Chapter

11

네트워크 프로그래밍과 java.net 패키지

*** 학습목표**

- 창구 역할을 하는 인터넷 주소, 포트, 소켓 등에 대해서 학습한다.
- 두 노드 간에 연결이 설정된 후 데이터의 송수신이 발생하는 방식인 연결형(TCP/IP)에 대해서 학습한다.
- 두 노드 간에 연결이 설정되지 않고도 데이터의 송수신이 발생하는 방식인 비연결형(UDP/IP)에 대해서 학습한다.

Section

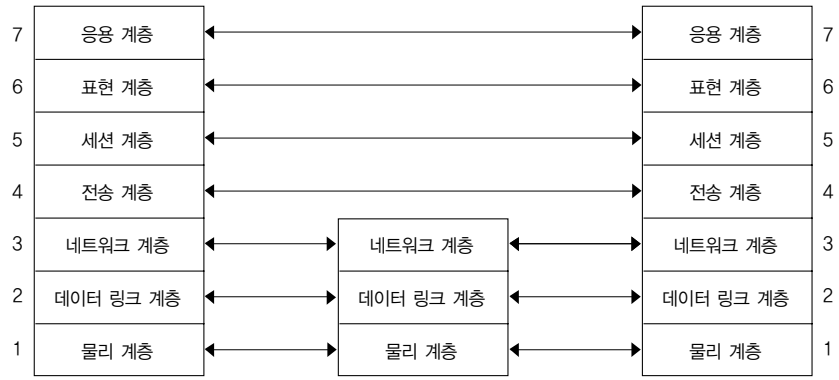
1 개요

이 장에서는 네트워킹 프로그램과 관련된 java.net 패키지에 대해서 설명한다. 자바에서의 통신은 창구 역할을 하는 인터넷 주소, 포트, 소켓 등을 통하여 이루어진다. 소켓을 이용하는 통신 방식을 소켓 통신이라고 하며, 소켓 통신에는 크게 연결형(TCP/IP)과 비연결형(UDP/IP)으로 구분된다. 연결형은 두 노드 간에 연결이 설정된 후에 데이터의 송수신이 발생하며, 비연결형은 두 노드 간에 연결이 설정되지 않고도 송수신이 발생하는 방식이다.

1. 네트워크의 계층 구조와 프로토콜

ISO의 OSI 7계층 프로토콜은 1977년 국제 표준화 기구(ISO, International Standards Organization) 위원회에서 제정한 표준 네트워크 구조를 위한 개방형 시스템 간 상호 접속(OSI, Open Systems Interconnection)에 관한 규정이다. 이 모델은 6년 이상 개발되어 1983년에 완성된 X.200으로 알려진 국제적 네트워크 표준이다.

OSI 7계층 프로토콜의 구성은 [그림 11-1]과 같이 7개의 계층으로 구성된다. 1계층은 물리 계층(Physical Layer), 2계층은 데이터 링크 계층(Data Link Layer), 3계층은 네트워크 계층(Network Layer), 4계층은 전송 계층(Transport Layer), 5계층은 세션 계층(Session Layer), 6계층은 표현 계층(Presentation Layer), 7계층은 응용 계층(Application Layer)이다. 1, 2, 3, 4계층인 하위 계층은 전달 기능이 있고, 5, 6, 7계층은 통신 기능이 있다. 즉, 하위 계층(전달 기능)에는 통신 기기 및 네트워크간의 통신 경로(path) 설정 및 유지, 해제 등에 대해서 규정하고, 상위 계층(통신 기능)에는 통신 경로를 이용한 정보 교환 등의 절차 등에 대해서 규정하고 있다.

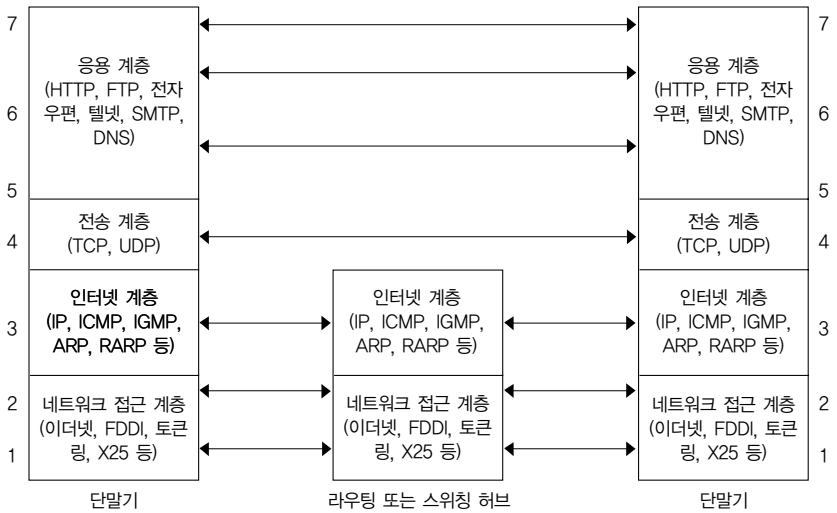


[그림 11-1] OSI 7계층 프로토콜의 구성

2. 인터넷과 프로토콜

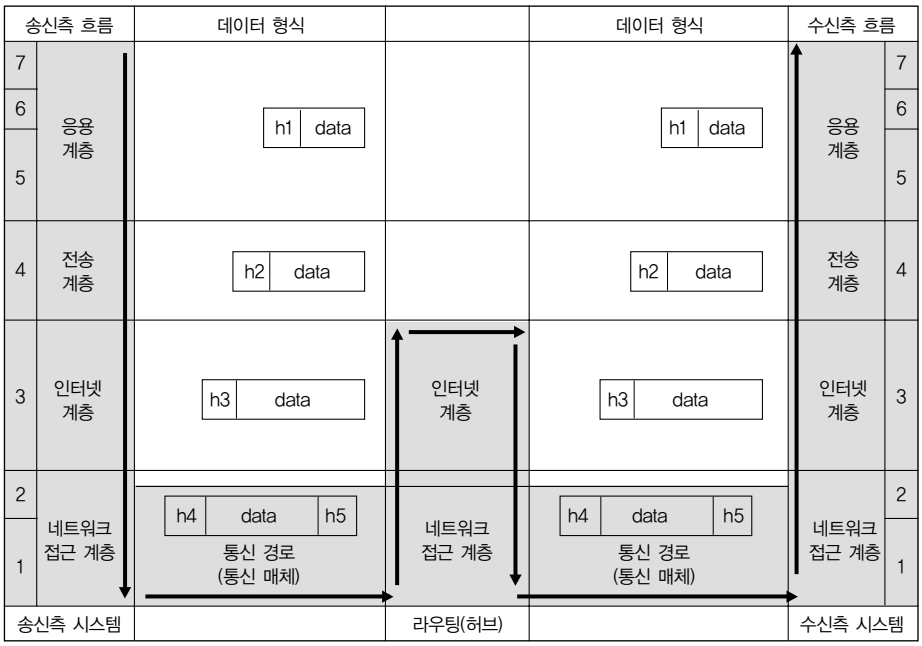
인터넷(Internet)은 전 세계에 널리 퍼져있는 근거리 통신망 또는 원거리 통신망들로 구성된 네트워크 중의 네트워크(Network of Networks)라고 할 수 있다.

