네트워크-UDP 프로그래밍

comsi.java@gmail.com
박 경 태

http://github.com/hopypark

1.UDP(User Datagram Protocol)란?

- UDP란?
  - TCP/IP 4계층에서 봤을 때, TCP와 같은 Transport Layer에 위치
  - TCP와 동급의 프로토콜로 데이터를 전송하기 위해서 사용되는 프로토콜
- TCP와는 다르게 비연결지향성(connectionless)
- 데이터의 흐름을 신뢰할 수 없다.
- TCP보다 빠르게 데이터를 주고 받을 수 있다.
- 단순히 데이터그램 위주의 통신을 하기 때문에 데이터 지향 프로토콜이라고 불린다.

1.1 UDP - 비연결지향성(connectionless)

- TCP는 통신 전에 상대방을 확인하는 절차를 가짐
  - 통신선로(session)를 연결하고 데이터의 흐림이 이루어짐

- UDP는 통신선로를 만들기 위한 작업이 없음
  - 그냥 보내고 받기 만하므로 신뢰성이 떨어짐
  - 클라이언트가 서버로 데이터를 보냈다 하더라도 서버로 데이터가 실제로 도착했는지 확인할 수 없음.

1.2 비신뢰성(unreliable)

- TCP는 프로토콜 자체에 메시지가 제대로 보내졌음을 확인하는 다양한 장치가 존재
  - 각 패킷에 순서를 매겨서 순서가 뒤엉키지 않도록 재조립함
  - 일정시간 동안 패킷이 도착하지 않으면 해당 패킷을 재요청

- UDP는 TCP와 같은 장치가 없다.
  - 전송된 패킷의 순서가 뒤바뀔 수 있거나 손실될 수도 있다.
  - 패킷의 순서가 바뀌었는지 손실되었는지 알 수 있는 방법이 없다.

- 신뢰성 확보
  - application차원에서 직접 코딩
  - 패킷 생성할 때 데이터 일련번호 등을 넣어서 보냄
  - 서버는 이에 대한 응답을 보내는 방식으로 패킷 신뢰성을 부여

1.3 UDP의 특징 및 사용처

- TCP보다 더 간단하고 더 빠른 처리를 보장
- 프로그래밍도 TCP에 비해 더욱 간단
  - TCP 서버와 같이 listen, accept를 할 필요가 없음
  - 소켓 생성하고 읽을 데이터가 있는지 기다리기만 한다(connectionless므로)
- 사용처
  - 음성 및 비디오를 위한 실시간 스트리밍 서비스
    - 통신의 품질보다는 통신의 연속성이 더 유용하게 사용되는 곳
  - 많은 패킷이 오가면서 중요하지 않은 몇 개의 데이터 손실 정도는 눈 감아 줄 수 있는 곳
  - UDP프로토콜을 사용하는 서비스는 DNS, NFS, SNMP, syslog등이 있다.

