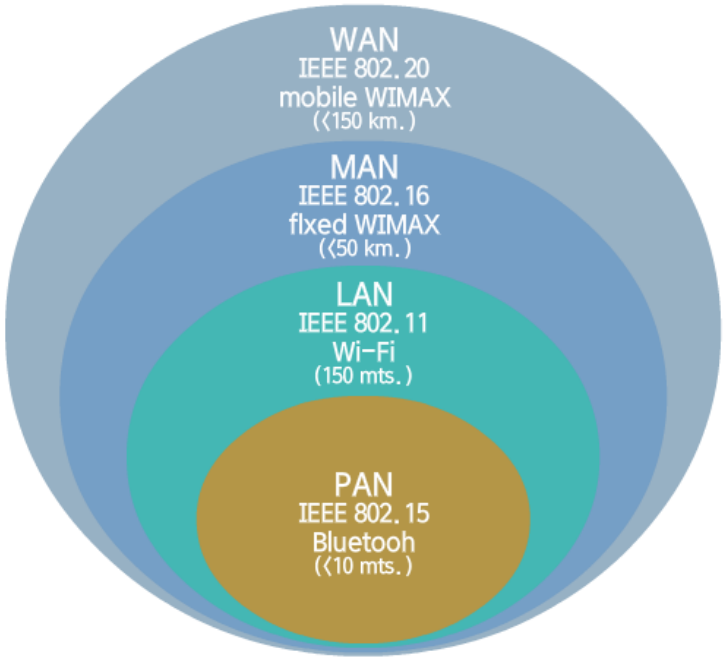
JAVA

- 네트워킹 Networking

기본 네트워크

기본 네트워크

- **네트워크**
  - 실시간으로 서로 데이터를 주고 받을 수 있는 컴퓨터 및 다른 장치들의 집합
  - 두 대 이상의 컴퓨터를 연결하고 서로 통신할 수 있는 것
  - 규모에 따른 네트워크 종류



WAN : 광대역 네트워크

MAN : 대도시 영역 네트워크

LAN : 근거리 영역 네트워크

PAN : 가장 작은 규모의 네트워크

기본 네트워크

- ## 네트워크 용어정리

1) 노드(Node)

- 네트워크에 연결된 장치
- 컴퓨터, 프린터, 라우터, 브릿지, 게이트웨이, 터미널 등

2) 주소(Address)

- 노드를 유일하게 구별해주는 일련의 바이트
- 네트워크 종류에 따라 다르게 할당됨

3) 도메인(Domain)

- 노드를 기억하기 쉽도록 노드에 부여한 이름
- 문자로 만든 인터넷 주소

4) 패킷 교환(Packet-switched)

- 데이터를 작은 단위(패킷)로 분리하여 네트워크 회선 공유
- 송신자와 수신자의 정보 포함

5) 프로토콜(Protocol)

- HTTP(Hypertext Transfer Protocol)
  : 웹 브라우저와 웹 서버가 통신하는 방법
- IEEE 802.3 : 물리적 전선에서 비트가 전기적인 신호로 바뀌는 방법 정의

6) IP(Internet Protocol)

- 군사적인 특성 내포
- 두 지점 사이의 여러 경로를 허용하고 손상된 라우터 주변 데이터의 패킷을 라우팅 할 수 있도록 설계

7) TCP(Transmission Control Protocol)

- IP 프로토콜 위에서 연결형 서비스 제공
- 각 연결의 끝에 IP 패킷의 수신 여부 확인 기능과 손상되거나 손실된 패킷에 대한 재전송 요청 기능 추가

기본 네트워크

- ## 네트워크 용어정리

8) UDP(User Datagram Protocol)

- 사용자 데이터그램 프로토콜
- 보내는 쪽에서 일방적으로 데이터를 전달하는 통신 프로토콜

11) 인터넷

- IP 프로토콜을 사용하는 컴퓨터들의 집단

8) 인터넷 주소 클래스

- InterNIC(Internet Network Information Center) 할당
- A, B, C Class는 네트워크용, D Class는 멀티캐스팅용, E Class는 예비용

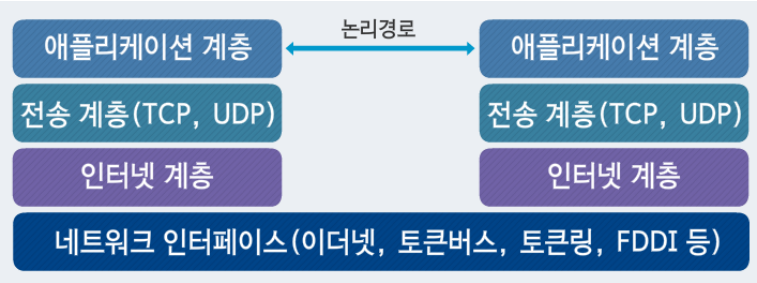
11) 9) 포트(Port)

- TCP와 UDP가 상위 계층에 제공하는 주소 표현 방식
- 1에서 65535까지의 숫자로 표현
- 1에서 1023까지는 FTP, HTTP와 같이 잘 알려진 서비스를 위해 미리 예약되어 있음

12) 대표적인 포트 할당

서비스명	서비스 내용	포트 번호	전송 계층
TCPMUX	TCP Port Service Multiplexer	1	UDP/TCP
ECHO	Echo	7	UDP/TCP
DAYTIME	Daytime	13	UDP/TCP
FTP-DATA	FTP 데이터 전송	20	TCP
FTP-CONTROL	FTP 데이터 전송 제어	21	TCP
TELNET	Telnet 터미널 에뮬레이션	23	TCP
SMTP	메일 메시지 전송 프로토콜	25	UDP/TCP
DNS	DNS 질의 응답	53	UDP/TCP
HTTP	웹 페이지	80	UDP/TCP
NTP	Network Time Protocol	123	UDP/TCP
BGP	BGP 라우팅 프로토콜	179	TCP

