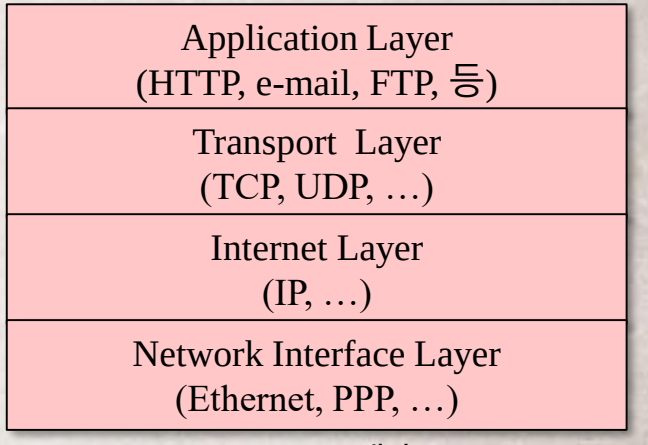
네트워크(TCP/IP)

comsi.java@gmail.com
박 경 태

TCP/IP 소개

- TCP 프로토콜
  - TCP는 Transmission Control Protocol
  - 두 시스템 간에 신뢰성 있는 데이터의 전송을 관장하는 프로토콜
    - 헤더체크섬은 TCP Header에 있는 손상을 감지
    - TCP는 각 패킷이 송신된 순서대로 수신되도록 책임을 짐
  - TCP에서 동작하는 응용프로그램 사례
    - e-mail, FTP, 웹(HTTP) 등



TCP/IP 4계층

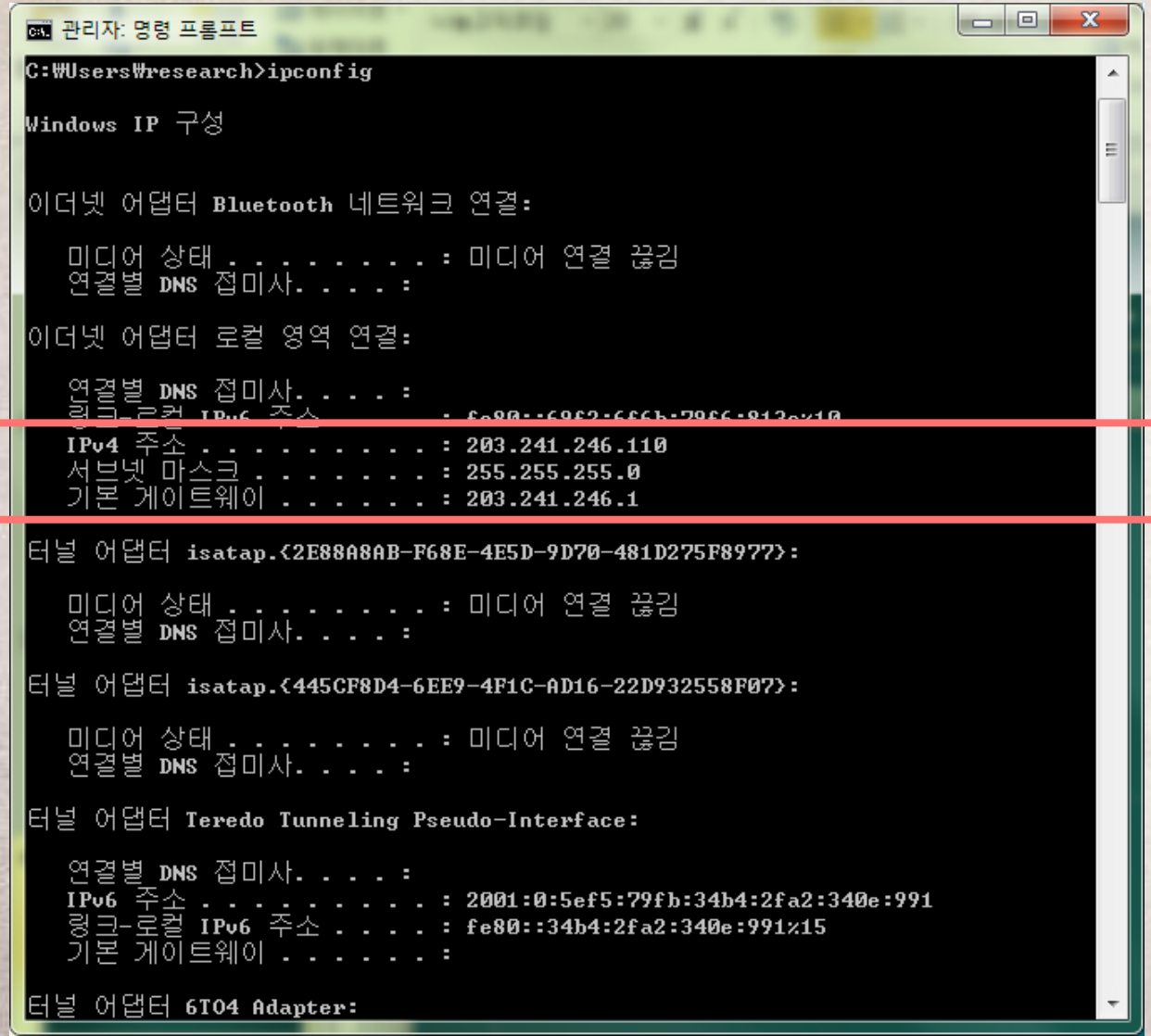
- IP
  - Internet Protocol
  - 통신 노드 간의 IP패킷을 전송하는 기능과 라우팅 기능을 담당하는 프로토콜
  - TCP보다 하위 레벨 프로토콜

IP 주소

- IP 주소
  - 네트워크 상에서 유일하게 식별될 수 있는 컴퓨터 주소
    - 숫자로 구성된 주소
    - 4개의 숫자가 '.'으로 연결
      - 예) 192.156.11.15
  - 숫자로 된 주소는 기억하기 어려우므로 "www.naver.com"과 같은 문자열로 구성된 도메인 이름으로 바꿔 사용
    - DNS(Domain Name Server)
      - 문자열로 구성된 도메인 이름을 숫자로 구성된 IP 주소로 자동 변환
  - 현재는 32비트의 IP 버전 4(IPv4)가 사용되고 있음
    - IP 주소 고갈로 인해 128비트의 IP 버전 6(IPv6)이 점점 사용되는 추세

