

# java 22강 - 네트워크

---

양 명 속

[[now4ever7@gmail.com](mailto:now4ever7@gmail.com)]



# 목차

---

- 네트워킹
- InetAddress
- URL
- URLConnection
- 소켓 프로그래밍
  - TCP 방식
  - UDP 방식



# 네트워킹

---

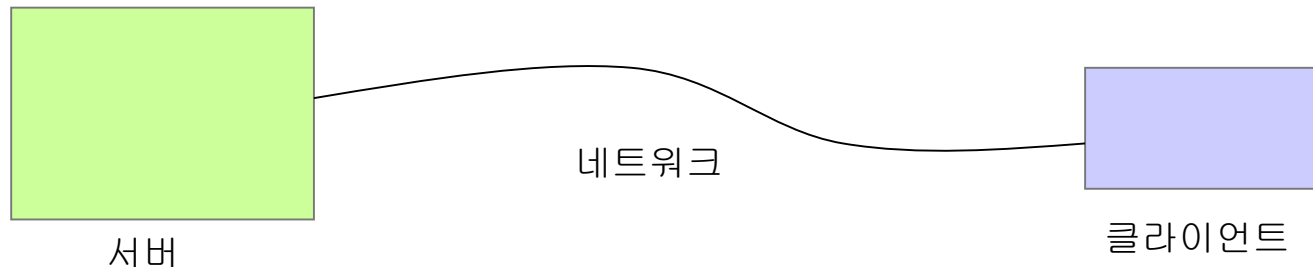
# 네트워킹(Networking)

## ■ 네트워킹(Networking)

- 두 대 이상의 컴퓨터를 케이블로 연결하여 네트워크를 구성하는 것
- 컴퓨터들을 서로 연결하여 데이터를 손쉽게 주고받거나 또는 자원 (프린터와 같은 주변기기)을 함께 공유
- `java.net` 패키지를 사용하면 네트워크 어플리케이션의 데이터 통신 부분을 쉽게 작성할 수 있음

## ■ 통신의 3대 요소

- 서버(Server)
- 클라이언트(Client)
- 네트워크(Network) : 서버와 클라이언트를 연결해줌

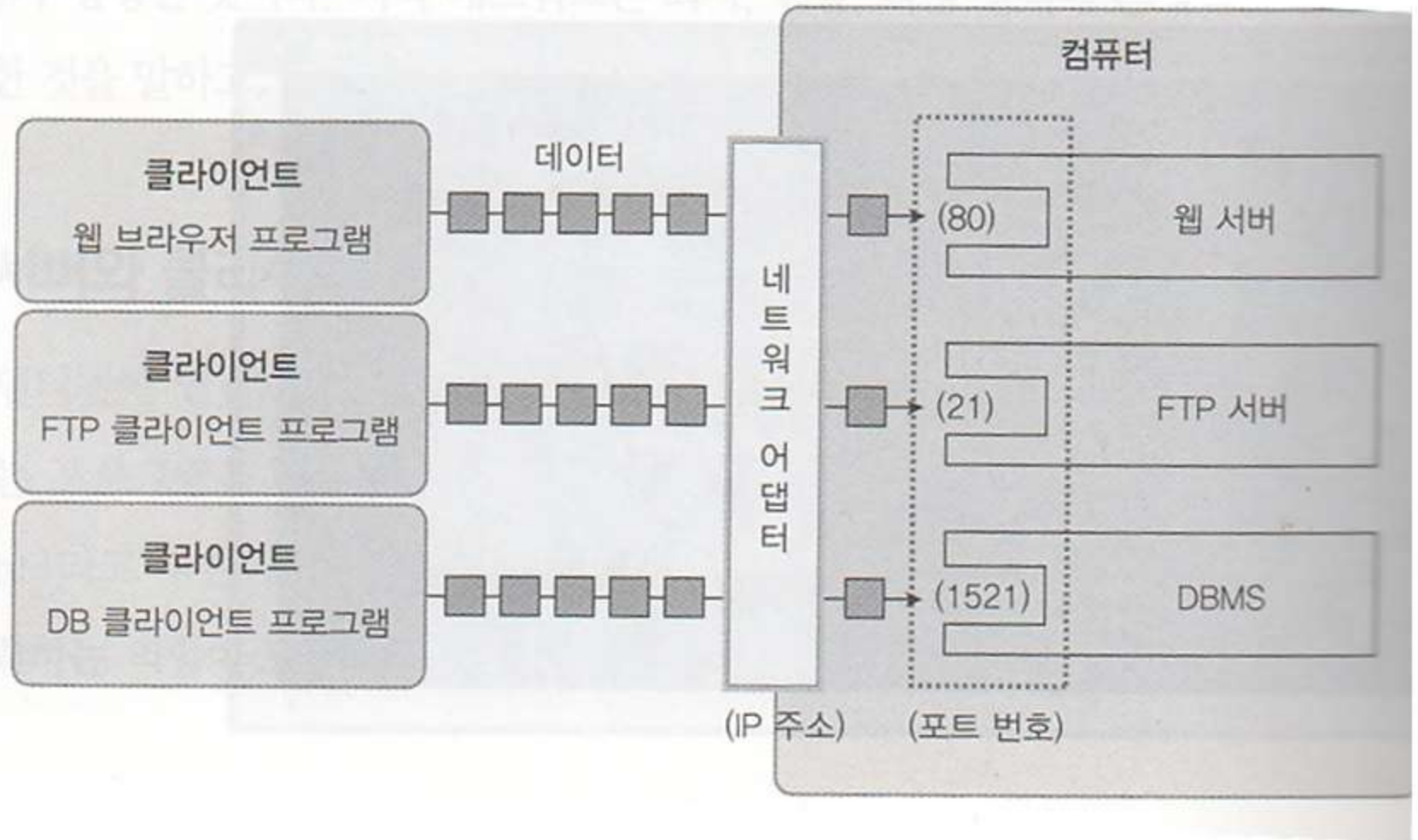




# 클라이언트/서버 (client/server)

- 클라이언트/서버 (client/server)
  - 컴퓨터간의 관계를 역할로 구분하는 개념
  - 서버 - 서비스를 제공하는 컴퓨터
  - 클라이언트 - 서비스를 사용하는 컴퓨터
  - 서비스 - 서버가 클라이언트로부터 요청 받은 작업을 처리하여 그 결과를 제공하는 것
  - 서버가 제공하는 서비스의 종류에 따라
    - 파일서버 - 클라이언트가 요청한 파일을 제공하는 서비스를 수행
    - 메일서버
    - 어플리케이션 서버
  - 서버가 서비스를 제공하기 위해서는 **서버 프로그램**이 있어야 하고,
  - 클라이언트가 서비스를 제공받기 위해서는 서버프로그램과 연결할 수 있는 **클라이언트 프로그램**이 있어야 함
    - 예) 웹서버에 접속하여 정보를 얻기 위해서는 웹브라우저(클라이언트 프로그램)가 있어야 하고, FTP 서버에 접속해서 파일을 전송받기 위해서는 알FTP와 같은 FTP 클라이언트 프로그램이 필요함

# 클라이언트/서버 (client/server)





# IP주소(IP address)

192.168.0.50

## ■ IP주소

- 컴퓨터(호스트, host)를 구별하는 데 사용되는 고유한 값으로 인터넷에 연결된 모든 컴퓨터는 IP 주소를 갖는다.
- IP 주소는 4byte (32bit)의 정수로 구성되어 있으며, 4개의 정수가 마침표를 구분자로 'a.b.c.d'와 같은 형식으로 표현됨
  - a, b, c, d => 부호없는 1byte 값(0~255 사이의 정수)
- IP 주소는 다시 네트워크 주소와 호스트 주소로 나눌 수 있는데, 32bit의 IP 주소 중에서 네트워크 주소와 호스트 주소가 각각 몇 bit를 차지하는 지는 네트워크를 어떻게 구성하였는지에 따라 달라짐
- 서로 다른 두 호스트의 IP주소의 네트워크주소가 같다는 것은 두 호스트가 같은 네트워크에 포함되어 있다는 것을 의미함

# IP주소(IP address)

- 예) ip주소 - 192.168.10.100
- subnet mask - 255.255.255.0

ip주소

192								168								10								100							
1	1	0	0	0	0	0	0	1	0	1	0	1	0	0	0	0	0	0	0	1	0	1	0	0	1	1	0	0	1	0	0
네트워크 주소																								호스트 주소							

서브넷 마스크

255								255								255								0							
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0		

- IP 주소와 서브넷 마스크를 비트연산자 & 로 연산하면 IP주소에서 네트워크 주소만을 뽑아낼 수 있음  
→ ip주소 - 192.168.10.100의 네트워크 주소는 24bit(192.168.10) 이라는 것과 호스트 주소는 마지막 8bit(100)라는 것을 알 수 있다
- 호스트 주소가 0 인 것은 네트워크 자신을 나타내고, 255는 브로드캐스트 주소로 사용





# InetAddress

- 자바에서는 IP 주소를 다루기 위한 클래스로 InetAddress를 제공한다
- InetAddress 의 메서드

메서드	설 명
byte[] getAddress()	IP주소를 byte배열로 반환한다
static InetAddress[] getAllByName(String host)	도메인명(host)에 지정된 모든 호스트의 IP 주소를 배열에 담아 반환한다
static InetAddress getByAddress(byte[] addr)	byte배열을 통해 IP주소를 얻는다
static InetAddress getByName(String host)	도메인명을 통해 IP주소를 얻는다
String getHostAddress()	호스트의 IP주소를 반환한다
String getHostName()	호스트의 이름을 반환한다
static InetAddress getLocalHost()	지역호스트의 IP주소를 반환한다
boolean isMulticastAddress()	IP주소가 멀티캐스트 주소인지 알려준다
boolean isLoopbackAddress()	IP주소가 loopback주소(127.0.0.1)인지 알려준다

# 예제

```
import java.net.*;
import java.util.*;
```

```
// InetAddress 클래스 : IP 주소를 다루기 위한 클래스
class NetworkEx1{
```

```
    public static void main(String[] args) {
        InetAddress ip = null;
        InetAddress[] ipArr = null;
```

```
        try {
            System.out.println("-----www.naver.com-----");

            ip = InetAddress.getByName("www.naver.com");
            System.out.println(ip);

            System.out.println("getHostName() :"+ip.getHostName());
            System.out.println("getHostAddress() :"+ip.getHostAddress());

            System.out.println("\n-----LocalHost-----");
            ip = InetAddress.getLocalHost();
            System.out.println("getHostName() :"+ip.getHostName());
            System.out.println("getHostAddress() :"+ip.getHostAddress());
            System.out.println();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

```
-----www.naver.com-----
www.naver.com/202.131.30.12
getHostName() :www.naver.com
getHostAddress() :202.131.30.12

-----LocalHost-----
getHostName() :yang
getHostAddress() :192.168.1.101

ipArr[0] :www.naver.com/202.131.30.12
ipArr[1] :www.naver.com/220.95.233.171
```



# 예제

```
        ipArr = InetAddress.getAllByName("www.naver.com");

        for(int i=0; i < ipArr.length; i++) {
            System.out.println("ipArr["+ i +"] :"+ ipArr[i]);
        }
    } catch (UnknownHostException e) {
        e.printStackTrace();
    }

