enable/disable SO_BROADCAST.
static void	setDatagramSocketImplFactory(DatagramSocketImplFactory fac) Sets the datagram socket implementation factory for the application.
void	connect(InetAddress address, int port) Connects the socket to a remote address for this socket.
void	connect(SocketAddress addr) Connects this socket to a remote socket address (IP address + port number).
void	disconnect() Disconnects the socket.

Class DatagramPacket

- DatagramPacket is a connectionless packet delivery service.
  UDP Socket을 개설하기 위해서는 DatagramSocket 클래스를 사용하고 UDP packet을 송수신하기 위해서는 IP packet을 담는 DatagramPacket 클래스를 사용

Datagram packets are used to implement a connectionless packet delivery service. Each message is routed from one machine to another based solely on information contained within that packet. Multiple packets sent from one machine to another might be routed differently, and might arrive in any order. Packet delivery is not guaranteed.

DatagramPacket Constructor

- UDP packet을 송수신하려면 송수신용 DatagramPacket 객체를 만들고 이 객체를 이용하여 패킷을 송수신 함.
  - 송신용 DatagramPacket: DatagramPacket(byte[] buf, int length, InetAddress addr, int port)
  - 수신용 DatagramPacket: DatagramPacket(byte[] buf, int length)

DatagramPacket Methods

- 주요 Method

Method Summary

byte[]	getData()
InetAddress	getAddress() Gets the remote IP address
int	getLength() Returns the packet length
int	getPort() Returns the remote port number.
void	setPort(int p) sets destination port for the packet.
void	setData(byte[] buf) replace buf with new value.
void	setAddress(InetAddress address) sets the remote IP address for this packet.

인터페이스

- xxxFactory: 개체를 생성하기 위한 **공장**을 규정

ContentHandlerFactory	URL 로부터 읽어오는 자원의 내용을 처리하기 위한 컨텐트 핸들러를 만드는 Factory 클래스가 갖추어야 할 요건을 규정한다.
URLStreamHandlerFactory	URL 스트림 프로토콜 핸들러를 만드는 Factory 클래스가 갖추어야 할 요건을 규정한다.
SocketImplFactory	소켓 이행 클래스 인스턴스를 만드는 Factory 클래스가 갖추어야 할 요건을 규정한다.
DatagramSocketImplFactory	데이터그램 소켓 이행 클래스 인스턴스를 만드는 Factory 클래스가 갖추어야 할 요건을 규정한다.
SocketOptions	소켓이 갖추어야 할 옵션을 지정하고 구하는 메쏘드들을 모아둔 인터페이스로, SocketImpl 및 DatagramSocketImpl 클래스에 의해 이행된다. 따라서 사용자 자신의 소켓을 만들기 위해서는 이 두 클래스를 확장하여 그 메쏘드들을 덮어쓰기하면 된다.
FileNameMap	파일명과 MIME 유형을 지정하는 스트링을 대응시키는(mapping) 메커니즘을 제공하는 인터페이스다.

클래스-URL 프로그래밍 관련

URI	Represents a Uniform Resource Identifier (URI) reference.
URL	Class URL represents a Uniform Resource Locator, a pointer to a "resource" on the World Wide Web.
URLClassLoader	This class loader is used to load classes and resources from a search path of URLs referring to both JAR files and directories.
URLConnection	The abstract class URLConnection is the superclass of all classes that represent a communications link between the application and a URL.
URLDecoder	Utility class for HTML form decoding.
URLEncoder	Utility class for HTML form encoding.
URLStreamHandler	The abstract class URLStreamHandler is the common superclass for all stream protocol handlers.
HttpURLConnection	A URLConnection with support for HTTP-specific features.
JarURLConnection	A URL Connection to a Java ARchive (JAR) file or an entry in a JAR file.
ContentHandler	The abstract class ContentHandler is the superclass of all classes that read an Object from a URLConnection

클래스-UDP 프로그래밍 관련

DatagramPacket	This class represents a datagram packet.
DatagramSocket	This class represents a socket for sending and receiving datagram packets.
DatagramSocketImpl	**Abstract** datagram and multicast socket implementation base class.
MulticastSocket	The multicast datagram socket class is useful for sending and receiving IP multicast packets.

클래스-TCP 프로그래밍 관련

ServerSocket	This class implements server sockets.
Socket	This class implements client sockets (also called just "sockets").
SocketImpl	The abstract class SocketImpl is a common superclass of all classes that actually implement sockets.
SocketPermission	This class represents access to a network via sockets.
SocketAddress	This class represents a Socket Address with no protocol attachment.
InetSocketAddress	This class implements an IP Socket Address (IP address + port number) It can also be a pair (hostname + port number), in which case an attempt will be made to resolve the hostname.