내 컴퓨터의 IP 주소 확인하기

- 내 컴퓨터의 윈도우에서 명령창을 열어 ipconfig 명령 수행

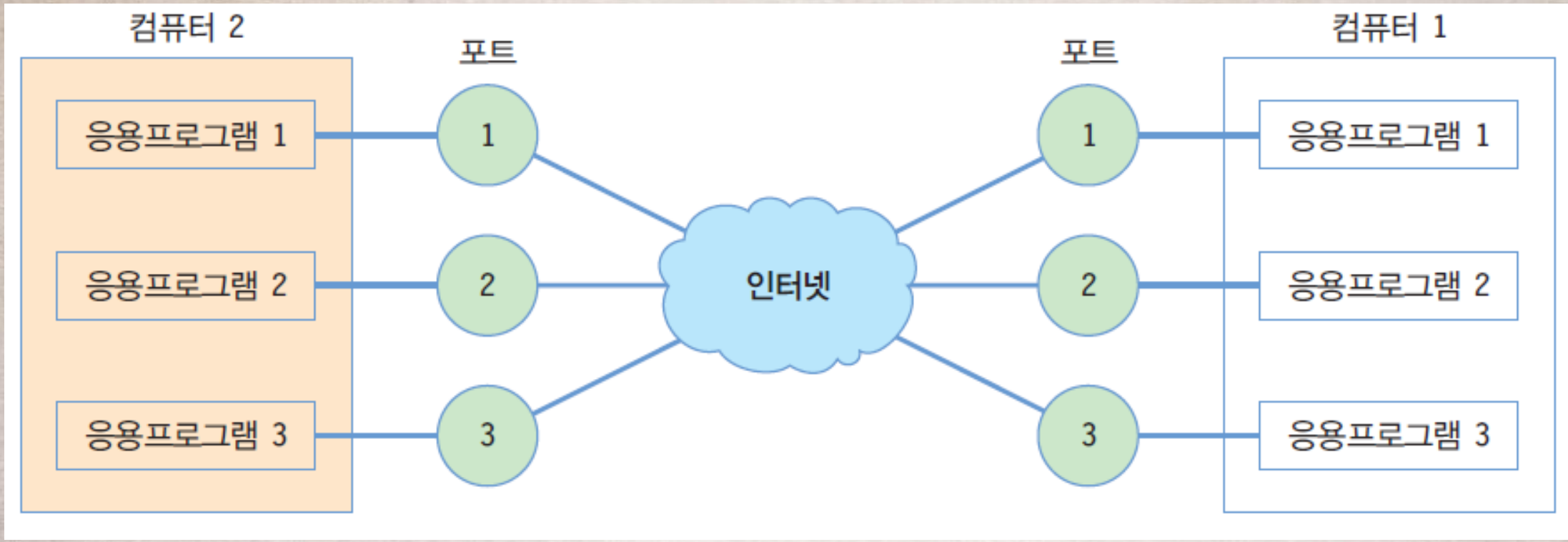


포트

- 포트
  - 통신하는 프로그램 간에 가상의 연결단 인 포트 생성
    - IP 주소는 네트워크 상의 컴퓨터 또는 시스템을 식별하는 주소
    - 포트 번호를 이용하여 통신할 응용프로그램 식별
  - 모든 응용프로그램은 하나 이상의 포트 생성 가능
    - 포트를 이용하여 상대방 응용프로그램과 데이터 교환
  - 잘 알려진 포트(well-know ports)
    - 시스템이 사용하는 포트 번호
    - 잘 알려진 응용프로그램에서 사용하는 포트 번호
      - 0부터 1023 사이의 포트 번호

      ex) 텔넷 23, HTTP 80, FTP 21
    - 잘 알려진 포트 번호는 개발자가 사용하지 않는 것이 좋음
      - 충돌 가능성 있음

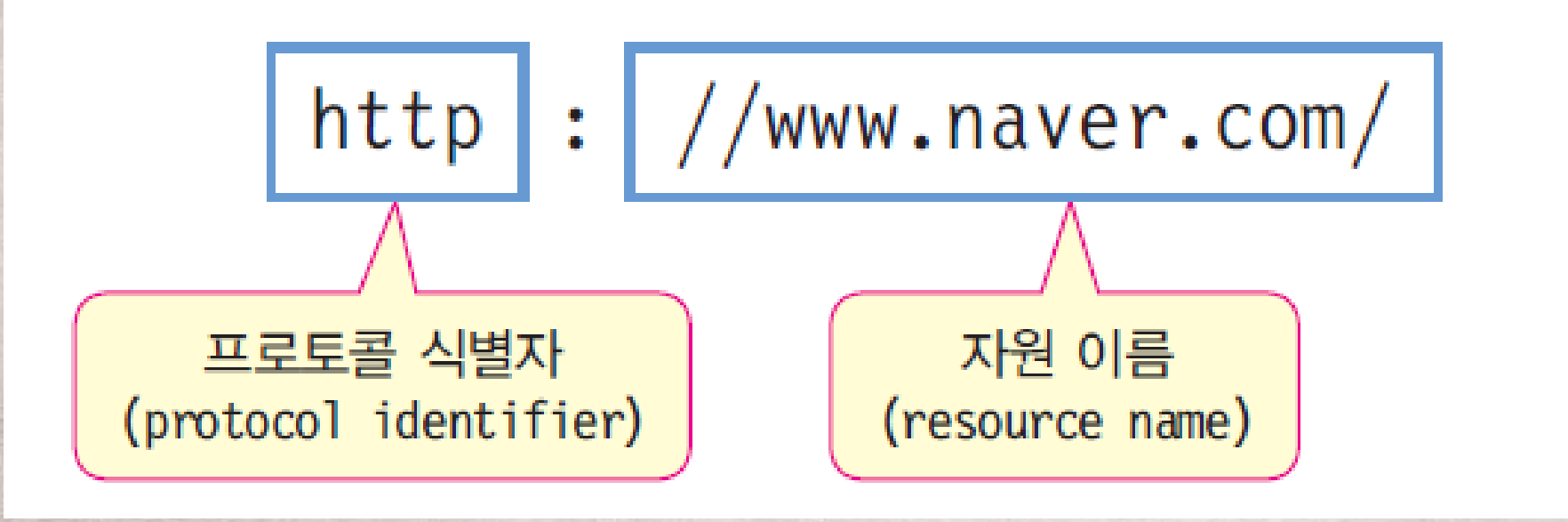
포트를 이용한 통신


컴퓨터 2
포트
포트
컴퓨터 1
응용프로그램 1
응용프로그램 2
응용프로그램 3
1
2
3
인터넷
1
2
3
응용프로그램 1
응용프로그램 2
응용프로그램 3

