네트워크 프로그래밍을 수행하는데 가장 중요한 것은 IP 주소에 대한 개념이다.

IP 주소에는 크게 2가지가 있다.

1) 사설 IP

이 녀석의 특징은 192.168로 시작하는 주소를 가진다는 것입니다.
이것은 특정한 지역에 있는 로컬망을 위한 주소입니다.
(로컬망이란 ? 외부로 나갈 순 없지만 내부에서 통신에 활용이 가능)

2) 공인 IP

우리가 네이버, 다음, 구글에 접속하기 위해서는 반드시 공인 IP가 필요합니다.
공인 IP가 없으면 인터넷에 접속이 불가능합니다.

???? 뭐지 ? 지금 우리는
192.168이라는 주소값을 가지고 인터넷을 하고 있지 않나요 ?

공유기) 공유기는 NAT 프로토콜을 사용해서
사설망의 IP를 공인 IP로 변환하여 인터넷을 사용할 수 있게 합니다.

학원) 스위치 혹은 라우터가 존재한다.
굉장한 고가의 스위치로는 L4 스위치가 존재한다.

기본적인 스위치의 동작 과정)

ifconfig를 입력하면 아래와 같은 정보가 나타날 것이다.

```
enp2s0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
```

Internet Protocol
: 네트워크상에서
다른컴퓨터와 구별 될수있는 고유 식별번호.

공인 IP는 전세계적으로 유일
→ 사설IP는 하나의 네트워크 안에서 유일

→ 1개의 공인 IP를 가지고 내부 네트워크들을 관리.

네트워크 사이에 데이터를 서로 전송.
IP주소를 통해 데이터를 전송

→ 라우팅 ? IP주소를 이용하여
두 네트워크간 데이터 전송 수행.

→ 같은 네트워크 내부에서 데이터를 서로 전송
Mac 주소를 통해 같은 네트워크 LAN 포트간 데이터 전송.

→ 네트워크 간 전송을 위해서는
L4 가필요.

기본적인 스위치의 동작 과정)

ifconfig를 입력하면 아래와 같은 정보가 나타날 것이다.

```
enp2s0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.0.9  netmask 255.255.255.0  broadcast 192.168.0.255
        inet6 fe80::2c3:b2e:3f8b:34f2  prefixlen 64  scopeid 0x20<link>
        ether xx:7b:ef:xx:dc:7e  txqueuelen 1000  (Ethernet)
        RX packets 12514664  bytes 10413031683 (10.4 GB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 6721535  bytes 2495119629 (2.4 GB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

ether라고 표시된 부분이 바로 그 유명한 MAC 주소입니다.
MAC 주소를 보고 내가 어디로 가야하는지를 파악합니다.
IP 주소는 사설망에 의해 같은 주소가 많이 생길 수 있지만
MAC 주소는 유일무이하다.
그러므로 이것은 털리면 해킹당할 수 있다.

로컬망의 동작 방식

(LAN)

192.168.0.9 <--- switch ---> 192.168.0.202
xx:7b:ef:xx:dc:7e xx:7b:ef:xx:dc:7e

특수한 스위치(L4)의 경우

공인IP를 만들어주는 장치?

네이버, 구글 <-> switch(L4) <-> router <-> switch <-> 우리의 사설망

스위치는 기본적으로 내부망(로컬망) 전용이므로
"야~ 난 이거 못하니까 잘 하는 라우터한테 토스할게"
라우터가 내용을 받고 자신의 라우팅 테이블을 확인합니다.

리눅스 명령어로는 route를 입력하면 내용을 확인할 수 있습니다.
라우터는 기본적으로 IP 주소를 보고 내가 어디로 가야하는지를 알 수 있습니다.

L4 스위치는 포트 번호까지 포함해서 데이터를 송수신 할 수 있습니다.
포트 번호란 ? 서비스의 종류를 나타내는 번호다.

netstat -ntlp 명령어를 통해
현재 컴퓨터에서 사용되는 서비스(포트 번호) 리스트를 얻을 수 있다.

실행하면 아래와 같은 결과를 얻게 됩니다.

