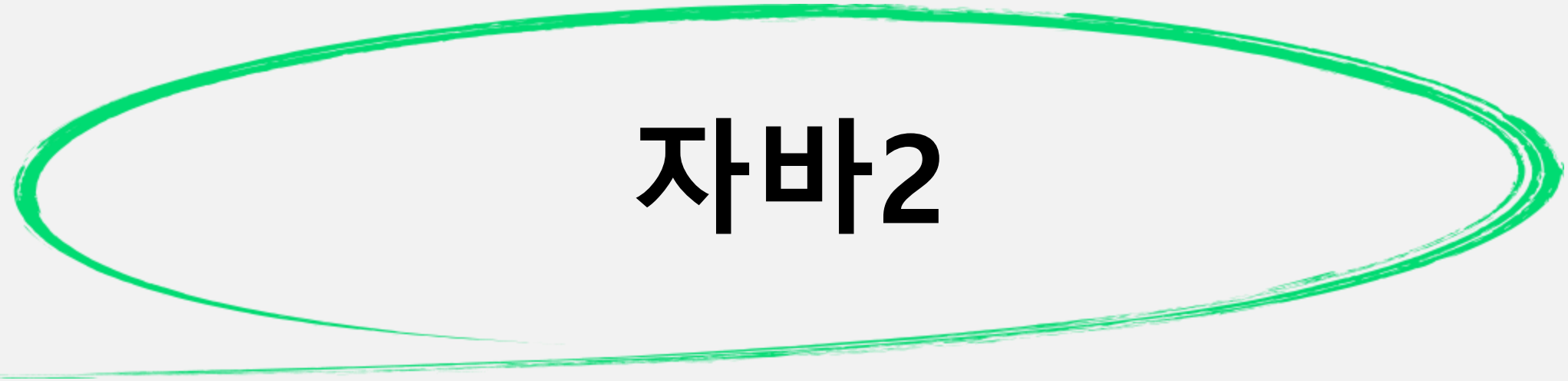


자바2

(네트워크 프로그래밍)

Chapter 11
네트워크 프로그래밍

Chapter11의 학습목표

- 네트워크의 개념에 대해 알아본다
- 자바에서 서버와 네트워크의 구현에 대해 학습한다

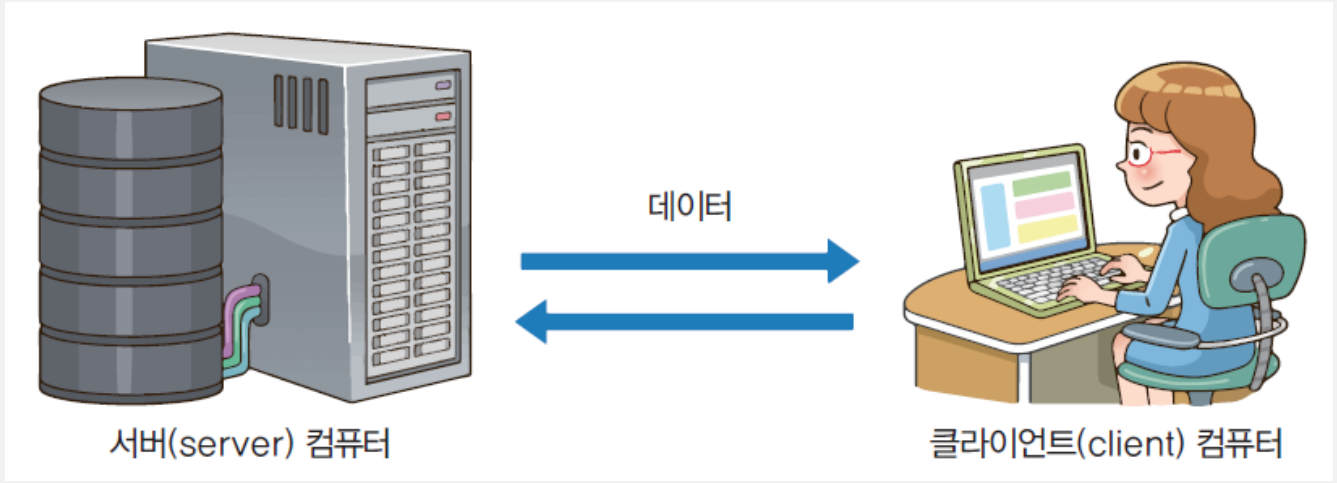
Network

네트워크 프로그래밍

- 네트워크(Network) : 분산되어 있는 두 개 이상의 컴퓨터를 통신망으로 연결한 것
- 네트워크 프로그래밍 : 다른 컴퓨터와 송수신 할 수 있는 프로그램을 만드는 것

서버와 클라이언트

- 서버 (Server) : 사용자들에게 서비스를 제공하는 컴퓨터. 일반적으로 매우 큰 용량과 성능을 가짐
  ex) 웹 서버
- 클라이언트 (Client) : 서버에게 서비스를 요청해서 사용하는 컴퓨터
  ex) 개인 컴퓨터



Network

IP 주소

- 컴퓨터와 네트워크에서 장치들이 서로를 인식하고 통신을 하기 위해 컴퓨터마다 부여되는 고유한 주소
- 인터넷에 연결된 모든 컴퓨터는 4byte(32bit)의 정수로 구성된 IP주소를 가지며 000.000.000.000 와 같은 형식으로 표현됨