} // main
}
```

## InetAddress의 주요 메서드 활용 예제

하나의 도메인명([www.naver.com](http://www.naver.com))에 여러 Ip주소가 맵핑될 수도 있고 또 그 반대의 경우도 가능하기 때문에 전자의 경우 `getAllByName()`을 통해 모든 IP주소를 얻을 수 있다

`getLocalHost()`를 사용하면 호스트명과 IP주소를 알아낼 수 있다.



# URL(Uniform Resource Location)

## ■ URL

- 인터넷에 존재하는 여러 서버들이 제공하는 **자원에 접근할 수 있는 주소**를 표현하기 위한 것
- 형태
  - '프로토콜://호스트명:포트번호/경로명/파일명?쿼리스트링#참조'

**http://www.java.com:80/book/source/test.jsp?id=hong#index1**

**프로토콜:** 자원에 접근하기 위해 서버와 통신하는데 사용되는 통신규약 (**http**)

**호스트명:** 자원을 제공하는 서버의 이름(**www.java.com**)

**포트번호:** 통신에 사용되는 서버의 포트번호 (**80**)

**경로명:** 접근하려는 자원이 저장된 서버상의 위치 (**/book/source/**)

**파일명:** 접근하려는 자원의 이름(**test.jsp**)

**쿼리스트링:** URL에서 ? 이후의 부분(**id=hong**)

**참조:** URL에서 # 이후의 부분(**index1**)



# URL

- 자바에서는 URL을 다루기 위한 클래스로 URL 클래스를 제공
- URL의 메서드

메서드	설 명
URL(String spec)	지정된 문자열 정보의 URL객체를 생성한다
URL(String protocol, String host, String file)	지정된 값으로 구성된 URL객체를 생성한다
URL(String protocol, String host, int port, String file)	지정된 값으로 구성된 URL객체를 생성한다
String getAuthority()	호스트명과 포트를 문자열로 반환한다
Object getContent()	URL의 Content 객체를 반환한다.
Object getContent(Class[] classes)	URL의 Content 객체를 반환한다.
int getDefaultPort()	URL의 기본포트를 반환한다.(http는 80)
String getFile()	파일명을 반환한다
String getHost()	호스트명을 반환한다
String getPath()	경로명을 반환한다
int getPort()	포트를 반환한다



# URL

메서드	설 명
String <b>getProtocol()</b>	프로토콜을 반환한다.
String <b>getQuery()</b>	쿼리를 반환한다.
String <b>getRef()</b>	참조(anchor)를 반환한다.
String <b>getUserInfo()</b>	사용자정보를 반환한다.
URLConnection <b>openConnection()</b>	URL과 연결된 URLConnection을 얻는다.
URLConnection <b>openConnection(Proxy proxy)</b>	URL과 연결된 URLConnection을 얻는다.
InputStream <b>openStream()</b>	URL과 연결된 URLConnection의 InputStream을 얻는다.
boolean <b>sameFile(URL other)</b>	두 URL이 서로 같은 것인지 알려준다
protected void <b>set(String protocol, String host, int port, String file, String ref)</b>	URL객체의 속성을 지정된 값으로 설정한다
protected void <b>set(String protocol, String host, int port, String authority, String userInfo, String path, String query, String ref)</b>	URL객체의 속성을 지정된 값으로 설정한다
String <b>toExternalForm()</b>	URL을 문자열로 변환하여 반환한다
URI <b>toURI()</b>	URL을 URI로 변환하여 반환한다



# URL

---

- URL 객체를 생성하는 방법

```
URL url = new URL("http://www.java.com:80/book/source/test.jsp");
```



# 예제

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

```
class NetworkEx2 {
    public static void main(String args[]) throws Exception {
        //URL url = new URL("http://www.yes24.com/24/category/bestseller?Gcode=100_012");
        URL url = new URL(args[0]);

        System.out.println("Protocol:" + url.getProtocol());
        System.out.println("Host:" + url.getHost());
        System.out.println("Port:" + url.getPort()); // -1 반환 => 기본 포트 사용한 경우
        System.out.println("File:" + url.getFile());

        URLConnection conn = url.openConnection();
        //System.out.println("conn.toString(): " + conn);
        System.out.println("WnWn파일 크기:" + conn.getContentLength());
        System.out.println("ContentType:" + conn.getContentType());
    }
}
```

```
Protocol:http
Host:www.yes24.com
Port:-1
File:/24/category/bestseller?Gcode=100_012

파일 크기:203404
ContentType:text/html; charset=ks_c_5601-1987
```

**URLConnection**을 생성하고 **get**메서드들을 통해서 관련 정보를 얻어서 출력





# 예제

```
//public final InputStream openStream() throws IOException
//URL과 연결된 URLConnection의 InputStream을 얻는다.
InputStream is=url.openStream();
BufferedReader br
    =new BufferedReader(new InputStreamReader(is));
String data="";
while ((data=br.readLine()) != null){
    System.out.println(data);
}
br.close();
}
```

- URL에 연결하여 그 내용을 읽어 온다
- 읽어올 데이터가 문자데이터이기 때문에 **BufferedReader**를 사용
- **openStream()**을 호출해서 URL의 **InputStream**을 얻은 후 데이터를 읽어온다.
- **openStream()** - URL에 연결해서 **InputStream**을 얻어온다.
  - public final InputStream openStream()



## 예제2

html 파일을 읽어서 텍스트 파일로 저장

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

```
public class NetworkEx5 {
    public static void main(String args[]) {
        String address="http://www.yes24.com/24/category/bestseller?Gcode=100_012";
        int ch = 0;

        try {
            URL url = new URL(address);
            InputStream in = url.openStream();
            FileOutputStream out = new FileOutputStream("text/yes24.txt");

            while((ch=in.read()) != -1) {
                out.write(ch); //파일에 출력
                System.out.write(ch); //화면 출력
            }
            in.close();
            out.close();
        } catch(Exception e) {
            e.printStackTrace();
        }
    } // main
}
```

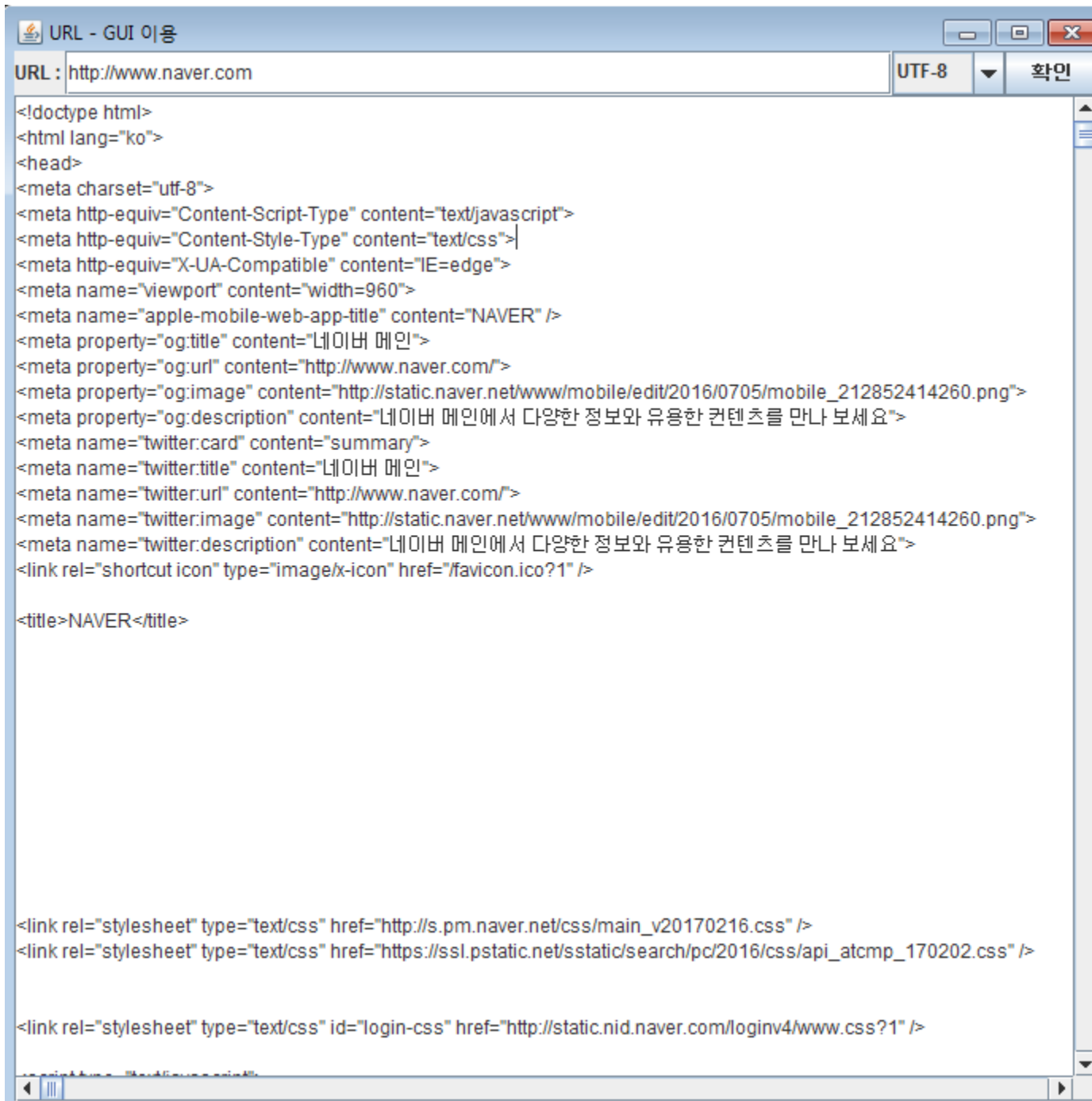


# 실습

---

- 예제2 수정하기

- 1. BufferedInputStream, BufferedOutputStream 이용해서 파일에 출력하기
  - =>yes1.txt
- 2. BufferedReader, BufferedWriter 이용해서 파일에 출력하기
  - =>yes2.txt



```
URL : http://www.naver.com UTF-8 확인

<!doctype html>
<html lang="ko">
<head>
<meta charset="utf-8">
<meta http-equiv="Content-Script-Type" content="text/javascript">
<meta http-equiv="Content-Style-Type" content="text/css">
<meta http-equiv="X-UA-Compatible" content="IE=edge">
<meta name="viewport" content="width=960">
<meta name="apple-mobile-web-app-title" content="NAVER" />
<meta property="og:title" content="네이버 메인">
<meta property="og:url" content="http://www.naver.com/">
<meta property="og:image" content="http://static.naver.net/www/mobile/edit/2016/0705/mobile_212852414260.png">
<meta property="og:description" content="네이버 메인에서 다양한 정보와 유용한 콘텐츠를 만나 보세요">
<meta name="twitter:card" content="summary">
<meta name="twitter:title" content="네이버 메인">
<meta name="twitter:url" content="http://www.naver.com/">
<meta name="twitter:image" content="http://static.naver.net/www/mobile/edit/2016/0705/mobile_212852414260.png">
<meta name="twitter:description" content="네이버 메인에서 다양한 정보와 유용한 콘텐츠를 만나 보세요">
<link rel="shortcut icon" type="image/x-icon" href="/favicon.ico?1" />

<title>NAVER</title>

<link rel="stylesheet" type="text/css" href="http://s.pm.naver.net/css/main_v20170216.css" />
<link rel="stylesheet" type="text/css" href="https://ssl.pstatic.net/sstatic/search/pc/2016/css/api_atcmp_170202.css" />

<link rel="stylesheet" type="text/css" id="login-css" href="http://static.nid.naver.com/loginv4/www.css?1" />
```

url을 읽어서 해당 웹페이지 정보를 **TextArea**에 출력하고, **file**에도 출력

BufferedReader  
BufferedWriter 이용

```

public class URLGuiTest extends JFrame implements ActionListener{
    private JLabel lb;
    private JTextField tfUrl;
    private JButton btOk;
    private JTextArea taResult;
    private JScrollPane scrollPane;
    private JPanel pnl, pnl2;
    private JComboBox<String> cbEncoding;
    public URLGuiTest() {
        super("URL - GUI 이용");
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        pnl=new JPanel(new BorderLayout());
        pnl2=new JPanel(new BorderLayout());

        lb=new JLabel("URL : ");
        tfUrl=new JTextField(15);
        btOk=new JButton("확인");
        cbEncoding = new JComboBox<String>();
        cbEncoding.addItem("EUC-KR");
        cbEncoding.addItem("UTF-8");

        pnl2.add(tfUrl,"Center");
        pnl2.add(cbEncoding,"East");

        pnl.add(lb,"West");
        pnl.add(pnl2,"Center");
        pnl.add(btOk,"East");
    }
}

```

```

taResult=new JTextArea();
scrollPane=new JScrollPane(taResult);

add(scrollPane,"Center");
add(pnl,"North");

setSize(700, 700);
setVisible(true);

btOk.addActionListener(this);
tfUrl.addActionListener(this);
}
public void actionPerformed(ActionEvent e) {
    if(e.getSource()==btOk || e.getSource()==tfUrl){
        readUrl();
    }
}
private void readUrl() {
    //url을 읽어서 해당 웹페이지 정보를 TextArea에 출력하고, file에도 출력
    BufferedReader br=null;
    BufferedWriter bw=null;
    StringWriter sw=null;
    try {
        URL url = new URL(tfUrl.getText());
        InputStream in = url.openStream();

```

```

        br=new BufferedReader(new InputStreamReader(in,
            cbEncoding.getSelectedItem().toString()));

```