인터넷의 기반이 되는 프로토콜은 TCP/IP와 UDP/IP가 있다. TCP/IP 기술은 가정과 대학, 학교, 기업 그리고 정부 연구 기관 등이 서로 연결된 전 세계적 규모의 인터넷을 구성하는 기반을 만들었다. 이 프로토콜의 계층 구조는 [그림 11-2]처럼 4계층으로 구성된다. 하위 계층에서 상위 계층으로 네트워크 접근 계층, 인터넷 계층, 전송 계층, 응용 계층이다. OSI 7계층과 비교하면 네트워크 접근 계층은 1, 2계층에 대응, 인터넷 계층은 3계층에 대응, 전송 계층은 4계층에 대응, 응용 계층은 5, 6, 7계층에 대응한다. 네트워크 접근 계층에는 이더넷, FDDI, 토큰링, X.25 등이 있으며, 인터넷 계층에는 IP, ICMP, IGMP, ARP, RARP 등이 있고, 전송 계층에는 TCP, UDP 등이 있다. 또한 응용 계층에는 HTTP, FTP, E-mail, Telnet, SMTP, DNS 등이 있다. 대부분의 응용 계층에는 클라이언트 및 서버(Client & Server) 모델 또는 P2P(Peer-to-Peer)로 구현된다. 클라이언트 및 서버 모델에서 클라이언트는 서버에게 작업 요청을 하는 장비나 프로세스이며, 서버는 클라이언트의 요청을 받아 작업을 처리하거나 결과를 클라이언트에게 통보하는 장비나 프로세스이다. P2P 모델에서는 각 시스템끼리 동등한 위치에서 서비스를 주고받는다.



[그림 11-2] 각 계층별 대표적인 프로토콜의 종류

인터넷상에서 데이터의 전송 과정은 [그림 11-3]과 같다. 송신측에서 최상위 계층(응용 계층)에서 어떤 프로세스에 의해서 발생된 데이터는 상위 계층에서 하위 계층으로 차례로 데이터가 전달된다. 이때 각각의 계층에서 전달 받은 데이터는 헤더(Header) 정보에 데이터를 추가하여 현재보다 한 단계 낮은 계층으로 전달한다. 한 단계 낮은 계층에서는 한 단계 높은 계층에서 전달 받은 데이터(헤더1+데이터)를 한 개의 데이터로 취급하고, 새로운 헤더(헤더2)를 데이터(헤더1+데이터)에 추가(헤더2+헤더1+데이터)하여 현재보다 한 단계 낮은 계층으로 전달한다. 이러한 과정을 거치면서 최하위 계층(네트워크 접근 계층)에 도달하게 된다. 송신측에 있는 최하위 계층의 데이터는 여러 개의 헤더가 씌어진 형태이며, 이러한 과정을 캡슐화(Encapsulation)라고 한다. 송신측 최하위 계층에서는 이러한 데이터를 전기 신호로 변환하여 전송 매체를 통하여 중계기(또는 교환기)를 거쳐서 수신측 최하위 계층인 네트워크 접근 계층으로 전송한다. 수신측 최하위 계층(네트워크 접근 계층)에서의 데이터는 여러 개의 헤더가 씌어진 형태이며, 각 계층의 헤더에 해당되는 부분을 벗겨내어 수신측의 최상위 계층으로 전달한다. 이러한 과정을 캡슐 해제(Decapsulation)라고 한다. 수신측의 최상위 계층(응용 계층)에서는 송신측의 최상위 계층에서 보냈던 원래의 데이터(헤더가 없는 데이터)가 정확하게 전달된다.



[그림 11-3] 인터넷에서 데이터의 전송 과정

3. 인터넷 주소, 포트와 소켓

인터넷 주소(Internet Adress)는 전화망에서 전화 연결을 하려면 발신지 및 수신지 전화 번호가 있어야 하고, 편지를 작성하여 보내려면 발신지 및 수신지 주소가 있어야 하는 것처럼 인터넷상에서의 시스템들은 각각의 주소를 반드시 갖고 있으며 그러한 주소를 IP 주소 또는 인터넷 주소라고 하며, 전 세계적으로 유일해야 한다. IP 주소 체계는 컴퓨터가 인식할 수 있는 주소 체계이며, 인터넷 주소 체계는 사람이 인식할 수 있는 주소 체계이다. IP 주소 또는 인터넷 주소는 인터넷 계층인 IP에서 정의되어 처리된다. IP 주소는 4개의 10진수와 .(점)으로 표현하며, 실제로는 32비트로 이루어진다. 각 10진수의 숫자는 0에서 255까지 사용 가능하며, 256개의 숫자인 8비트가 필요하다. 그러므로 전체적으로 32비트가 필요하다. 즉, 인터넷 주소는 하나의 컴퓨터에 할당된 번호이다. 예를 들면, 211.102.111.012이다.

포트

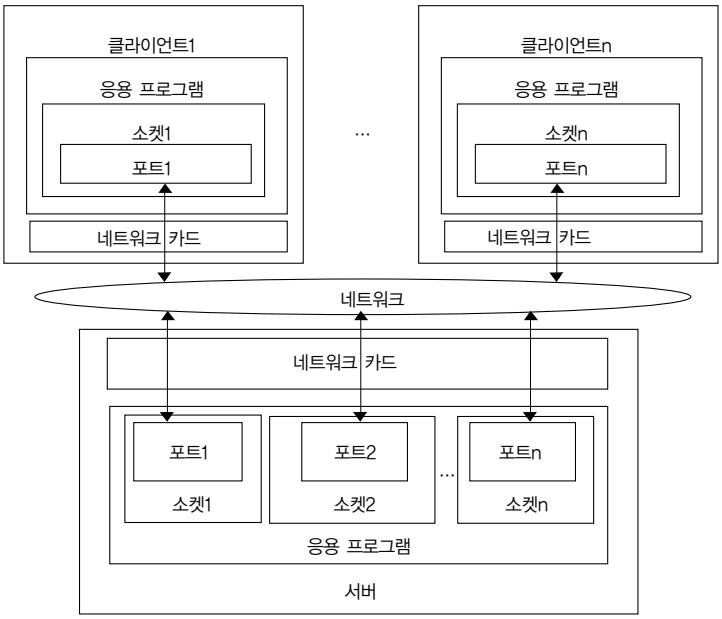
포트(Port)는 하드웨어가 아니라 소프트웨어적으로 접속이 가능한 연결 단자로 가상적인 채널의 단자 역할을 하는 것이다. 하나의 컴퓨터 안에 여러 개의 프로세스(혹은 프로그램)가 동시에 실행되면, 여러 개의 채널이 열려 있는 경우이다. 예를 들면, 서버에서 HTTP 서비스(80번), FTP 서비스(21번), Telnet 서비스(23번), SMTP tjqltm(25번), ECHO 서비스(7번) 등을 동시에 수행할 때 물리적 통신선은 하나이지만 가상의 채널인 포트를 이용하면 여러 서비스를 동시에 수행할 수 있다. 포트 번호는 TCP/IP 시스템에서 보통 16비트 크기를 가지며, 범위는 1~65,535이다. 그러나 0~1,024까지는 시스템에서 주로 사용하므로 사용하지 않고, 1025~65,535까지는 사용자가 사용 가능하다.

소켓

소켓(Socket)은 응용 프로그램 간에 정보 교환을 위한 매체로 접속의 끝 부분을 의미하며, 포트보다 고수준의 개념이다. 즉, 데이터를 송수신하는 창구 역할을 하는 것이다. 소켓 기능은 과거의 운영체제에서 지원하지 않았지만 요즘에는 운영체제 안에서 소켓 기능을 제공하는 경우가 많다. 예를 들면, 과거의 MS-Windows 3.1 등의 이하 버전은 소켓 기능을 제공하지 않았지만 MS-Windows 98/NT/XP 등에서는 소켓 기능을 제공한다.

소켓을 제공하는 운영체제에서 통신 프로그램을 하기 위해서는 소켓을 생성하도록 운영체제에 요청해야 하고, 시스템에서는 소켓을 생성하여 소켓을 식별하는 고유 번호 등을 반환한다. 그러므로 네트워크 관련 응용 프로그램을 작성하기 위해서는 고유 번호를 갖는 생성된 소켓에 데이터를 쓰거나 읽는다. 인터넷 관련 소켓에는 TCP(Transmission Control Protocol)와 UDP(User Datagram Protocol) 소켓이 있으며, 자바에서는 이 두 가지 소켓 관련 클래스들을 java.net 패키지 내에서 지원하고 있다. 소켓을 이용하는 통신 방식을 소켓 통신이라고 한다.