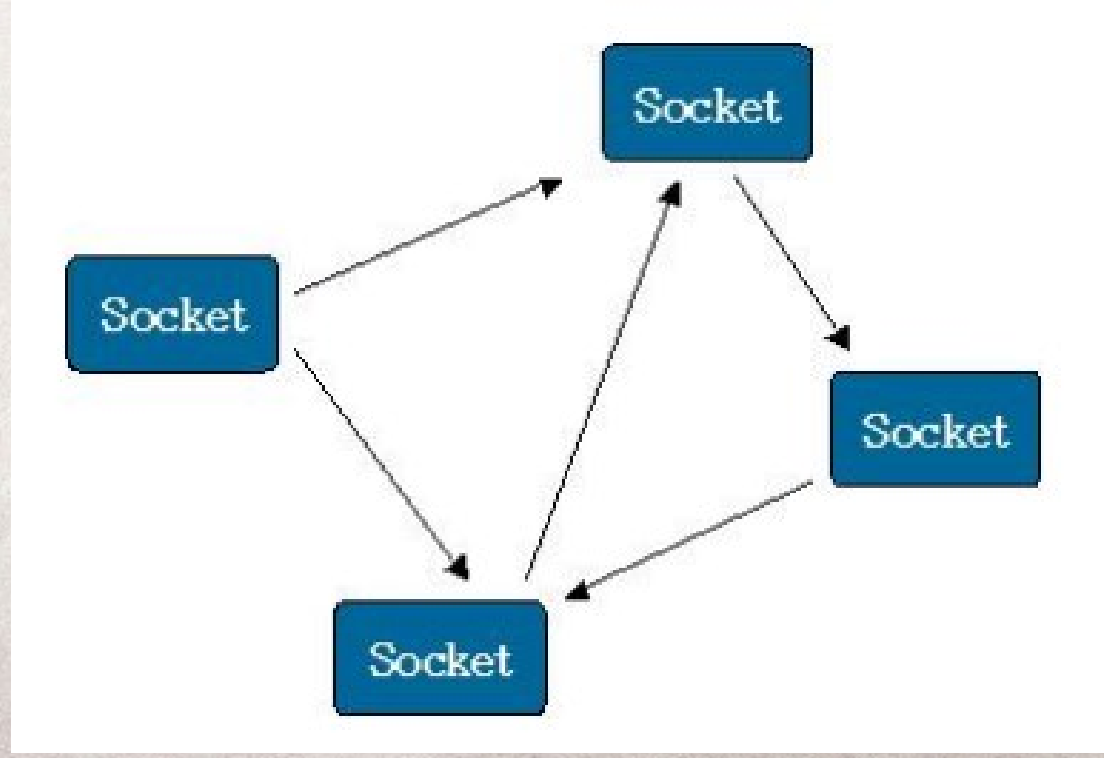
TCP vs. UDP

구분	TCP	UDP
신뢰성	• Reliable	• Unreliable
연결성	• 연결지향성 • 연결을 맺고 통신	• 비연결성
재전송	• 재전송 요청함 (오류 및 패킷 손실 검출시)	• 재 전송 없음
특징	• flow control 사용 • 속도는 다소 느리지만 신뢰성 제공 • 1:1 송수신 • 스트림 전송으로 전송 데이터의 크기 무제한	• 신뢰성 보장은 없지만 고속데이터 전송 • 실시간 전송에 적합 • 1:1 혹은 1:N 연결 • 데이터 그램 단위 전송이며, 하나의 데이터그램은 65535바이트로 제한(그 이상은 잘라서 보내야함)
용도	• 신뢰성이 필요한 통신 • HTTP, FTP, …	• 총 패킷수가 적은 통신 • 동영상 및 음성 등 멀티미디어 통신

2.1 UDP기반 서버/클라이언트

- UDP 서버는 클라이언트와 연결되어 있지 않다.
  - 데이터를 주고 받기 위한 연결 설정 과정이 필요 없다.
    - accept 함수와 connect 함수의 호출이 불필요
  - 소켓의 생성과 주소 할당 만이 UDP서버와 클라이언트가 구현해야 할 전부임.

- UDP 서버의 소켓은 오로지 하나
  - UDP 소켓 하나로 여러 호스트들과 데이터 송수신 할 수 있음
    - UDP 기반으로 통신하는 호스트들은 여러 개의 소켓을 생성하지 않음.



2.2 UDP 에코 클라이언트

- 시나리오

DatagramSocket의 인스턴스 구성
(로컬주소와 포트 지정)

DatagramSocket의 send()와 receive() 메소드를 사용해
DatagramPacket의 인스턴스를 주고 받음

통신이 끝나면 DatagramSocket의 close() 메소드를 사용하여 소켓 자원을 돌려준다.

2.2.1 데이터그램 패킷(DatagramPacket)

- TCP의 스트림 대신에 UDP에서 사용되는 독립된 메시지

- DatagramPacket의 송신은 먼저 DatagramPacket 인스턴스를 구성하고 DatagramSocket의 send() 메소드를 통해 송신

- 수신은 할당된 공간(byte[])을 가진 DatagramPacket 인스턴스를 구성하고 DatagramSocket의 receive() 메소드를 통해 수신

2.2.1 데이터그램 패킷(DatagramPacket)

- 생성자

문 법	내 용
DatagramPacket(byte[] buffer, int length)	•목적지 주소가 지정되지 않음(setAddress()로 지정)
DatagramPacket(byte[] buffer, int offset, int length)	•기본적으로 받기 위한 DatagramPacket을 구성하는 데 사용
DatagramPacket(byte[] buffer, int length, InetAddress remoteAddr, int remotePort)	•기본적으로 받기 위한 DatagramPacket을 구성하는 데 사용
DatagramPacket(byte[] buffer, int offset, int length, InetAddress remoteAddr, int remotePort)	

buffer	데이터그램의 실제 전송 정보
length	실제로 사용된 버퍼의 바이트 수.
offset	버퍼 배열에 있는 송수신될 메시지 데이터의 첫번째 바이트의 위치(기본 0으로 지정)
remoteAddr	데이터그램의 주소(보통은 목적지)
remotePort	데이터그램의 포트(보통은 목적지)