기본 네트워크

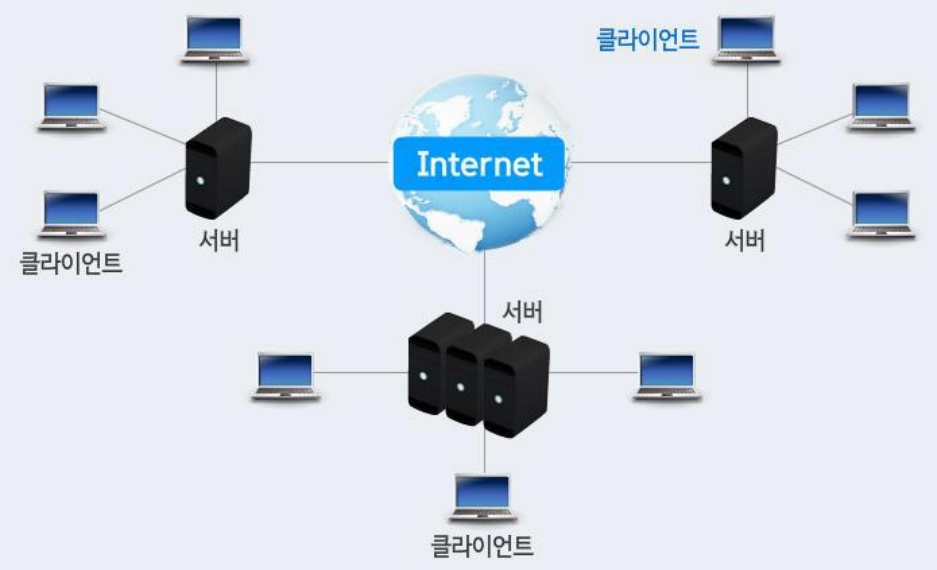


- **TCP/IP 4계층**

1. **네트워크 계층**
   - OSI의 물리계층과 데이터 링크계층의 혼합
   - 특정 네트워크 인터페이스가 물리적인 연결을 통해 로컬 네트워크 및 외부로 IP 데이터그램을 보내는 방법 정의
2. **인터넷 계층**
   - OSI 모델의 네트워크 계층(network layer) 역할
   - 데이터의 비트와 바이트를 상위 그룹인 패킷으로 구성하는 방법
   - 다른 컴퓨터가 서로를 찾을 수 있도록 주소 체계
   - IP는 가장 널리 사용되는 네트워크 계층 프로토콜
3. **전송 계층**
   - 애플리케이션 프로그램 간의 통신을 구현하는 것
   - 보낸 순서대로 수신되도록 보장할 책임이 있음
   - 패킷 손실 시 해당 패킷에 대해 송신자에게 재전송 요청
4. **애플리케이션 계층**
   - OSI 참조 모델의 세션, 프레젠테이션, 애플리케이션 계층의 개념을 애플리케이션 프로그램 안에서 구현
   - 예 : HTTP는 웹 브라우저가 그래픽 이미지를 그림으로 출력되게 함

기본 네트워크

- **클라이언트와 서버**



  - 클라이언트-서버 구조
    - 클라이언트와 서버가 각자의 역할에 맞게 구성되어 있다는 의미
    - 월드 와이드 웹 : 웹 서버는 서버 역할, 웹 브라우저는 클라이언트 프로그램

인터넷 주소

인터넷 주소

- **IP 주소란?**
  - 네트워크를 구분하여 주는 ID로써 인터넷 상에 존재하는 호스트들을 구분하기 위한 32 비트 주소 체계를 의미함
  - 일반적으로 점이 찍힌 십진수 표현방식을 사용해서 IP 주소를 표현하는데, 점에 의해 구분되는 각각의 십진수 값은 1바이트로 표현함

    예) A라는 회사의 최 대리에게 데이터를 전송한다고 가정 이 회사는 하나의 로컬네트워크로 이루어져 있음. 이러한 회사의 권 대리가 사용하는 컴퓨터에 데이터를 전송하기 위해서, 일단은 A 회사의 네트워크로 데이터를 전송해 주는 것이 우선임

    즉, 처음부터 IP 주소 4바이트 모두를 참조해서 최 대리의 컴퓨터로 바로 찾아 가는 것이 아니라, 4바이트 IP 주소 중에서 네트워크 주소만을 참조해서 일단 A회사의 네트워크를 먼저 찾아감. A회사의 네트워크를 찾았다면 이제는 호스트 주소를 참조해서 최 대리의 컴퓨터로 찾아갈 차례가 되는 것임. 다시 말해 같은 네트워크 상에서는 모든 컴퓨터가 동일한 네트워크 주소를 가지고 각각의 컴퓨터가 서로 다른 호스트 주소를 가짐

인터넷 주소

- **IP 주소란?**
  - 네트워크를 구분하여 주는 ID로써 인터넷 상에 존재하는 호스트들을 구분하기 위한 32 비트 주소 체계를 의미함
  - 일반적으로 점이 찍힌 십진수 표현방식을 사용해서 IP 주소를 표현하는데, 점에 의해 구분되는 각각의 십진수 값은 1바이트로 표현함

0~255.0~255.0~255.0~255

InetAddress 클래스

- **InetAddress 클래스**
  - IP 주소를 표현하고 제어하는 기능을 제공하는 클래스
    - 호스트 네임 자체를 가지고 주소를 표현해 그에 관련된 기능을 제공
    - 인스턴스 생성법
      - **생성자 없음, static 메서드를 사용하여 인스턴스를 생성**

public static InetAddress getByName(String Host) throws UnknownHostException

→ 호스트의 이름 또는 주소를 InetAddress 객체로 반환

public static InetAddress getLocalHost() throws UnknownHostException

→ 로컬 호스트 네임을 InetAddress 객체로 반환

→ 만약 방화벽으로 가려진 경우 127.0.0.1을 반환

public static InetAddress[] getAllByName(String Host)) throws UnknownException

→ 호스트에 대한 모든 IP 주소를 InetAddress 객체 배열로 반환

InetAddress 클래스

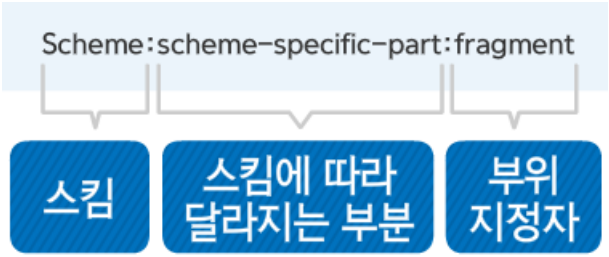
- InetAddress 클래스

메서드	설명
• boolean equals(InetAddress other)	– 현 객체가 other 객체와 같은 주소를 가지면 true, 아니면 false를 반환
• byte[] getAddress()	– 주소를 나타내는 4개의 요소(IP 주소)를 가진 바이트 배열을 반환
• String getHostAddress()	– 주소 정보를 나타내는 문자열을 반환 – IP 주소를 점으로 구분하는 10진수 형태로 반환 ex) 66.94.230.48 로 수정
• String getHostName()	– 컴퓨터 이름을 나타내는 문자열(도메인명)을 반환 ex) http://www.e-koreatech.ac.kr
• static InetAddress getLocalHost() throws UnknownHostException	– 현재 컴퓨터를 나타내는 InetAddress 객체를 반환
• static InetAddress getByName (String HostName) throws UnknownHostException	– 호스트 네임으로 지정된 컴퓨터를 나타내는 InetAddress 객체를 반환
• static InetAddress[] getAllByName (String HostName) throws UnknownHostException	– 호스트 네임으로 지정된 모든 컴퓨터 – 하나의 도메인 이름으로 여러 대의 컴퓨터를 사용하는 경우를 나타내는 InetAddress 객체들의 배열을 반환 – 호스트의 모든 IP 주소에 대한 정보를 InetAddress 배열로 반환