URL을 이용한 웹 프로그래밍

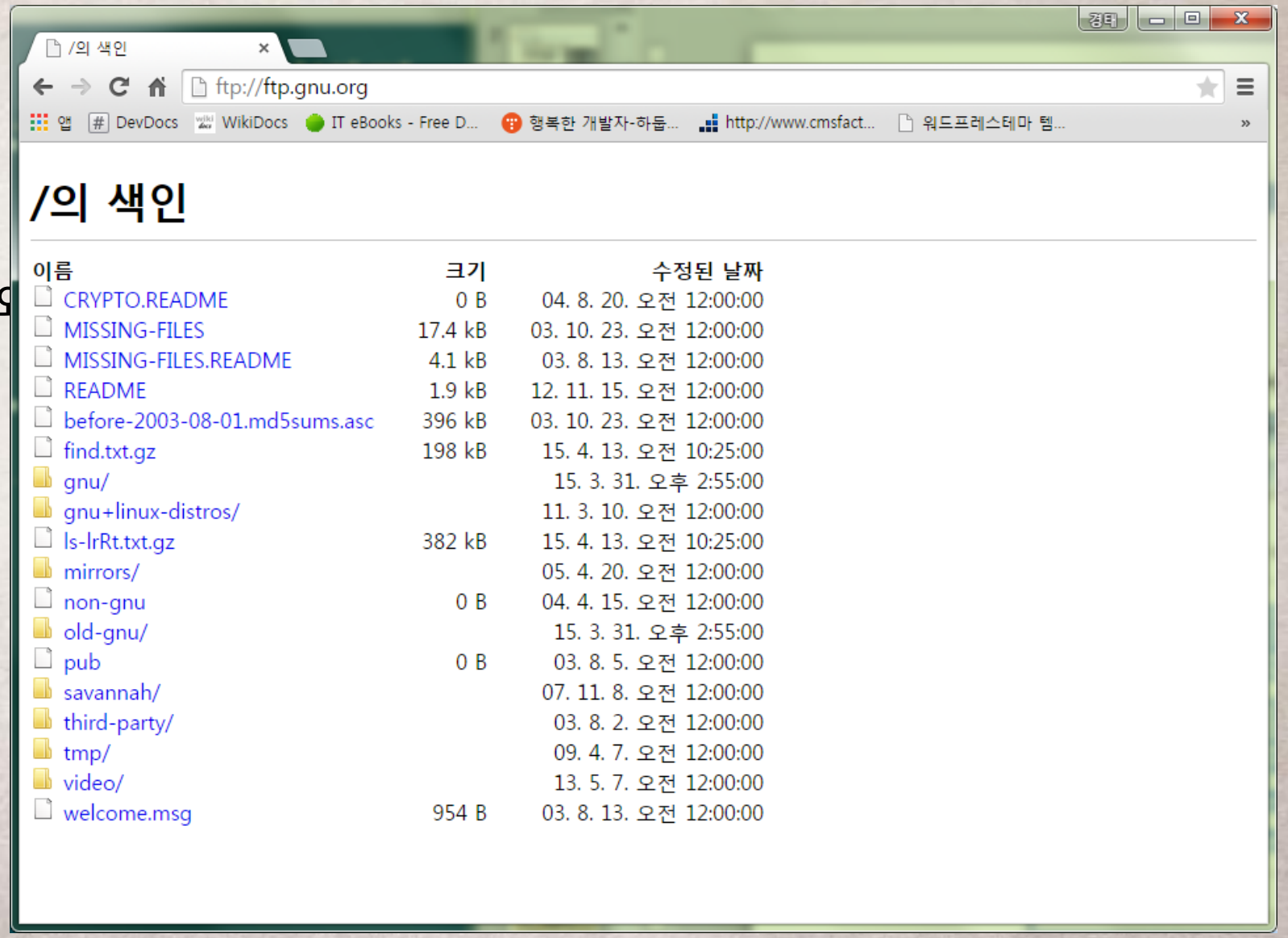
- URL이란?
  - URL은 Uniform Resource Locator
  - 인터넷 상의 리소스에 대한 주소(HTML 문서, 이미지, …)

- URL 구조



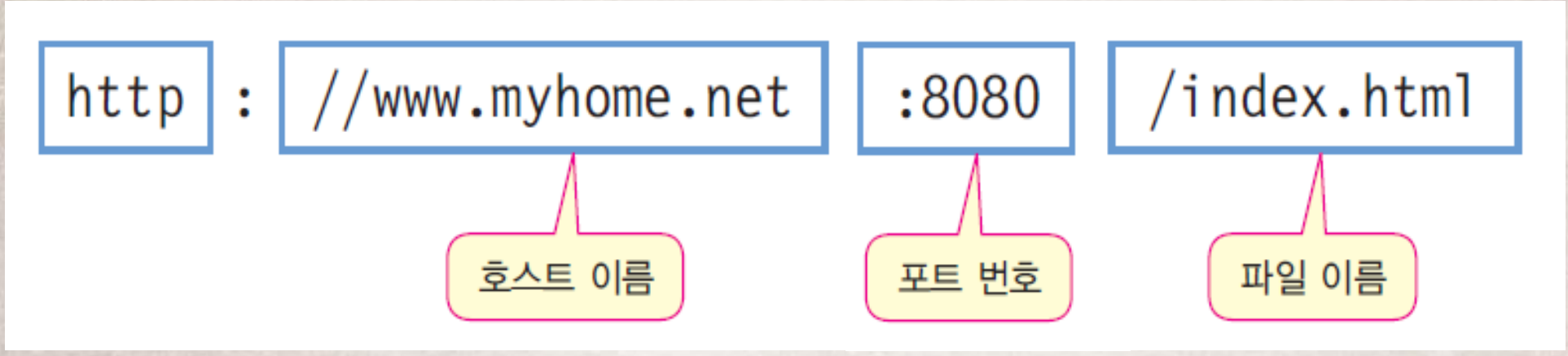
프로토콜 식별자

- 프로토콜 식별자
  - 인터넷상의 자원을 가져올
  - 종류
    - HTTP, FTP, TELNET
  - 대부분의 브라우저들은 HTTP 오



자원 이름

- 자원 이름
  - 자원 이름은 사용되는 프로토콜에 따라서 그 구성이 달라짐



HTTP의 경우

자바의 URL 클래스

- URL 클래스
  - java.net 패키지에 포함
  - 웹 상의 자원을 지정하는 URL을 나타냄

생성자	설 명
URL(String spec)	문자열이 지정하는 자원에 대한 URL 객체를 생성
URL(String protocol, String host, int port, String file)	프로토콜 식별자, 호스트 주소, 포트 번호, 파일 이름이 지정하는 자원에 대한 URL 객체 생성
URL(String protocol, String host, String file)	프로토콜 식별자, 호스트 주소, 파일 이름이 지정하는 자원에 대한 URL 객체 생성
URL(URL context, String spec)	URL context객체 내에 spec에서 지정하는 자원에 대한 URL 객체 생성

자바의 URL 클래스의 주요 메소드

- 주요 메소드

메소드	설 명
Object getContent()	URL의 컨텐트를 반환
String getFile()	URL 주소의 파일 이름 반환
String getHost()	URL 주소의 호스트 이름 반환
String getPath()	URL 주소의 경로 부분 반환
int getPort()	URL 주소의 포트 번호 반환
int getLocalPort()	소켓이 연결된 로컬 포트 번호 반환
int getPort()	소켓이 연결한 서버의 포트 번호 반환
InputStream openStream()	URL에 대해 연결을 설정하고 이 연결로부터 입력을 받을 수 있는 InputStream 객체 반환
URLConnection openConnection()	URL 주소의 원격 객체에 접속한 뒤 통신할 수 있는 URLConnection 객체 리턴