2.2.1 데이터그램 패킷(DatagramPacket)

- 메소드

메 소 드	내 용
InetAddress getAddress()	Datagram을 보내거나 받을 서버의 IP address 반환
void setAddress(InetAddress iaddr)	Datagram이 보내질 서버의 IP address 설정
int getPort()	Datagram을 보내거나 받을 서버의 포트 번호 반환
void setPort(int port)	Datagram을 보내거나 받을 서버의 포트 번호 설정
byte[] getData()	Datagram과 연관된 버퍼 반환. 반환된 객체는 가장 최근에 이 DatagramPacket에 대해 생성자나 setData()에 의해 연관된 바이트 배열에 대한 참조이다.
void setData(byte[] buffer) void setData(byte[] buffer, int offset, int length)	주어진 바이트 배열을 데이터그램 버퍼로 만든다. 첫번째는 바이트 배열 전체를 버퍼로 만든다. 두번째 형식은 오프셋에서 오프셋+길이-1까지의 바이트를 버퍼로 만든다.
int getLength()	Datagram 내부 길이를 반환.
void setLength(int length)	Datagram 내부 길이를 설정. 명시적으로 생성자나 setLength()메소드에 의해 설정. 연관된 버퍼의 길이보다 더 크게 만들려면 IllegalArgumentException 발생

2.2.2 데이터그램 소켓(DatagramSocket)

- 생성자

문 법	내 용
DatagramSocket()	• 로컬 포트가 지정되지 않았다면, 소켓은 가능한 아무 로컬 포트로 나 설정
DatagramSocket(int localPort)	
DatagramSocket(int localPort, InetAddress localAddr)	• 로컬 주소가 지정되지 않으면 로컬 주소 중의 하나가 선택

로컬포트; 0의 로컬 포트는 생성자가 아무 가능한 포트를 선택하게 해준다.