URL과 URI

URL과 URI

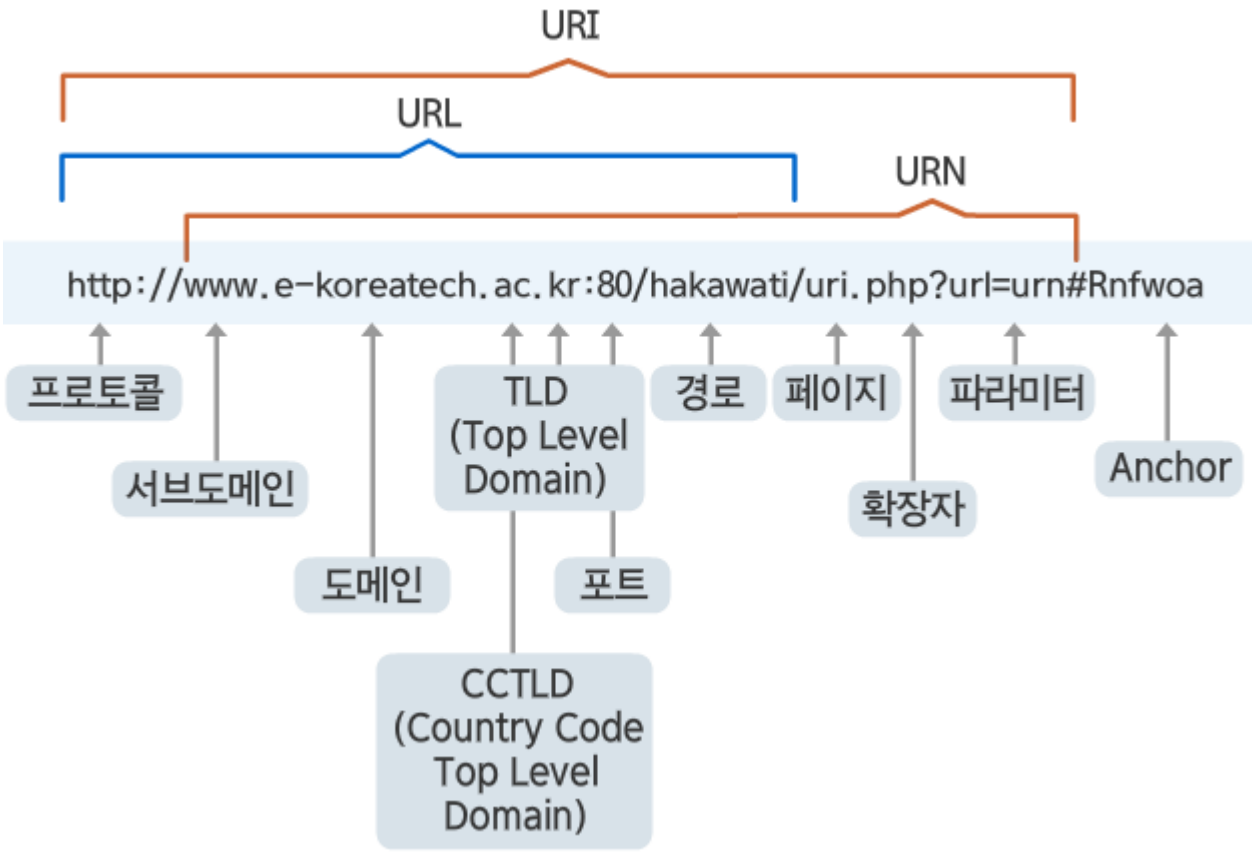
- **URI(Uniform Resource Identifier)**
  - 인터넷에 있는 어떤 자원에 접근하기 위한 유일한 주소 또는 키
  - URI만 있으면 웹에 있는 자원에 간단하게 접속 가능
    → 인터넷에서 요구되는 기본조건으로 인터넷 프로토콜에 항상 붙어 다님
- **URI 구성**



- **현재 사용되는 스킴**
  - data : 링크에 직접 포함된 Base64로 인코딩된 데이터
  - file : 로컬 디스크에 있는 파일
  - ftp : FTP 서버
  - http : HTTP(Hypertext Transfer Protocol)를 사용하는 www 서버
  - mailto : 메일 주소
  - telnet : 텔넷 기반 서비스 연결

URL과 URI

- URI(Uniform Resource Identifier)



URL과 URI

- **자바의 URI 클래스**
  - URL을 일반화(Generalization) 시킨 것
  - java.net 패키지에 포함
  - java.net.URL 클래스와 차이점
    - URI 클래스는 순수하게 리소스를 식별하고 URI를 분석하는 기능만 제공
    - URI 클래스는 URL 클래스보다 관련된 스펙 및 표준 사항을 더 잘 준수
    - URI 객체는 상대적인 URI를 표현할 수 있음
      → URL 객체 : 네트워크 전송을 위한 애플리케이션 계층 프로토콜을 표현
      → URI 객체 : 순수하게 문자열 분석과 조작을 위한 객체

URL과 URI

- **자바의 URI 클래스**
  - URI 생성하기

public URI(String uri) throws URISyntaxException

- 적절한 문자열로부터 새 URI 객체 생성

public URI(String scheme, String host, String path, String fragment) throws URISyntaxException

- http, ftp, URL 같은 계층적인 URI에 사용됨
- 이 생성자에서는 호스트와 경로가 URI의 스킴에 따라 다른 인자 부분을 구성함

public URI(String scheme, String schemeSpecificPart, String fragment) throws URISyntaxException

- 주로 비계층 URI에 사용됨
- 스킴 : http, urn, tel과 같은 URI의 프로토콜

public URI(String scheme, String authority, String path, String query, String fragment) throws URISyntaxException