```

        bw=new BufferedWriter(new FileWriter("text/url.txt"));
        sw=new StringWriter();
        String line="";
        while((line=br.readLine())!=null){
            bw.write(line);
            bw.newLine();

            sw.write(line+"Wn");
        }
        taResult.setText(sw.toString());
    } catch (MalformedURLException e) {
        e.printStackTrace();
    } catch (IOException e) {
        e.printStackTrace();
    }finally{
        try {
            if(br!=null) br.close();
            if(bw!=null) bw.close();
            if(sw!=null) sw.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}

public static void main(String[] args) {
    new URLGuiTest();
}
}

```



# URLConnection

---

## ■ URLConnection

- 어플리케이션과 URL간의 통신연결을 나타내는 클래스의 최상위 클래스로 추상클래스임
- URLConnection을 상속받아 구현한 클래스로는 HttpURLConnection과 JarURLConnection이 있음
- URL의 프로토콜이 http 프로토콜이라면 openConnection()은 HttpURLConnection을 반환함
- URLConnection을 사용해서 연결하고자하는 자원에 접근하고 읽고 쓰기를 할 수 있다.



# URLConnection의 메서드

메서드	설명
void addRequestProperty(String key, String value)	지정된 키와 값을 RequestProperty에 추가한다. 기존에 같은 키가 있어도 값을 덮어쓰지 않는다.
void connect()	URL에 지정된 자원에 대한 통신연결을 연다.
boolean getAllowUserInteraction()	UserInteraction의 허용여부를 반환한다.
int getConnectTimeout()	연결종료시간을 천분의 일초로 반환한다.
Object getContent()	content객체를 반환한다.
Object getContent(Class[] classes)	content객체를 반환한다.
String getContentEncoding()	content의 인코딩을 반환한다.
int getContentLength()	content의 크기를 반환한다.
String getContentType()	content의 type을 반환한다.
long getDate()	헤더(header)의 date필드의 값을 반환한다.
boolean getDefaultAllowUserInteraction()	defaultAllowUserInteraction의 값을 반환한다.
String getDefaultRequestProperty(String key)	RequestProperty에서 지정된 키의 디폴트 값을 얻는다.
boolean getDefaultUseCaches()	useCache의 디폴트 값을 얻는다.
boolean getDoInput()	doInput필드값을 얻는다.
boolean getDoOutput()	doOutput필드값을 얻는다.
long getExpiration()	자원(URL)의 만료일자를 얻는다.(천분의 일초단위)
FileNameMap getFileNameMap()	FileNameMap(mimetable)을 반환한다.
String getHeaderField(int n)	헤더의 n번째 필드를 읽어온다.
String getHeaderField(String name)	헤더에서 지정된 이름의 필드를 읽어온다.
long getHeaderFieldDate(String name,long Default)	지정된 필드의 값을 날짜값으로 변환하여 반환한다. 필드값이 유효하지 않을 경우 Default값을 반환한다.

메서드	설명
int getHeaderFieldInt(String name,int Default)	지정된 필드의 값을 정수값으로 변환하여 반환한다. 필드값이 유효하지 않을 경우 Default값을 반환한다.
String getHeaderFieldKey(int n)	헤더의 n번째 필드를 읽어온다.
Map getHeaderFields()	헤더의 모든 필드와 값이 저장된 Map을 반환한다.
long getIfModifiedSince()	ifModifiedSince(변경여부)필드의 값을 반환한다.
InputStream getInputStream()	URLConnection에서 InputStream을 반환한다.
long getLastModified()	LastModified(최종변경일)필드의 값을 반환한다.
OutputStream getOutputStream()	URLConnection에서 OutputStream을 반환한다.
Permission getPermission()	Permission(허용권한)을 반환한다.
int getReadTimeout()	읽기제한시간의 값을 반환한다.(천분의 일초)
Map getRequestProperties()	RequestProperties에 저장된 (키, 값)을 Map으로 반환한다.
String getRequestProperty(String key)	RequestProperty에서 지정된 키의 값을 반환한다.
URL getURL()	URLConnection의 URL의 반환한다.
boolean getUseCaches()	캐쉬의 사용여부를 반환한다.
String guessContentTypeFromName(String fname)	지정된 파일(fname)의 content-type을 추측하여 반환한다.
String guessContentTypeFromStream(InputStream is)	지정된 입력스트림(is)의 content-type을 추측하여 반환한다.
void setAllowUserInteraction(boolean allowuserinteraction)	UserInteraction의 허용여부를 설정한다.
void setConnectTimeout(int timeout)	연결종료시간을 설정한다.
void setContentHandlerFactory(ContentHandlerFactory fac)	ContentHandlerFactory를 설정한다.
void setDefaultAllowUserInteraction(boolean defaultallowuserinteraction)	UserInteraction허용여부의 기본값을 설정한다.
void setDefaultRequestProperty(String key, String value)	RequestProperty의 기본 키쌍(key-pair)을 설정한다.
void setDefaultUseCaches(boolean defaultusecaches)	캐쉬 사용여부의 기본값을 설정한다.
void setDoInput(boolean doinput)	DoInput필드의 값을 설정한다.
void setDoOutput(boolean dooutput)	DoOutput필드의 값을 설정한다.
void setFileNameMap(FileNameMap map)	FileNameMap을 설정한다.



# URLConnection의 메서드

메서드	설명
<code>void setIfModifiedSince(long ifmodifiedsince)</code>	ModifiedSince필드의 값을 설정한다.
<code>void setReadTimeout(int timeout)</code>	읽기제한시간을 설정한다.(천분의 일초)
<code>void setRequestProperty(String key, String value)</code>	ReqeustProperty에 (key, value)를 저장한다.
<code>void setUseCaches(boolean usecaches)</code>	캐쉬의 사용여부를 설정한다.



## 예제2

---

```
InputStream in = url.openStream();
```



```
URLConnection con = url.openConnection();  
InputStream in = con.getInputStream();
```



# 소켓 프로그래밍

---



# 소켓 프로그래밍

---

- 소켓 프로그래밍
  - 소켓을 이용한 통신 프로그래밍
  - 클라이언트와 서버간의 일대일 통신
  - 소켓(socket) - 프로세스간의 통신에 사용되는 **양쪽 끝 단**을 의미함
  - 서로 멀리 떨어진 두 사람이 통신하기 위해서 전화기가 필요한 것처럼, **프로세스간의 통신을 위해서는 소켓이 필요**함
  - java.net 패키지 이용



# 통신 방식

---

## ■ 1. TCP 통신 방식

- 양방향 연결 기반 통신 방식(Connection Oriented 방식)
- 신뢰성 있는 데이터 전송(수신여부 확인)
- 연결 후 통신 - 전화 통신 방식
- 데이터가 손실되면 재전송됨
  
- 서버와 클라이언트가 TCP 방식으로 통신하려면
  - 서버와 클라이언트가 있어야 하고
  - 서로 연결이 이뤄진 후
  - 데이터를 주고 받는다.

## ■ 2. UDP 통신방식

- 비연결 기반 통신 방식
- 신뢰성 없는 데이터 전송(수신여부 확인 안함)
- 연결 없이 통신 - 우편물 배달방식
- 데이터가 손실될 수 있다
- 전송속도가 빠르다

# TCP 와 UDP

## ■ TCP/IP 프로토콜

- 이기종 시스템간의 통신을 위한 표준 프로토콜
- TCP와 UDP 모두 TCP/IP 프로토콜에 포함되어 있음
- TCP와 UDP는 전송방식이 다르며, 각 방식에 따른 장단점이 있음

항목	TCP	UDP
연결방식	.연결기반(connection-oriented) - 연결 후 통신(전화기) - 1:1 통신방식	.비연결기반(connectionless-oriented) - 연결없이 통신(소포) - 1:1, 1:n, n:n 통신방식
특징	.데이터의 경계를 구분안함 (byte-stream) .신뢰성 있는 데이터 전송 - 데이터의 전송순서가 보장됨 - 데이터의 수신여부를 확인함 (데이터가 손실되면 재전송됨) - 패킷을 관리할 필요가 없음 .UDP보다 전송속도가 느림	.데이터의 경계를 구분함.(datagram) .신뢰성 없는 데이터 전송 - 데이터의 전송순서가 바뀔 수 있음 - 데이터의 수신여부를 확인안함 (데이터가 손실되어도 알 수 없음) - 패킷을 관리해주어야 함 .TCP보다 전송속도가 빠름
관련 클래스	.Socket .ServerSocket	.DatagramSocket .DatagramPacket .MulticastSocket





# TCP 와 UDP

## ■ TCP

- 데이터를 전송하기 전에 먼저 상대방과 연결을 한 후에 데이터를 전송하며 잘 전송되었는지 확인하고 전송에 실패했다면 해당 데이터를 재전송하기 때문에 신뢰성 있는 데이터의 전송이 요구되는 통신에 적합함
- 전화에 비유
- 예) 파일을 주고받는데 적합

## ■ UDP

- 상대방과 연결하지 않고 데이터를 전송하며, 데이터를 전송하지만 데이터가 바르게 수신되었는지 확인하지 않기 때문에 데이터가 전송되었는지 확인할 길이 없음
- 데이터를 보낸 순서대로 수신한다는 보장이 없다.
- 이러한 확인과정이 필요하지 않기 때문에 TCP에 비해 빠른 전송이 가능함
- 예) 게임이나 동영상의 데이터를 전송하는 경우 - 데이터가 중간에 손실되어 좀 끊기더라도 빠른 전송이 필요할 때 적합



# TCP 소켓 프로그래밍

- TCP 소켓 프로그래밍
  - 클라이언트와 서버간의 일대일 통신
  - 먼저 서버 프로그램이 실행되어 클라이언트 프로그램의 연결요청을 기다리고 있어야 함
- 서버 프로그램과 클라이언트 프로그램간의 통신과정
  1. 서버 프로그램에서는 **서버소켓을 사용해서** 서버 컴퓨터의 특정 포트에서 클라이언트의 **연결 요청**을 처리할 **준비**를 한다.
  2. 클라이언트 프로그램은 접속할 서버의 **IP** 주소와 포트정보를 가지고 **소켓**을 생성해서 **서버에 연결을 요청**한다.
  3. 서버소켓은 클라이언트의 연결요청을 받으면 서버에 **새로운 소켓을 생성**해서 **클라이언트의 소켓과 연결되도록** 한다.
  4. 이제 클라이언트의 소켓과 새로 생성된 서버의 소켓은 서버소켓과 관계 없이 **일대일 통신**을 한다.

# TCP 소켓 프로그래밍

## ■ 서버소켓(ServerSocket)

- 포트와 결합되어 포트를 통해 원격 사용자의 연결요청을 기다리다가 연결요청이 올 때마다 새로운 소켓을 생성하여 상대방 소켓과 통신할 수 있도록 연결함
- 여기까지가 서버소켓의 역할
- 실제적인 데이터 통신은 서버소켓과 관계없이 소켓과 소켓 간에 이루어짐

## ■ 포트(port)

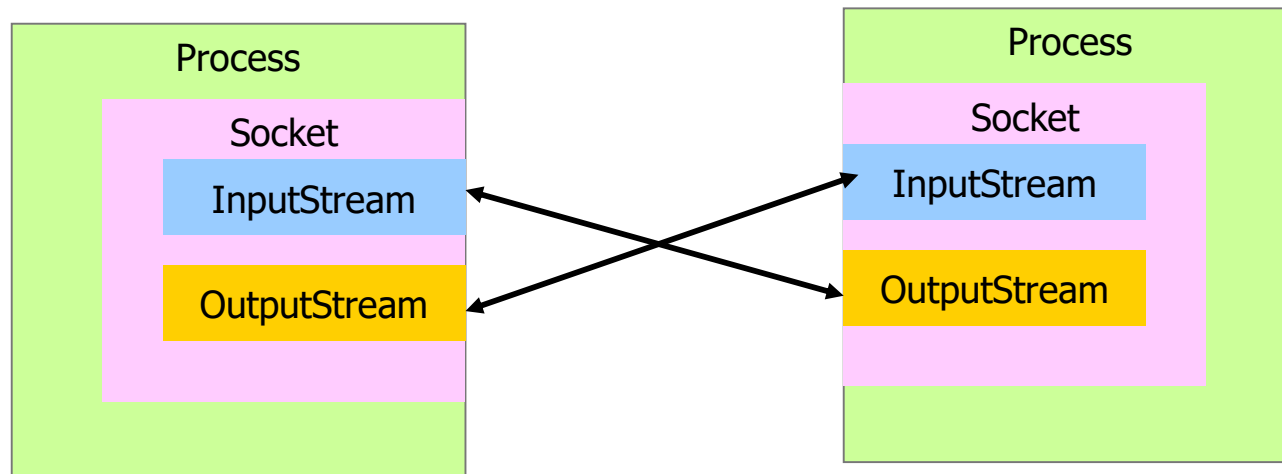
0~1024 사이의 값은 시스템에서 사용하는 포트번호  
예) http : 80, telnet : 23, ftp : 21

- 관문
- **호스트(컴퓨터)가 외부와 통신을 하기 위한 통로**
- 하나의 호스트가 65,536개의 포트를 가지고 있으며 포트는 번호로 구별됨
- 포트의 번호 : 0~65,535의 범위에 속하는 값

- 서버소켓 - 소켓간의 연결만 처리하고, 실제 데이터는 소켓들끼리 서로 주고 받는다.
- 소켓들이 데이터를 주고받는 연결 통로는 입출력 스트림이다.

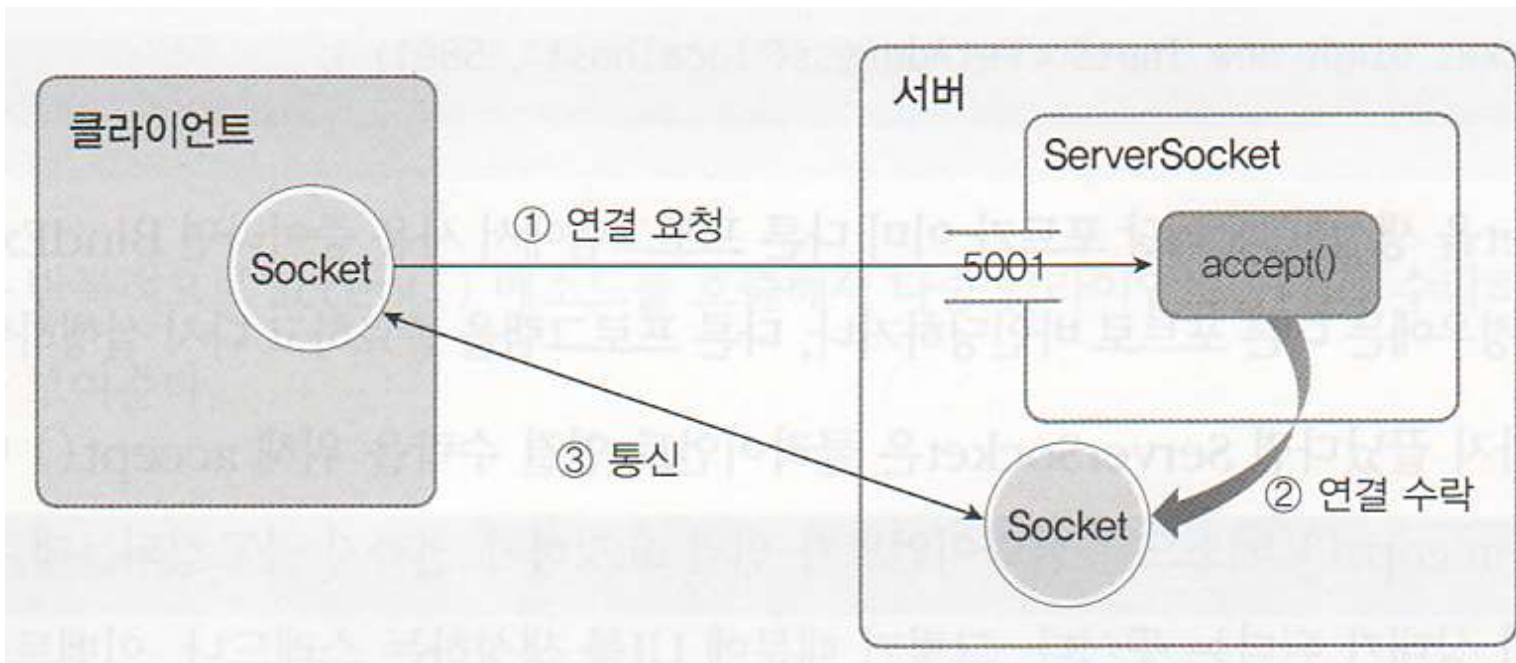
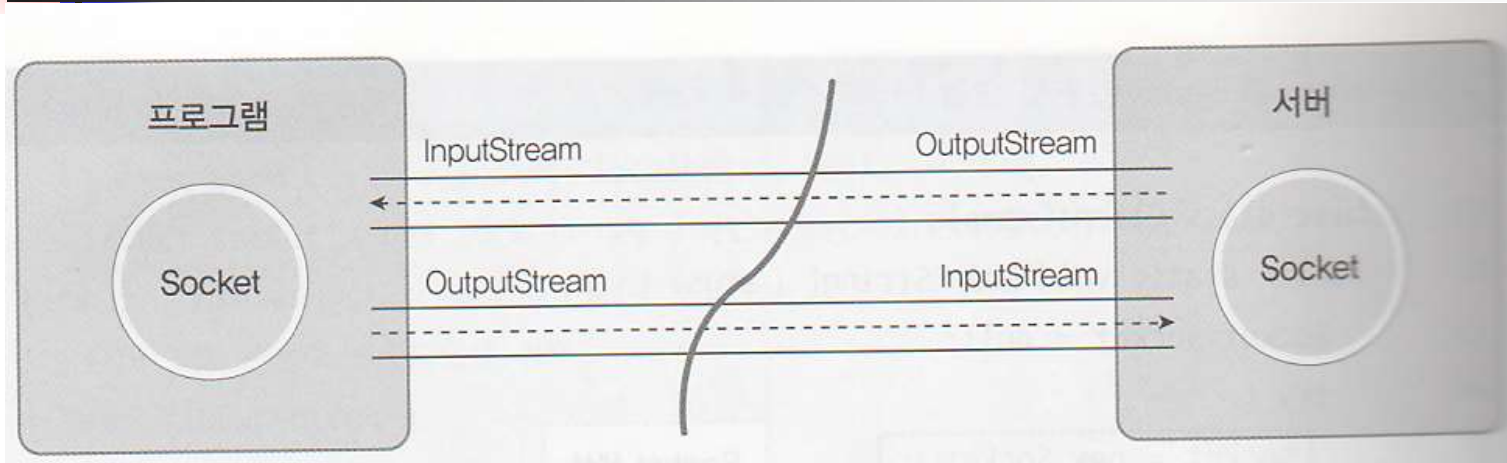
# TCP 소켓 프로그래밍

- 소켓은 두 개의 스트림, 입력 스트림과 출력 스트림을 가지고 있으며, 이 스트림들은 연결된 상대방 소켓의 스트림들과 교차연결됨
- 한 소켓의 입력스트림은 상대방 소켓의 출력스트림과 연결되고, 출력스트림은 입력스트림과 연결됨
- 한 소켓에서 출력스트림으로 데이터를 보내면 상대방 소켓에서는 입력스트림으로 받게 됨



소켓간의 입출력 스트림 연결

# TCP 소켓 프로그래밍





# TCP 소켓 프로그래밍

- TCP를 이용한 소켓 프로그래밍을 위해 Socket과 ServerSocket 클래스 제공

- **Socket** - 프로세스간의 통신을 담당하며, **InputStream**과 **OutputStream**을 가지고 있다.

이 두 스트림을 통해 프로세스간의 통신(입출력)이 이루어진다.

- **ServerSocket** - 포트와 연결(**bind**)되어 외부의 연결요청을 기다리다 연결요청이 들어오면, **Socket**을 생성해서 소켓과 소켓간의 통신이 이루어지도록 한다. 한 포트에 하나의 **ServerSocket**만 연결할 수 있다.

# 예제-TcpIpServer