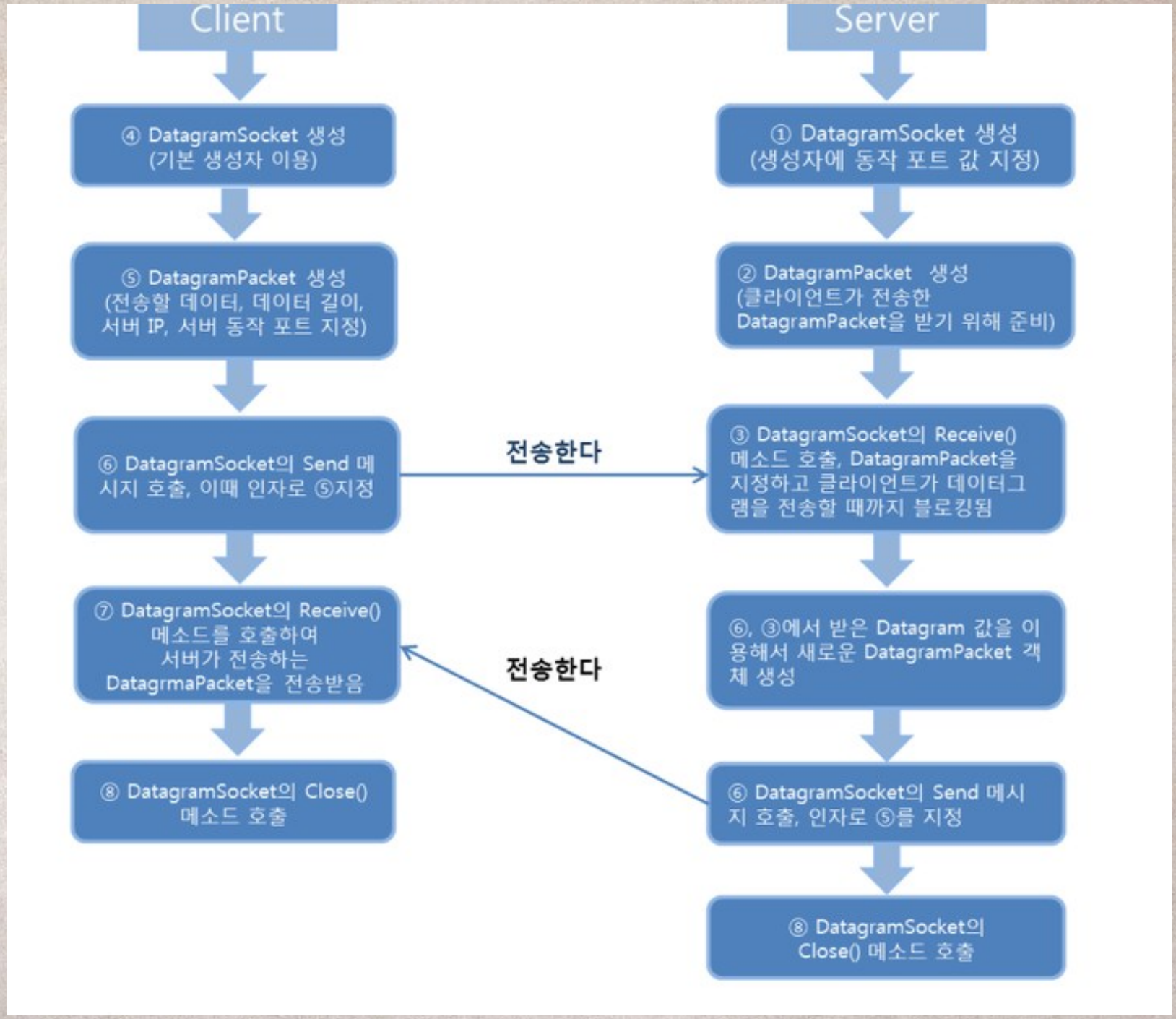
로컬주소

2.2.2 데이터그램 소켓(DatagramSocket)

- 메소드

메 소 드	내 용
void close()	소켓 닫기. 데이터 그램은 해제 후 이 소켓을 사용하여 송수신할 수 없다.
void connect(InetAddress remoteAddr, int remotePort)	소켓의 원격 주소와 포트를 설정한다. 다른 주소를 가진 데이터그램을 송신하려면 예외가 발생할 수 있고, 다른 포트나 주소에서의 데이터 그램은 무시된다.
void disconnect()	소켓에 지정된 원격주소와 포트를 제거한다.
void receive(DatagramPacket packet)	이 소켓으로부터 데이터그램을 수신한다.
void send(DatagramPacket packet)	이 소켓으로부터 데이터그램을 송신한다.
int getSoTimeout() void setSoTimout(int timeout)	이 소켓에서 receive()에서 멈출 수 있는 최대 시간을 반환하거나 설정한다. 데이터가 사용 가능하기 전에 특정 시간이 흘러 버리면, InterruptedIOException이 발생한다(0의 값은 데이터가 사용 가능할 때까지 수신이 멈추어 있는다).

클라이언트와 서버의 동작 순서



Echo Server/Client

UDP 클라이언트: UDPEchoClientTimeout.java

UDPEchoClientTimeout.java | UDPEchoServer.java

```java
import java.io.IOException;
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.InetAddress;
import java.net.SocketException;
import java.net.SocketTimeoutException;
import java.net.UnknownHostException;


public class UDPEchoClientTimeout {

    private static final int TIMEOUT_MSEC   = 3000;
    private static final int MAXTRIES       = 5;
    private static final String HOSTNAME    = "localhost";
    private static final int PORT           = 5000;

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

        InetAddress serverAddress = null;
        // 서버 주소
        try {
            serverAddress = InetAddress.getByName(HOSTNAME);
        } catch (UnknownHostException e) {
            System.out.println("알려지지 않은 호스트입니다.");
            System.out.println(e.getMessage());
        }
        // 전송할 문자열을 byte[]로 변환
        byte[] bytesToSend = "Hello COMSI~한글".getBytes();

        // 사용할 소켓
```

UDP 클라이언트: UDPEchoClientTimeout.java

```
30          // 사용할 소켓
31          DatagramSocket socket = null;
32          try {
33              socket = new DatagramSocket();
34              // 데이터 수신 최대 대기시간
35              socket.setSoTimeout(TIMEOUT_MSEC);
36          } catch (SocketException e) {
37              // TODO Auto-generated catch block
38              e.printStackTrace();
39          }
40
41          // 전송할 패킷
42          DatagramPacket sendPacket = new DatagramPacket(bytesToSend,
43                  bytesToSend.length, serverAddress, PORT);
44          // 수신할 패킷
45          DatagramPacket receivePacket =
46                  new DatagramPacket(new byte[bytesToSend.length], bytesToSend.length);
47
48          int tries = 0;
49          boolean receivedResponse = false;
50
```