```
netstat -ntlp
(Not all processes could be identified, non-owned process info
 will not be shown, you would have to be root to see it all.)
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State       PID/Program name
tcp        0      0 127.0.0.53:53           0.0.0.0:*               LISTEN      -
tcp        0      0 127.0.0.1:631           0.0.0.0:*               LISTEN      -
tcp6       0      0 127.0.0.1:63342         :::*                    LISTEN      10027/java
tcp6       0      0 127.0.0.1:42675         :::*                    LISTEN      10027/java
tcp6       4      0 :::33333                :::*                    LISTEN      11949/java
tcp6       0      0 127.0.0.1:35189         :::*                    LISTEN      10027/java
tcp6       0      0 ::1:631                 :::*                    LISTEN      -
tcp6       0      0 127.0.0.1:6942          :::*                    LISTEN      10027/java
tcp6       0      0 127.0.0.1:35905         :::*                    LISTEN      10027/java
```

케이스 비교를 위해 새로운 서비스를 띄워보도록 하겠습니다.
대표적인 보안 통신 서비스인 ssh를 사용해보도록 합니다.

```
sudo apt-get install openssh-server
```

설치 이후에 22번이 새롭게 추가된 것을 볼 수 있을 것이다.
22가 바로 ssh(보안 통신) 서비스 포트 번호에 해당한다.

```
netstat -ntlp
(Not all processes could be identified, non-owned process info
 will not be shown, you would have to be root to see it all.)
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State       PID/Program name
tcp        0      0 127.0.0.53:53           0.0.0.0:*               LISTEN      -
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN      -
tcp        0      0 127.0.0.1:631           0.0.0.0:*               LISTEN      -
tcp6       0      0 127.0.0.1:63342         :::*                    LISTEN      10027/java
tcp6       0      0 127.0.0.1:42675         :::*                    LISTEN      10027/java
tcp6       4      0 :::33333                :::*                    LISTEN      11949/java
tcp6       0      0 127.0.0.1:35189         :::*                    LISTEN      10027/java
tcp6       0      0 :::22                   :::*                    LISTEN      -
tcp6       0      0 ::1:631                 :::*                    LISTEN      -
tcp6       0      0 127.0.0.1:6942          :::*                    LISTEN      10027/java
tcp6       0      0 127.0.0.1:35905         :::*                    LISTEN      10027/java
```

포트 번호는 아무렇게나 할당해도 되나요 ?

1) 그때그때 다릅니다.
   고정된 포트를 번호를 가진 서비스들이 존재함
   (ssh: 22, http: 80, ftp: 20, 21, telnet: 23 https: 443 등등)
   이런 고정된 포트는 할당하면 다른 서비스들과 충돌을 유발하게 될 것이다.

결국 현재 포트 번호를 보면 내가 어떤 서비스에 접근해야하는지를 알 수 있게 된다.

서버와 클라이언트 개념은 무엇인가 ?

Server(서버): 서비스를 제공하는 사람: 서비스를 제공하는 측
programmer(프로그래머): 프로그램을 만드는 사람

요청을 하면 ? 클라이언트 → 고객
요청을 처리하면 ? 서버

* 예외 처리

: Exception (예외) 가 발생한 것을 대비하여 미리 예측해 이를 소스상에서 제어하고 처리하도록 만드는것.

① 일반 예외. - 컴파일 시 발생하는 Exception.
② 실행 예외 - 프로그램 실행시 발생하는 Exception.

처리코드
: try - catch - finally

① try
: 실제 코드가 들어가는 곳. Exception이 발생할 수 있는 코드.

② catch
: try에서 Exception 이 발생하면, 코드순서가 catch로 옮겨짐
즉, 예외에 대한 처리 코드.

③ finally
: try에서 Exception 발생 유무와 상관 X, 무조건 수행되는 코드.
(생략가능함).