- -기본적으로 세 번째 생성자와 같으며 쿼리 문자열이 추가됨

URL과 URI

- **자바의 URI 클래스**
  - URI 생성하기

    public URI(String scheme, String userInfo, String host, int port, String path, String query, fragment) throws URISyntaxException

    - 이전의 계층적인 URI를 생성하는 두 생성자가 호출하는 기본 URI 생성자
    - 기관을 사용자 정보, 호스트, 포트 부분으로 분리하며, 각 부분에는 각각의 구문 규칙이 있음
    - 생성된 URI는 여전히 일반적인 URI 구문 규칙을 따라야 함
  - 상대 URI 변환하기

    public URI resolve(URI uri)

    public URI resolve(String uri)

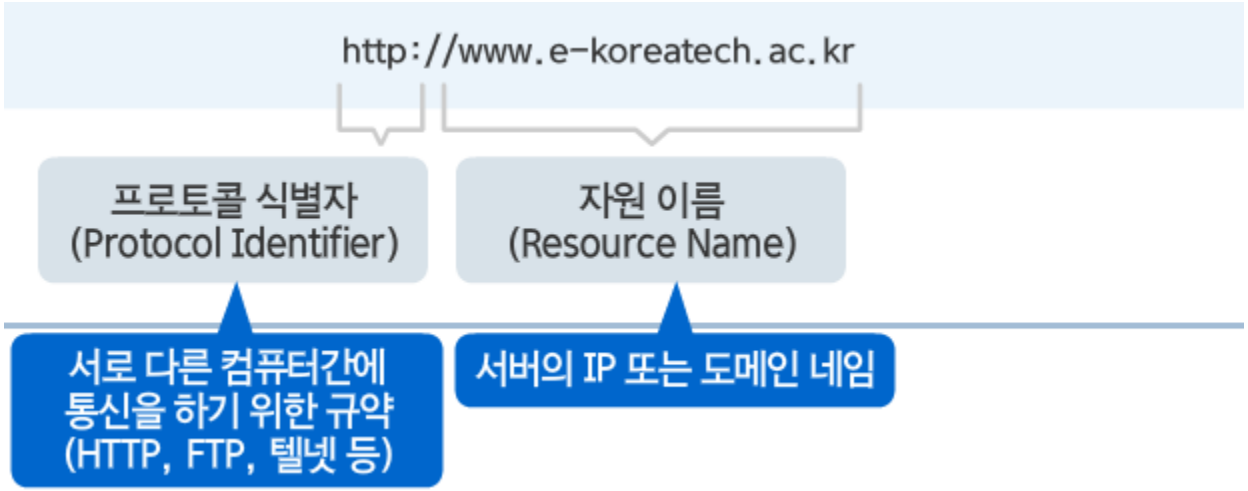
    public URI relativize(URI uri)

```
URI absolute = new URI(http://www.example.com);
URI relative = new URI("image.logo.png");
URI resolved = absolute.resolve(relative);

실행 결과  : http://www.example.com/images/logo.png
```

URL과 URI

- **URL(Uniform Resource Locator)**
  - 인터넷 상의 리소스에 대한 주소
  - 프로토콜의 이름, 도메인 이름, 경로명 등
  - URL 구조



URL과 URI

- **자바의 URL 클래스**
  - java.net 패키지에 포함
  - java.net.URL : 여러 종류의 생성자 제공

생성자	설명
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 객체 context에 대한 상대 경로가 지정하는 자원에 대한 URL 객체 생성

URL과 URI

- **자바의 URL 클래스**
  - 주요메소드

메서드	설명
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
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 반환

URL(Uniform Resource Location)

인터넷에 존재하는 서버들의 자원에 접근할 수 있는 주소.

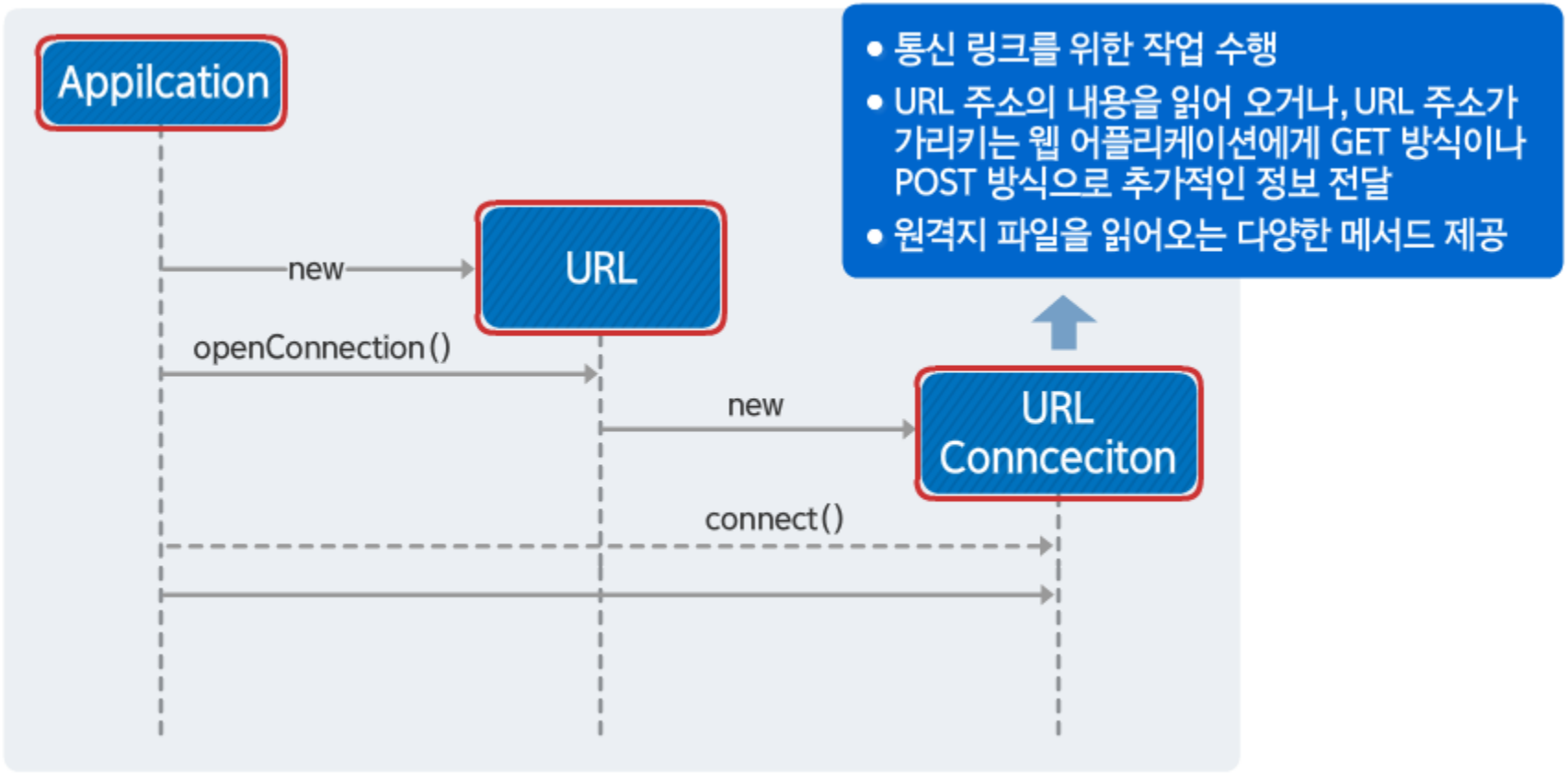
http://www.naver.com:80/sample/hello.html?referer=test#index1