UDP 클라이언트: UDPEchoClientTimeout.java

```
50
51          do{
52              try {
53                  socket.send(sendPacket);
54              } catch (IOException e) {
55                  e.printStackTrace();
56              }
57
58              try{
59                  socket.receive(receivePacket); // 수신시도
60                  if(!receivePacket.getAddress().equals(serverAddress))
61                      throw new IOException("알려지지 않은 서버로부터 패킷을 받음.");
62                  receivedResponse = true;
63
64              }catch(SocketTimeoutException s){ // 지정된 시간동안 아무것도 받지 못할 경우
65                  tries+= 1;
66                  System.out.println("Timed out, " + (MAXTRIES - tries) + " more tries...");
67              }
68
69          } while((!receivedResponse) && (tries < MAXTRIES));
70
71          if(receivedResponse){
72              System.out.println("Received: " + new String(receivePacket.getData()));
73          }else{
74              System.out.println("0 response -- giving up");
75          }
76          socket.close();
77      }
78  }
79
```

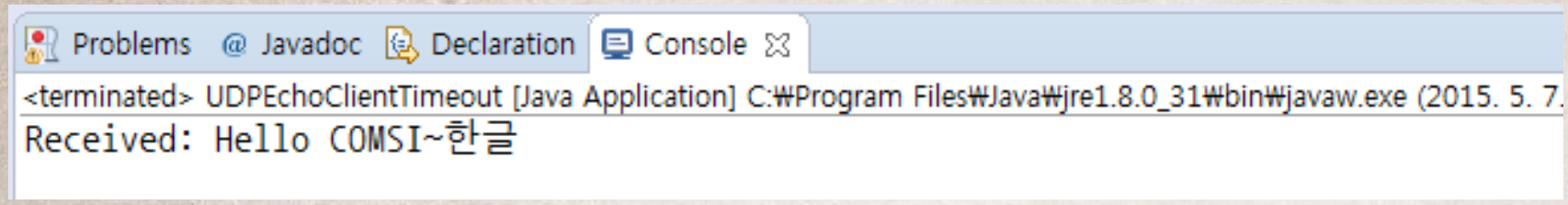
UDP서버: UDPEchoServer.java

UDPEchoClientTimeout.java | UDPEchoServer.java

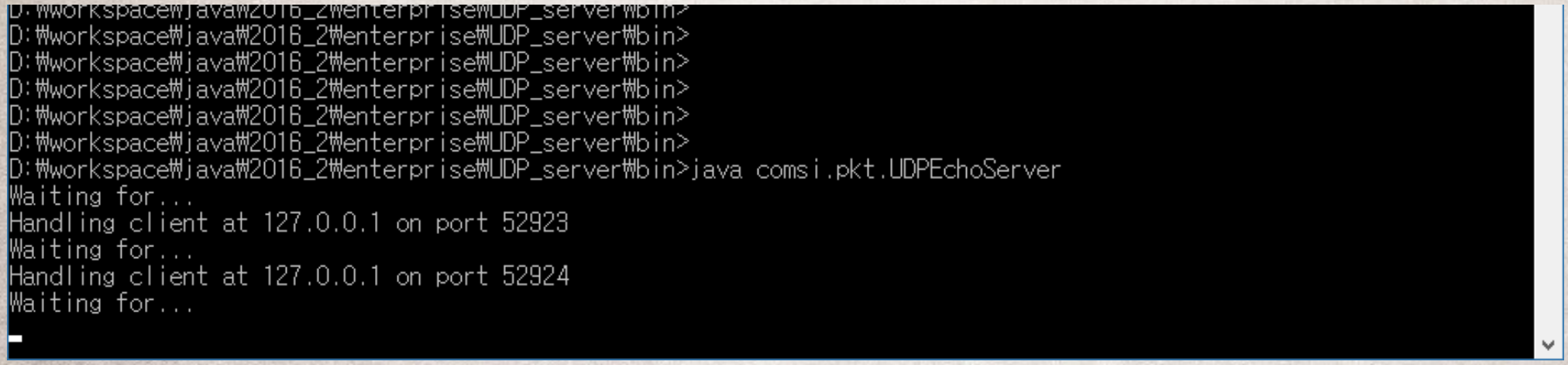
```java
import java.io.IOException;
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.SocketException;


public class UDPEchoServer {

    // datagram의 최대 크기
    private static final int ECHO_MAX_BYTE  = 255;
    private static final int PORT           = 5000;

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

        DatagramSocket socket = null;
        DatagramPacket packet = null;

        try {
            socket = new DatagramSocket(PORT);
        } catch (SocketException e) {
            e.printStackTrace();
        }

        packet = new DatagramPacket(new byte[ECHO_MAX_BYTE], ECHO_MAX_BYTE);

        for(;;){ // datagram 받고 보내기 무한반복
            // 클라이언트로부터 패킷을 받는다.
            socket.receive(packet);
            System.out.println("Handling client at "
                + packet.getAddress().getHostAddress() + " on port " + packet.getPort());
            socket.send(packet); // 클라이언트에 받은 패킷을 보냄.
            packet.setLength(ECHO_MAX_BYTE);    // 버퍼크기가 최소되지 않도록 재설정
        }
    }
}
```