Class ServerSocket

http://java.sun.com/j2se/1.5.0/docs/api/java/net/ServerSocket.html

- public class **ServerSocket** extends Object
- A server socket waits for requests to come in over the network. It performs some operation based on that request, and then possibly returns a result to the requester. The actual work of the server socket is performed by an instance of the SocketImpl class. An application can change the socket factory that creates the socket implementation to configure itself to create sockets appropriate to the local firewall.

Constructor Summary
ServerSocket() Creates an unbound server socket.
ServerSocket(int port) Creates a server socket, bound to the specified port.
ServerSocket(int port, int backlog) Creates a server socket and binds it to the specified local port number, with the specified backlog.
ServerSocket(int port, int backlog, InetAddress bindAddr) Create a server with the specified port, listen backlog, and local IP address to bind to.

ServerSocket method

Method Summary	
Socket	accept() Listens for a connection to be made to this socket and accepts it.
void	bind(SocketAddress endpoint) Binds the ServerSocket to a specific address (IP address and port number).
void	bind(SocketAddress endpoint, int backlog) Binds the ServerSocket to a specific address (IP address and port number).
void	close() Closes this socket.
ServerSocketC	getChannel() Returns the unique ServerSocketChannel object associated with this socket, if any.
InetAddress	getInetAddress() Returns the local address of this server socket.
int	getLocalPort() Returns the port on which this socket is listening.
SocketAddress	getLocalSocketAddress() Returns the address of the endpoint this socket is bound to, or null if it is not bound yet.
int	getReceiveBufferSize() Gets the value of the SO_RCVBUF option for this ServerSocket, that is the proposed buffer size that will be used for Sockets accepted from this ServerSocket.
boolean	getReuseAddress() Tests if SO_REUSEADDR is enabled.
int	getSoTimeout() Retrive setting for SO_TIMEOUT.
protected void	implAccept(Socket s) Subclasses of ServerSocket use this method to override accept() to return their own subclass of socket.
boolean	isBound() Returns the binding state of the ServerSocket.
boolean	isClosed() Returns the closed state of the ServerSocket.
void	setPerformancePreferences(int connectionTime, int latency, int bandwidth) Sets performance preferences for this ServerSocket.
void	setReceiveBufferSize(int size) Sets a default proposed value for the SO_RCVBUF option for sockets accepted from this ServerSocket.
void	setReuseAddress(boolean on) Enable/disable the SO_REUSEADDR socket option.
static void	setSocketFactory(SocketImplFactory fac) Sets the server socket implementation factory for the application.
void	setSoTimeout(int timeout) Enable/disable SO_TIMEOUT with the specified timeout, in milliseconds.
String	toString() Returns the implementation address and implementation port of this socket as a String.

Class Socket

- public class **Socket** extends Object
- This class implements client sockets (also called just "sockets"). A socket is an endpoint for communication between two machines.
- The actual work of the socket is performed by an instance of the SocketImpl class. An application, by changing the socket factory that creates the socket implementation, can configure itself to create sockets appropriate to the local firewall.

Constructor Summary	
	Socket() Creates an unconnected socket, with the system-default type of SocketImpl.
	Socket(InetAddress address, int port) Creates a stream socket and connects it to the specified port number at the specified IP address.
	Socket(InetAddress host, int port, boolean stream) **Deprecated.** *Use DatagramSocket instead for UDP transport.*
	Socket(InetAddress address, int port, InetAddress localAddr, int localPort) Creates a socket and connects it to the specified remote address on the specified remote port.
	Socket(Proxy proxy) Creates an unconnected socket, specifying the type of proxy, if any, that should be used regardless of any other settings.
protecte	Socket(SocketImpl impl) Creates an unconnected Socket with a user-specified SocketImpl.
	Socket(String host, int port) Creates a stream socket and connects it to the specified port number on the named host.
	Socket(String host, int port, boolean stream) **Deprecated.** *Use DatagramSocket instead for UDP transport.*
	Socket(String host, int port, InetAddress localAddr, int localPort) Creates a socket and connects it to the specified remote host on the specified remote port.

클래스-기타 네트워크 관련

Authenticator	The class Authenticator represents an object that knows how to obtain authentication for a network connection.
NetPermission	This class is for various network permissions.
PasswordAuthentication	The class PasswordAuthentication is a data holder that is used by Authenticator.
Proxy	This class represents a proxy setting, typically a type (http, socks) and a socket address.
ProxySelector	Selects the proxy server to use, if any, when connecting to the network resource referenced by a URL.
CacheRequest	Represents channels for storing resources in the ResponseCache.
CacheResponse	Represent channels for retrieving resources from the ResponseCache.
ResponseCache	Represents implementations of URLConnection caches.
SecureCacheResponse	Represents a cache response originally retrieved through secure means, such as TLS.
CookieHandler	A CookieHandler object provides a callback mechanism to hook up a HTTP state management policy implementation into the HTTP protocol handler.
Inet4Address	This class represents an Internet Protocol version 4 (IPv4) address.
Inet6Address	This class represents an Internet Protocol version 6 (IPv6) address.
InetAddress	This class represents an Internet Protocol (IP) address.
NetworkInterface	This class represents a Network Interface made up of a name, and a list of IP addresses assigned to this interface.