프로토콜 : 자원에 접근하기 위해 서버와 통신하는데 사용되는 통신규약(http)
호스트명 : 자원을 제공하는 서버의 이름(www.javachobo.com)
포트번호 : 통신에 사용되는 서버의 포트번호(80)
경로명 : 접근하려는 자원이 저장된 서버상의 위치(/sample/)
파일명 : 접근하려는 자원의 이름(hello.html)
쿼리(query) : URL에서 '?'이후의 부분(referer=javachobo)
참조(anchor) : URL에서 '#'이후의 부분(index1)

URLConnection 클래스

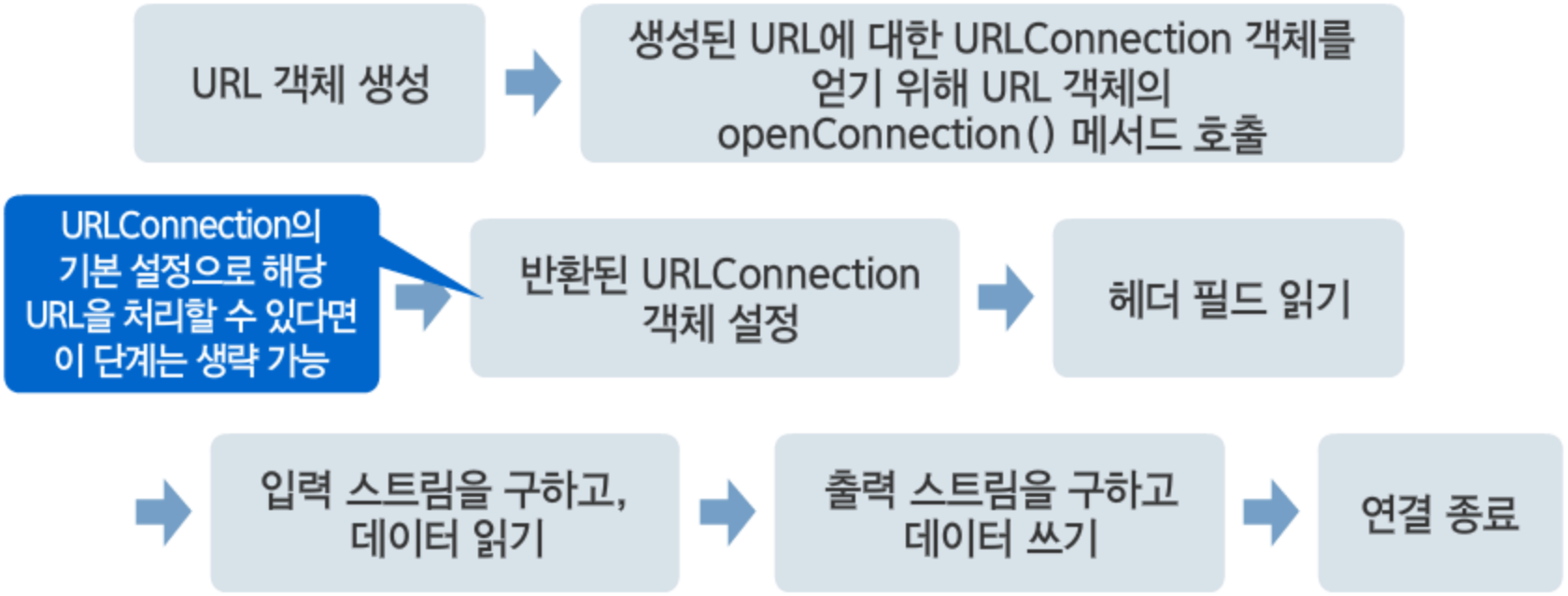
URLConnection 클래스

- URLConnection 클래스



URLConnection 클래스

- URLConnection 클래스 사용 절차



URL(Uniform Resource Location)

```
package httptest;

public class UrlConnectionTest {

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

        // URLConnection : http 기반의 통신 처리
        // URL 클래의 openConnection() 메서드로 URLConnection 객체 반환

        URL url = null;
        String urlStr = "http://www.ctware.net"; // index.html

        url = new URL(urlStr);

        URLConnection conn = url.openConnection();

        BufferedReader in = null;

        //in = new BufferedReader(new InputStreamReader(url.openStream()));
        in = new BufferedReader(new InputStreamReader(conn.getInputStream()));

        String str = null;

        while(true) {
                str = in.readLine();
                if(str == null) {
                        break;
                }
                System.out.println(str);

        }
```

URL(Uniform Resource Location)