```java
package Fifteenth;

import java.io.*;
import java.net.ServerSocket;
import java.net.Socket;
import java.util.Date;

public class SocketServerTest {
    public static void main(String[] args) {
        // 문자열을 숫자로 바꾸는 기
        int port = Integer.parseInt("33333");

        try {
            // 소켓이란 ? 네트워킹을 할 수 있는 클래스 객체
            // 서버 소켓 생성시 서비스 번호를 부여해야 하는데
            // 이 서비스 번호로 port(33333)을 설정했다.법
            // 네트워크 포트가 16비트를 사용하므로 제한은 0 ~ 65535
            // (단, 고정된 포트를 사용하면 안됨)
            ServerSocket servSock = new ServerSocket(port);

            System.out.println("Server: Listening - " + port);
```

① 포트 번호를 선언. → 포트번호 프린트.
② Exception 확인.
try
1) Port 번호에서 예외발생 경우를 확인
2) 만약 Exception이 없으면, Client가 accept() 승인여부 확인
3) Client의 IP 확인
4) 서버에 프린트할 내용은 어떠 (OutputStream)
서버쪽
소켓 서버를 만드는 클래스
5) 출력은 Client에게 보내주는 객체생성 (PrintWriter)
6) Client에게 보내고싶은 내용기록
클라이언트쪽
7) Client가 입력하실내용 객체 생성. (InputStream)
8) 서버에서 출력될것 객체 생성. (readLine)

```
12
13          try {
14              // 소켓이란 ? 네트워킹을 할 수 있는 클래스 객체
15              // 서버 소켓 생성시 서비스 번호를 부여해야 하는데
16              // 이 서비스 번호로 port(33333)을 설정했다.법
17              // 네트워크 포트가 16비트를 사용하므로 제한은 0 ~ 65535
18              // (단, 고정된 포트를 사용하면 안됨)
19              ServerSocket servSock = new ServerSocket(port);
20
21              System.out.println("Server: Listening - " + port);
22
23              while(true) {
24                  // accept()의 경우 클라이언트가 접속을 요청했는지 체크해서
25                  // 만약 요청 있었다면 요청을 승인한다.
26                  // (accept()는 블로킹 연산이다)
27
28                  // 결국 sock은 클라이언트 소켓을 의미하게 된다.
29                  // 그래서 좀 더 가독성이 좋은 코드는 Socket clntSock으로 작성하면 좋을
30                  // (전화왔을때 통화하기 슬라이드가 accept()라 보면됨)
31                  Socket sock = servSock.accept();
32
33                  // 접속한 클라이언트의 IP를 확인하는 코드
34                  System.out.println(
35                          "[" + sock.getInetAddress() +
36                          "] client connected"
37                  );
38
```

```
33                  // 접속한 클라이언트의 IP를 확인하는 코드
34                  System.out.println(
35                          "[" + sock.getInetAddress() +
36                          "] client connected"
37                  );
38
39                  // 출력을 위한 객체를 만듭니다.
40                  // 클라이언트에게 출력할 객체를 만든다.
41                  OutputStream out = sock.getOutputStream();
42                  // println의 결과를 out으로 전송한다라는 뜻
43                  // 내가 서버에 요청하는 것: 출력이 아닌 입력이다.
44                  // 서버가 처리해서 돌려주는 것: 이것이 출력이다.
45                  PrintWriter writer = new PrintWriter(out, true);
46                  // 즉 여기서 println의 출력은 클라이언트에게 간다.
47                  // Date()는 시간을 가져온다.
48                  // toString()은 문자열로 만듬
49
50                  // 접속 상대에게 보내고 싶은 데이터를 이곳에 기록한다.
51                  writer.println(new Date().toString());
52
53                  // 입력: 클라이언트
54                  // 클라이언트가 서버에게 보낸 것
55                  InputStream in = sock.getInputStream();
```

accept() 이후부터 OutPut Stream 까지가 서버 화면에 나올 것들 입력.

클라이언트에게 출력보낼 것들은 적으면 된다.

```
57                  // 소켓을 통해 데이터를 읽을땐 무조건 아래 형식으로 진행합니다.
58                  // 바뀌지 않으니 항상 고정해서 사용하면 됩니다.
59                  // 이 버퍼에 클라이언트가 보낸 내용이 들어있다.
60                  BufferedReader reader =
61                          new BufferedReader(new InputStreamReader(in));
62
63                  // 그러므로 reader.readLine()을 통해서
64                  // 내용을 읽으면 클라이언트가 보낸 내용을 출력할 수 있게 된다.
65                  System.out.println("msg: " + reader.readLine());
66              }
67          } catch (IOException e) {
68              // Exception은 예외 처리로
69              // I/O 예외가 발생하면 무엇인가 잘못되었음을 감지하고
70              // 어디가 잘못되었는지 출력하도록 구성된다.
71              // ex) 통신중에 갑자기 네트워크 불안정으로 통신이 끊기면
72              //      catch가 I/O 예외를 감지하고 아래 코드가 동작하게 된다.
73              // 아래 코드는 에러 메시지를 출력하는 코드로 언제나 동일하게 작성하면 됨
74              System.out.println("Server Exception: " + e.getMessage());
75
76              // 콜 스택(매서드 호출)이 어떤식으로 이뤄졌는지 상태를 보여줌
77              // 디버깅을 위해서 많이 사용하는 편
78              e.printStackTrace();
79          }
80      }
81  }
```

→ 예외가 발생시 처리하는 메서드.

예외 발생한 내용을 print