서버와 클라이언트 모델에서의 소켓과 포트의 관계는 [그림 11-4]와 같다.



[그림 11-4] 서버와 클라이언트 모델에서의 소켓과 포트의 관계

❹ 인터넷의 전송 계층에서의 대표적인 프로토콜의 종류

전송 계층에서의 역할은 포트를 사용한 두 개의 종단 호스트(End Hosts)간에 데이터를 전달하는 기능이다. 대표적인 프로토콜의 종류에는 TCP(Transmission Control Protocol)와 UDP(User Datagram Protocol)가 있다.

TCP

TCP는 두 개의 종단 간에 연결을 설정한 후에 데이터를 바이트 스트림으로 교환하는 연결형(Connection-Oriented) 프로토콜로 신뢰성이 있다.

UDP

UDP는 두 개의 종단 간에 연결을 설정하지 않고 데이터를 교환하는 비연결형(Connectionless) 프로토콜로 신뢰성이 떨어진다.

❺ 네트워크 프로그래밍과 java.net 패키지

네트워크 프로그래밍을 하기 위해서는 [표 11-1]처럼 java.net 패키지에 포함되어 있는 클래스를 이용해야 한다.

[표 11-1] java.net 패키지에 포함되어 있는 클래스

분류	클래스명	기능
소켓	DatagramPacket	UDP와 같은 비연결형 프로토콜에 사용되는 데이터 그램이다.
	DatagramSocket	데이터 그램의 전송과 수신을 위하여 사용된다.
	ServerSocket	TCP와 같은 연결형 프로토콜의 서버에서 사용된다.
	Socket	연결형 또는 비연결형 프로토콜의 서버와 클라이언트에서 사용된다.
	SocketImpl	소켓 구현을 위한 추상 클래스로 직접 사용할 수 없다.
	SocketImplFactory	SocketImpl 객체를 생성하기 위한 인터페이스이다.
호스트명 분리	InetAddress	호스트명과 인터넷 주소를 분리 및 인터넷 주소를 표현하는 데 사용된다.
URL	ContentHandler	MIME 객체를 생성하기 위한 콘텐츠 처리자의 추상 클래스를 직접 사용할 수 없다.
	ContentHandlerFactory	MIME형의 ContentHandler 객체를 생성하여 돌려주는 인터페이스로 직접 사용할 수 없다.
	URL	URL을 파싱하거나 URL의 호스트 연결을 생성하는 데 사용한다.
	URLConnection	URL 호스트의 연결을 생성하고 데이터를 처리하는 데 사용된다.
	URLEncoder	문자열을 x-www-form-urlencoded 형식으로 부호화하기 위하여 사용된다.
	URLStreamHandler	URL의 프로토콜(HTTP, FTP 등)을 처리하기 위한 추상 클래스이다.
	URLStreamHandlerFactory	URLStreamHandler의 인스턴스를 생성하기 위한 인터페이스로 직접 사용될 수 없다.
Exception (예외 처리)	MalformedURLException	URL 생성을 위한 인자의 구문이 잘못되면 예외 처리 된다.
	ProtocolException	잘못된 형의 소켓에 연결하고자 하는 경우에 예외 처리 된다.
	SocketException	제공하지 않는 소켓을 사용하고자 시도하거나 이미 다른 프로세스가 열어 둔 소켓을 연결하고자 하는 경우에 예외 처리 된다.
	UnknownHostException	호스트명이 인터넷 주소를 해석할 수 없는 경우에 예외 처리 된다.
	UnknownServiceException	URL 연결에 의하여 제공되지 않는 서비스를 사용하고자 시도할 때 예외 처리 된다.

Section

2 연결형 소켓 클래스와 TCP 프로토콜

1 연결형 소켓 클래스의 메소드

연결형 소켓 클래스와 관련된 메소드는 [표 11-2]와 같다.

[표 11-2] 연결형 소켓 클래스 관련 메소드

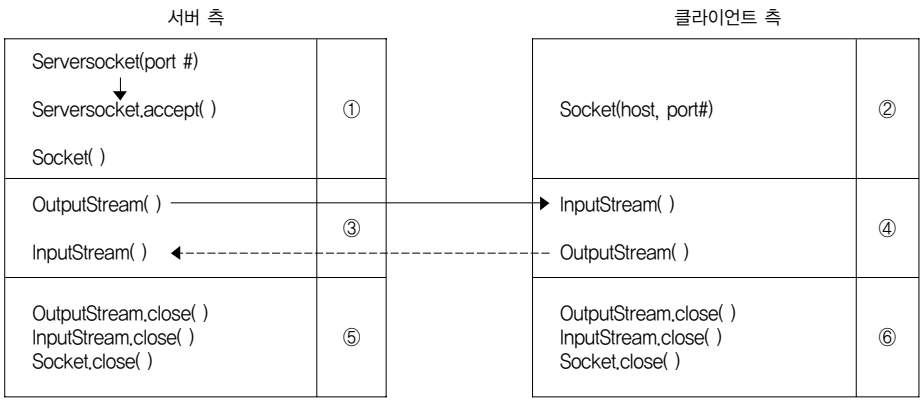
클래스명	메소드명	기능
ServerSocket	ServerSocket( )	명시한 로컬 포트와 결합된 서버 소켓을 생성한다.
	accept( )	서버 소켓의 연결을 가져온다.
	close( )	서버 소켓을 닫는다.
	getInetAddress( )	서버 소켓이 생성된 컴퓨터의 주소를 가져온다.
	getLocalPort( )	서버 소켓의 로컬 포트 번호를 읽어온다.
Socket	Socket( )	명시한 특정 목적지에 대한 소켓을 생성한다.
	close( )	소켓을 닫는다.
	getInputStream( )	소켓에 대한 입력 스트림을 생성한다.
	getOutputStream( )	소켓에 대한 출력 스트림을 생성한다.
	getInetAddress( )	소켓이 연결된 원격 호스트의 주소를 읽어온다.
	getLocalPort( )	소켓의 로컬 포트 번호를 읽어온다.
	getPort( )	소켓으로 데이터를 송수신하는 원격 포트 번호를 읽어온다.

2 TCP 프로토콜의 동작 원리

TCP 프로토콜의 동작 원리는 [그림 11-5]와 같다.

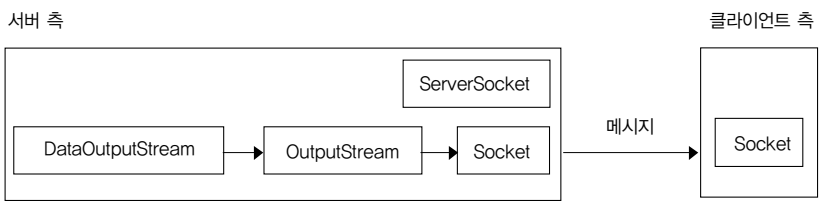
① 서버 측에서는 먼저 java.net 패키지에 포함되어 있는 Serversocket 클래스를 이용해 포트 번호를 할당한다. 클라이언트 측으로부터의 새로운 연결을 얻기 위하여 accept( ) 메소드를 호출하여 기다린다.

② 클라이언트가 소켓 객체를 생성하여 연결을 요청하면 서버 측에서는 accept( ) 메소드를 이용해 소켓을 열어주고, 클라이언트는 서버 주소의 포트 번호로 연결한다.
③ 서버 측에서는 클라이언트 측과 정보를 송수신하기 위하여 스트림을 사용한다. 상대 측에게 보낼 때는 출력 스트림인 OutputStream( )이 사용되며, 상대 측으로부터 받을 때는 입력 스트림인 InputStream( )이 사용된다.
④ 클라이언트 측에서는 서버 측과 정보를 송수신하기 위하여 스트림을 사용한다. 상대 측에게 보낼 때는 출력 스트림인 OutputStream( )이 사용되며, 상대 측으로부터 받을 때는 입력 스트림인 InputStream( )이 사용된다.
⑤ 서버 측에서는 출력 스트림, 입력 스트림과 서버 소켓을 종료하기 위하여 닫는다.
⑥ 클라이언트 측에서는 출력 스트림, 입력 스트림과 소켓을 종료하기 위하여 닫는다.