결과화면

Client



Server



메시지 주고 받기(객체)

DataInputStream, DataOutputStream

- 자바 기본 데이터 타입의 데이터를 input/output 하기 위한 클래스
- DataInputStream은 inputstream을 통해서 DataOutputStream은 outputstream 통해서 데이터를 입출력 할 수 있다.

Constructors

Constructor and Description
`DataInputStream(InputStream in)` Creates a DataInputStream that uses the specified underlying InputStream.

Constructors

Constructor and Description
`DataOutputStream(OutputStream out)` Creates a new data output stream to write data to the specified underlying output stream.

DataInputSream - methods

Methods

Modifier and Type	Method and Description
int	**read**(byte[] b) Reads some number of bytes from the contained input stream and stores them into the buffer array b.
int	**read**(byte[] b, int off, int len) Reads up to len bytes of data from the contained input stream into an array of bytes.
boolean	**readBoolean**() See the general contract of the readBoolean method of DataInput.
byte	**readByte**() See the general contract of the readByte method of DataInput.
char	**readChar**() See the general contract of the readChar method of DataInput.
double	**readDouble**() See the general contract of the readDouble method of DataInput.
float	**readFloat**() See the general contract of the readFloat method of DataInput.
void	**readFully**(byte[] b) See the general contract of the readFully method of DataInput.
void	**readFully**(byte[] b, int off, int len) See the general contract of the readFully method of DataInput.
int	**readInt**() See the general contract of the readInt method of DataInput.

DataOutputSream - methods

Methods

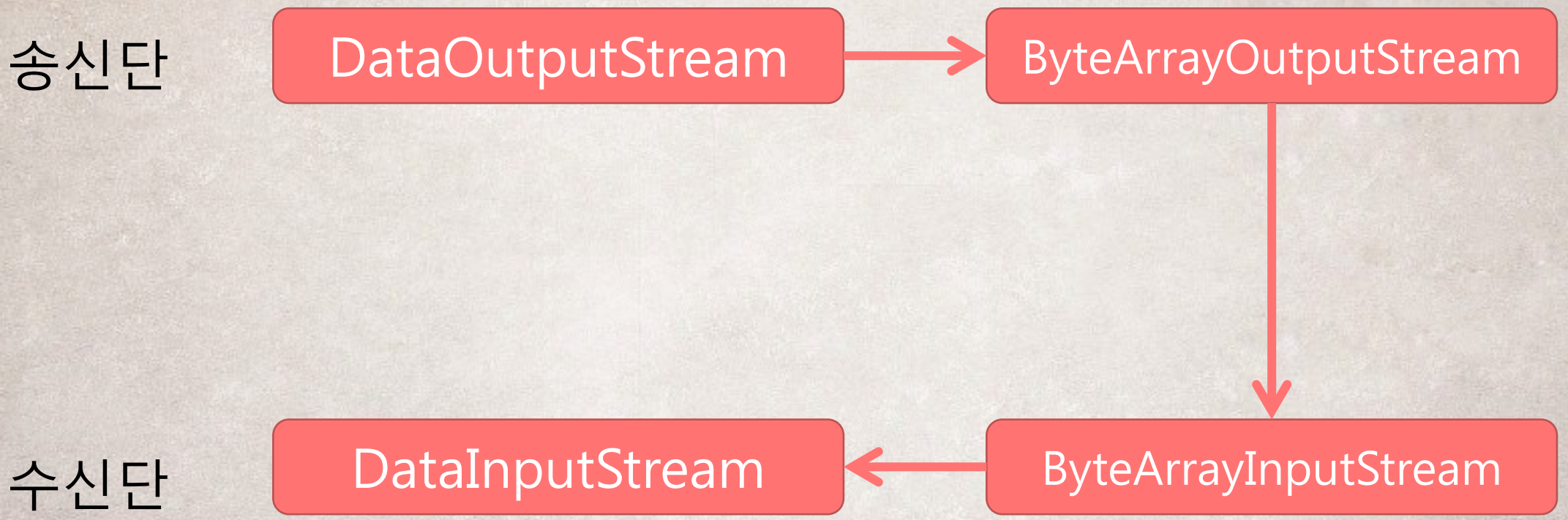
Modifier and Type	Method and Description
void	**flush**() Flushes this data output stream.
int	**size**() Returns the current value of the counter written, the number of bytes written to this data output stream so far.
void	**write**(byte[] b, int off, int len) Writes len bytes from the specified byte array starting at offset off to the underlying output stream.
void	**write**(int b) Writes the specified byte (the low eight bits of the argument b) to the underlying output stream.
void	**writeBoolean**(boolean v) Writes a boolean to the underlying output stream as a 1-byte value.
void	**writeByte**(int v) Writes out a byte to the underlying output stream as a 1-byte value.
void	**writeBytes**(String s) Writes out the string to the underlying output stream as a sequence of bytes.
void	**writeChar**(int v) Writes a char to the underlying output stream as a 2-byte value, high byte first.
void	**writeChars**(String s) Writes a string to the underlying output stream as a sequence of characters.
void	**writeDouble**(double v) Converts the double argument to a long using the doubleToLongBits method in class Double, and then writes that long value to the underlying output stream as an 8-byte quantity, high byte first.
void	**writeFloat**(float v) Converts the float argument to an int using the floatToIntBits method in class Float, and then writes that int value to the underlying output stream as a 4-byte quantity, high byte first.
void	**writeInt**(int v) Writes an int to the underlying output stream as four bytes, high byte first.