```java
package Fifteenth;

import java.io.*;
import java.net.Socket;
import java.net.UnknownHostException;

public class SocketClientTest {
    public static void main(String[] args) {
        // 내가 접속할 서버의 IP 주소를 적습니다.
        String hostname = "192.168.0.9";
        // 서버에 여러 서비스가 있을 수 있는데
        // 그 중에서 내가 사용하고자 하는 서비스의 포트 번호를 적습니다.
        int port = 33333;

        for(int i = 0; i < 10; i++) {
            try {
                // Socket 객체를 할당해서
                // 서버의 IP, 포트 번호를 가지고 접속을 요청합니다.
                // 서버에 대한 소켓을 획득하게 됩니다.
                // 이 요청이 들어갈때 서버의 accept()가 동작하게 됩니다.
                // 예를 들자면 이 행위는 전화를 거는것과 같다.
                // (서버쪽 주석을 살펴보면 감이 더 잘 올 것이다)
                Socket sock = new Socket(hostname, port);

                // 서버의 출력을 획득
                // 즉 서버가 수신하게 만들도록 설정을 해주는 것
                OutputStream out = sock.getOutputStream();
```

접속할 IP주소.
// Port number

① 예외확인(Exception)

try「 1) IP주소, Port 입력 값을 받고
2) Client가 입력할 수 있는 객체생성.
(OutStream)
3) Client가 입력 받을수 있는 객체생성.
(InputStream, InputStreamReader
readLine)」

Catch「예외 발생 시 처리할 내용
(Server와 동일)」

for 쓰는 이유를 안고 살습니다.

접속할 → IP주소도필요

Server Socket과 달리 여기서는

accept하는 client의 과정.

```
29                  String str = "Hello Network Programming";
30                  // 위의 문자열을 바이트 단위로 쪼개서 서버로 전송한다.
31                  out.write(str.getBytes());
32
33                  // 서버의 입력을 생성(수신 준비)
34                  InputStream in = sock.getInputStream();
35                  BufferedReader reader =
36                          new BufferedReader(new InputStreamReader(in));
37
38                  // 서버가 보낸 내용을 time에 저장하고 출력한다.
39                  String time = reader.readLine();
40                  System.out.println(time);
41
42                  // UnknownHostException: 내가 접속하려는 IP를 찾지 못할 때
43              } catch (UnknownHostException e) {
44                  System.out.println("Server Not Found: " + e.getMessage());
45
46              } catch (IOException e) {
47                  System.out.println("I/O Error: " + e.getMessage());
48              }
49          }
50      }
51  }
```

out Stream 부터 out.write 까지가 서버에 보낼 내용들.

서버에서 입력 받기위한 객체.

다른곳에서 보낸내용 저장장소.

1. 가위 바위 보 게임.

1) Server

① 기존 SocketServertest 의 Client Connected 까지 동일 하게 작성 (37 line)
② 스캐너 선언 → Server 에서 가위 바위 보 입력 가능하게 만들기.
③ if 문 통해 결과도 프린트 할 수있게 작성 하기.
④ OutPut Stream 선언
⑤ PrintWriter 선언 → Server의 Scanner 출력.
⑥ Writer.println에 → 스캐너 입력값 넣기.

2) Client

① 기존 Socket sock = new Socket 까지 동일하게 작성 (28 line)
② 스캐너 선언 → Client 에서 가위바위보 입력 가능하게 만들기.
③ if문 통해 게임결과도 프린트할수 있게 작성.
④ OutPut Stream 선언
⑤ Out.write에 스캐너 입력값 넣기.