예제 7-1 : URL 파싱하기

- U 름, 호스트 주소, 포트 번

BufferedIOEx.java | FileClassExample.java | ParseURL.java

```java
import java.net.MalformedURLException;
import java.net.URL;


public class ParseURL {

    public static void main(String[] args) {
        URL opinion = null;
        URL homepage = null;

        try{
            homepage = new URL("http://news.hankooki.com:80");
            opinion = new URL(homepage, "opionion/editorial.html");
        }catch(MalformedURLException e){
            System.out.println("잘못된 URL입니다.");
        }

        System.out.println("protocol = "+opinion.getProtocol());
        System.out.println("host = "+opinion.getHost());
        System.out.println("port = "+opinion.getPort());
        System.out.println("path = "+opinion.getPath());
        System.out.println("filename = "+opinion.getFile());

    }

}
```

```
protocol = http
host = news.hankooki.com
port = 80
path = /opionion/editorial.html
filename = /opionion/editorial.html
```

URLConnection 클래스

- URLConnection 클래스
  - 주어진 원격지의 주소 URL에 네트워크 접속 후 데이터를 보내거나 받을 수 있도록 하는 기능
  - URL 객체 생성 방법
    - URL.openConnection() 이용

```
URL aURL = new URL("http://www.naver.com");
URLConnection uc = aURL.openConnection(); // 원격지와 연결한다.
```

    - URLConnection 생성자 이용

```
URL aURL = new URL("http://www.naver.com");
URLConnection uc = new URLConnection(aURL);
uc.connect(); // 원격지와 연결한다.
```

      - 연결하기 전에 여러 가지 인자들과 요청과 관련된 속성들을 설정 가능

URLConnection 클래스 주요 메소드

메소드	설명
abstract void connect()	URL에 의해 참조되는 외부 리소스와 통신 연결 설정
Object getContent()	URL 연결에서 컨텐트를 가져옴
String getContentEncoding()	컨텐트 인코딩 필드를 반환
int getContentLength()	컨텐트 길이 필드 반환
String getContentType()	컨텐트 타입 필드 반환
boolean getDoInput()	URLConnection 객체의 doInput 필드 값 반환
boolean getDoOutput()	URLConnection 객체의 doOutput 필드 값 반환
InputStream getInputStream()	설정된 연결에서 데이터를 읽을 입력 스트림 반환
OutputStream getOutputStream()	설정된 연결로 데이터를 출력할 출력 스트림 반환
URL getURL()	URLConnection 객체의 URL 필드 값 반환
void setDoInput(boolean doInput)	URLConnection 객체의 doInput 필드 값 설정
void setDoOutput(boolean doOutput)	URLConnection 객체의 doOutput 필드 값 설정

- doInput 필드가 true로 설정되면 URLConnection 객체로 표현되는 URL 연결이 입력을 위해 사용됨을 의미.
- doOutput 필드가 true로 설정되면 출력을 위해 사용됨을 의미

URLConnection 객체를 이용하여 원격지 데이터 받기

- URLConnection 객체에서 데이터 읽기
  - URLConnection 객체에서 getInputStream() 메소드를 이용하여 입력 스트림을 얻은 후에 스트림 입력을 수행

예제 7-2 : URLConnection으로 원격지에서 데이터 읽기

- 하여 데이터를

*URLConnectionReader.java

```
1  import java.io.BufferedReader;
2  import java.io.IOException;
3  import java.io.InputStreamReader;
4  import java.net.URL;
5  import java.net.URLConnection;
6
7
8  public class URLCor
9
10     public static v
11         try{
12             URL url
13
14             URLConn
15             Buffere
16             String
17
18             while((
19                 Sys
20
21             }
22             in.clos
23         }catch(IOE
24             System.
25         }
26
27     }
28
29 }
30
```

<terminated> URLConnectionReader [Java Application] C:₩Program Files₩Java₩jre1.8.0_31₩bin₩javaw.exe (2015. 4. 14. 오후 1:58:42)

```
.vertical_song .link_more {position:absolute;top:18px;right:11px;padding-right:7px;font-size:11px;line-height:14px;back
.tiles_story .vertical_reader {display:inline;float:right;padding:0 1px;background-color:#e1e1e1}
.vertical_reader .wrap_bestreader {overflow:hidden;position:relative;width:188px;height:339px;border-top:1px solid #e1e
.vertical_reader .tit_bestreader {display:block;overflow:hidden;width:174px;height:29px;padding:18px 0 0 14px;line-heig
.vertical_reader .list_thumb {overflow:hidden;width:188px;padding-top:15px;border-top:1px solid #ececec}
.vertical_reader .list_thumb li {float:none;margin:0;padding:0 15px 20px;font-size:0}
.vertical_reader .list_thumb .link_cont {width:158px;height:50px;cursor:pointer}
.vertical_reader .list_thumb .info_thumb {float:left;margin-right:10px}
.vertical_reader .list_thumb .info_cont {float:none}
.vertical_reader .info_thumb .img_daum {overflow:hidden;position:absolute;top:0;left:0;z-index:1;width:15px;height:15px
.vertical_reader .info_thumb .num1 {background-position:-290px -55px}
.vertical_reader .info_thumb .num2 {background-position:-310px -55px}
.vertical_reader .info_thumb .num3 {background-position:-330px -55px}
.vertical_reader .info_thumb .num4 {background-position:-350px -55px}
.vertical_reader .frame_g {width:58px;height:48px}
.vertical_reader .tit_story {display:block;overflow:hidden;float:left;width:88px;height:34px;margin-top:-1px;font-size
.vertical_reader .readers_info {display:block;font-size:11px;line-height:14px;color:#878787}
.vertical_reader .readers_info .ico_daum {float:left;width:15px;height:13px;margin:4px 2px 0 0;text-indent:-9999px}
.vertical_reader .readers_info .ico_reader {background-position:-80px -250px}
.vertical_reader .readers_info .num_reader {overflow:hidden;float:left;width:71px;margin-top:4px;white-space:nowrap;tex
.vertical_reader .link_more {position:absolute;top:18px;right:11px;padding-right:7px;font-size:11px;line-height:14px;ba
.layout_headline_vertical .vertical_img {float:left}
.layout_vertical .default_img .box_tit {height:43px;padding-top:12px}
.layout_promotion .default_img .box_tit {height:43px;padding-top:12px}
.wrap_cafe .layout_promotion .default_img .box_tit {height:45px;padding-top:10px}
.wrap_storyball .ico_newarr {background-position:0 -35px}
.wrap_storyball .ico_recom {background-position:-55px -35px}
```

소켓 프로그래밍

- 소켓 (socket)
  - 소켓은 네트워크 상에서 수행되는 두 프로그램 간의 양방향 통신 링크의 한쪽 끝 단을 의미
  - 소켓은 특정 포트 번호와 연결되어 있음
    - TCP에서 데이터를 보낼 응용프로그램을 식별할 수 있음.
  - 자바에서의 데이터 통신 시 소켓 사용
  - 소켓 종류
    - 서버 소켓과 클라이언트 소켓

소켓 주소

- IP는 호스트와의 통신을 확인하기 위해 32비트 이진 주소 사용한다.
  - 자바에서 주소는 숫자형 주소(169.1.1.1)이나 이름(comsi.inje.ac.kr)같은 스트링을 사용하여 표현
- 클라이언트는 반드시 서버 프로그램이 통신을 시작할 때 작동되는 호스트의 IP 주소를 알아야 한다.
- InetAddress 클래스를 이용해 IP주소 확인
  - getByNames()나 getAllByNames()는 이름이나 IP주소를 가지고 이에 상응하는 InetAddress 반환
  - getLocalHost()는 지역 호스트 주소를 포함하는 InetAddress인스턴스 반환