IP 주소 = 네트워크 주소 + 호스트 주소

메소드	설명
byte[] getAddress()	IP주소를 byte 배열로 반환
static InetAddress[] getAllByName(String host)	도메인명에 지정된 모든 호스트의 IP 주소를 배열에 담아 반환
static InetAddress getByName(String hos)	도메인명을 통해 IP주소를 얻음
String getHostAddress()	호스트의 IP주소를 반환
String getHostName()	호스트의 이름을 반환
boolean isMulticastAddress()	IP 주소가 멀티캐스트 주소인지 확인
Boolean isLoopbackAddress()	IP 주소가 loopback 조시인지 알려줌

Network

InetAddress

- IP 주소를 표현한 클래스. IP주소를 객체화.
- 자바에서는 모든 IP주소를 InetAddress 클래스를 이용하여 사용하고 있음

종류	메소드	반환형	설명
생성자 메소드	getAllByName(String host)	static InetAddress[]	매개변수 host에 대응되는 InetAddress 배열을 반환
	getByAddress(byte[] addr)	static InetAddress	매개변수 addr에 대응되는 InetAddress 객체를 반환 ex) addr[0] = (byte) 209 addr[1] = (byte) 209 addr[2] = (byte) 209 addr[3] = (byte) 209 InetAddress iaddr = InetAddress.getByAddress(addr);
	getBAddress(String host, byte[] addr)		매개변수 host와 addr로 InetAddress객체를 생성
	getByName(String host)		매개변수 host에 대응되는 InetAdress객체를 반환
	getLocalHost()		로컬 호스트의 InetAdress객체를 반환
InetAddress 클래스 메소드	getAddress()	byte[]	InetAdress 객체의 실제 IP주소를 바이트 배열로 반환
	getHostAddress()	String	IP 주소를 문자열로 반환
	getHostName()		host 이름을 문자열로 반환
	toString()		IP 주소를 스트링 문자열로 오버라이딩 한 메소드

Network

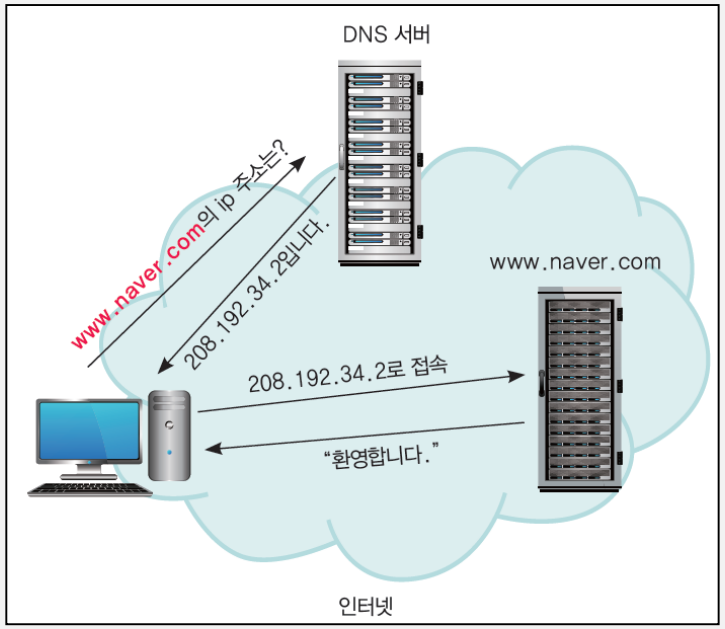
IP 주소 : 네이버의 IP주소를 출력하는 프로그램

```
import java.net.*;

public class A_Class {
    public static void main(String[] args) throws Exception{
        String hostname = "www.naver.com";
        try
        {
            InetAddress address = InetAddress.getByName(hostname);
            System.out.println("IP 주소: " + address.getHostAddress());
        }
        catch ( UnknownHostException e )
        {
            System.out.println(hostname + "의 IP 주소를 찾을 수 없습니다.   " );
        }

    }
}
```

Network

DNS와 URL

- DNS (Domain Name System) : 숫자 대신 기호를 사용하는 주소. IP주소를 영문 이름의 주소로 바꿔주는 시스템
- DNS 서버 : 기호 주소를 숫자 주소로 변환해주는 서버
- URL (Uniform Resourse Locator) : 인터넷 상의 자원을 나타내는 약속. 자원에 대한 주소를 지정하는 방법
- 프로토콜(Protocol) : 클라이언트와 서버 간의 통신 규약. 서버와 클라이언트 들이 상호간 접속 방식, 데이터 형식, 데이터를 주고받는 방법 등을 정하는 방법. 대표적인 인터넷 표준 프로토콜으로 TCP와 UDP가 있다.