```
[09:06:26]서버가 준비되었습니다.  
[09:06:26]연결요청을 기다립니다.  
[09:07:52]127.0.0.1로부터 연결요청이 들어왔습니다.  
[09:07:52]데이터를 전송했습니다.  
[09:07:52]연결요청을 기다립니다.
```

```
import java.net.*;  
import java.io.*;  
import java.util.Date;  
import java.text.SimpleDateFormat;
```

```
public class TcpIpServer {  
    public static void main(String args[]) {  
        ServerSocket serverSocket = null;
```

```
        try {  
            // 서버소켓을 생성하여 7777번 포트와 결합(bind)시킨다.  
            serverSocket = new ServerSocket(7777);  
            System.out.println(getTime()+"서버가 준비되었습니다.");
```

```
        } catch(IOException e) {  
            e.printStackTrace();  
        }
```

```
        while(true) {  
            try {  
                System.out.println(getTime()+"연결요청을 기다립니다.");  
                // 서버소켓은 클라이언트의 연결요청이 올 때까지 실행을 멈추고 계속 기다린다.  
                // 클라이언트의 연결요청이 오면 클라이언트 소켓과 통신할 새로운 소켓을 생성한다.  
                Socket socket = serverSocket.accept();  
                System.out.println(getTime()+ socket.getInetAddress()  
                    + "로부터 연결요청이 들어왔습니다.");
```

간단한 TCP/IP 서버 프로그램



# 예제-TcpIpServer

```
// 소켓의 출력스트림을 얻는다.
OutputStream out = socket.getOutputStream();
DataOutputStream dos = new DataOutputStream(out);

// 원격 소켓(remote socket)에 데이터를 보낸다.
dos.writeUTF("[Notice] Test Message1 from Server.");
System.out.println(getTime()+"데이터를 전송했습니다.");

// 스트림과 소켓을 닫아준다.
dos.close();
socket.close();
} catch (IOException e) {
    e.printStackTrace();
}
} // while
} // main

// 현재시간을 문자열로 반환하는 함수
static String getTime() {
    SimpleDateFormat f = new SimpleDateFormat("[hh:mm:ss]");
    return f.format(new Date());
}
} // class
```





# 예제 – TcpIpClient

TCP/IP 서버와 통신하기 위한 클라이언트 프로그램

```
import java.net.*;
import java.io.*;
public class TcpIpClient {
    public static void main(String args[]) {
        try {
            String serverIp = "127.0.0.1";

            System.out.println("서버에 연결중입니다. 서버IP : " + serverIp);
            // 소켓을 생성하여 연결을 요청한다.
            Socket socket = new Socket(serverIp, 7777);

            // 소켓의 입력스트림을 얻는다.
            InputStream in = socket.getInputStream();
            DataInputStream dis = new DataInputStream(in);

            // 소켓으로 부터 받은 데이터를 출력한다.
            System.out.println("서버로부터 받은 메세지 : "+dis.readUTF());
            System.out.println("연결을 종료합니다.");