예제7-3.소켓 주소 알아보기

InetAddressEx.java

```java
import java.net.*;

public class InetAddressEx {

    public static void main(String[] args) {
        try{
            InetAddress address = InetAddress.getLocalHost();
            System.out.println("Local Host:");
            System.out.println("\t"+address.getHostName());
            System.out.println("\t"+address.getHostAddress());
        }catch(UnknownHostException e){
            System.out.println("Unable to determine this host's address");
        }

        for(int i = 0; i < args.length; i++){
            try{
                InetAddress[] addressList = InetAddress.getAllByName(args[i]);
                System.out.println(args[i] + ":");
                System.out.println("\t"+addressList[0].getHostName());
                for(int j=0; j<addressList.length; j++){
                    System.out.println("\t"+addressList[j].getHostAddress());
                }
            }catch(UnknownHostException e){
                System.out.println("Unable to find address for "+args[i]);
            }
        }
    }
}
```

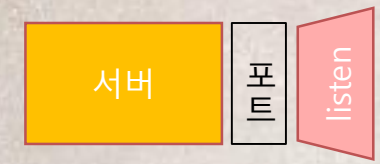
<terminated> PointVectorEx [Java Application] C:₩Program Files₩Java₩j

```
Local Host:
        tae-pc
        192.168.0.8
comsi.inje.ac.kr:
        comsi.inje.ac.kr
        211.220.195.180
```

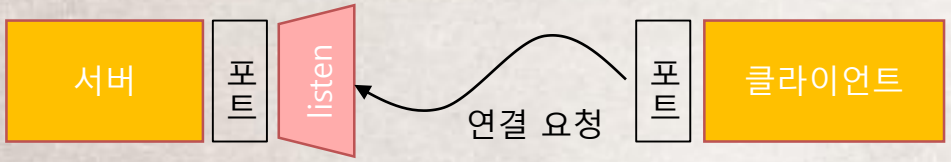
소켓을 이용한
서버 클라이언트 통신 프로그램

클라이언트와 서버 연결 순서

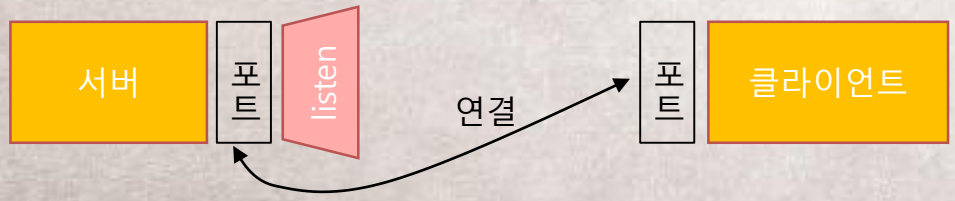
- 클라이언트와 서버 연결
  - 서버는 서버 소켓으로 들어오는 연결 요청을 기다림



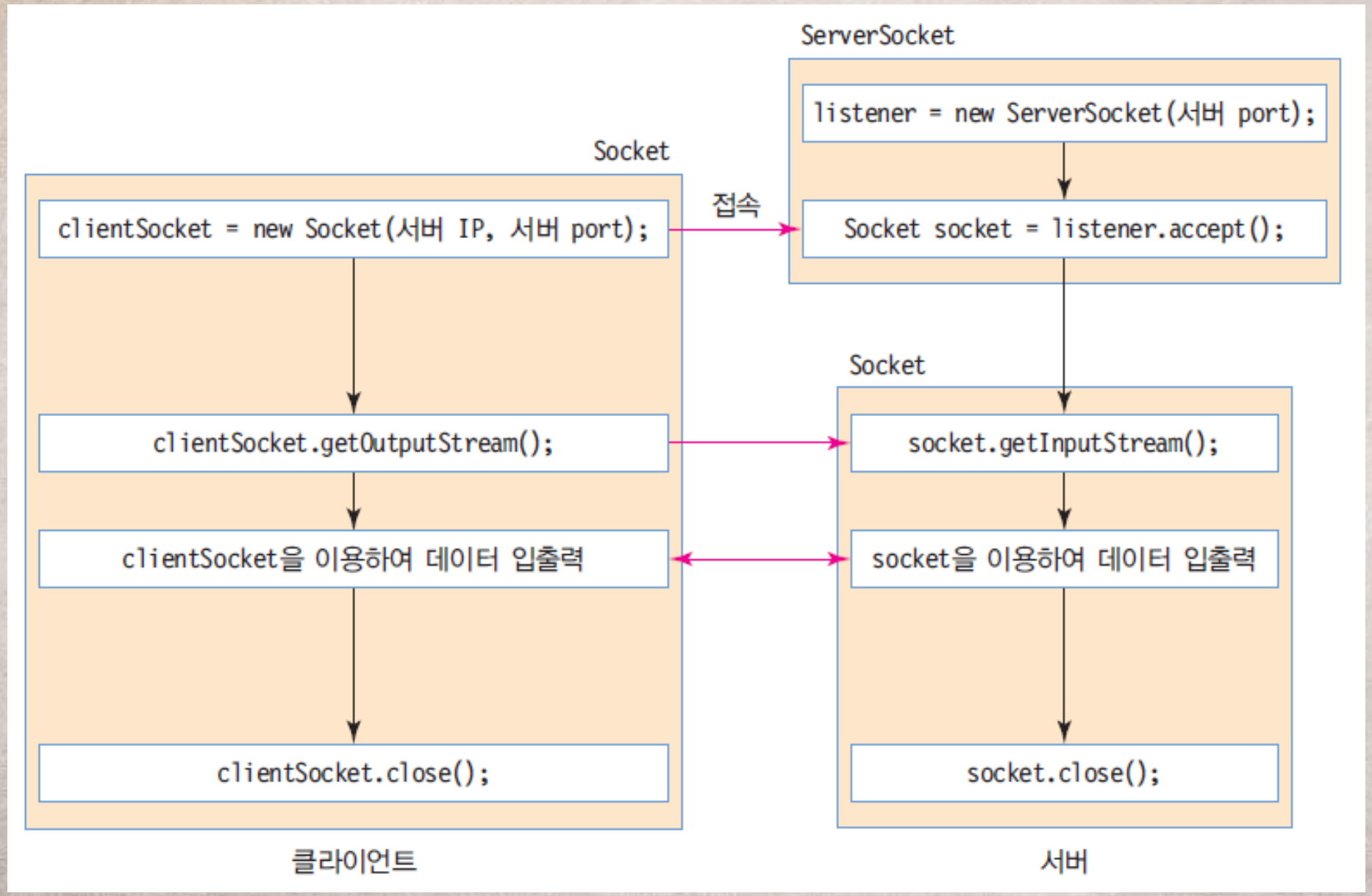
  - 클라이언트가 서버에게 연결 요청



  - 서버가 연결 요청 수락하고 새로운 소켓을 만들어 클라이언트와 연결 생성



소켓을 이용한 서버 클라이언트 통신 프로그램의 구조



Socket 클래스, 클라이언트 소켓

- Socket 클래스
  - 클라이언트 소켓에 사용되는 클래스
  - java.net 패키지에 포함
  - 주요 생성자

생성자	설명
Socket(InetAddress address, int port)	소켓을 생성하여 지정된 IP 주소와 포트 번호에 연결한다.
Socket(String host, int port)	소켓을 생성하여 지정된 호스트와 포트 번호에 연결한다. 호스트 이름이 null인 경우는 루프백(loopback) 주소로 가정한다.