Network

URL 클래스

- URL을 추상화하여 만들어진 클래스. final로 설정되어있어 상속해서 사용이 불가능

생성자	설명	객체 생성 예
URL(String spec)	매개변수 spec으로 URL객체를 생성	new URL("http://java.sun.com/index.html");
URL(String protocol, String host, int port, String file)	protocol과 host, port와 file로 URL객체를 생성. file은 path와 query를 의미	new URL("http","java.sun.com",80,"index.html");
URL(String protocol, String host, String file)	protocol과 host, file로 URL 객체를 생성	new URL("http","java.sun.com","index.html");

메소드	반환형	설명
getAuthority()	String	URL의 호스트명과 포트를 결합한 문자열 반환
getPort()	int	URL에 명시된 포트를 반환. 없으면 -1을 반환
getDefaultPort()	String	URL에 상관없이 프로토콜의 default 포트번호를 반환
getFile()		URL의 path와 query를 결합한 문자열 반환
getHost()		URL의 host를 반환
getPath()		URL의 query를 반환
getProtocol()		URL의 protocol을 반환
getQuery()		URL의 query를 반환
getRef()		URL의 reference를반환
openConnection()	URLConnection	URLConnection의 객체를 생성하여 반환
getInputStream()	InputStream	InputStream객체를 생성하여 반환. 이 메소드로 url의 자원을 가져옴
toExternalForm()	String	URL을 문자열로 반환

Network

웹 사이트의 정보를 출력하는 프로그램을 작성

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

public class A_Class {
    public static void main(String[] args) throws Exception{
        URL site = new URL("https://www.naver.com/");
        URLConnection url = site.openConnection();
        BufferedReader in = new BufferedReader(
                                new InputStreamReader(
                                url.getInputStream(), "UTF-8"));
        String inLine;

        while ((inLine = in.readLine()) != null)
            System.out.println(inLine);
        in.close();
    }
}
```

Network

네이버 영화 '기생충'의 한줄평 5개를 가져와서 출력하는 프로그램

```java
public class NetworkTest {

    public static void main(String[] args) throws Exception {
        // TODO Auto-generated method stub
        URL site = new URL("https://movie.naver.com/movie/bi/mi/basic.naver?code=161967");

        URLConnection url = site.openConnection();

        BufferedReader in = new BufferedReader(new InputStreamReader(
                url.getInputStream(), "UTF-8"));

        String inLine;
        boolean check = false;

        while((inLine = in.readLine()) != null) {
            if(inLine.contains("/p")) {
                check = false;
            }

            if(check && !inLine.trim().equals("")) {
                System.out.println(inLine.trim());
            }

            if(inLine.contains("스포일러 컨텐츠로 처리되는지 여부")) {
                check = true;
            }
        }