            // 스트림과 소켓을 닫는다.
            dis.close();
            socket.close();
            System.out.println("연결이 종료되었습니다.");
        } catch (ConnectException ce) {
            ce.printStackTrace();
        } catch (IOException ie) {
            ie.printStackTrace();
        } catch (Exception e) {
            e.printStackTrace();
        }
    } // main
} // class
```



# 예제

---

## ■ TCP/IP 서버 프로그램

- 서버소켓이 7777번 포트에서 클라이언트 프로그램의 연결요청을 기다린다.
- 클라이언트의 요청이 올 때까지 진행을 멈추고 계속 기다린다.
- 클라이언트 프로그램이 서버에 연결을 요청하면, 서버소켓은 새로운 소켓을 생성하여 클라이언트 프로그램의 소켓(원격소켓)과 연결한다
- 새로 생성된 소켓은 "[Notice] Test Message1 from Server." 라는 데이터를 원격소켓에 전송하고 연결을 종료한다.
- 서버소켓은 다시 클라이언트 프로그램의 요청을 기다린다.



# 예제

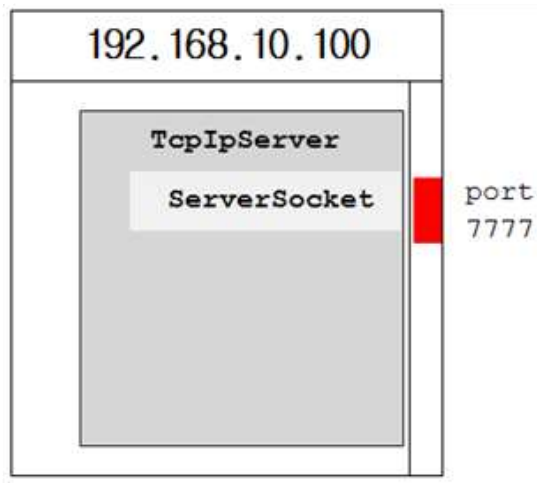
---

## ■ TCP/IP 서버와 통신하기 위한 클라이언트 프로그램

- 연결하고자 하는 서버의 **IP**와 포트번호를 가지고 소켓을 생성하면 자동적으로 서버에 연결요청을 하게 된다
- 서버 프로그램이 실행되고 있지 않거나 서버의 전원이 꺼져 있어서 서버와 연결을 실패하면 **ConnectException** 이 발생
- 서버와 연결되면 소켓의 입력스트림을 얻어서 서버가 전송한 데이터를 읽을 수 있다.
- 서버와의 작업이 끝나면 소켓과 스트림을 닫아야 한다.

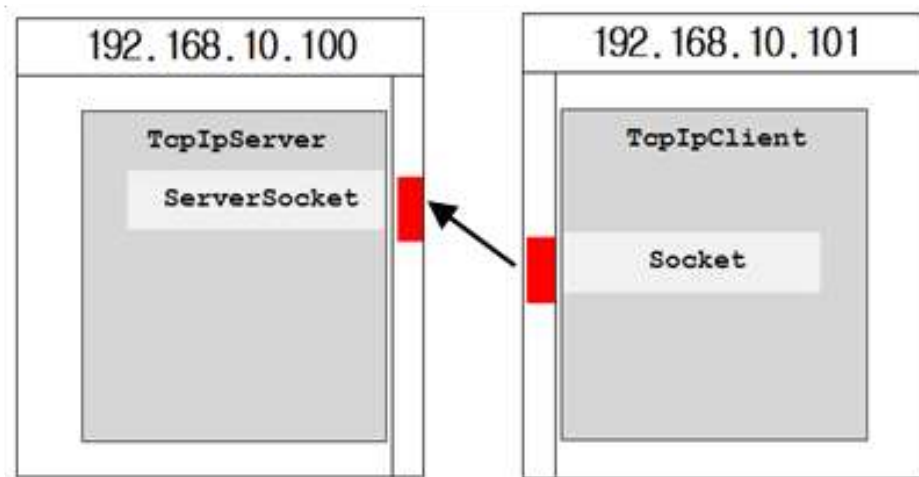
# 예제-서버와 클라이언트의 연결과정

- 서버와 클라이언트의 연결과정
  - 서버 IP : 192.168.10.100
  - 클라이언트 IP : 192.168.10.101
- 1. 서버프로그램(TcpIpServer.java)를 실행한다.  
d:\Wjava> java TcpIpServer
- 2. 서버 소켓을 생성한다  
`ServerSocket serverSocket = new ServerSocket(7777); //TcpIpServer.java`



# 예제-서버와 클라이언트의 연결과정

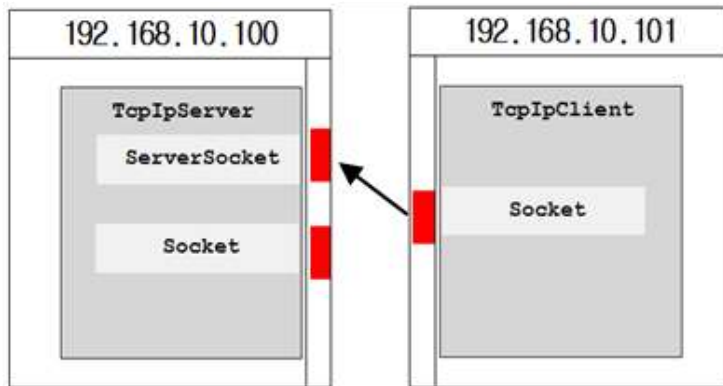
- 3. 서버소켓이 클라이언트 프로그램의 연결요청을 처리할 수 있도록 대기상태로 만든다.  
(클라이언트 프로그램의 연결요청이 오면 새로운 소켓을 생성해서 클라이언트 프로그램의 소켓과 연결한다.)  
`Socket socket = serverSocket.accept(); //TcplpServer.java`
- 4. 클라이언트 프로그램(TcplpClient.java)에서 소켓을 생성하여 서버소켓에 연결을 요청한다.  
`Socket socket = new Socket("192.168.10.100", 7777); //TcplpClient.java`



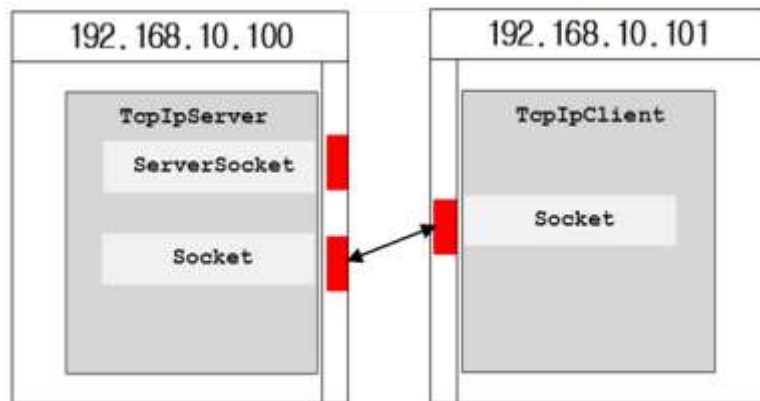
# 예제-서버와 클라이언트의 연결과정

- 5. 서버소켓은 클라이언트 프로그램의 연결요청을 받아 새로운 소켓을 생성하여 클라이언트 프로그램의 소켓과 연결한다.

```
Socket socket = serverSocket.accept(); //TcpIpServer.java
```



- 6. 새로 생성된 서버의 소켓(서버소켓 아님)은 클라이언트의 소켓과 통신한다.





## 예제-TcplpServer 수정

---

```
import java.net.*;
import java.io.*;
import java.util.Date;
import java.text.SimpleDateFormat;

public class TcplpServer11 {
    public static void main(String args[]) {
        ServerSocket serverSocket = null;

        try {
            // 서버소켓을 생성하여 7777번 포트와 결합(bind)시킨다.
            serverSocket = new ServerSocket(7777);
            System.out.println(getTime()+"서버가 준비되었습니다.");

        } catch(IOException e) {
            e.printStackTrace();
        }

        while(true) {
            try {
                System.out.println(getTime()+"연결요청을 기다립니다.");
```

```

// 클라이언트의 연결요청이 오면 클라이언트 소켓과 통신할 새로운 소켓을 생성한다.
Socket socket = serverSocket.accept();
System.out.println(getTime()+ socket.getInetAddress()
                    + "로부터 연결요청이 들어왔습니다.");

// 소켓의 출력스트림을 얻는다.
OutputStream out = socket.getOutputStream();
DataOutputStream dos = new DataOutputStream(out);

// 원격 소켓(remote socket)에 데이터를 보낸다.
dos.writeUTF("[Notice] Test Message1 from Server.");
System.out.println(getTime()+"데이터를 전송했습니다.");

//-----
// 소켓의 입력스트림을 얻는다.
InputStream in = socket.getInputStream();
DataInputStream dis = new DataInputStream(in);

// 원격 소켓으로 부터 받은 데이터를 출력한다.
System.out.println(getTime()+"클라이언트로부터 받은 메세지 :"+dis.readUTF());
//-----

// 스트림과 소켓을 닫아준다.
dis.close();
dos.close();
socket.close();

```





## 예제-TcplpClient 수정

---

```
import java.net.*;
import java.io.*;
public class TcplpClient11 {
    public static void main(String args[]) {
        try {

            String serverIp = "127.0.0.1";

            System.out.println("서버에 연결중입니다. 서버IP : " + serverIp);
            // 소켓을 생성하여 연결을 요청한다.
            Socket socket = new Socket(serverIp, 7777);

            // 소켓의 입력스트림을 얻는다.
            InputStream in = socket.getInputStream();
            DataInputStream dis = new DataInputStream(in);

            // 소켓으로 부터 받은 데이터를 출력한다.
            System.out.println("서버로부터 받은 메세지 : "+dis.readUTF());
            System.out.println("연결을 종료합니다.");
```



# 예제-TcpIpClient

```
//-----  
// 소켓의 출력스트림을 얻는다.  
OutputStream out = socket.getOutputStream();  
DataOutputStream dos = new DataOutputStream(out);  
  
//소켓에 데이터를 보낸다.  
dos.writeUTF("안녕하세요! 클라이언트입니다.");  
System.out.println("데이터를 전송했습니다.");  
//-----  
// 스트림과 소켓을 닫아준다.  
dos.close();  
dis.close();  
socket.close();  
System.out.println("연결이 종료되었습니다.");  
} catch (ConnectException ce) {  
    ce.printStackTrace();  
} catch (IOException ie) {  
    ie.printStackTrace();  
} catch (Exception e) {  
    e.printStackTrace();  
}  
  
} // main } // class
```

## 예제2

```
import java.net.*;
import java.io.*;
import java.util.Date;
import java.text.SimpleDateFormat;
```

```
public class TcpIpServer2 {
    public static void main(String args[]) {
        ServerSocket serverSocket = null;
```

```
        try {
```

```
            // 서버소켓을 생성하여 7777번 포트와 결합(bind)시킨다.
            serverSocket = new ServerSocket(7777);
            System.out.println(getTime()+"서버가 준비되었습니다.");
```

```
        } catch(IOException e) {
            e.printStackTrace();
        }
```

```
        while(true) {
            try {
```

```
                // 서버소켓
```

```
                System.out.println(getTime()+"연결요청을 기다립니다.");
```

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

```
                System.out.println(getTime()+ socket.getInetAddress() + "로부터
```

```
연결요청이 들어왔습니다.");
```

```
[07:31:53]서버가 준비되었습니다.
[07:31:53]연결요청을 기다립니다.
[07:32:01]127.0.0.1로부터 연결요청이 들어왔습니다.
getPort():2417
getLocalPort():7777
[07:32:01]데이터를 전송했습니다.
[07:32:01]연결요청을 기다립니다.
```

- Socket 클래스에 정의된 `getPort()`, `getLocalPort()`를 사용해서 TCP/IP 통신에서 소켓이 사용하고 있는 포트를 알아낼 수 있음
- `getPort()` 가 반환하는 값 : 상대방 소켓(원격소켓)이 사용하는 포트
- `getLocalPort()` 가 반환하는 값 : 소켓(서버소켓 아님) 자신이 사용하는 포트



## 예제2

```
System.out.println("getPort():"+socket.getPort());
System.out.println("getLocalPort():" + socket.getLocalPort());

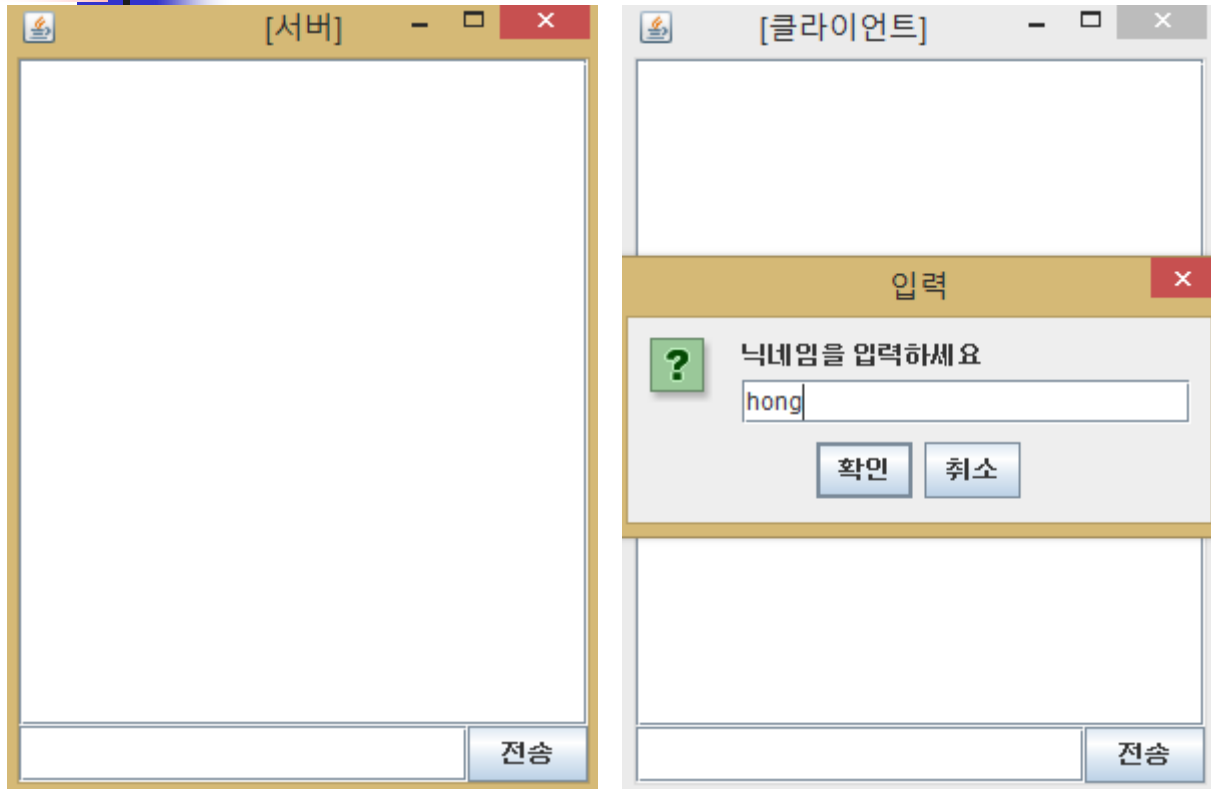
// 소켓의 출력스트림을 얻는다.
OutputStream out = socket.getOutputStream();
DataOutputStream dos = new DataOutputStream(out);