[그림 11-5] TCP 프로토콜의 동작 원리

TCP 프로토콜의 동작 원리에서 서버 측에서 클라이언트 측으로 문자열을 보내는 과정은 [그림 11-6]과 같다.



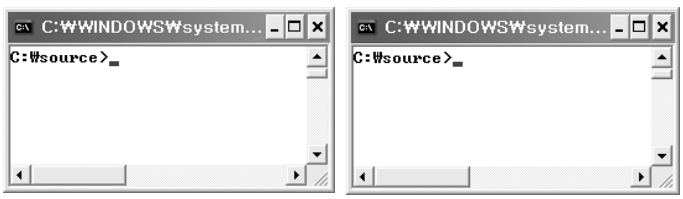
[그림 11-6] 서버 측에서 클라이언트 측으로 문자열을 보내는 과정

실습하기[11-1] 연결형(TCP/IP)을 이용해 서버와 클라이언트측 간에 상호 통신

소켓 통신에는 크게 연결형(TCP/IP)과 비연결형(UDP/IP)으로 구분된다. 여기서는 연결형을 이용해 서버와 클라이언트 측간에 상호 통신이 가능한 애플리케이션(콘솔) 프로그램을 작성하시오.

서버 측 프로그램을 먼저 실행 시킨 후에 '클라이언트 대기중 ....' 메시지를 자체적으로 출력하면서 기다리고 있다가 클라이언트 측의 접속 요청과 클라이언트 측에서부터 서버 측으로 문자 메시지 '<전송 시작>I love JEJUDO!(client → server)<전송 마침>' 를 전송하면, '클라이언트 접속 성공!' 을 자체적으로 출력하고 서버 측은 클라이언트 측으로부터 수신 받은 문자 메시지 '<전송 시작>I love JEJUDO!(client → server)<전송 마침>' 을 서버 측 화면에 출력한다. 또한, 클라이언트 측은 서버 측으로부터 수신 받은 문자 메시지 'Welcom to connect to TCP Server!(server → client)' 를 클라이언트 측 화면에 출력한다. 실행 방법은 다음과 같다.

1 C:\source 폴더에서 실행한다고 가정하고, 두 개의 콘솔 창을 각각 띄운다.

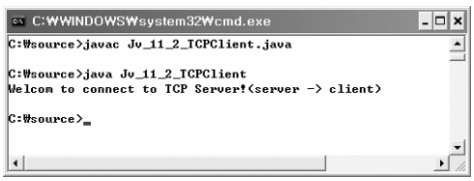


[그림 11-7] 두 개의 콘솔 창

2 먼저, 서버 측 프로그램을 실행시킨다(강제 종료 시에는 Ctrl+C 키를 누른다).

[그림 11-8] 서버 측 프로그램의 실행

3 클라이언트 측 프로그램을 실행시킨다. 그러면 다음과 같은 화면이 나타난다.



[그림 11-9] 클라이언트 측 프로그램의 실행

4 서버 측에 다음과 같은 결과가 출력된다.

[그림 11-10] 서버 측 결과 화면

[예제 11-1] Jv_11_1_TCPServer.java

```
01  import java.net.*;
02  import java.io.*;
03
04  class Jv_11_1_TCPServer {
05    public static void main(String args[]) {
06     ServerSocket s1 = null;
07     Socket s2;
08     OutputStream os1;
09     DataOutputStream os2;
10     InputStream is1;
11     DataInputStream is2;
12
13     try {
14        s1 = new ServerSocket(5432);
15         } catch (IOException e) {e.printStackTrace();}
16     while(true) {
17        try {
18             System.out.println("클라이언트 대기중 ....");
19             s2 = s1.accept();
20             System.out.println("클라이언트 접송 성공!");
```

```
21
22              os1 = s2.getOutputStream();
23              os2 = new DataOutputStream(os1);
24              os2.writeUTF("Welcom to connect to TCP Server!(server ->
                client)");
25
26              is1 = s2.getInputStream();
27              is2 = new DataInputStream(is1);
28              String st = new String(is2.readUTF());
29              System.out.println(st);
30
31              is1.close();
32              is2.close();
33              os1.close();
34              os2.close();
35              s2.close();
36          } catch(IOException e) {e.printStackTrace();}
37      }
38
39    }
40  }
```

01행: 네트워크 관련 클래스를 이용하기 위하여 java.net 패키지를 import한다.

06행: 자바에서 통신을 하려면 소켓을 만들어야 하는 데, 여기에서는 초기값이 null인 ServerSocket 객체 s1을 선언한다. 이러한 소켓을 이용하는 통신 방식을 소켓 통신이라고 한다. 소켓 통신에는 크게 연결형과 비연결형으로 구분된다. 연결형은 두 노드 간에 연결이 설정된 후에 데이터의 송수신이 발생하며, 비연결형은 두 노드 간에 연결이 설정되지 않고도 송수신이 발생한다.

14행: 5432 포트 번호를 이용해 ServerSocket 객체 s1을 생성한다. 포트 번호는 16비트의 크기를 가지며, 그 범위는 0~65,535이지만 1023번 이하의 포트 번호는 시스템이 미리 지정된 서비스용(HTTP : 80, FTP : 21 등)으로 사용하기 때문에 1,023번 이하는 사용하지 않는 것이 바람직하다.

07행: Socket 객체 s2를 선언한다.

19행: s1에 대한 accept( ) 메소드를 호출하여 클라이언트와 연결된 소켓을 s2에 할당한다. s1.accept( )의 의미는 클라이언트가 서버에 접속할 때까지 프로그램을 잠시 멈추고 기다린다는 뜻이다. 이렇게 잠시 멈추는 것을 블록(Block) 또는 블록킹(Blocking)이라고 한다. 즉, 새로운 소켓이 생성되어 s2라는 참조 변수에 저장한다.

여기에서는 소켓 통신에서 연결형에 해당되는 부분이다. 두 노드 간에 연결이 설정될 때까지 데이터의 송수신이 이루어지지 않고 기다린다.

16행: 무한 반복을 하는데, 클라이언트의 계속적인 요청을 받아들이기 위한 것이다. 여기에서 서버 프로그램은 무한 반복을 하면서 계속 수행이 된다.

08행: OutputStream os1을 선언한다. 이 객체는 서버 측에서 클라이언트 측으로 응답을 보내기 위한 객체이다.

09행: DataOutputStream 객체 os2를 선언한다. 이 객체는 데이터를 바이트 스트림으로 서버 측에서 클라이언트 측으로 출력하기 위한 객체이다.

22행: s2에 대한 OutputStream 객체 os1을 생성한다.

23행: os1을 매개 변수로하여 os2를 생성한다.

24행: 서버 측에서 클라이언트 측으로 응답을 보내기 위하여 os2 객체에 writeUTF( ) 메소드를 이용해 매개 변수로 입력된 문자들 "Welcom to connect to TCP Server!(server → client)"를 스트림으로 출력한다.

10행: InputStream 객체 is1을 선언한다. 이 객체는 요청 메시지를 받기 위한 객체이다.

11행: DataInputStream 객체 is2를 선언한다. 이 객체는 데이터를 바이트 스트림으로 입력받기 위한 객체이다.

26행: s2에 대한 InputStream 객체 is1을 생성한다.

27행: is1을 매개 변수로 하여 is2를 생성한다.

28행: 클라이언트 측에서 서버 측으로 보내 문자 메시지를 받기 위하여 is2 객체에 readUTF( ) 메소드를 이용해 변수 s에 저장한다.

29행: 변수 s에 저장된 문자 메시지를 콘솔 화면에 출력한다.

31-35행: 클래스 InputStream, DataInputStream, OutputStream, DataOutStream, Socket에 의해서 생성된 각각의 객체 is1, is2, os1, os2, s2를 종료한다.