        in.close();
    }
}
```

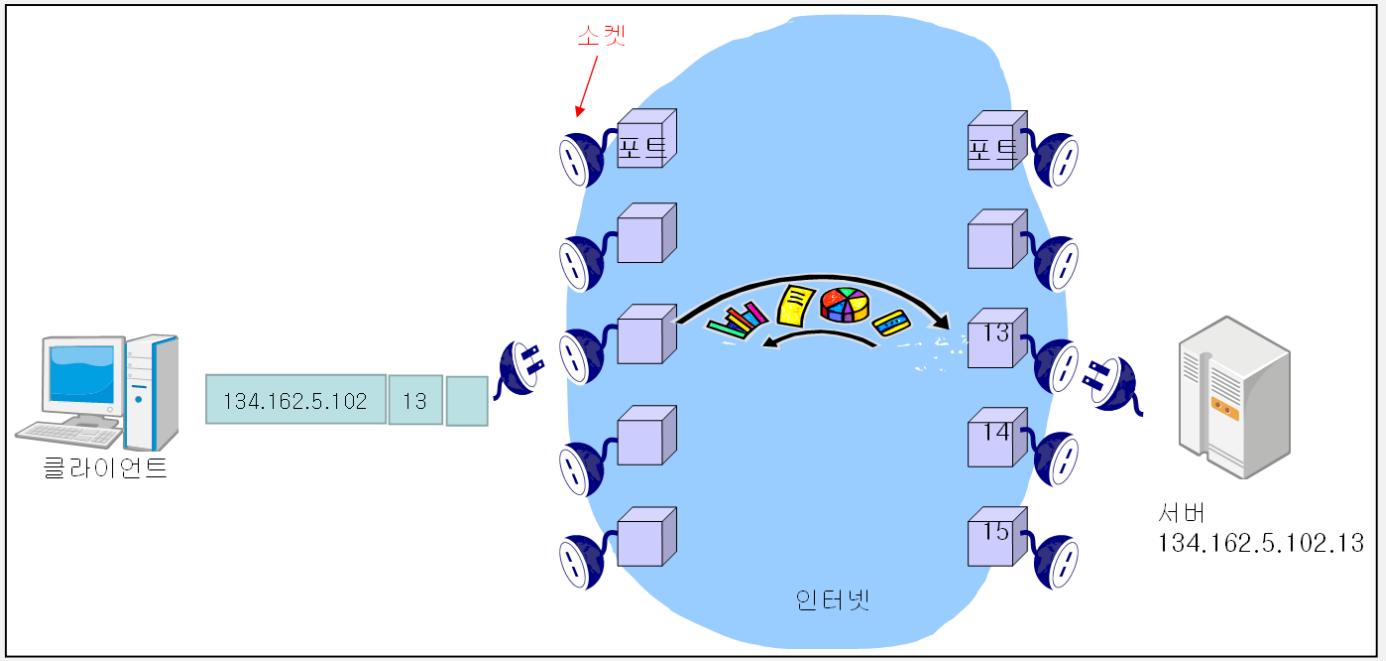
Network

소켓 프로그래밍 (Socket)

- 소켓을 이용한 통신을 하는 프로그램
- 소켓(Socket) : 프로세스간의 통신에 사용되는 양쪽 끝단(endpoint)를 의미하며, 네트워크상에서 서버, 클라이언트 두 개의 프로그램이 특정 포트를 통해 양방향 통신이 가능하도록 만들어주는 장치
- 서버 소켓(ServerSocket) : 포트와 연결(bind)되어 클라이언트의 연결을 기다리다가 연결이 되면 Socket을 생성해서 소켓끼리 연결되게 함

포트 (Port)

- 응용 프로그램과 통신하는 가상의 통신 선로
- 0 ~ 65535까지의 번호를 사용



Network

TCP/UDP – 연결지향 프로토콜/비연결지향 프로토콜

- 소켓을 이용하여 통신을 진행
- TCP (Transmission Control Protocol) : 신뢰성있게 통신하기 위하여 먼저 서로 간에 연결을 설정하고 데이터를 주고받는 방식
- UDP (User Datagram Protocol) : 데이터를 몇 개의 고정 길이의 패킷으로 분할하여 전송하는 방식

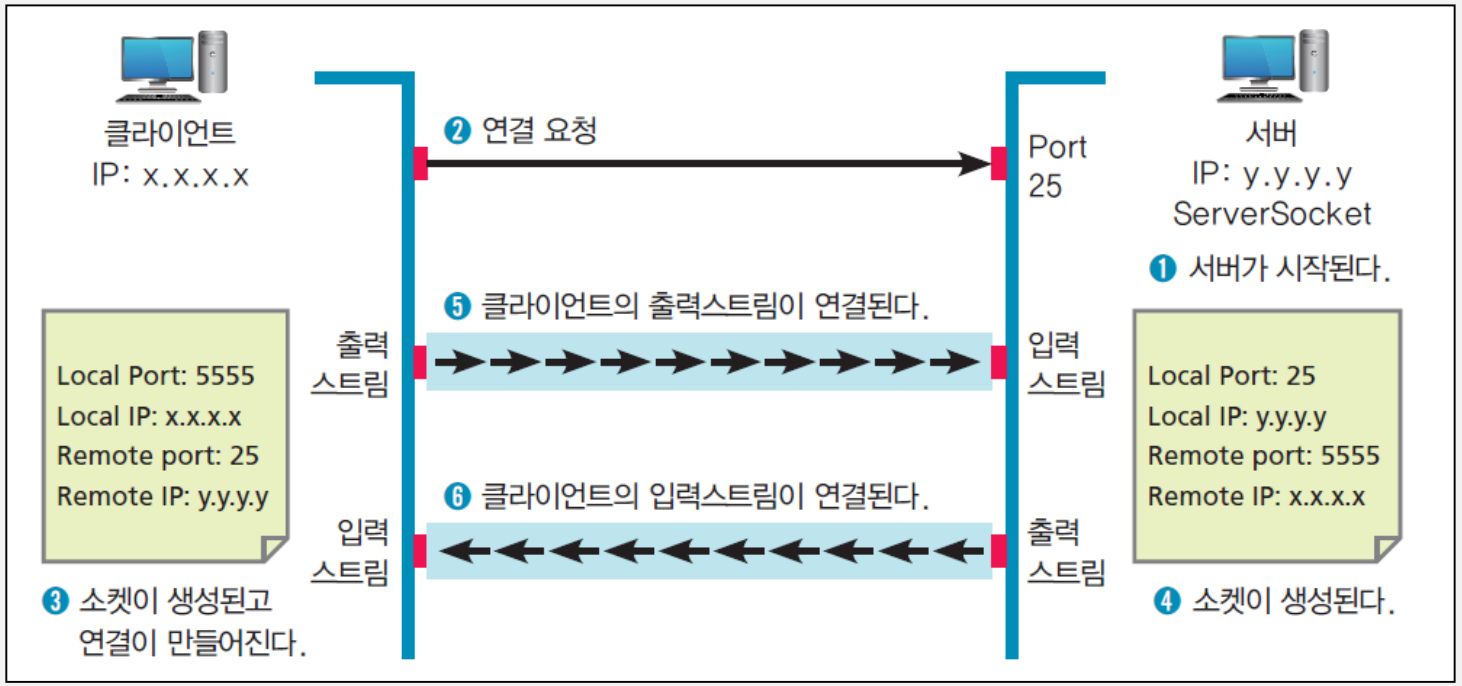
항목	TCP	UDP
연결방식	• 연결 후 통신 • 1:1 통신방식	• 연결하지 않고 통신 • 1:1부터 n:n 통신방식
특징	• 데이터의 전송순서가 보장 • 데이터의 수신여부 확인 • UDP보다 느림	• 데이터의 전송순서 보장되지 않음 • 순서가 바뀔 수 있음 • 데이터의 수신여부 확인안함 • TCP보다 속도가 빠름
방식 예	전화	택배 혹은 편지

Network

TCP 서버와 클라이언트 제작

- 서버는 연결 요청을 받는 소켓을 따로 가짐

1. SeverSocket 객체 생성
2. accept() 메소드 호출
3. 소켓으로부터 스트림 객체를 얻음
4. 스트림을 이용한 상호 대화
5. close() (종료)



Network

Socket 클래스

- 호스트를 찾을 수 없거나 서버의 포트가 열려 있지 않은 경우 UnKnownHostException 예외가 발생
- 네트워크 실패, 또는 방화벽에 의해 서버에 접근이 불가능 할 때 IOException 예외가 발생

생성자	설명
Socket(InetAddress address, int port)	InetAddress 객체와 port를 이용하여 Socket객체를 생성
Socket(String host, int port)	host와 port를 이용하여 Socket객체를 생성

메소드	반환형	설명
close()	void	소켓 객체를 닫는다
getInetAddress()	InetAddress	소켓 객체를 InetAddress 객체로 반환
getInputAddress()	InputStream	소켓 객체로 입력할 수 있는 InputStream 객체를 반환
getLocalAddress()	InetAddress	소켓 객체의 로컬 주소를 반환
getPort()	int	소켓 객체의 포트를 반환
isClosed()	boolean	소켓 객체가 닫혀있으면 true, 열려있으면 false를 반환
isConnected()	boolean	소켓 객체가 연결되어 있으면 true, 연결되어 있지 않으면 false 반환
setSoTimeout(int timeout)	void	소켓 객체의 시간을 밀리 세컨드로 설정

Network

TCP Server Socket 클래스

- 클라이언트와의 TCP 연결을 받기 위해서는 java.net.ServerSocket 클래스의 객체를 생성해야 함
- ServerSocket클래스는 java.net.Socket 객체를 생성하는 역할을 함
- accept()메소드는 클라이언트의 요청, 연결이 있을 때까지 블로킹됨

생성자	설명
ServerSocket(int port)	port를 이용하여 ServerSocket 객체를 생성

메소드	반환형	설명
accept()	Socket	클라이언트의 Socket 객체가 생성될때까지 블로킹되는 메소드
close()	void	ServerSocket 객체를 닫음
getLocalPort()	int	ServerSocket 객체가 청취하고 있는 포트번호를 반환
getSoTimeout()	int	ServerSocket 클래스의 accept() 메소드가 유효할 수 있는 시간을 밀리세컨드로 반환. 0 이면 무한대를 의미함
isClosed()	boolean	Serversocket 객체의 닫힌 상태를 반환
setSoTimeout(int timeout)	void	ServerSocket 클래스의 accept() 메소드가 유효할 수 있는 시간을 밀리세컨드로 설정

Network

현재 사용중인 TCP통신 Port번호를 출력하는 프로그램