Class NetworkInterface

- public final class **NetworkInterface** extends Object
- This class represents a Network Interface made up of a name, and a list of IP addresses assigned to this interface. It is used to identify the local interface on which a multicast group is joined. Interfaces are normally known by names such as "le0".

Method Summary	
boolean	equals(Object obj) Compares this object against the specified object.
static NetworkInterface	getByInetAddress(InetAddress addr) Convenience method to search for a network interface that has the specified Internet Protocol (IP) address bound to it.
static NetworkInterface	getByName(String name) Searches for the network interface with the specified name.
String	getDisplayName() Get the display name of this network interface.
Enumeration<InetAddress>	getInetAddresses() Convenience method to return an Enumeration with all or a subset of the InetAddresses bound to this network interface.
String	getName() Get the name of this network interface.
static Enumeration<NetworkInterface>	getNetworkInterfaces() Returns all the interfaces on this machine.
int	hashCode() Returns a hash code value for the object.
String	toString() Returns a string representation of the object.

Project1: Building your own chatting program

Ref:
TCP/IP Sockets in Java: Practical Guide for Programmers

Kenneth L. Calvert
Michael J. Donahoo

TCP Client/Server Interaction

Server starts by getting ready to receive client connections...

Client

1. Create a TCP socket
2. Communicate
3. Close the connection

Server

1. Create a TCP socket
2. Repeatedly:
   a. Accept new connection
   b. Communicate
   c. Close the connection

TCP Client/Server Interaction

ServerSocket servSock = new ServerSocket(servPort);

Client

1. Create a TCP socket
2. Communicate
3. Close the connection

Server

1. Create a TCP socket
2. Repeatedly:
   a. Accept new connection
   b. Communicate
   c. Close the connection

TCP Client/Server Interaction

```
for (;;) {
    Socket clntSock =   servSock.accept();
```

Client

1. Create a TCP socket
2. Communicate
3. Close the connection

Server

1. Create a TCP socket
2. Repeatedly:
   a. Accept new connection
   b. Communicate
   c. Close the connection

TCP Client/Server Interaction

Server is now blocked waiting for connection from a client

Client

1. Create a TCP socket
2. Communicate
3. Close the connection

Server

1. Create a TCP socket
2. Repeatedly:
   a. Accept new connection
   b. Communicate
   c. Close the connection

TCP Client/Server Interaction

Later, a client decides to talk to the server...

Client

1. Create a TCP socket
2. Communicate
3. Close the connection

Server

1. Create a TCP socket
2. Repeatedly:
   a. Accept new connection
   b. Communicate
   c. Close the connection

TCP Client/Server Interaction

Socket socket = new Socket(server, servPort);

Client

1. Create a TCP socket
2. Communicate
3. Close the connection

Server

1. Create a TCP socket
2. Repeatedly:
   a. Accept new connection
   b. Communicate
   c. Close the connection

TCP Client/Server Interaction

```
OutputStream out = socket.getOutputStream();
out.write(byteBuffer);
```

Client

1. Create a TCP socket
2. Communicate
3. Close the connection

Server

1. Create a TCP socket
2. Repeatedly:
   a. Accept new connection
   b. Communicate
   c. Close the connection

TCP Client/Server Interaction

Socket clntSock = servSock.accept();

Client

1. Create a TCP socket
2. Communicate
3. Close the connection

Server

1. Create a TCP socket
2. Repeatedly:
   a. Accept new connection
   b. Communicate
   c. Close the connection

TCP Client/Server Interaction

```
InputStream in = clntSock.getInputStream();
recvMsgSize = in.read(byteBuffer);
```

Client

1. Create a TCP socket
2. Communicate
3. Close the connection

Server

1. Create a TCP socket
2. Repeatedly:
   a. Accept new connection
   b. Communicate
   c. Close the connection

TCP Client/Server Interaction

close(sock);

close(clntSocket)

Client

1. Create a TCP socket
2. Establish connection
3. Communicate
4. Close the connection

Server

1. Create a TCP socket
2. Bind socket to a port
3. Set socket to listen
4. Repeatedly:
   a. Accept new connection
   b. Communicate
   c. Close the connection

TCP Tidbits

- Client knows server address and port
- No correlation between `send()` and `recv()`

Client	Server
out.write("Hello Bob")	
	in.read() -> "Hello "
	in.read() -> "Bob"
	out.write("Hi ")
	out.write("Jane")
in.read() -> "Hi Jane"	

Closing a Connection

- close() used to delimit communication
- Analogous to EOF

Client	Server
out.write(*string*)	
	in.read(*buffer*)
while (not received entire string)	while(client has not closed connection)
in.read(*buffer*)	out.write(*buffer*)
out.write(*buffer*)	in.read(*buffer*)
close(*socket*)	
	close(*client socket*)