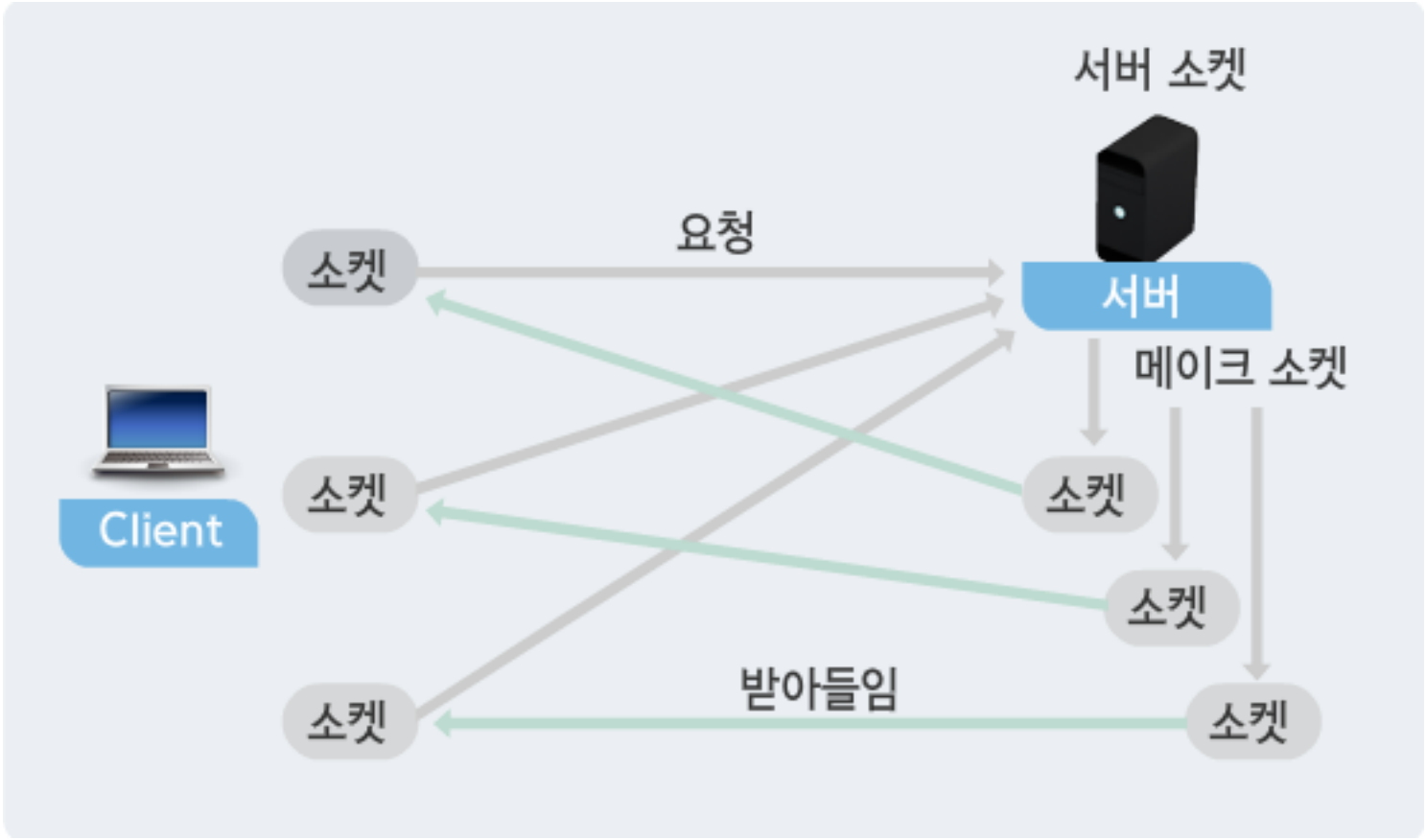
```
            /*
              System.out.println("conn.toString():" + conn);
              System.out.println("getAllowUserInteraction():" +
              conn.getAllowUserInteraction()); System.out.println("getConnectTimeout():" +
              conn.getConnectTimeout()); System.out.println("getContent():" +
              conn.getContent()); System.out.println("getContentEncoding():" +
              conn.getContentEncoding()); System.out.println("getContentLength():" +
              conn.getContentLength()); System.out.println("getContentType():" +
              conn.getContentType()); System.out.println("getDate():" + conn.getDate());
              System.out.println("getDefaultAllowUserInteraction():" +
              conn.getDefaultAllowUserInteraction());
              System.out.println("getDefaultUseCaches():" + conn.getDefaultUseCaches());
              System.out.println("getDoInput():" + conn.getDoInput());
              System.out.println("getDoOutput():" + conn.getDoOutput());
              System.out.println("getExpiration():" + conn.getExpiration());
              System.out.println("getHeaderFields():" + conn.getHeaderFields());
              System.out.println("getIfModifiedSince():" + conn.getIfModifiedSince());
              System.out.println("getLastModified():" + conn.getLastModified());
              System.out.println("getReadTimeout():" + conn.getReadTimeout());
              System.out.println("getURL():" + conn.getURL());
              System.out.println("getUseCaches():" + conn.getUseCaches());
             */
    }

}
```

소켓(Socket)