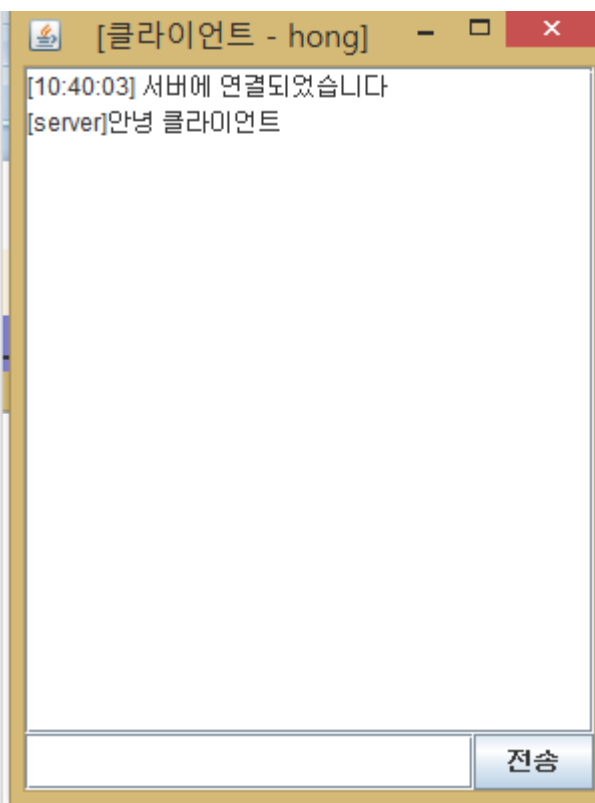
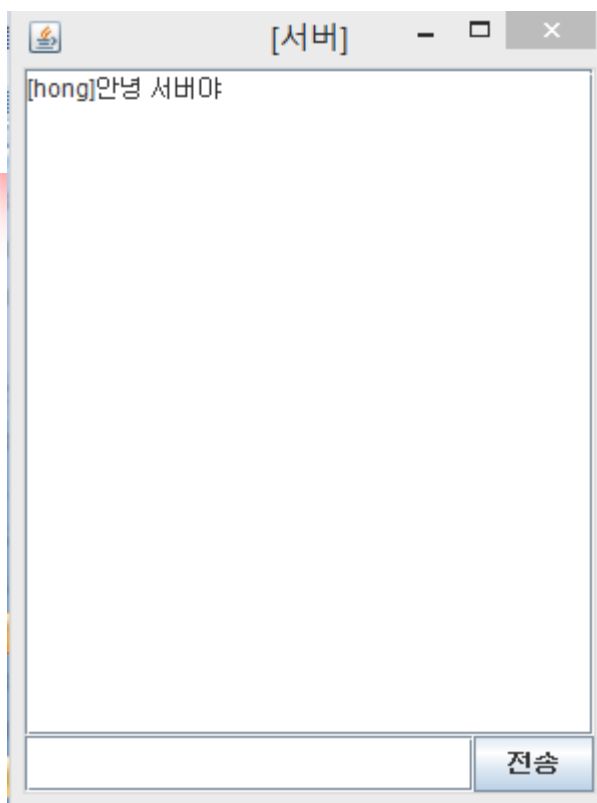
// 원격 소켓(remote socket)에 데이터를 보낸다.
dos.writeUTF("[Notice] Test Message1 from Server.");
System.out.println(getTime()+"데이터를 전송했습니다.");

// 스트림과 소켓을 닫아준다.
dos.close();
socket.close();
} catch (IOException e) {
    e.printStackTrace();
}
} // while
} // main

// 현재시간을 문자열로 반환하는 함수
static String getTime() {
    SimpleDateFormat f = new SimpleDateFormat("[hh:mm:ss]");
    return f.format(new Date());
}
} // class
```

# Client/Server 와 1대1 통신





```

public class TcpServer extends JFrame implements ActionListener{
    JTextArea taList;
    JScrollPane scrollPane;
    JTextField tfChat;
    JButton btSend;
    JPanel pl;

    Socket socket;
    DataOutputStream dos;
    String name;

    public TcpServer(){
        super("[ 서버 ]");

        taList = new JTextArea();
        taList.setEditable(false);

        scrollPane=new JScrollPane(taList);
        pl=new JPanel(new BorderLayout());
        tfChat=new JTextField();
        btSend=new JButton("전송");
        pl.add(btSend, "East");
        pl.add(tfChat,"Center");

        this.add(scrollPane, "Center");
        this.add(pl, "South");

        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
}

```

```

        this.setSize(300, 400);
        tfChat.addActionListener(this);
        btSend.addActionListener(this);
    }

    public void startMain(){
        System.out.println("[===Server===]\n\n");
        try {
            ServerSocket serverSocket = new ServerSocket(8888);
            System.out.println(DateUtil.getTime()+"서버가 준비되었습니다.\n");
            //while(true){
                //클라이언트에서 요청이 들어오면 쓰레드를 생성
                System.out.println("연결 요청을 기다립니다.\n");
                Socket socket = serverSocket.accept();
                System.out.println(socket.getInetAddress()+" , "
                    + socket.getPort() +"에서 연결 요청이 들어왔습니다.\n");
                name="server";
                //출력용 스트림
                dos= new DataOutputStream(socket.getOutputStream());
                //쓰레드 실행
                ServerReceiver th = new ServerReceiver(socket);
                th.start();

                //}
            } catch (IOException e) {
                e.printStackTrace();
            }
        }

        public static void main(String[] args) {
            TcpServer f = new TcpServer();
            f.setVisible(true);
            f.startMain();
        }
    }

```



```

class ServerReceiver extends Thread{
    Socket socket;
    DataInputStream dis;
    public ServerReceiver(Socket socket) {
        this.socket=socket;
        //입력용 스트림
        try {
            dis=new DataInputStream(socket.getInputStream());
        } catch (IOException e) {e.printStackTrace(); }
    } //생성자
    public void run() {
        try {
            while(dis!=null){
                String data =dis.readUTF();
                taList.append(data+"\n");
            }
        } catch (IOException e) {e.printStackTrace(); }
    } //run()
} //내부 class

public void actionPerformed(ActionEvent e) {
    if(e.getSource()==btSend || e.getSource()==tfChat){
        try {
            if(dos!=null){
                dos.writeUTF("[ "+name+" ]"+ tfChat.getText() +"\n");
                tfChat.setText("");
                tfChat.requestFocus();
            }
        } catch (IOException e1) { e1.printStackTrace(); }
    }
} //class

```

```

public class TcpClient extends JFrame implements ActionListener{
    JTextArea taList;
    JScrollPane scrollPane;
    JTextField tfChat;
    JButton btSend;
    JPanel pl;
    Socket socket;
    DataOutputStream dos;
    String name;
    public TcpClient(){
        super("[클라이언트]");
        taList = new JTextArea();
        taList.setEditable(false);
        scrollPane=new JScrollPane(taList);
        pl=new JPanel(new BorderLayout());
        tfChat=new JTextField();
        btSend=new JButton("전송");
        pl.add(btSend, "East");
        pl.add(tfChat,"Center");
        this.add(scrollPane, "Center");
        this.add(pl, "South");
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        this.setSize(300, 400);
        tfChat.addActionListener(this);
        btSend.addActionListener(this);
    }
    public void startMain(){
        name=JOptionPane.showInputDialog(this, "닉네임을 입력하세요");
    }
}

```

```

this.setTitle("[클라이언트 - "+ name + "]);
try {
    socket=new Socket("127.0.0.1", 8888);
    taList.append(DateUtil.getTime() + "서버에 연결되었습니다\r\n");
    //입력용 쓰레드 실행
    ClientReceiver th = new ClientReceiver(socket);
    th.start();
    //출력용 스트림
    dos= new DataOutputStream(socket.getOutputStream());
} catch (IOException e) {
    e.printStackTrace();
}
} //startMain()
//내부 클래스 - 입력용 쓰레드
class ClientReceiver extends Thread{
    Socket socket;
    DataInputStream dis;
    //생성자
    public ClientReceiver(Socket socket) {
        this.socket=socket;
        //입력용 스트림
        try {
            dis=new DataInputStream(socket.getInputStream());
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
    public void run() {
        try {
            while(dis!=null){
                String data =dis.readUTF();
                taList.append(data+"\r\n");
            }
        }
    }
}

```

```

        } catch (IOException e) {
            //e.printStackTrace();
        }

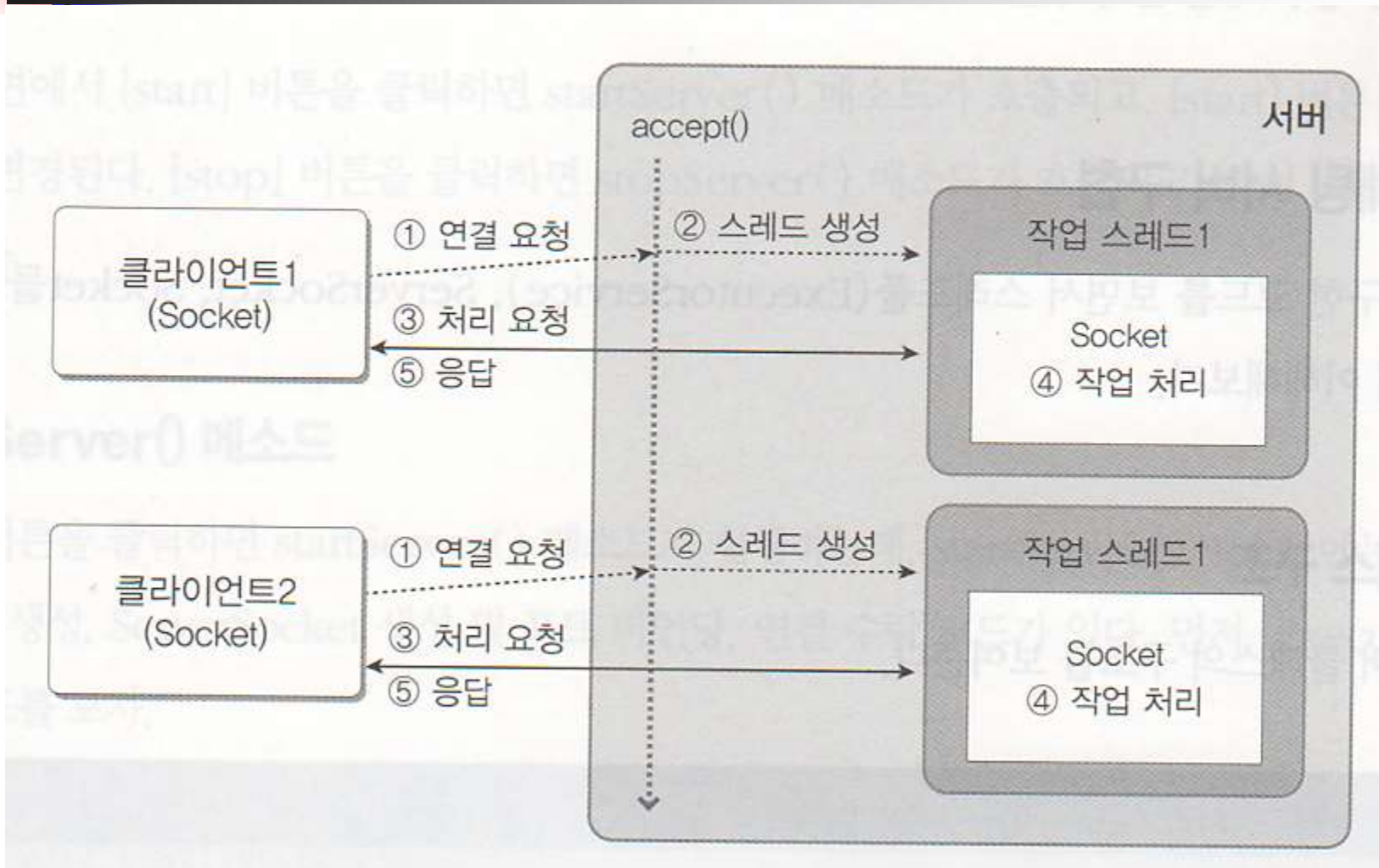
    } //run()
} //내부클래스

public static void main(String[] args) {
    TcpClient f = new TcpClient();
    f.setVisible(true);
    f.startMain();
}