Member 클래스

UDPServer.java | UDPClient.java | Member.java | MemberEncoderBin.java | MemberDecoderBin.java

```java
package comsi.pkt;

public class Member {
    public String name;
    public int age;
    public int weight;
    public int height;

    public Member(String name, int age, int weight, int height){
        this.name = name;
        this.age = age;
        this.weight = weight;
        this.height = height;
    }

    public String toString(){
        return "name=" + name + "\n" +
                        "age=" + age + "\n" +
                        "weight=" + weight + "\n" +
                        "height=" + height;

    }
}
```

데이터 송수신 흐름도


송신단
DataOutputStream
ByteArrayOutputStream
수신단
DataInputStream
ByteArrayInputStream

자바 객체 전송 - 부호화 전송

UDPServer.java | UDPClient.java | Member.java | MemberEncoderBin.java | MemberDecoderBin.java

```java
package comsi.pkt;

import java.io.ByteArrayOutputStream;
import java.io.DataOutputStream;
import java.io.IOException;

public class MemberEncoderBin {
    public final int MAX_LENGTH = 1024;
    public final String ENCODING = "ISO-8859-1";


    public byte[] encode(Member member) throws Exception{

        ByteArrayOutputStream buf = new ByteArrayOutputStream();
        DataOutputStream out = new DataOutputStream(buf);

        out.writeInt(member.age);
        out.writeInt(member.weight);
        out.writeInt(member.height);
        byte[] encodedName = member.name.getBytes(ENCODING);

        if (encodedName.length > MAX_LENGTH)
            throw new IOException("Encoded length too long");
        out.writeInt(encodedName.length);
        out.write(encodedName);

        out.flush();
        System.out.println(encodedName.length + ":" + encodedName);

        return buf.toByteArray();

    }
}
```

자바 객체 전송 – 부호화 전송

UDPServer.java | UDPClient.java | Member.java | MemberEncoderBin.java | MemberDecoderBin.java

```java
package comsi.pkt;

import java.io.ByteArrayInputStream;
import java.io.DataInputStream;
import java.io.EOFException;
import java.io.IOException;
import java.net.DatagramPacket;

public class MemberDecoderBin {

    public final int MAX_LENGTH = 1024;
    public final String ENCODING = "ISO-8859-1";

    public Member decode(DatagramPacket p) throws IOException{
        ByteArrayInputStream packet
            = new ByteArrayInputStream(p.getData(), p.getOffset(), p.getLength());

        DataInputStream src = new DataInputStream(packet);
        int age = src.readInt();
        int weight = src.readInt();
        int height = src.readInt();
        int strLength = src.readInt();

        if (strLength == -1) throw new EOFException();

        byte[] buf = new byte[strLength];
        src.read(buf);
        String name = new String(buf, ENCODING);

        return new Member(name, age, weight, height);
    }
}
```

read(buf, 0, buf.length)