[예제 11-2] Jv_11_2_TCPClient.java

```
01  import java.net.*;
02  import java.io.*;
03
04  class Jv_11_2_TCPClient {
05    public static void main(String args[]) {
06    try {
07     Socket s1;
08     InputStream is1;
09     DataInputStream is2;
10     OutputStream os1;
11     DataOutputStream os2;
```

```
12
13      String sendString = "I love JEJUDO!(client -> server)";
14      s1 = new Socket("127.0.0.1", 5432);
15      is1 = s1.getInputStream();
16      is2 = new DataInputStream(is1);
17      String st = new String(is2.readUTF());
18      System.out.println(st);
19
20      os1 = s1.getOutputStream();
21      os2 = new DataOutputStream(os1);
22      os2.writeUTF("<전송 시작>" + sendString + "<전송 마침>");
23
24      os2.close();
25      os1.close();
26      is2.close();
27      is2.close();
28      s1.close();
29      } catch(ConnectException connExc) {
30         System.err.println("서버 연결 실패");
31        }
32        catch(IOException e) {e.printStackTrace();}
33
34    }
35  }
```

01행: 네트워크 관련 클래스를 이용하기 위하여 java.net 패키지를 import한다.

07행: Socket 객체 s1을 선언한다.

14행: 5432 포트 번호를 이용해 Socket 객체 s1을 생성한다. 포트 번호는 16비트의 크기를 가지며, 그 범위는 0~65,535이지만 1,023번 이하의 포트 번호는 시스템이 미리 지정된 서비스용(HTTP : 80, FTP : 21 등)으로 사용하기 때문에 1,023 번 이하는 사용하지 않는 것이 바람직하다. 호스트 주소를 "127.0.0.1"로 사용하는 이유는 자신의 컴퓨터에서 송수신을 하기 위함이다.

08행: InputStream 객체 is1을 선언한다. 이 객체는 요청 메시지를 받기 위한 객체이다.

09행: DataInputStream 객체 is2를 선언한다. 이 객체는 데이터를 바이트 스트림으로 입력받기 위한 객체이다.

15행: s2에 대한 InputStream 객체 is1을 생성한다.

16행: is1을 매개 변수로 하여 is2를 생성한다.

17행: 서버 측에서 클라이언트 측으로 보내 문자 메시지를 받기 위하여 is2 객체에 readUTF( ) 메소드를 이용해 변수 st에 저장한다.

18행: 변수 s에 저장된 문자 메시지를 콘솔 화면에 출력한다.

10행: OutputStream os1을 선언한다. 이 객체는 클라이언트 측에서 서버 측으로 응답을 보내기 위한 객체이다.

11행: DataOutputStream 객체 os2를 선언한다. 이 객체는 데이터를 바이트 스트림으로 클라이언트 측에서 서버 측으로 출력하기 위한 객체이다.

20행: s2에 대한 OutputStream 객체 os1을 생성한다.

21행: os1을 매개 변수로하여 os2를 생성한다.

22행: 클라이언트 측에서 서버 측으로 응답을 보내기 위하여 os2 객체에 writeUTF( ) 메소드를 이용해 매개변수로 입력된 문자들 "〈전송 시작〉 I love JEJUDO!(client → server)sendString 〈전송 마침〉"을 스트림으로 출력한다.

24~28행: 클래스 InputStream, DataInputStream, OutputStream, DataOutStream, Socket에 의해서 생성된 각각의 객체 is1, is2, os1, os2, s1을 종료한다.

Section

3 비연결형 소켓 클래스와 UDP 프로토콜

1 비연결형 소켓 클래스의 메소드

비연결형 소켓 클래스와 관련된 메소드는 [표 11-3]과 같다.

[표 11-3] 비연결형 소켓 클래스 관련 메소드

클래스명	메소드명	기능
DatagramSocket	DatagramSocket( )	데이터 그램 소켓을 생성한다.
	close( )	데이터 그램 소켓을 종료한다.
	receive( )	소켓을 사용해 데이터 그램을 읽어온다.
	send( )	소켓을 사용해 목적지로 데이터 그램을 전송한다.
	getLocalPort( )	소켓이 결합된 로컬 포트를 읽어온다.
DatagramPacket	DatagramPacket( )	송수신할 데이터를 위한 데이터 그램 패킷의 인스턴스를 생성한다.
	getAddress( )	패킷의 목적지 주소를 얻는다.
	getData( )	패킷의 데이터 내용을 얻는다.
	getLength( )	패킷의 길이를 얻는다.
	getPort( )	패킷 목적지의 포트를 얻는다.

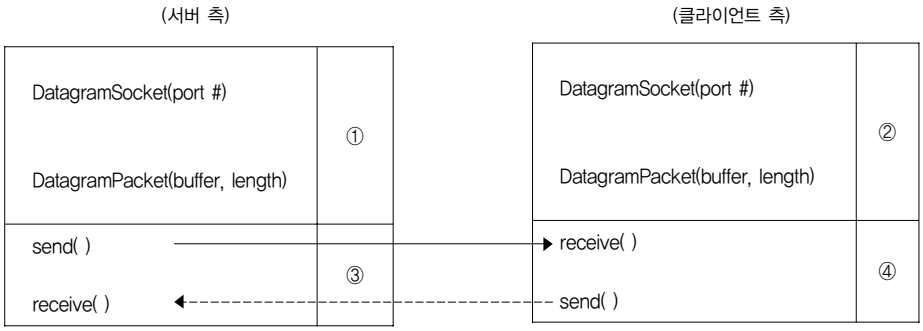
2 UDP 프로토콜의 동작 원리

UDP 프로토콜의 동작 원리는 [그림 11-11]과 같다.

① 서버 측에서는 먼저 java.net 패키지에 포함되어 있는 DatagramSocket, DatagramPacket 클래스를 생성한다. TCP처럼 클라이언트 측으로부터의 새로운 연결을 얻기 위한 accept( ) 메소드 사용 등은 필요가 없다.

② 클라이언트 측에서도 java.net 패키지에 포함되어 있는 DatagramSocket, DatagramPacket 클래스를 생성한다.

③ 서버 측에서는 클라이언트 측과 정보를 송수신하기 위하여 상대 측에게 보낼 때는 send( ) 메소드를 사용하고, 상대 측으로부터 받을 때는 receive( ) 메소드를 사용한다.
④ 클라이언트 측에서는 서버 측과 정보를 송수신하기 위하여 상대측에 보낼 때는 send( ) 메소드를 사용하고, 상대 측으로부터 받을 때는 receive( ) 메소드를 사용한다.

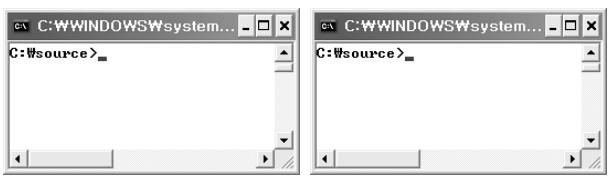


[그림 11-11] UDP 프로토콜의 동작 원리

실습하기[11-2] 비연결형(UDP/IP)을 이용해 클라이언트 측에서 서버 측으로 문자를 송신

소켓 통신에는 크게 연결형과 비연결형으로 구분된다. 여기에서는 비연결형을 이용해 클라이언트 측에서 서버 측으로 문자를 송신 가능한 애플리케이션(콘솔) 프로그램을 작성하시오. 서버 측 프로그램을 먼저 실행 시킨 후에 서버 측에서 기다리고 있다가 클라이언트 측의 실행 시에 보내려는 문자 메시지를 서버 측은 클라이언트 측으로부터 수신 받은 문자 메시지를 서버 측 화면에 출력한다. 실행 방법은 다음과 같다.