```java
import java.io.IOException;
import java.net.ServerSocket;

public class NetworkTest {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        ServerSocket serverSocket = null;

        System.out.println("Port스캔을 시작합니다.");

        for(int i = 1; i < 65536; i++) {
            try {
                serverSocket = new ServerSocket(i);
                serverSocket.close();
            } catch (IOException e) {
                System.out.println(i + "번 포트가 사용중입니다. -> ");
            }
        }
    }
}
```

```
Port스캔을 시작합니다.
135번 포트가 사용중입니다. ->
139번 포트가 사용중입니다. ->
445번 포트가 사용중입니다. ->
1028번 포트가 사용중입니다. ->
1030번 포트가 사용중입니다. ->
1035번 포트가 사용중입니다. ->
3250번 포트가 사용중입니다. ->
3262번 포트가 사용중입니다. ->
3306번 포트가 사용중입니다. ->
3598번 포트가 사용중입니다. ->
3651번 포트가 사용중입니다. ->
3668번 포트가 사용중입니다. ->
3669번 포트가 사용중입니다. ->
3670번 포트가 사용중입니다. ->
3671번 포트가 사용중입니다. ->
3672번 포트가 사용중입니다. ->
3686번 포트가 사용중입니다. ->
3707번 포트가 사용중입니다. ->
```

Network

TCP 서버와 클라이언트 제작 – Echo 프로그램 (서버)

서버

```
public class SeverClass {
    public static void main(String[] args) {
        ServerSocket serverSocket = null;
        try {
            serverSocket = new ServerSocket(8000);
            System.out.println("서버가 준비되었습니다.");
        } catch (Exception e) {
            e.printStackTrace();
        }
```

ServerSocket을 생성하고 8000번 포트에 연결(bind)

Network

TCP 서버와 클라이언트 제작 – Echo 프로그램 (서버)

```
while (true) {
    try {
        System.out.println("연결 요청을 기다리는 중");
        Socket socket = serverSocket.accept();
        System.out.println(socket.getInetAddress() + "연결 요청이 들어왔습니다.");
        OutputStream out = socket.getOutputStream();
        DataOutputStream dos = new DataOutputStream(out);
        dos.writeUTF("TEST 확인");
        System.out.println("메시지를 전송합니다.");
        dos.close();
        socket.close();
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

클라이언트로부터의 연결이 올 때까지 계속 이 상태로 대기

OutputStream을 통해 "TEST 확인"이라는 메시지를 클라이언트에게 전송

사용 완료된 소켓과 스트림을 종료시키고 클라이언트의 연결을 기다림

Network

TCP 서버와 클라이언트 제작 – Echo 프로그램 (클라이언트)