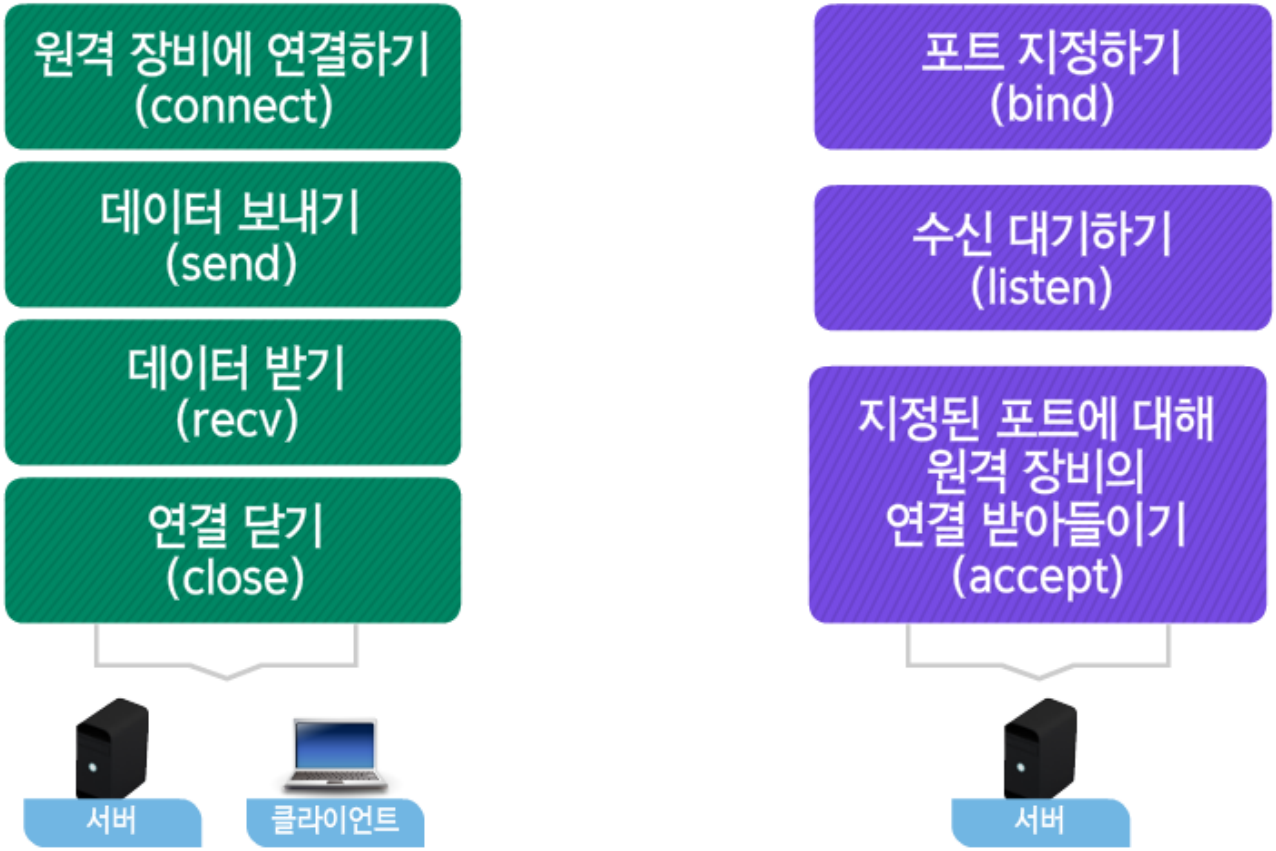
클라이언트 소켓

- **소켓(Socket)이란?**
  - 네트워크상에서 서버와 클라이언트 두 개의 프로그램이 특정 포트를 통해 양방향 통신이 가능하도록 만들어 주는 소프트웨어 장치



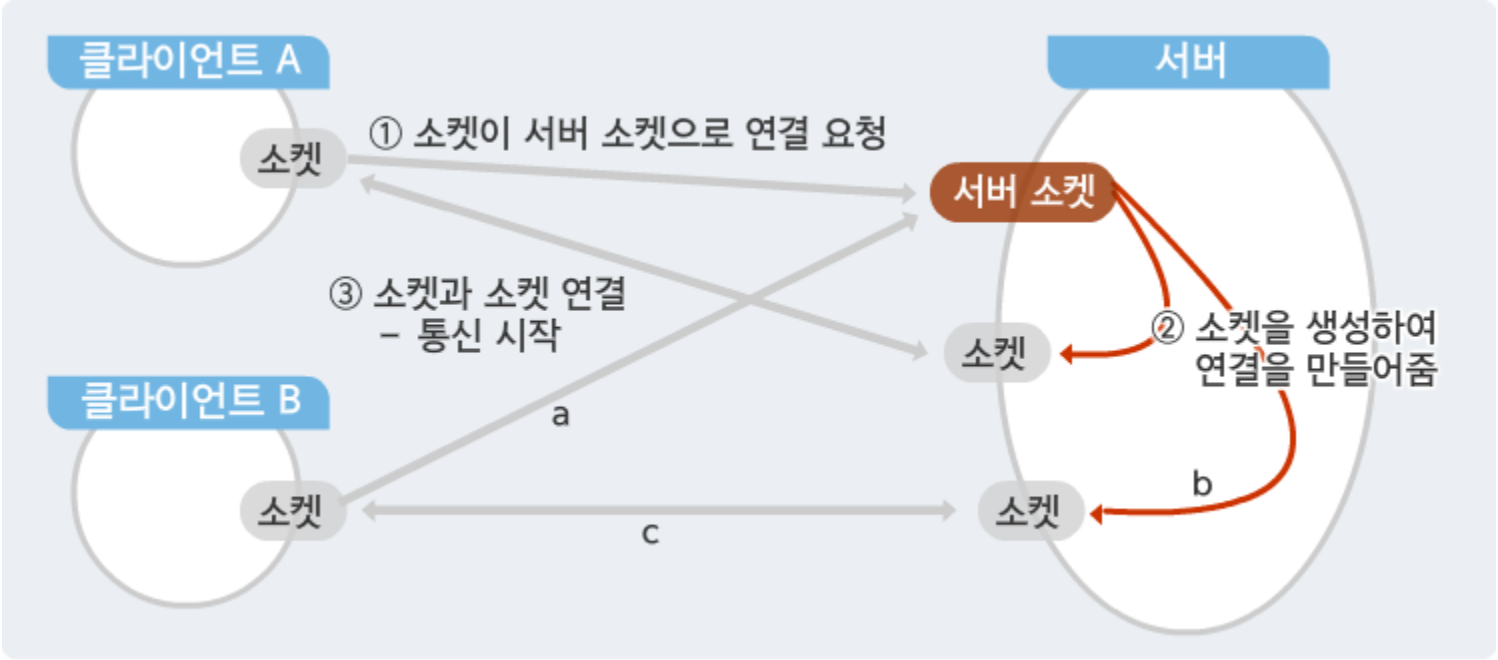
클라이언트 소켓

- **소켓의 기능**
  - 자바의 소켓 클래스는 각각에 해당하는 메서드 제공



서버 소켓

- **서버 소켓이란?**
  - 서버 소켓 클래스 : 자바에서 서버를 작성하기 위해 필요한 모든 것을 제공함



TCP와 UDP

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

TCP소켓 프로그래밍

클라이언트와 서버 간의1:1 소켓통신.

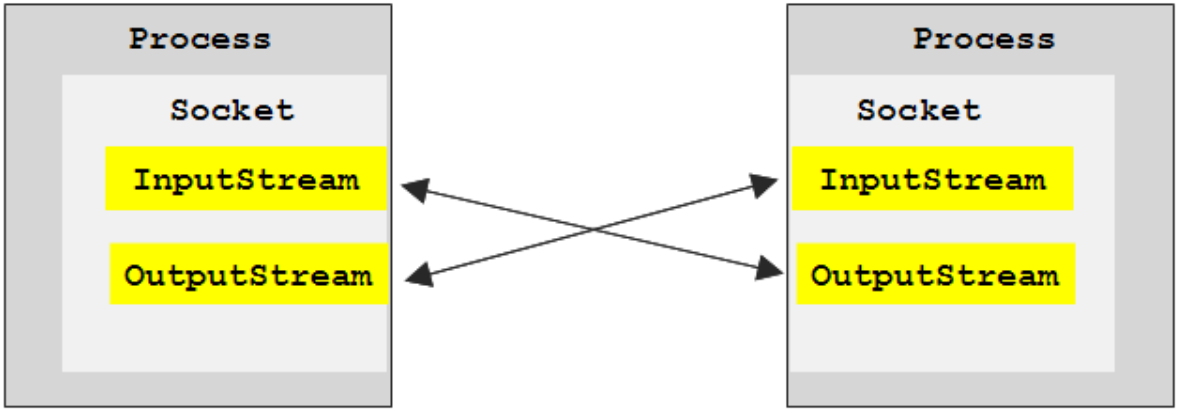
- 서버가 먼저 실행되어 클라이언트의 연결 요청을 기다리고 있어야 한다.