1 C:\source 폴더에서 실행한다고 가정하고, 두 개의 콘솔 창을 각각 띄운다.

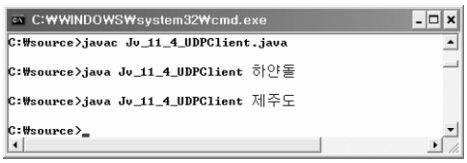


[그림 11-12] 두 개의 콘솔 창

2 먼저 서버 측 프로그램을 실행시킨다(강제 종료 시에는 Ctrl+C 키를 누른다).

[그림 11-13] 서버 측 프로그램의 실행

3 클라이언트 측 프로그램을 실행시킨다.



[그림 11-14] 클라이어트 측 프로그램의 실행

4 서버 측 화면에 다음과 같은 결과가 출력된다.

[그림 11-15] 결과 화면

[예제 11-3] Jv_11_3_UDPServer.java

```
01  import java.net.*;
02  import java.io.*;
03  import java.lang.*;
04
05  public class Jv_11_3_UDPServer {
06    public static void main(String args[]) {
07     try {
08         DatagramSocket ds = new DatagramSocket(9999);
09         byte[] bf = new byte[30];
10         DatagramPacket dp = new DatagramPacket(bf, bf.length);
11
12     while(true) {
13        try {
```

```
14          ds.receive(dp);
15          String rs1 = new String(dp.getData());
16          String rs2 = new String(rs1.trim());
17          System.out.println("주소:" + dp.getAddress() + "  포트번호:" +
            dp.getPort());
18          System.out.println("수신된 메시지(Client->Server): " + rs2);
19         } catch(IOException e){}
20       }
21      } catch(IOException e){}
22     }
23  }
```

01행: 네트워크 관련 클래스를 이용하기 위하여 java.net 패키지를 import한다.

08행: 자바에서 통신을 하려면 소켓을 만들어야 하는 데 이러한 소켓을 이용하는 통신 방식을 소켓 통신이라고 한다. 소켓 통신에는 크게 연결형과 비연결형으로 구분된다. 여기에서의 비연결형은 두 노드 간에 연결이 설정되지 않고도 송수신이 발생하는 방식이다. 9999 포트 번호를 이용해 데이터 그램 소켓(DatagramSocket) 객체 ds를 생성한다. 포트 번호는 16비트의 크기를 가지며, 범위는 0~65,535이지만 1,023번 이하의 포트 번호는 시스템이 미리 지정된 서비스용(HTTP : 80, FTP : 21 등)으로 사용하기 때문에 1,023번 이하는 사용하지 않는 것이 바람직하다.

09행: 데이터 그램 소켓으로부터 데이터를 받기 위하여 크기가 30인 바이트 배열을 선언한다.

10행: 수신된 패킷을 담을 데이터 그램 패킷(DatagramPacket) 객체 dp를 생성한다.

12행: 무한 반복을 하는데 클라이언트의 계속적인 요청을 받아들이기 위한 것이다. 여기에서 서버 프로그램은 무한 반복을 하면서 계속 수행된다.

14행: 클라이언트 측에서 서버 측으로 보내 문자 메시지를 받기 위하여 receive 메소드를 이용해 데이터 그램 소켓을 통하여 패킷을 받는다.

15행: 받은 패킷의 바이트 배열을 스트링으로 변환한다.

16행: trim( ) 메소드를 이용해 스트링의 공백을 없앤다.

17행: 패킷의 목적지 주소와 포트 번호를 얻는다.

18행: 변수 rs2에 저장된 문자 메시지를 콘솔 화면에 출력한다.

[예제 11-4] Jv_11_4_UDPClient.java

```
01  import java.net.*;
02  import java.io.*;
03  public class Jv_11_4_UDPClient {
04    public static void main(String args[]) {
05     try {
06           DatagramSocket ds = new DatagramSocket();
07           InetAddress ia = InetAddress.getByName("localhost");
08           byte[] bf = args[0].getBytes();
09           DatagramPacket dp = new DatagramPacket(bf, bf.length, ia,
             9999);
10
11           ds.send(dp);
12
13         } catch(Exception e){
14           System.out.println(e);
15         }
16    }
17  }
```

01행: 네트워크 관련 클래스를 이용하기 위하여 java.net 패키지를 import한다.

06행: 자바에서 통신을 하려면 소켓을 만들어야 하는 데 이러한 소켓을 이용하는 통신 방식을 소켓 통신이라고 한다. 여기에서의 비연결형은 두 노드 간에 연결이 설정되지 않고도 송수신이 발생하는 방식이다. 데이터 그램 소켓(DatagramSocket) 객체 ds를 생성한다.

07행: InetAddress 클래스의 getByName( ) 메소드는 호스트의 인터넷 주소를 얻기 위하여 사용된다. 여기에서는 "localhost"를 사용하므로 자기 자신의 컴퓨터를 의미한다.

08행: 실행 시에 명령 라인으로부터 입력 받은 문자열을 바이트 배열로 변환한다.

09행: 수신된 패킷을 담을 데이터 그램 패킷(DatagramPacket) 객체 dp를 생성한다. 이때 매개 변수로는 데이터, 데이터의 길이, 연결할 서버 주소, 포트 번호를 사용한다. 여기에서는 9999번을 사용하는데 이 번호는 서버의 포트 번호와 일치시켜야 한다. 즉, 9999 포트 번호를 이용해 데이터 그램 패킷 객체 dp를 생성한다.

11행: 클라이언트 측에서 서버 측으로 문자 메시지를 보내기 위하여 send 메소드를 이용해 데이터 그램 소켓을 통하여 패킷을 보낸다.

Section

04 인터넷 주소와 포트 관련 클래스

1. InetAddress 클래스

InetAddress 클래스의 메소드는 [표 11-4]와 같다.

[표 11-4] InetAddress 클래스의 메소드

클래스명	메소드명	기능
InetAddress	getAllByName( )	호스트가 갖고 있는 모든 인터넷 주소를 얻는다.
	getByName( )	호스트의 인터넷 주소를 얻는다.
	getLocalHost( )	현재 사용 중인 지역 호스트에 대한 InetAddress 객체를 얻는다.
	getAddress( )	네트워크 상의 IP 주소를 byte 배열로 얻는다.
	getHostAddress( )	IP 주소의 점 문자열 형식을 가져온다. 예 127.0.0.1
	getHostName( )	InetAddress 객체의 호스트명을 얻는다.

2. URL 클래스

URL 클래스의 생성자는 [표 11-5]와 같으며, URL 클래스의 메소드는 [표 11-6]과 같다.

[표 11-5] URL 클래스의 생성자

클래스명	메소드명	기능
URL	URL(String spc)	매개 변수 spec으로 URL 객체를 생성한다.
	URL(String protocol, String host, int port, String file)	protocol과 host, port와 file로 URL 객체를 생성한다.
	URL(String protocol, String host, String file)	protocol과 host, file로 URL 객체를 생성한다.

[표 11-6] URL 클래스의 메소드

클래스명	메소드명	기능
URL	getAuthority( )	URL의 호스트명과 포트를 결합한 문자열을 얻는다.
	getPort( )	URL의 포트 번호를 얻는다.
	getDefaultPort( )	프로토콜의 default 포트 번호를 얻는다.
	getFile( )	URL의 파일명을 얻는다.
	getHost( )	URL의 호스트 컴퓨터명을 얻는다.
	getPath( )	URL의 경로를 얻는다.
	getProtocol( )	URL의 프로토콜을 얻는다.
	getQuery( )	URL의 쿼리를 문자열로 얻는다.
	getRef( )	URL의 참조를 문자열로 얻는다.
	openConnection( )	URLConnection 객체를 생성해 준다.
	openStream( )	InputStream의 객체를 생성해 준다.
	toExternalForm( )	URL을 문자열로 얻는다.

❸ URConnection 클래스

URLConnection 클래스의 생성자는 [표 11-7]과 같으며, URLConnection 클래스의 메소드는 [표 11-8]과 같다.

[표 11-7] URLConnection 클래스의 생성자

클래스명	메소드명	기능
URLConnection	URLConnection(URL url)	매개 변수 url으로 URL 객체를 생성한다.