루프백주소

- 컴퓨터의 네트워크 입출력 기능을 시험하기 위하여 가상으로 할당한 인터넷 주소(127.0.0.1)
- 웹 서버나 인터넷 소프트웨어의 네트워크 동작 기능을 시험하는 데 사용.

클라이언트 소켓 생성(서버 접속, 입출력 스트림 생성)

- 클라이언트 소켓 생성 및 서버에 접속

```
Socket clientSocket = new Socket("128.12.1.1", 5550);
```

  - Socket 객체의 생성되면 곧 바로 128.12.1.1의 주소로 자동 접속
- 네트워크 입출력 스트림 생성

```
BufferedReader in = new BufferedReader(
                        new InputStreamReader(clientSocket.getInputStream()));
BufferedWriter out = new BufferedWriter(
                        new OutputStreamWriter(clientSocket.getOutputStream()));
```

  - 일반 스트림을 입출력 하는 방식과 동일
- 서버로 데이터 전송

```
out.write("hello"+"\n");
out.flush();
```

  - Write()는 버퍼에 데이터를 저장
  - flush()를 호출하면 스트림 속(버퍼)에 데이터를 남기지 않고 모두 전송
- 서버로부터 데이터 수신

```
int x = in.read();          // 서버로부터 한 개의 문자 수신
String line = in.readline(); //서버로부터 한 행의 문자열 수신
```

- 네트워크 접속 종료

```
clientSocket.close();
```

ServerSocket 클래스, 서버 소켓

- ServerSocket 클래스
  - 서버 소켓에 사용되는 클래스
  - java.net 패키지에 포함
  - 주요 생성자

생성자	설명
ServerSocket(int port)	소켓을 생성하여 지정된 포트 번호에 연결한다.

  - 주요 메소드

메소드	설명
Socket accept()	연결 요청을 기다리다 연결 요청이 들어오면 수락하고 새 Socket 객체를 반환
void close()	서버 소켓을 닫는다.
InetAddress getInetAddress()	서버 소켓에 연결된 로컬 주소 반환
int getLocalPort()	서버 소켓이 연결 요청을 모니터링하는 클라이언트 포트 번호 반환
boolean isBound()	서버 소켓이 로컬 주소에 연결되어있으면 true 반환
boolean isClosed()	서버 소켓이 닫혀있으면 true 반환
void setSoTimeout(int timeout)	accept()에 대한 타임 아웃 시간 지정. 0이면 타임아웃이 해제.

서버 소켓 생성(클라이언트 접속, 입출력 스트림 생성)

- 서버 소켓 생성

```
ServerSocket serverSocket = new ServerSocket(5550);
```

  - 이미 사용 중인 포트 번호를 지정하면 오류가 발생

- 클라이언트로부터 접속 기다림

```
Socket socket = serverSocket.accept();
```

  - accept() 메소드는 연결 요청이 오면 새로운 Socket 객체 반환
  - 서버에서 클라이언트와의 데이터 통신은 새로 만들어진 Socket 객체를 통해서 이루어짐
  - ServerSocket 클래스는 Socket 클래스와 달리 주어진 연결에 대해 입출력 스트림을 만들어주는 메소드가 없음

- 네트워크 입출력 스트림 생성

```
BufferedReader in = new BufferedReader(new InputStreamReader(socket.getInputStream()));
BufferedWriter out = new BufferedWriter(new OutputStreamWriter(socket.getOutputStream()));
```

  - accept() 메소드에서 얻은 Socket 객체의 getInputStream()과 getOutputStream() 메소드를 이용하여 데이터 스트림 생성
  - 일반 스트림을 입출력하는 방식과 동일하게 네트워크 데이터 입출력

클라이언트로 데이터 송수신

- 클라이언트로부터 데이터 수신

```
int x = in.read();              // 클라이언트로부터 한 개의 문자 수신
String line = in.readline(); //클라이언트로부터 한 행의 문자열 수신
```

- 클라이언트로 데이터 전송

```
out.write("Hi!, Client"+"\n");
out.flush();
```

  - Write()는 버퍼에 저장
  - flush()를 호출하면 스트림 속(버퍼)에 데이터를 남기지 말고 모두 전송
- 네트워크 접속 종료

```
socket.close();
```