1. 서버는 서버소켓을 사용해서 서버의 특정 포트에서 클라이언트의 연결요청을 처리 할 준비를 한다.
2. 클라이언트는 접속 할 서버의 IP주소와 포트 정보로 소켓을 생성해서 서버에 연결을 요청한다.
3. 서버 소켓은 클라이언트의 연결 요청을 받으면 서버에 새로운 소켓을 생성해서 클라이언트의 소켓과 연결 되도록 한다.
4. 이제 클라이언트의 소켓과 새로 생성된 서버의 소켓은 서버 소켓과 관계없이 1:1 통신을 한다.

TCP소켓 프로그래밍

Socket - 프로세스간의 통신을 담당하며, InputStream과 OutputStream을 가지고 있다.
이 두 스트림을 통해 프로세스간의 통신(입출력)이 이루어진다.

ServerSocket - 포트와 연결(bind)되어 외부의 연결요청을 기다리다 연결요청이 들어오면,
Socket을 생성해서 소켓과 소켓간의 통신이 이루어지도록 한다.
한 포트에 하나의 ServerSocket만 연결할 수 있다.
(프로토콜이 다르면 같은 포트를 공유할 수 있다.)



TCP소켓 프로그래밍

```
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.");
```

TCP소켓 프로그래밍

```
// 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.");

                                        // 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
```

TCP소켓 프로그래밍

```
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 | catch(IOException | ce) {
                        ce.printStackTrace();
                } catch(Exception e) {
                        e.printStackTrace();
                }
        } // main
} // class
```

소켓을 이용한 멀티체팅 프로그램

소켓을 이용한 멀티체팅 프로그램 : chatting Server

```
package httptest;

import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.IOException;
import java.net.ServerSocket;
import java.net.Socket;
import java.util.Collections;
import java.util.HashMap;
import java.util.Iterator;

public class TcpIpMultichatServer {

        // 1. 접속한 사용자들을 Map 저장 -> 모든 사용자에 메시지 전송
        // 2. name, 사용자별 소켓을 이용한 쓰레스(기능:받고, 전체 사용자에게 전송 )
        // 쓰레드 생성 : 1. 전체 메시지 보내기 2. Map 저장
        // 3. 멀티 쳇 : 누가 언제 들어와도 체팅이 가능한 체팅 기능 - while(true) , socket.accept()

        HashMap<String,Object> clients;
        public TcpIpMultichatServer() {
                clients = new HashMap<String,Object>();
                Collections.synchronizedMap(clients);
        }
        // 프로그램의 시작
        public static void main(String[] args) throws IOException {
                new TcpIpMultichatServer().start();
        }
```

소켓을 이용한 멀티체팅 프로그램 : chatting Server

```
// 서버 시작
public void start() throws IOException {
        ServerSocket serverSocket = null;
        Socket socket = null;

        serverSocket = new ServerSocket(7777);
        System.out.println("서버가 시작되었습니다.");

        while (true) {
            socket = serverSocket.accept();
            System.out.println("["+socket.getInetAddress() + ":"+socket.getPort() +"] 사용자 접속");
            ServerReceiver receiver = new ServerReceiver(socket);
            receiver.start();
        }
}

void sendToAll(String msg) {
        Iterator itr = clients.keySet().iterator();
        while(itr.hasNext()) {
                DataOutputStream dout = (DataOutputStream) clients.get(itr.next());
                try {
                        dout.writeUTF(msg);
                } catch (IOException e) {
                        e.printStackTrace();
                }
        }
}
```

소켓을 이용한 멀티체팅 프로그램 : chatting Server

```
class ServerReceiver extends Thread {    // 내부 클래스 : 데이터 받기 처리하는 쓰레드
        Socket socket;  DataInputStream in; DataOutputStream out;
        public ServerReceiver(Socket s) {
                socket = s;
                try {
                        in = new DataInputStream(s.getInputStream());
                        out = new DataOutputStream(s.getOutputStream());
                } catch (IOException e) {
                        e.printStackTrace();
                }
        }
        public void run() {
                String name = "";
                try {           // 전체 사용자에게 접속 안내 메시지를 보내기
                        name = in.readUTF();
                        clients.put(name, out);
                        sendToAll("###" + name + "님이 접속하셨습니다.");
                        while(in!=null) { sendToAll(in.readUTF()); }
                } catch (IOException e) {
                        e.printStackTrace();
                } finally {
                        System.out.println(name + "님이 나가셨습니다.");
                        clients.remove(name);
                }
        }
}
}
```

소켓을 이용한 멀티체팅 프로그램 : chatting client

```
package httptest;

import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.IOException;
import java.net.ConnectException;
import java.net.Socket;
import java.util.Scanner;

public class TcpIpMultichatClient3 {

	public static void main(String[] args) {
		try {
			String serverIp = "127.0.0.1";
			// 소켓을 생성하여 연결을 요청한다.
			Socket socket = new Socket(serverIp, 7777);
			System.out.println("서버에 연결되었습니다.");
			Thread sender = new Thread(new ClientSender(socket, "전현무"));
			Thread receiver = new Thread(new ClientReceiver(socket));
			sender.start();
			receiver.start();
		} catch (ConnectException ce) {
			ce.printStackTrace();
		} catch (Exception e) {
		}

	}
```

소켓을 이용한 멀티체팅 프로그램 : chatting client

```
static class ClientSender extends Thread {
        Socket socket;
        DataOutputStream out;
        String name;

        ClientSender(Socket socket, String name) {
                this.socket = socket;
                try {
                        out = new DataOutputStream(socket.getOutputStream());
                        this.name = name;
                } catch (Exception e) {
                }
        }

        public void run() {
                Scanner scanner = new Scanner(System.in);
                try {
                        if (out != null) {
                                out.writeUTF(name);
                        }
                        while (out != null) {
                                out.writeUTF("[" + name + "]" + scanner.nextLine());
                        }
                } catch (IOException e) {
                }
        } // run()
}
```

소켓을 이용한 멀티체팅 프로그램 : chatting client

```
        static class ClientReceiver extends Thread {
                Socket socket;
                DataInputStream in;

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

                public void run() {
                        while (in != null) {
                                try {
                                        System.out.println(in.readUTF());
                                } catch (IOException e) {
                                }
                        }
                } // run
        }

}
```