[표 11-8] URLConnection 클래스의 메소드

클래스명	메소드명	기능
URLConnection	getContentEncoding( )	헤더 필드의 content-encoding에 대한 값을 얻는다.
	getContentLength( )	헤더 필드의 content-length에 대한 값을 얻는다.
	getHeaderField(String name)	헤더 필드의 이름에 대한 값을 얻는다.
	getHeaderFields( )	헤더 필드의 구조를 map으로 변환한다.
	getInputStream( )	입력하기 위하여 InputStream 객체를 얻는다.
	getOutputStream( )	출력하기 위하여 OutputStream 객체를 얻는다.
	getURL( )	URL 필드의 값을 얻는다.

실습하기[11-3] URL 클래스, URLConnection 클래스, InetAddress 클래스의 메소드를 이용

URL 클래스의 메소드, URLConnection 클래스의 메소드, InetAddress 클래스의 메소드를 이용해 인터넷 주소와 포트 등 관련 정보를 얻을 수 있는 애플리케이션(콘솔) 프로그램을 작성하시오. 각 컴퓨터 및 통신 환경에 따라 결과가 다르게 나타날 수 있다. 만약, C:\source 폴더에서 실행한다고 가정하면 한 개의 콘솔 창을 띄운 다음 프로그램을 실행시킨다.

[그림 11-16] 인터넷 주소와 포트 등 관련 정보 출력

[예제 11-5] Jv_11_5.java

```
01  import java.net.*;
02
03  class Jv_11_5 {
04    public static void main(String args[])
05            throws Exception, MalformedURLException {
06     URL u = new URL(args[0]);
07     System.out.println("=== URL 클래스의 메소드 ===");
08     System.out.println("프로토콜:" + u.getProtocol());
09     System.out.println("포트: " + u.getPort());
10     System.out.println("호스트: " + u.getHost());
11     System.out.println("파일(경로포함): " + u.getFile());
12     System.out.println("전체URL: " + u.toExternalForm());
13
14     System.out.println("\n=== URLConnection 클래스의 메소드 ===");
```

```
15        URLConnection uc = u.openConnection();
16        int len = uc.getContentLength();
17        System.out.println("문서의 길이: " + len + "바이트");
18        URL uu = uc.getURL();
19        System.out.println("URL: " + uu);
20
21      System.out.println("\n=== InetAddress 클래스의 메소드 ===");
22      InetAddress addr = null;
23      addr = InetAddress.getByName("www.bu.ac.kr");
24      System.out.println("컴퓨터의 이름과 IP주소");
25      System.out.println(addr);
26      InetAddress Na[] = InetAddress.getAllByName("www.yonsei.ac.kr");
27      System.out.println("www.yonsei.ac.kr에 대한 정보");
28      for (int i=0; i<Na.length; i++)
29          System.out.println(Na[i]);
30
31       addr = InetAddress.getLocalHost();
32       System.out.println("로컬 컴퓨터의 이름: " + addr.getHostName());
33       byte ip[] = addr.getAddress();
34       System.out.println("로컬 컴퓨터의 IP 주소 : ");
35       for (int i=0; i<ip.length; i++) {
36            if ( i > 0) {
37                     System.out.print(".");
38            }
39            System.out.print(ip[i] & 0xff);
40        }
41        System.out.println("\n로컬 컴퓨터의 IP 주소 : " + addr.getHost
          Address());
42    }
43  }
```

06행: 명령 라인으로부터 문자열을 입력 받아 URL 객체 u를 생성한다.

08~12행: URL 클래스 관련 정보를 얻어 출력한다.

15행: URLConnection 객체 uc를 생성한다.

16~19행: URLConnection 클래스 관련 정보를 얻어 출력한다.

22행: InetAddress 클래스 객체 addr를 생성시켜 null로 초기화 한다.

23~41행: InetAddress 클래스 관련 정보를 얻어 출력한다.

요약

1 포트

포트(Port)는 하드웨어가 아니라 소프트웨어적으로 접속이 가능한 연결 단자로 가상적인 채널의 단자 역할을 하는 것이다. 하나의 컴퓨터 안에 여러 개의 프로세스(혹은 프로그램)가 동시에 실행되면, 여러 개의 채널이 열려 있는 경우이다.

2 소켓

소켓(Socket)은 응용 프로그램 간에 정보 교환을 위한 매체로 접속의 끝 부분을 의미하며, 포트보다 고수준의 개념이다. 즉, 데이터를 송수신하는 창구 역할을 하는 것이다. 소켓 기능은 과거의 운영체제에서 지원하지 않았지만 요즘에는 운영체제 안에서 소켓 기능을 제공하는 경우가 많다. 예를 들면, 과거의 MS-Windows 3.1 등의 이하 버전은 소켓 기능을 제공하지 않았지만, MS-Windows 98/NT/XP 등에서는 소켓 기능을 제공한다.

3 인터넷의 전송 계층에서 대표적인 프로토콜의 종류

전송 계층에서의 역할은 포트를 사용한 두 개의 종단 호스트(End Hosts) 간에 데이터를 전달하는 기능이다. 대표적인 프로토콜의 종류에는 TCP가 있다.

4 TCP

TCP(Transmission Control Protocol)는 두 개의 종단 간에 연결을 설정한 후에 데이터를 바이트 스트림으로 교환하는 연결형(Connection-Oriented) 프로토콜로 신뢰성이 있다.

5 UDP

UDP(User Datagram Protocol)는 두 개의 종단 간에 연결을 설정하지 않고 데이터를 교환하는 비연결형(Connectionless) 프로토콜로 신뢰성이 떨어진다.

6 네트워크 프로그래밍과 java.net 패키지

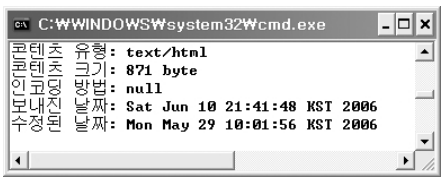
네트워크 프로그래밍을 하기 위해서는 다음 표와 같이 java.net 패키지에 포함되어 있는 클래스들을 이용하는 것이 필요하다.

요약

분류	클래스명	기능
소켓	DatagramPacket	UDP와 같은 비연결형 프로토콜에 사용되는 데이터 그램이다.
	DatagramSocket	데이터 그램의 전송과 수신을 위하여 사용된다.
	ServerSocket	TCP와 같은 연결형 프로토콜의 서버에서 사용된다.
	Socket	연결형 또는 비연결형 프로토콜의 서버와 클라이언트에서 사용된다.
	SocketImpl	소켓 구현을 위한 추상 클래스로 직접 사용할 수 없다.
	SocketImplFactory	SocketImpl 객체를 생성하기 위한 인터페이스이다.
호스트명 분리	InetAddress	호스트명과 인터넷 주소를 분리 및 인터넷 주소를 표현하는 데 사용된다.
URL	ContentHandler	MIME 객체를 생성하기 위한 콘텐츠 처리자의 추상 클래스를 직접 사용할 수 없다.
	ContentHandlerFactory	MIME 형의 ContentHandler 객체를 생성하여 돌려주는 인터페이스로 직접 사용할 수 없다.
	URL	URL을 파싱하거나 URL의 호스트 연결을 생성하는 데 사용한다.
	URLConnection	URL 호스트의 연결을 생성하고 데이터를 처리하는 데 사용된다.
	URLEncoder	문자열을 x-www-form-urlencoded 형식으로 부호화하기 위하여 사용된다.
	URLStreamHandler	URL의 프로토콜(HTTP, FTP 등)을 처리하기 위한 추상 클래스이다.
	URLStreamHandlerFactory	URLStreamHandler의 인스턴스를 생성하기 위한 인터페이스로 직접 사용될 수 없다.
Exception (예외 처리)	MalformedURLException	URL 생성을 위한 인자의 구문이 잘못되면 예외처리 된다.
	ProtocolException	잘못된 형의 소켓에 연결하고자 하는 경우에 예외처리 된다.
	SocketException	제공하지 않는 소켓을 사용하고자 시도하거나 이미 다른 프로세스가 열어 둔 소켓을 연결하고자 하는 경우에 예외 처리 된다.
	UnknownHostException	호스트명이 인터넷 주소를 해석할 수 없는 경우에 예외 처리 된다.
	UnknownServiceException	URL 연결에 의하여 제공되지 않는 서비스를 사용하고자 시도할 때 예외 처리 된다.