- 서버 응용프로그램 종료

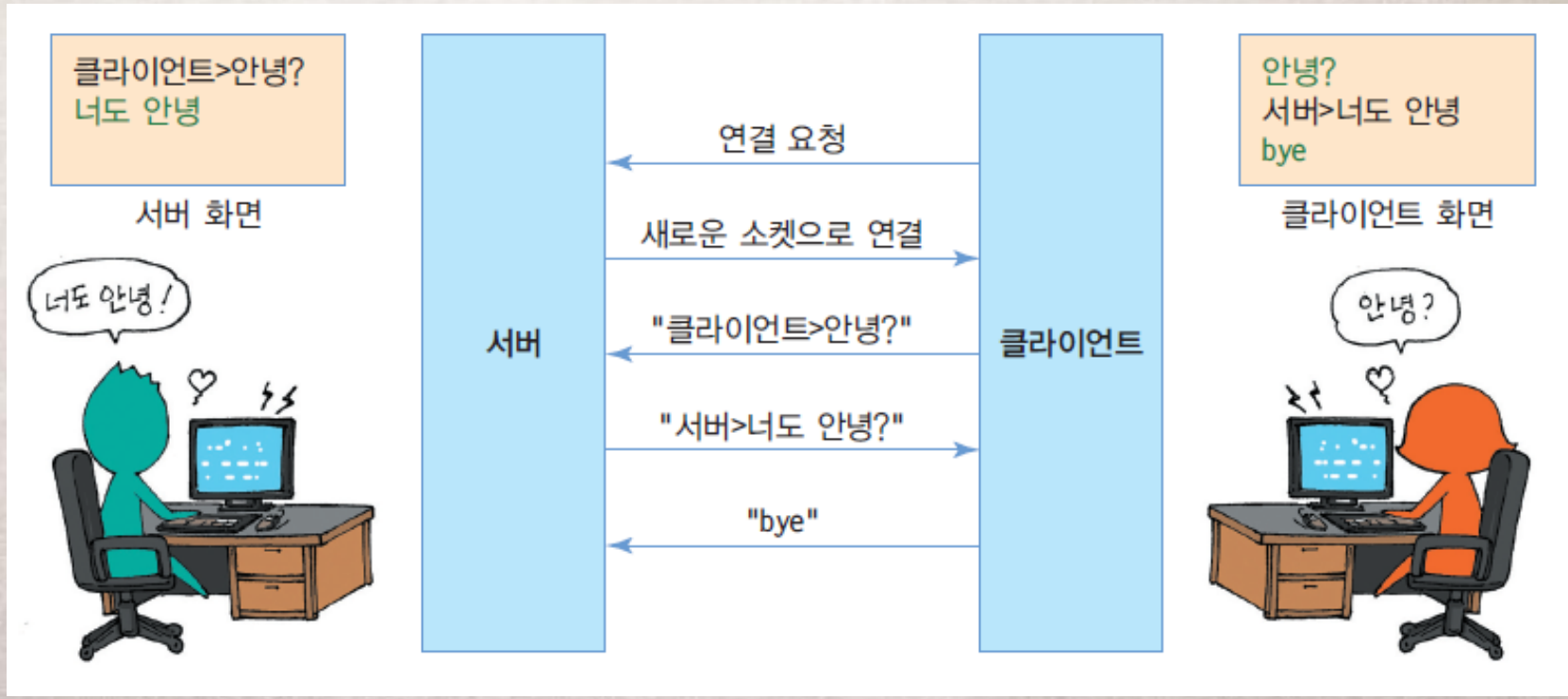
```
serverSocket.close();
```

  - 더 이상 클라이언트의 접속을 받지 않고 서버 응용 프로그램을 종료하고자 하는 경우 ServerSocket 종료

Practice – Chatting program

소켓을 이용한 클라이언트/서버 채팅 예제

- 간단한 채팅 프로그램 예제
  - 서버와 클라이언트가 1:1로 채팅 하는 간단한 예제
  - 서버와 클라이언트 간의 메시지 구분을 위해 서버는 메시지 앞에 “서버〉”을 접두어로 붙여 메시지를 전송하며 클라이언트는 “클라이언트〉”를 접두어로 붙여 메시지 전송
  - 서버와 클라이언트가 번갈아 가면서 메시지 전송 및 수신
  - 클라이언트가 bye를 보내면 프로그램 종료



서버 프로그램

- 서버 소켓 생성

```
ServerSocket servSocket = new ServerSocket(9999);
```

  - 시스템에서 사용되지 않은 포트 번호로 서버 소켓 생성
- 클라이언트 요청 대기

```
Socket socket = servSocket.accept();
```

  - 클라이언트가 연결 요청이 올 때까지 소켓 기다림
  - 해당 포트 번호로 연결 요청이 오면
    - 수락과 함께 새로운 소켓을 생성
    - 새 소켓으로 클라이언트와 통신
  - 새로운 소켓의 포트 번호는 자동으로 할당

서버 프로그램

- 클라이언트와 통신을 위한 입출력 스트림 생성

```
InputStream in = socket.getInputStream();
OutputStream = socket.getOutputStream();
```

  - 스트림을 생성하여 클라이언트와 데이터 송수신
    - 데이터의 종류에 따라 바이트 스트림 또는 문자 스트림을 생성
    - 채팅과 같이 문자열을 송수신하는 경우는 문자 스트림 사용
    - 효율적 입출력을 위하여 버퍼 스트림(Buffered Stream) 사용
- 클라이언트로부터 데이터 수신 및 송신

```
int ch;
wihle((ch = in.read()) != -1){
    System.out.print((char)ch);
   outStream.write(ch);
};
```

  - 스트림 생성 이후는 데이터 입력 받는 방법과 동일
  - 클라이언트에서 한 행의 문자열(-1)을 보내올 때까지 기다림
  - flush() 메소드로 스트림의 모든 데이터를 클라이언트로 송신가능

서버 프로그램

- 연결 종료

```
socket.close();
servSocket.close();
```

  - 데이터의 송수신이 끝나면 소켓을 닫아야 함
  - 소켓을 닫으면 소켓의 입출력 스트림도 같이 닫힘
  - 서버 소켓을 닫으면 클라이언트 연결 요청을 받을 수 없음

클라이언트 프로그램

- 연결 요청

```
socket = new Socket("localhost", 9999);
```

  - 소켓 생성
    - 서버의 호스트 주소
    - 서버가 연결 요청을 모니터링하는 포트 번호로 소켓 생성
  - 예제는 호스트 이름을 “localhost”로 지정
    - 동일한 시스템에서 서버와 클라이언트가 동작하기 때문
- 클라이언트와 통신을 위한 입출력 스트림 생성

```
InputStream in = socket.getInputStream();
OutputStream out = socket.getOutputStream();
```

  - 스트림을 생성하여 서버와 데이터 송수신
    - 데이터의 종류에 따라 바이트 스트림 또는 문자 스트림 사용
    - 채팅과 같이 문자열을 송수신하는 경우는 문자 스트림 사용
    - 효율적 입출력을 위하여 버퍼 스트림(Buffered Stream) 사용

클라이언트 프로그램

- 서버에 데이터 송신

```
outStream.write(“Hello World!”.getBytes());
//out.flush();
```

  - 스트림 생성 이후는 데이터 출력 방법과 동일
  - 콘솔에서 입력 받은 문자열을 서버로 송신
  - flush() 메소드로 스트림의 모든 데이터를 서버로 송신

- 클라이언트의 데이터 수신

```
while(receivedTotalByte < sendByteBuffer.length){
    if((bytesRcvd = inStream.read(sendByteBuffer, receivedTotalByte,
        sendByteBuffer.length - receivedTotalByte)) == -1) throw new
                SocketException("Connection close prematurely");
   receivedTotalByte += bytesRcvd;
}
```

  - 스트림 생성 이후는 데이터 입력 방법과 동일
  - 서버로 보낸 바이트 수만큼 받을 때까지 반복해서 바이트를 받는다.
  - read()함수의 파라메터–받을 버퍼, 처음으로 받은 바이트가 있어야 오프셋, 버퍼에 담을 수 있는 바이트의 최대값

클라이언트 프로그램

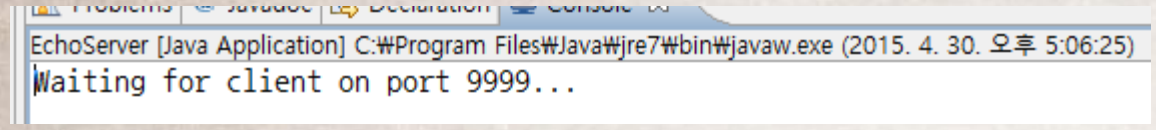
- 연결 종료

```
socket.close();
```

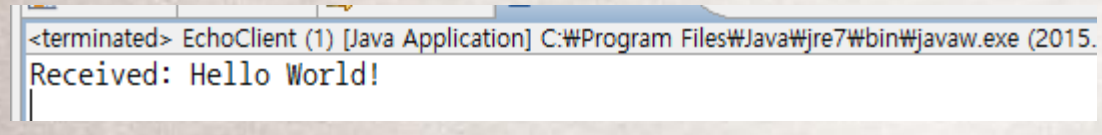
  - 데이터의 송수신이 끝나면 소켓을 닫아야 함
  - 소켓을 닫으면 소켓의 입출력 스트림도 같이 닫힘