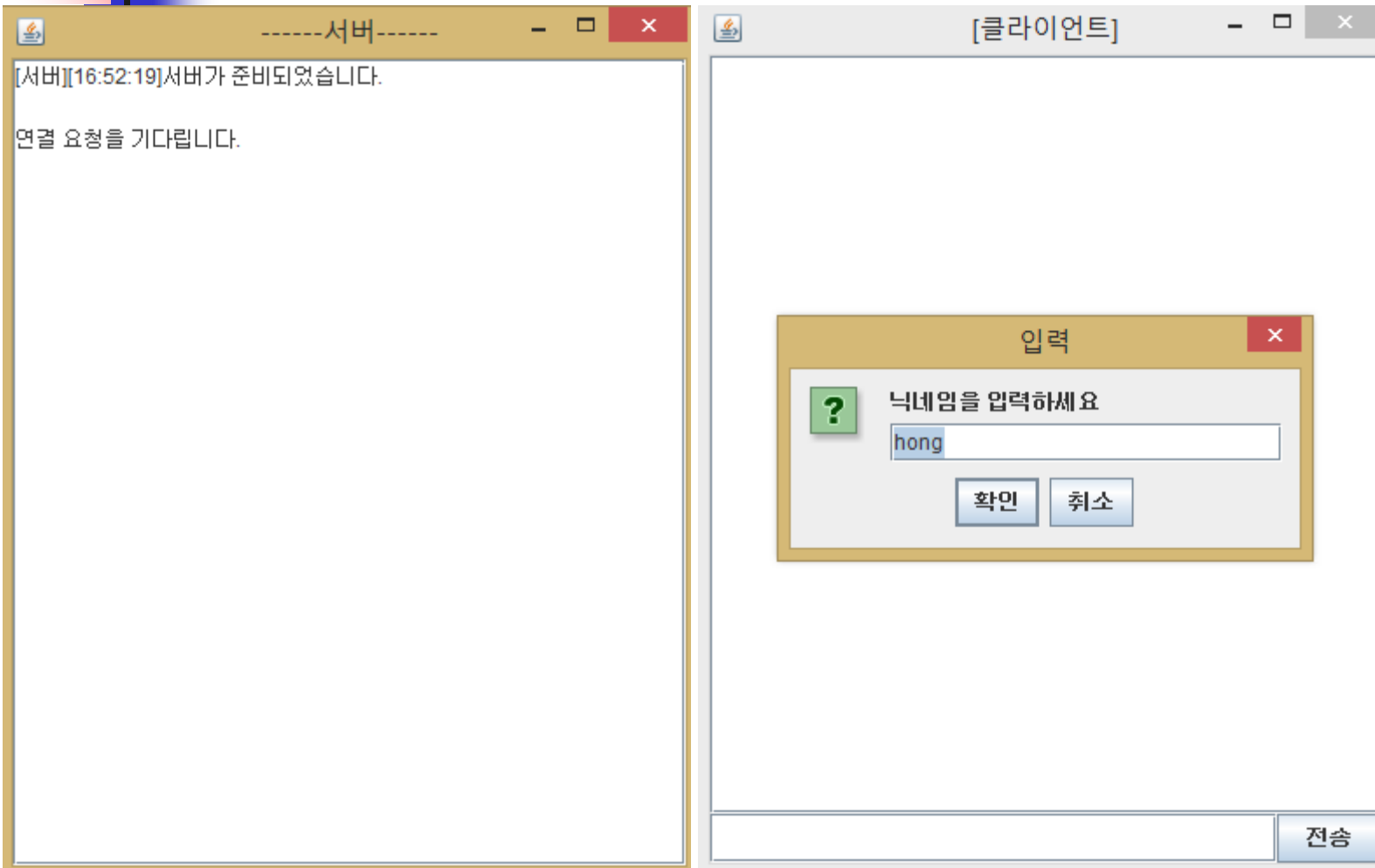
@Override
public void actionPerformed(ActionEvent e) {
    if(e.getSource()==btSend || e.getSource()==tfChat){
        try {
            if(dos!=null){
                dos.writeUTF("[ "+name+" ]"+ tfChat.getText() + "\n");
                tfChat.setText("");
                tfChat.requestFocus();
            }
        } catch (IOException e1) {
            e1.printStackTrace();
        }
    }
}
}
}
}

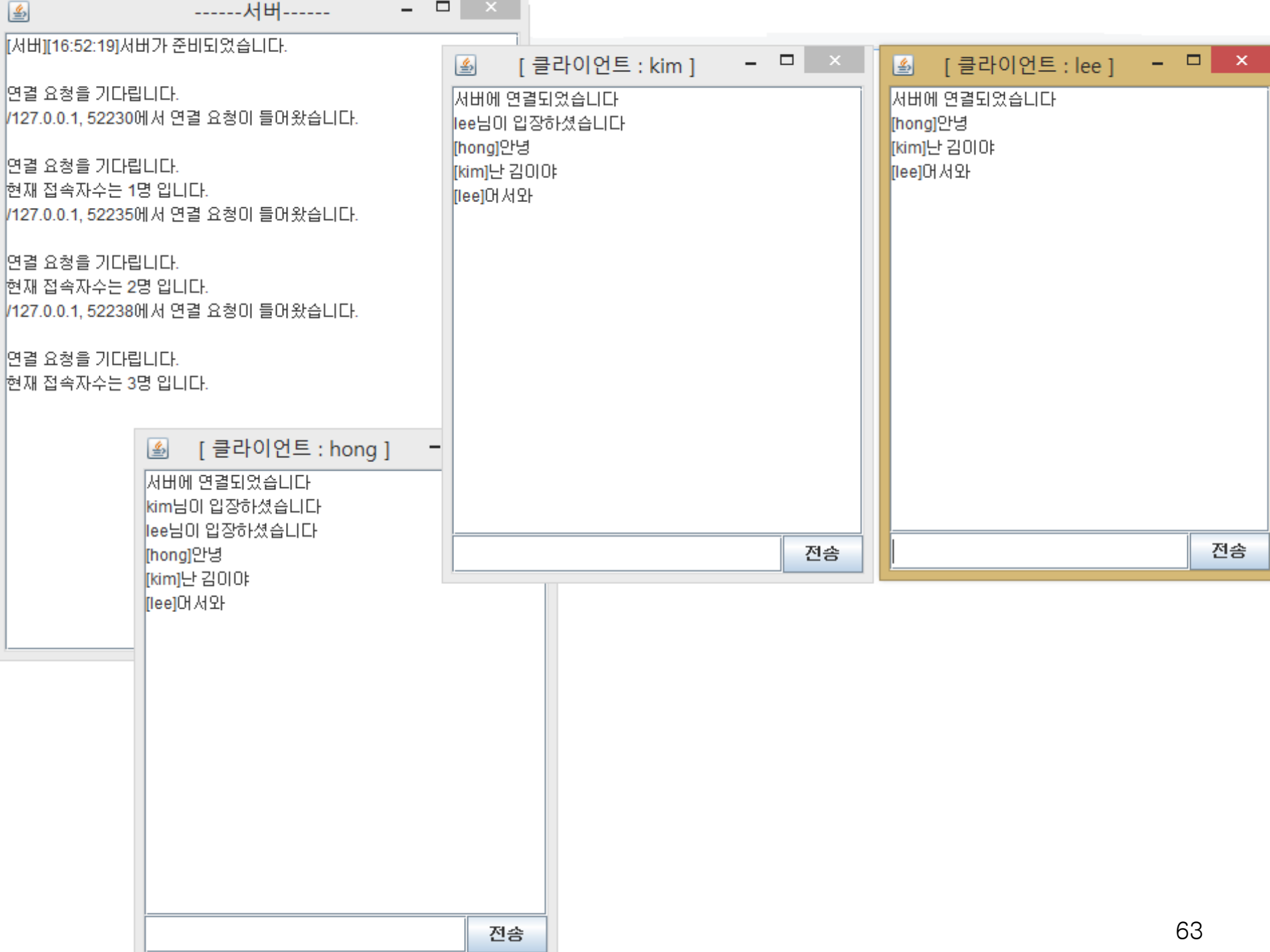
```

# 멀티 채팅



# 멀티 채팅





[서버][16:52:19]서버가 준비되었습니다.

연결 요청을 기다립니다.

/127.0.0.1, 52230에서 연결 요청이 들어왔습니다.

연결 요청을 기다립니다.

현재 접속자수는 1명 입니다.

/127.0.0.1, 52235에서 연결 요청이 들어왔습니다.

연결 요청을 기다립니다.

현재 접속자수는 2명 입니다.

/127.0.0.1, 52238에서 연결 요청이 들어왔습니다.

연결 요청을 기다립니다.

현재 접속자수는 3명 입니다.

[127.0.0.1,52238] 에서 접속을 종료하였습니다.

현재 서버 접속자수는 2

[ 클라이언트 : kim ]

서버에 연결되었습니다

lee님이 입장하셨습니다

[hong]안녕

[kim]난 김이야

[lee]어서와

[lee]바이

lee님이 나갔습니다.

[ 클라이언트 : hong ]

서버에 연결되었습니다

kim님이 입장하셨습니다

lee님이 입장하셨습니다

[hong]안녕

[kim]난 김이야

[lee]어서와

[lee]바이

lee님이 나갔습니다.

- 여러 클라이언트가 서버에 접속해서 채팅을 할 수 있는 멀티채팅 서버 프로그램
- 서버에 접속한 클라이언트를 **HashMap**에 저장해서 관리
- 클라이언트가 멀티채팅서버에 접속하면 **HashMap**에 저장되고, 접속을 해제하면 **HashMap**에서 제거
- 클라이언트가 데이터를 입력하면, 멀티채팅서버는 **HashMap**에 저장된 모든 클라이언트에게 데이터를 전송함
- 멀티 채팅 서버의 **ServerReceiver** 스레드는 클라이언트가 추가될 때마다 생성되며, 클라이언트의 입력을 서버에 접속된 모든 클라이언트에게 전송하는 일을 함



```

public class TcpMultiServer2 extends JFrame{
    JTextArea taList;
    JScrollPane scrollPane;
    HashMap<String, DataOutputStream> hashMap = new HashMap<String, DataOutputStream>();

    public TcpMultiServer2(){
        super("----- 서버 -----");
        taList=new JTextArea();
        taList.setEditable(false);
        scrollPane=new JScrollPane(taList);
        this.add(scrollPane);
        this.setSize(400,500);
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        //HashMap동기화 - 동시에 들어오지 못하게 한 명씩 들어올 수 있게 동기화 시켜준다.
        Collections.synchronizedMap(hashMap);
    }

    public static void main(String[] args) {
        TcpMultiServer2 obj = new TcpMultiServer2();
        obj.setVisible(true);
        obj.startMain();
    }

    public void startMain(){
        taList.append("[ 서버 ]");
        try {
            ServerSocket serverSocket = new ServerSocket(7777);
            taList.append(DateUtil.getTime()+ "서버가 준비되었습니다.\n");
        }
    }
}

```

```

while(true){
    taList.append("Wn연결 요청을 기다립니다.Wn");
    Socket socket = serverSocket.accept();

    taList.append(socket.getInetAddress()+" ",
        +socket.getPort() + "에서 연결 요청이 들어왔습니다.Wn");

    ServerReceiver th = new ServerReceiver(socket);
    th.start();
}
} catch (IOException e) {
    e.printStackTrace();
}
}

```

// 내부클래스 - 스레드 : 연결 요청하는 모든 클라이언트들을 HashMap에 저장하고  
 // 클라이언트가 전송한 데이터를 모든 클라이언트들에게 전달한다.