연습문제

1 URLConnection 클래스의 메소드를 이용해 콘텐츠의 헤더 정보를 출력하는 애플리케이션(콘솔)이다. 빈 부분에 적당한 문장을 채우시오.



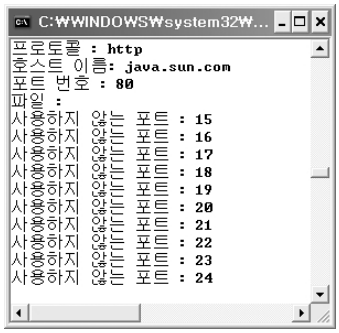
처리조건

① URLConnection 클래스의 메소드를 이용할 것
② 콘텐츠의 헤더 정보를 출력할 것

```
// file name: Jv_11_b1.java
import java.io.*;
import java.util.*;
import      ①

public class Jv_11_b1 {
    public static void main(String[] args) {
                URL u = null;
                URLConnection uc = null;
                try {
                    u = new URL("http://www.bu.ac.kr");
                    uc =      ②

          System.out.println("콘텐츠 유형: " + uc.getContentType());
          System.out.println("콘텐츠 크기: " +      ③      + " byte");
          System.out.println("인코딩 방법: " + uc.getContent Encoding ());
          System.out.println("보내진 날짜: " + new Date(      ④      );
          System.out.println("수정된 날짜: " + new Date (uc.
          getLastModified()));
     }catch(MalformedURLException e) {
          System.out.println(e);
     }catch(IOException e) {
     System.out.println(e);
     }
  }
}
```

연습문제

2 URL 클래스의 메소드를 이용해 관련 정보를 얻고, 소켓 클래스를 이용해 사용 중인 포트 및 사용 하지 않는 포트를 알아내고 출력하는 애플리케이션(콘솔)이다. 빈 부분에 적당한 문장을 채우시오.



처리조건

① "http://java.sun.com:80"에 대해서 URL 클래스의 메소드를 이용할 것

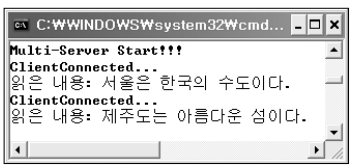
② 소켓 클래스를 이용해 사용 중인 포트 및 사용 하지 않는 포트를 알아낼 것. 그 범위는 15번에서 24번까지 실행할 것. 환경마다 결과가 변할 수 있음

```
// file name: Jv_11_b2.java
import java.net.*;
import java.io.*;

public  class Jv_11_b2 {
        public static void main(String[] args) {
                URL u = null;
                try {
                      u = new URL("http://java.sun.com:80");
                }catch(MalformedURLException e) {
                       System.out.println(e);
                       System.exit(0);
                }

                System.out.println("프로토콜 : "       + [ ① ] );
                System.out.println("호스트 이름: "  + [ ② ] );
                System.out.println("포트 번호 : "       + [ ③ ] );
                System.out.println("파일 : "             + u.getFile());

                for(int i=15 ; i<25 ; i++) {
                    try {
                            Socket s = new [ ④ ]("localhost", i);
```

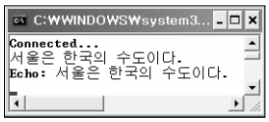
```
                System.out.println("사용중인 포트 :" + i);
                s.close();
            }catch(IOException e) {
                System.out.println("사용하지 않는 포트 : " + i);
            }
        }

    }
}
```

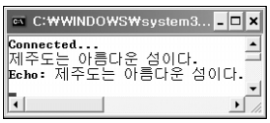
3 소켓 통신에는 크게 연결형(TCP/IP)과 비연결형(UDP/IP)으로 구분된다. 여기서는 연결형(TCP/IP)을 이용해 서버와 클라이언트 측간에 상호 통신이 가능하면서 다중 클라이언트 처리가 가능한 애플리케이션(콘솔) 프로그램을 작성하시오.

〈서버측〉



〈클라이언트 측〉



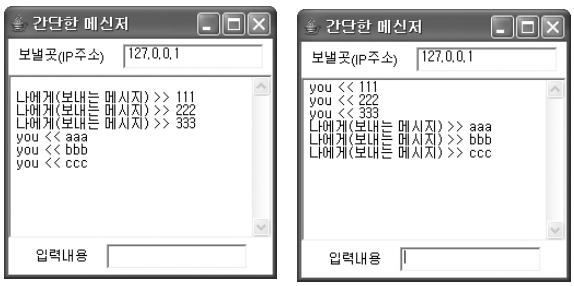


처리조건

① 다중 클라이언트 처리가 가능할 것
② (실행 방법)
만일, 'C:\source' 폴더에서 실행한다고 가정하면 3개 이상의 콘솔 창을 각각 띄운다.
– 먼저 서버 측 프로그램을 실행시킨다(강제 종료 시에는 Ctrl+C 키를 누른다).
DOS C:\source>java Jv_11_x1_MTCPServer
– 그다음 클라이언트 측 프로그램을 실행시킨다.
DOS C:\source>java Jv_11_x1_MTCPClient localhost
–> '서울은 한국의 수도이다.' 입력
– 그다음 클라이언트 측 프로그램을 실행시킨다.
DOS C:\source>java Jv_11_x1_MTCPClient localhost
–> '제주도는 아름다운 섬이다.' 입력
③ 서버 측 프로그램을 먼저 실행 시킨 후에 서버 측에서는 'Multi-Server Start!!!' 메시지를 자체적으로 출력하면서 기다리고 있다가 클라이언트 측의 접속 요청과 클라이언트 측에서부터 서버 측으로 문자 메시지 '서울은 한국의 수도이다.'를 전송하면, 'ClientConnected...'를 자체적으로 출력하고 서버 측은 클라이언트 측으로부터 수신 받은 문자 메시지 '서울은 한국의 수도이다.'를 서버 측 화면에 출력한다.
④ 또한, 클라이언트 측은 자체적으로 입력하여 서버 측으로 보낸 문자 메시지 '서울은 한국의 수도이다.'를 클라이언트 측 화면에 출력한다. 또 다른 클라이언트 창을 띄워서, '제주도는 아름다운 섬이다.' 입력한다.

연습문제

4 소켓 통신에는 크게 연결형(TCP/IP)과 비연결형(UDP/IP)으로 구분된다. 여기에서는 비연결형(UDP/IP)을 이용해 두 클라이언트 측 간에 상호 통신이 가능한 애플리케이션(콘솔) 프로그램을 작성하시오.



처리조건

① 서버와 클라이언트 구분을 하지 않는다.
② 두 클라이언트끼리 메시지를 주고받을 수 있을 것
③ 실행 방법
  만일, C:\source 폴더에서 실행한다고 가정하면 2개의 콘솔 창을 각각 띄운다.
  – 먼저 클라이언트 측 프로그램을 실행시킨다.
    예 C:\source>java Jv_11_x2_Messenger1
  – 그다음 클라이언트 측 프로그램을 실행시킨다.
    예 C:\source>java java Jv_11_x2_Messenger2
④ Jv_11_x2_Messenger1.java 프로그램과 Jv_11_x2_Messenger2.java 프로그램은 거의 동일할 것
  단, Jv_11_x2_Messenger1.java 프로그램의 포트 번호는
  (RECEIVE_PORT = 3000, SEND_PORT = 5000)
  Jv_11_x2_Messenger2.java 프로그램의 포트 번호는
  (RECEIVE_PORT = 5000, SEND_PORT = 3000)으로 설정할 것