실행 보기

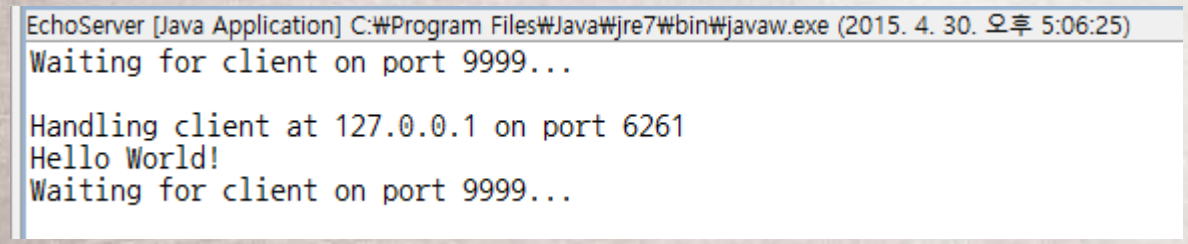
- 서버 시작



- 클라이언트가 서버로 전송 후 서버의 응답을 받음



- 클라이언트의 접속 후 서버의 상태



예제7-3. 서버 프로그램

EchoServer.java | EchoClient.java

```
1  import java.io.IOException;
2  import java.io.InputStream;
3  import java.io.OutputStream;
4  import java.net.ServerSocket;
5  import java.net.Socket;
6
7
8  public class EchoServer {
9
10     // 클라이언트의 요청을 받기 위한 서버소켓 포트
11     private static final int SERVER_SOCKET_PORT = 9999;
12     private static final int BUFFER_SIZE = 32;
13
14     public static void main(String[] args) {
15         ServerSocket  servSocket = null;
16         byte[] byteBuffer = new byte[BUFFER_SIZE];
17         try {
18             // 클라이언트의 컨넥션 요청을 받기 위한 서버소켓 생성
19             servSocket = new ServerSocket(SERVER_SOCKET_PORT);
20
21             for(;;){
22                 System.out.println("Waiting for client on port " + servSocket.getLocalPort() + "...\n");
23
24                 // 클라이언트의 소켓을 얻는다.
25                 Socket clientSocket = servSocket.accept();
26
27                 System.out.println("Handling client at " +
28                             clientSocket.getInetAddress().getHostAddress() + " on port " +
29                             clientSocket.getPort());
30                 InputStream inStream = clientSocket.getInputStream();
31                 OutputStream outStream = clientSocket.getOutputStream();
```

예제7-3. 서버 프로그램

```
32
33                  int ch;
34
35                  while((ch = inStream.read()) != -1){
36                      System.out.print((char)ch);
37                      outStream.write(ch);
38                  }
39
40                  System.out.println(new String(byteBuffer));
41
42                  clientSocket.close();
43              }
44
45          } catch (IOException e) {
46              System.out.println("스트림입출력 중에 오류가 발생했습니다.");
47              System.out.println(e.getMessage());
48
49          }
50      }
51
52  }
53
```

예제7-3. 클라이언트 프로그램

EchoServer.java | EchoClient.java

```
1  import java.io.IOException;
2  import java.io.InputStream;
3  import java.io.OutputStream;
4  import java.net.Socket;
5  import java.net.SocketException;
6  import java.net.UnknownHostException;
7
8
9  public class EchoClient {
10
11     // 클라이언트에 요청을 보내기 위한 서버소켓 포트
12     private static final int SERVER_SOCKET_PORT = 9999;
13     private static final String SERVER_ADDRESS = "localhost";
14
15
16     public static void main(String[] args) {
17
18         byte[] sendByteBuffer = "Hello World!".getBytes();
19         Socket socket = null;
20
21         try {
22             // 해당 서버(SERVER_ADDRESS)의 포트(SERVER_SOCKET_PORT)에 연결한다.
23             socket = new Socket(SERVER_ADDRESS, SERVER_SOCKET_PORT);
24
25             InputStream inStream = socket.getInputStream();
26             OutputStream outStream = socket.getOutputStream();
27
28             // 서버에 문자열을 보낸다.
29             outStream.write(sendByteBuffer);
30
```

예제7-3. 클라이언트 프로그램

```
31              // 서버로부터 돌려 받기 위한 처리
32              int receivedTotalByte = 0; // 전체 받은 바이트 수
33              int bytesRcvd; // 마지막에 read에서 받은 바이트 수
34              while(receivedTotalByte < sendByteBuffer.length){
35                  if((bytesRcvd = inStream.read(sendByteBuffer, receivedTotalByte,
36                          sendByteBuffer.length - receivedTotalByte)) == -1)
37                      throw new SocketException("Connection close prematurely");
38                  receivedTotalByte += bytesRcvd;
39              }
40
41              System.out.println("Received: " + new String(sendByteBuffer));
42
43              socket.close();
44          } catch (UnknownHostException e) {
45              System.out.println("정상적인 Host가 아닙니다.");
46              System.out.println(e.getMessage());
47          } catch (IOException e) {
48              System.out.println("스트림입출력 중에 오류가 발생했습니다.");
49              System.out.println(e.getMessage());
50
51          }
52
53      }
54
55  }
56
```

int read(byte[] b, int off, int len) throws IOException

read

```
public int read(byte[] b,
       int off,
       int len)
         throws IOException
```

Reads up to len bytes of data from the input stream into an array of bytes. An attempt is made to read as many as len bytes, but a smaller number may be read. The number of bytes actually read is returned as an integer.

This method blocks until input data is available, end of file is detected, or an exception is thrown.

If len is zero, then no bytes are read and 0 is returned; otherwise, there is an attempt to read at least one byte. If no byte is available because the stream is at end of file, the value -1 is returned; otherwise, at least one byte is read and stored into b.

The first byte read is stored into element b[off], the next one into b[off+1], and so on. The number of bytes read is, at most, equal to len. Let *k* be the number of bytes actually read; these bytes will be stored in elements b[off] through b[off+*k*-1], leaving elements b[off+*k*] through b[off+len-1] unaffected.

In every case, elements b[0] through b[off] and elements b[off+len] through b[b.length-1] are unaffected.

The read(b, off, len) method for class InputStream simply calls the method read() repeatedly. If the first such call results in an IOException, that exception is returned from the call to the read(b, off, len) method. If any subsequent call to read() results in a IOException, the exception is caught and treated as if it were end of file; the bytes read up to that point are stored into b and the number of bytes read before the exception occurred is returned. The default implementation of this method blocks until the requested amount of input data len has been read, end of file is detected, or an exception is thrown. Subclasses are encouraged to provide a more efficient implementation of this method.

Parameters:

b - the buffer into which the data is read.

off - the start offset in array b at which the data is written.

len - the maximum number of bytes to read.

Returns:

the total number of bytes read into the buffer, or -1 if there is no more data because the end of the stream has been reached.

Throws:

IOException - If the first byte cannot be read for any reason other than end of file, or if the input stream has been closed, or if some other I/O error occurs.

NullPointerException - If b is null.

IndexOutOfBoundsException - If off is negative, len is negative, or len is greater than b.length - off