클라이언트

```
public class ClientClass {
    public static void main(String[] args) {
        try {
            String serverIp = "127.0.0.1";
            System.out.println("서버에 연결 시도 중");

            Socket socket = new Socket(serverIp, 8000);
            InputStream in = socket.getInputStream();
            DataInputStream dis = new DataInputStream(in);
            System.out.println("서버로부터 받은 메시지: " + dis.readUTF());

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

서버의 포트 번호와 통신하기 위해 소켓을 생성하고 연결 시도

소켓의 InputStream을 통해 서버로부터 메세지를 받고 출력

Network

DatagramPacket클래스 - UDP

- DatagramPacket 클래스는 어플리케이션에서 주고받을 데이터와 관련된 클래스
- DatagramSocket 클래스는 실제 데이터의전송을 책임짐
- DatagramPacket 클래스는 데이터를 송신하기 위한 기능과 수신하기 위한 기능으로 분리되어 각각 생성자에서 제공

생성자	설명
DatagramPacket(byte[] buf, int length)	데이터를 수신하기 위한 생성자로 바이트 배열 buf의 length만큼 저장
DatagramPacket(byte[] buf, int length, InetAddress address, int port)	데이터를 송신하기 위한 생성자로 address와 port로 바이트 배열 buf의 length만큼 저장
DatagramPacket(byte[] buf, int offset, int length)	데이터를 수신하기 위한 생성자로 바이트 배열 buf의 offset 위치에서 length만큼 저장
DtagramPacket(byte[] buf, int offset, int length, InetAddress address, int port)	데이터를 송신하기 위한 생성자로 address와 port로 바이트 배열 buf의 offset위치에서 length만큼 저장

Network

DatagramPacket클래스 - UDP

메소드	반환형	설명
InetAddress()	InetAddress	데이터그램에 대한 목적지 또는 출발지 주소를 반환
getData()	byte[]	버퍼에 들어있는 실제 데이터를 바이트 배열로 반환
getLength()	int	버퍼에 들어있는 실제 데이터의 길이를 반환
getOffset()		버퍼에 들어있는 실제 데이터이 시작 위치를 반환
getPort()		데이터그램에 대한 목적지 또는 출발지 포트를 반환
setAddress(InetAddress iaddr)	void	데이터그램을 보낸 호스트 주소를 설정
setData(byte[] buf)		버퍼에 들어있는 실제 데이터를 바이트 배열 buffer로 설정
setData(byte[] buf, int offset, int length)		버퍼에 들어있는 실제 데이터를 바이트 배열 buffer의 offset위치에서 length만큼 설정
setLength(int length)		버퍼에 들어있는 실제 데이터의 길이를 설정
setPort(int port)		데이터그램에 대한 목적지 또는 출발지 포트를 설정

Network

DatagramSocket클래스 - UDP

- 스트림 소켓이 하나의 스트림에 연결되는 것과 다르게 데이터그램 소켓은 다중의 목적지로부터 수신하고 다중의 목적지로 전송
- DatagramPacket을 보내거나 받을 수 있는 메소드를 제공

생성자	설명
DatagramSocket()	할당된 특정한 포트번호가 중요하지 않다면 사용가능한 임시 UDP 포트로 소켓을 생성하여 DatagramSocket 객체를 생성
DatatgramSocket(int port)	매개변수 포트로 소켓을 생성하여 DatagramSocket 객체를 생성
DatagramSocket(int port, InetAddress iaddr)	매개변수 port와 iaddr로 소켓을 생성하여 DatagramSocket 객체를 생성

메소드	반환형	설명
send(DatagramPacket dp)	void	UDP 데이터그램(dp)를 전송하는 메소드
receive(DatagramPacket dp)		UDP 데이터그램ㅇ르 받아서 이미 존재하는 DatagramPacket객체인 dp에 저장
close()		데이터그램 소켓이 점유하고 있는 포트를 자유롭게 해줌
getLocalPort()	int	현재 소켓이 데이터그램을 기다리고 있는 로컬 포트가 몇 번인지 리턴
connect(InetAddress address, int port)	void	데이터그램 소켓이 지정된 호스트의 지정된 포트하고만 패킷을 주고받을 것이라고 지정
disconnect()		현재 연결된 데이터그램 소켓의 연결을 끊음
getPort()	int	데이터그램 소켓이 연결되어 있다면 소켓이 연결되어 있는 포트번호를 반환
getInetAddress()	InetAddress	데이터그램 소켓이 연결되어 있다면 소켓이 연결되어 있는 주소를 반환

Network

UDP를 이용한 통신

- DategramSocket() : UDP 프로토콜을 사용하는 소켓을 생성
- DatagramPacket() : UDP 패킷을 생성

데이터를 보낼 때

```
public class Sender {
    public static void main(String[] args) throws IOException {
        DatagramSocket socket = null;
        socket = new DatagramSocket();
        String s = "우리는 여전히 우리 운명의 주인이다.";
        byte[] buf = s.getBytes();