자바 객체 전송 - 전송 클라이언트

UDPServer.java | UDPClient.java | Member.java | MemberEncoderBin.java | MemberDecoderBin.java

```java
package comsi.pkt;

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

public class UDPClient {

    private static final String HOSTNAME    = "localhost";
    private static final int PORT           = 5000;

    public static void main(String[] args) throws Exception{
        InetAddress serverAddress = null;

        serverAddress = InetAddress.getByName(HOSTNAME);

        DatagramSocket socket = new DatagramSocket();
        Member member = new Member("Kyungtae Park", 40, 85, 185);
        MemberEncoderBin encoder = new MemberEncoderBin();

        byte[] codedMember = encoder.encode(member);

        DatagramPacket packet = new DatagramPacket(codedMember, codedMember.length, serverAddress, PORT);
        socket.send(packet);

        socket.close();
    }
}
```

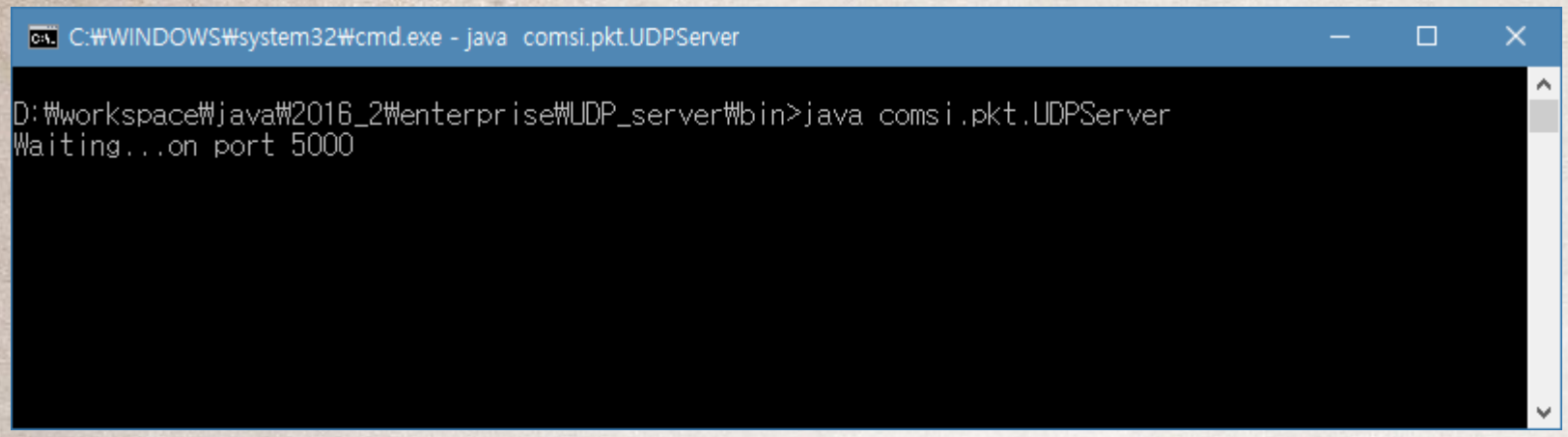
자바 객체 전송 - 수신 서버

UDPServer.java | UDPClient.java | Member.java | MemberEncoderBin.java | MemberDecoderBin.java

```java
package comsi.pkt;

import java.net.DatagramPacket;

public class UDPServer {

    private final static int MAX_LENGTH     = 255;
    private final static int PORT           = 5000;

    public static void main(String[] args) throws Exception{
        // TODO Auto-generated method stub
        DatagramSocket socket = new DatagramSocket(PORT);
        System.out.println("Waiting...on port "+ PORT);
        MemberDecoderBin decoder = new MemberDecoderBin();

        DatagramPacket packet = new DatagramPacket(new byte[MAX_LENGTH], MAX_LENGTH);

        socket.receive(packet);
        Member member = decoder.decode(packet);
        System.out.println(member);

        socket.close();
    }
}
```

실행

Server 실행 – 대기 상태



Server 실행 – 클라이언트 접속 후 받은 메시지