```

class ServerReceiver extends Thread{
    Socket socket;
    DataInputStream dis;
    DataOutputStream dos;

    public ServerReceiver(Socket socket) {
        this.socket = socket;
        try {
            dis = new DataInputStream(socket.getInputStream());
            dos = new DataOutputStream(socket.getOutputStream());
        } catch (IOException e) {e.printStackTrace(); }
    }
}

```

```

public void run() {
    String name = "";
    try {

        //클라이언트에서 보내는 닉네임
        name = dis.readUTF();
        sendToAll(name, name + "님이 입장하셨습니다.Wn");

        //각 클라이언트를 HashMap에 저장
        hashMap.put(name, dos);
        taList.append("현재 접속자수는 " + hashMap.size()+"명 입니다.Wn");

        //클라이언트가 보낸 데이터를 모든 클라이언트들에게 전달
        while(dis!=null){
            String data = dis.readUTF();
            sendToAll(name, data);
        }
    } catch (IOException e) { //e.printStackTrace();
    } finally {
        sendToAll(name, name+"님이 나갔습니다.Wn");

        //hashmap에서 제거
        hashMap.remove(name);

        taList.append("[ "+socket.getInetAddress()+", "
            +socket.getPort()+"] 에서 접속을 종료하였습니다.Wn");
        taList.append("현재 서버 접속자수는 " + hashMap.size()+"Wn");
    }
}
}

```

```
public void sendToAll(String name, String data) {
    Iterator<String> iter = hashMap.keySet().iterator();
    while(iter.hasNext()){
        String key = iter.next();

        DataOutputStream dos = hashMap.get(key);
        try {
            dos.writeUTF(data);
        } catch (IOException e) {
            //e.printStackTrace();
        }
    }
}

//while
}

class
```

- 클라이언트가 새로 추가되었을  
언트의 출력 스트림을 **HashMap**  
가 입력한 데이터를 전송하는 데

- 클라이언트가 새로 추가되었을 때 클라이언트의 이름을 **key**로 클라이언트의 출력 스트림을 **HashMap**인 **clients**에 저장해서 다른 클라이언트가 입력한 데이터를 전송하는 데 사용함
- 클라이언트가 종료되어 클라이언트의 입력 스트림(**in**)이 **null**이 되면 **while** 문을 빠져나가서 **clients**의 목록에서 해당 클라이언트를 제거

```
public class TcpMultiClient2 extends JFrame implements ActionListener {
```

```
    JTextArea ta;
```

```
    JScrollPane scrollPane;
```

```
    JTextField tfChat;
```

```
    JButton btSend;
```

```
    JPanel pl;
```

```
    Socket socket;
```

```
    DataOutputStream dos1;
```

```
    String name1;
```

```
    public TcpMultiClient2(){
```

```
        super("[클라이언트]");
```

```
        ta=new JTextArea();
```

```
        scrollPane = new JScrollPane(ta);
```

```
        tfChat=new JTextField();
```

```
        btSend=new JButton("전송");
```

```
        pl=new JPanel(new BorderLayout());
```

```
        pl.add(tfChat,"Center");
```

```
        pl.add(btSend, "East");
```

```
        this.add(scrollPane, "Center");
```

```
        this.add(pl, "South");
```

```
        this.setSize(300,400);
```

```
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```

```
        btSend.addActionListener(this);
```

```
        tfChat.addActionListener(this);
```

```
    }
```

```
    public static void main(String[] args) {
```

```
        TcpMultiClient2 obj = new TcpMultiClient2();
```

```

    obj.setVisible(true);
    obj.startMain();
}

public void startMain(){
    String str=JOptionPane.showInputDialog(this, "닉네임을 입력하세요", "hong");
    name1=str;
    this.setTitle("[ 클라이언트 : "+str +" ]");
    try {
        socket = new Socket("127.0.0.1", 7777);
        ta.append("서버에 연결되었습니다Wn");

        //입력용 쓰레드 start
        ClientReceiver th = new ClientReceiver(socket);
        th.start();

        dos1 = new DataOutputStream(socket.getOutputStream());
        if(dos1!=null) dos1.writeUTF(name1);
    } catch (IOException e) {
        e.printStackTrace();
    }
}

//내부클래스 - 입력용 쓰레드
class ClientReceiver extends Thread{
    Socket socket;
    DataInputStream dis;
}

```

```

public ClientReceiver(Socket socket) {
    this.socket = socket;
    try {
        dis = new DataInputStream(socket.getInputStream());
    } catch (IOException e) {
        e.printStackTrace();
    }
}

public void run() {
    while(dis!=null){
        try {
            String data = dis.readUTF();
            ta.append(data);
        } catch (IOException e) {e.printStackTrace();}
    }
}

}

public void actionPerformed(ActionEvent e) {
    if(e.getSource()==btSend || e.getSource()==tfChat){
        if(dos1!=null){
            try {
                dos1.writeUTF("[ "+name1+" ]"+tfChat.getText()+"\n");
                tfChat.setText("");
                tfChat.requestFocus();
            } catch (IOException e1) {e1.printStackTrace();}
        }
    }
}

}
}

```

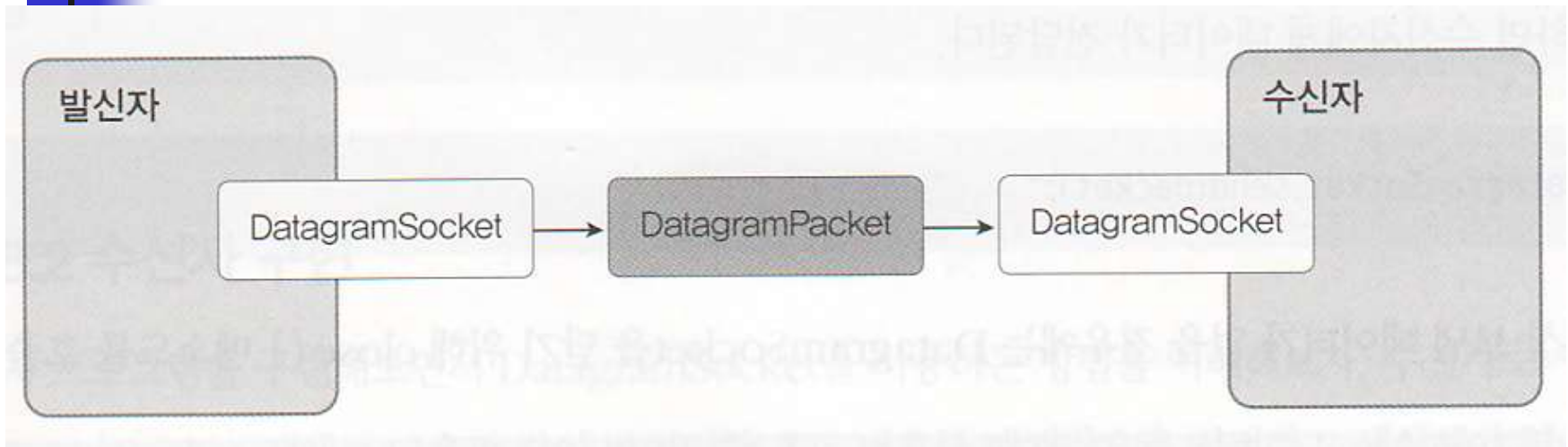


# UDP 소켓 프로그래밍

- TCP소켓 프로그래밍에서는 Socket과 ServerSocket을 사용하지만,
- UDP소켓 프로그래밍에서는 DatagramSocket과 DatagramPacket을 사용.
- UDP는 연결지향적이지 않으므로 연결요청을 받아줄 서버소켓이 필요 없다.
- DatagramSocket간에 데이터(DatagramPacket)를 주고 받는다.
- UDP - 서버 소켓과 소켓의 구분이 없으며, DatagramSocket이 두 가지 역할을 동시에 함
- DatagramSocket은 연결설정 절차 필요없이 DatagramPacket 을 상대 측에 전달할 수 있고 또 전달받을 수 있음
- DatagramPacket 클래스
  - 바이트 데이터를 보내기 위한 클래스
  - 송신할 때: 상대 ip 주소와 port 번호 모두 필요
  - 수신할 때: 버퍼(바이트 배열)만 필요



# UDP 소켓 프로그래밍



# 예제-UDPClient

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

```
//데이터를 보내는 측
```

```
class UDPClient {
    public static final int port=3000;

    public static void main(String[] args) throws Exception {
        InetAddress inet=InetAddress.getByName("127.0.0.1");
        //키보드로 보낼 메시지를 입력받는다
        Scanner sc = new Scanner(System.in);

        String msg="";
        System.out.println("보낼 내용 입력 : ");
        DatagramPacket pack=null;
        DatagramSocket ds=new DatagramSocket();
        while ((msg=sc.nextLine())!=null){
            if(msg.equalsIgnoreCase("x")) break; //종료
            byte[] data=msg.getBytes();
            pack=new DatagramPacket(data, data.length, inet, port);
            ds.send(pack);
            System.out.println("보낼 내용 입력 : ");
        }
    }
}
//main()
}
//
```

데이터 그램 소켓 생성 후 소켓 객체의 **send()** 메서드에 데이터그램 패킷 객체를 매개변수로 넣어서 전송.



# 예제 – UDPServer

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

```
//데이터를 받는 측
```

```
class UDPServer{
```

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

```
        byte[] buffer=new byte[100]; //클라이언트(송신자)가 보낸 데이터를 담아줄 바이트 배열을 생성
```

```
        DatagramSocket ds=new DatagramSocket(3000); //DatagramSocket 생성시 클라이언트 포트 번호를 매개변수로 지정
```

```
        DatagramPacket dp
```

```
            =new DatagramPacket(buffer, buffer.length); //버퍼 배열을 DatagramPacket 객체 생성시 매개변수로 지정
```

```
        while (true){
```

```
            ds.receive(dp); //DatagramSocket의 receive()메서드를 이용하여 패킷을 받는다
```

```
            byte bmsg[] =dp.getData(); //버퍼 배열에 담긴 데이터를 문자열로 만들어 출력
```

```
            String msg=new String(bmsg, 0, dp.getLength());
```

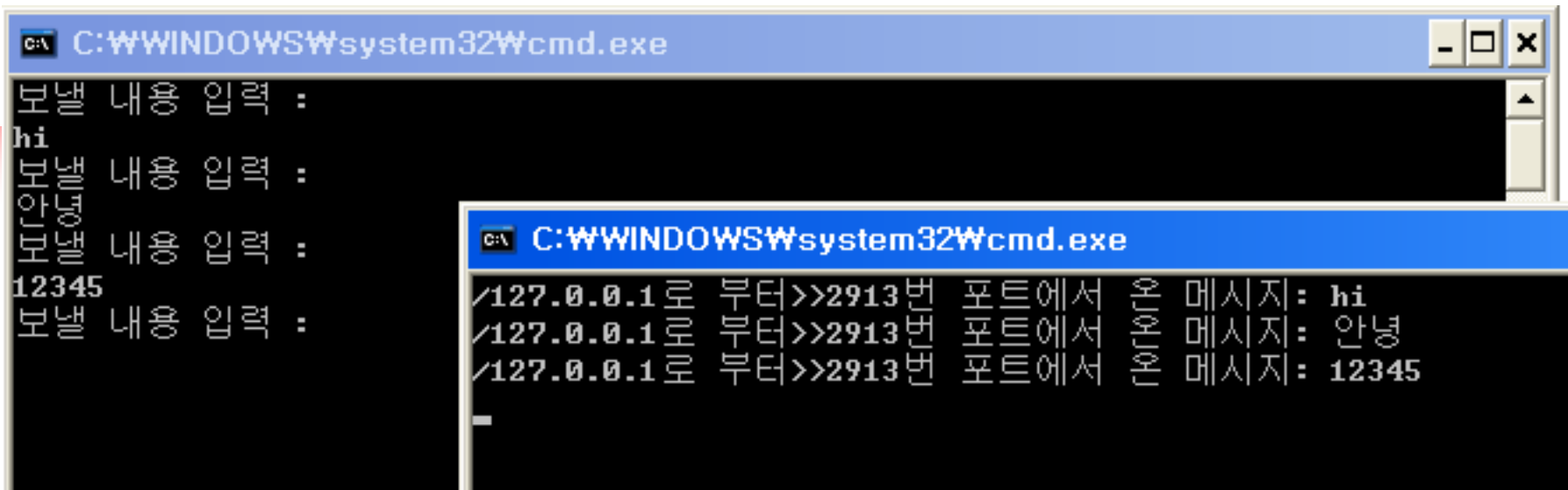
```
            System.out.println(dp.getAddress() + "로부터>>" +
```

```
                dp.getPort() + "번 포트에서 온 메시지: " + msg);
```

```
        } //while
```

```
    } //main
```

```
} //
```



The image shows two overlapping Windows command prompt windows. The background window has a title bar 'C:\WINDOWS\system32\cmd.exe' and contains the following text:

```
패킷 내용 인력 :  
hi  
패킷 내용 인력 :  
패킷 내용 인력 :  
12345  
패킷 내용 인력 :
```

The foreground window also has a title bar 'C:\WINDOWS\system32\cmd.exe' and contains the following text:

```
/127.0.0.1로 부터>>2913번 포트에서 온 메시지: hi  
/127.0.0.1로 부터>>2913번 포트에서 온 메시지: 안녕  
/127.0.0.1로 부터>>2913번 포트에서 온 메시지: 12345  
-
```