        // "address"의 "port"에 있는 클라이언트에게 데이터를 보낸다.
        InetAddress address = InetAddress.getByName("127.0.0.1"); // 로컬 호스트
        DatagramPacket packet = new DatagramPacket(buf, buf.length, address, 5000);
        socket.send(packet);
        socket.close();
    }
}
```

Network

UDP를 이용한 통신

- DategramSocket() : UDP 프로토콜을 사용하는 소켓을 생성
- DatagramPacket() : UDP 패킷을 생성

데이터를 받을 때

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

        byte[] buf = new byte[256];

        DatagramSocket socket = new DatagramSocket(5000);  // 포트 번호: 5000
        DatagramPacket packet = new DatagramPacket(buf, buf.length);
        socket.receive(packet);
        System.out.println(new String(buf));
  }
}
```

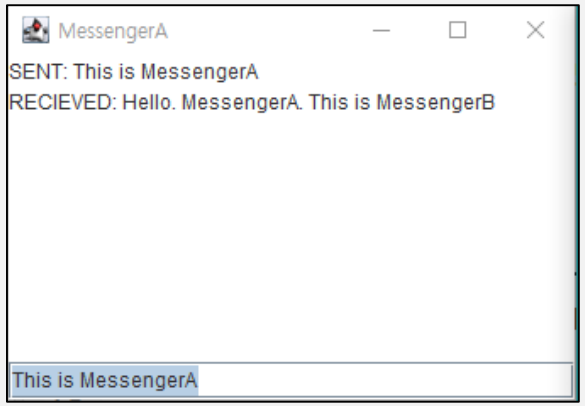
IO

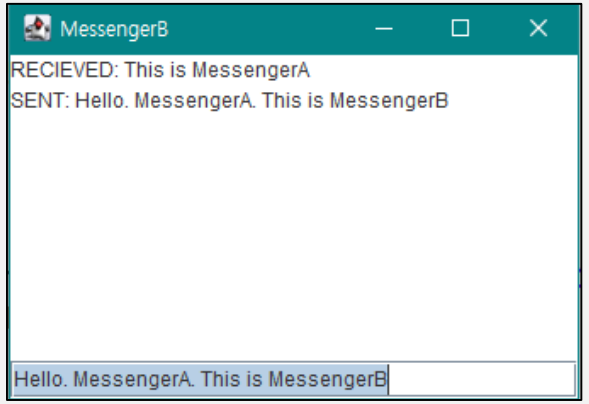
UDP를 이용한 서버와 클라이언트 작성

간단한 채팅을 할 수 있는 메신저를 작성

MessengerA

```
public class MessengerA {
    protected JTextField textField;
    protected JTextArea textArea;
    DatagramSocket socket;
    DatagramPacket packet;
    InetAddress address = null;
    final int myPort = 5000;          // 수신용 포트 번호
    final int otherPort = 6000; // 송신용 포트 번호

    public MessengerA() throws IOException {
          MyFrame f=new MyFrame();
          address = InetAddress.getByName("127.0.0.1");
          socket = new DatagramSocket(myPort);
    }
```





IO

UDP를 이용한 서버와 클라이언트 작성

```
public void process() {
    while (true) {
        try
        {
                byte[] buf = new byte[256];
                packet = new DatagramPacket(buf, buf.length);
                socket.receive(packet); // 패킷을 받는다.
                // 받은 패킷을 텍스트 영역에 표시한다.
                textArea.append("RECIEVED: " + new String(buf) + "\n");
        }
        catch (IOException ioException) {
                ioException.printStackTrace();
        }
    }
}
```

IO

UDP를 이용한 서버와 클라이언트 작성

내부클래스로 GUI작성

```
class MyFrame extends JFrame implements ActionListener {
    public MyFrame() {
            super("MessengerA");
            setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

            textField = new JTextField(30);
            textField.addActionListener(this);

            textArea = new JTextArea(10, 30);
            textArea.setEditable(false);

            add(textField, BorderLayout.PAGE_END);
            add(textArea, BorderLayout.CENTER);
            pack();
            setVisible(true);
    }
```

```
    public void actionPerformed(ActionEvent evt) {
        String s = textField.getText();
        byte[] buffer = s.getBytes();
        DatagramPacket packet;

        // 패킷을 생성한다.
        packet = new DatagramPacket(buffer, buffer.length, address,
                    otherPort);
        try {
                socket.send(packet);    // 패킷을 보낸다.
        } catch (IOException e) {
                e.printStackTrace();
        }
        textArea.append("SENT: " + s + "\n");
        textField.selectAll();
        textArea.setCaretPosition(textArea.getDocument().getLength());
    }
}
```

IO

UDP를 이용한 서버와 클라이언트 작성

내부클래스로 GUI작성

```
public static void main(String[] args) throws IOException {
    MessengerA m = new MessengerA();
    m.process();
}
```

```
public class MessengerB {
    final int myPort = 6000;          // 수신용 포트 번호
    final int otherPort = 5000; // 송신용 포트 번호

public static void main(String[] args) throws IOException {
    MessengerB m = new MessengerB();
    m.process();
}
```

MessengerB

클래스 이름과 포트번호 이외의 코드는 동일

Network

유니캐스트 / 브로드캐스트 / 멀티캐스트 – UDP통신

- **유니캐스트 통신(Unicast) :**
  UDP통신에 참여하는 대상이 1:1로 연결
  전송하는 대상이 데이터를 전송받는 대상의 IP주소와 Port 번호를 알고 있어야 함

- **브로드캐스트 통신 (Broadcast) :**
  UDP통신에 참여하는 대상이 1:N으로 연결 (1대 다수)
  유니캐스트 통신을 보완한 방식으로, 브로드캐스트 통신을 지원하는 모든 IP주소 대역의 사람들에게 동시에 데이터를 전송하는 방식. 255번 ip는 브로드 캐스팅 전용 ip주소임
  ex) 10.211.55.xxx 대역의 모든 사람에게 전송하려면 10.211.55.255로 전송

- **멀티캐스트 통신 (Multicast) :**
  224.0.0.0 ~ 239.255.255.255까지 D클래스의 IP주소를 사용하는 방식
  UDP통신에 참여하는 대상이 1:N으로 연결 (1대 다수)
  원하는 대상만 데이터를 전송받을 수 있다는 점에서 브로드캐스팅과 차이가 있음
  클라이언트들은 join()과 leave()메소드를 사용하여 수신 여부를 결정
  네트워크나 라우터가 지원해야 가능한 방식임

Network

브로드캐스팅 예제 - Sender

Sender

```
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.InetAddress;
import java.util.Scanner;

public class BroadcastSender{
    public static void main(String[] args) throws Exception {
        Scanner input = new Scanner(System.in);
        DatagramSocket socket = new DatagramSocket();

        while(true) {
            System.out.println("Message = ");
            String message = input.nextLine();
            InetAddress remote_addr = InetAddress.getByName("xxx.xxx.xxx.255");
            DatagramPacket packet = new DatagramPacket(
                    message.getBytes(), message.getBytes().length, remote_addr, 8000);
            socket.send(packet);
        }
    }
}
```

Network

브로드캐스팅 예제 - Receiver

Receiver

```java
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.InetAddress;

public class BroadCastClient {
    public static void main(String[] args) throws Exception {
        DatagramSocket socket = new DatagramSocket(8000);
        System.out.println("server readey...");
        try {
            while(true) {
                DatagramPacket packet = new DatagramPacket(new byte[65508], 65508);
                socket.receive(packet);

                InetAddress client_addr = packet.getAddress();
                String message = new String(packet.getData()).trim();
                System.out.println(client_addr + " >> " + message);
            }
        } catch (Exception e) {
            // TODO: handle exception
        }
        finally {
            socket.close();
        }
    }
}
```

Network

MulticastSocket

- IP주소는 미리 클라이언트와 서버 사이에 약속되어있어야 함
- 주소는 224.0.0.0 ~ 239.255.255.255 이내여야 함

생성자	설명
MulticastSocket	UDP 통신으로 데이터를 전송하는 MulticatSocket 클래스의 객체를 생성
MulticatsSocket(int port)	특정 포트로 설정되는 MulticastSocket 클래스의 객체를 생성

메소드	반환형	설명
getLoopbackMode()	**boolean**	데이터의 로컬 루프백 설정 여부를 확인
getTimeTolive()	**int**	데이터의 TTL값을 가져옴
joinGroup(InetAddress addr)	**void**	멀티캐스트용 IP 주소에 연결
leaveGroup(InetAddress addr)		멀티캐스트용 IP 주소에 연결을 해제
send(DatagramPacket dp)		데이터를 전송
setLoopbackMode(Boolean bool)		데이터의 로컬 루프백을 설정
setTimeToLive(int ttl)		데이터의 TTL 값을 설정

Network

멀티캐스팅 예제 - Receiver

Sender

```
import java.net.DatagramPacket;
import java.net.InetAddress;
import java.net.MulticastSocket;
import java.util.Scanner;

public class MulticastSender {
    public static void main(String[] args) throws Exception {
        // TODO Auto-generated method stub
        Scanner input = new Scanner(System.in);
        MulticastSocket socket = new MulticastSocket();

        while(true) {
            System.out.println("Message = ");
            String message = input.nextLine();
            InetAddress remote_addr = InetAddress.getByName("224.0.0.1");
            DatagramPacket packet = new DatagramPacket(
                    message.getBytes(), message.getBytes().length, remote_addr, 8000);
            socket.send(packet);
        }
    }
}
```

Network

멀티캐스팅 예제 - Receiver

Receiver

```java
import java.net.DatagramPacket;
import java.net.InetAddress;
import java.net.MulticastSocket;

public class MulticastReceiver {
    public static void main(String[] args) throws Exception {
        MulticastSocket socket = new MulticastSocket(8000);
        socket.joinGroup(InetAddress.getByName("224.0.0.1"));
        System.out.println("server readey...");
        try {
            while(true) {
                DatagramPacket packet = new DatagramPacket(new byte[65508], 65508);
                socket.receive(packet);

                InetAddress client_addr = packet.getAddress();
                String message = new String(packet.getData()).trim();
                System.out.println(client_addr + " >> " + message);
            }
        } catch (Exception e) {
            // TODO: handle exception
        }
        finally {
            //socket.leaveGroup(InetAddress.getByName("224.0.0.1"));
            //socket.close();
        }
    }
}
```

수고하셨습니다.