

Java Network Programming

Java Network programming conception!

Network Protocol

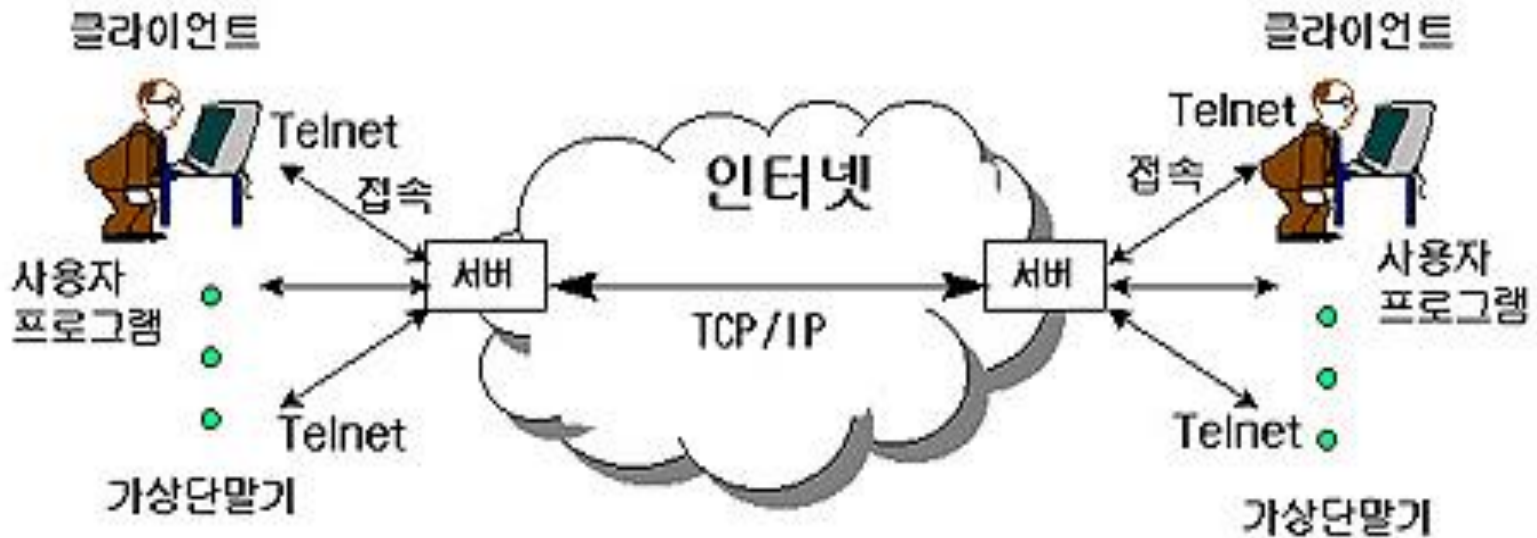
- Protocol
 - 무엇을, 어떻게, 언제 교신할 것인가에 관련된 개체 사이에서 상호 수용 가능한 협정
 - 메시지 전달 방법, 메시지 형식, 에러 발생 시 해결 방법을 기술
 - 어떤 컴퓨터와 통신을 시작해야 하는지?
 - 응답이 어떻게 처리되는지?
 - 데이터 표현은 어떻게 하며, 어떻게 송,수신하는지?
 - 통신중 발생하는 error는 어떻게 처리하는지?
- TCP/IP를 중심으로 한 프로토콜(응용프로그램 계층)
 - Telnet, FTP, SLIP/PPP

Telnet

- 주로 유닉스 시스템의 네트워크로 연결된 원격터미널에서 호스트의 셸 모드를 흉내내는 프로그램, 원격지의 가상 터미널에서 호스트에서 작업하는 것과 같은 효과를 낼 수 있다.(제한된 텍스트 모드)
- 지역(local area)의 시스템에서 원격지(remote area)의 시스템으로 접속을 할 수 있도록 하는 네트워크 유틸리티(Telnet은 TCP 세션을 통해 연결된다.)
- TCP/IP 프로토콜을 기본으로 하는 유닉스 운영체제는 보통 서비스마다 고유의 포트 번호가 있는데 이것은 보통 /etc/services 라는 파일에 정의되어 있다.

Telnet

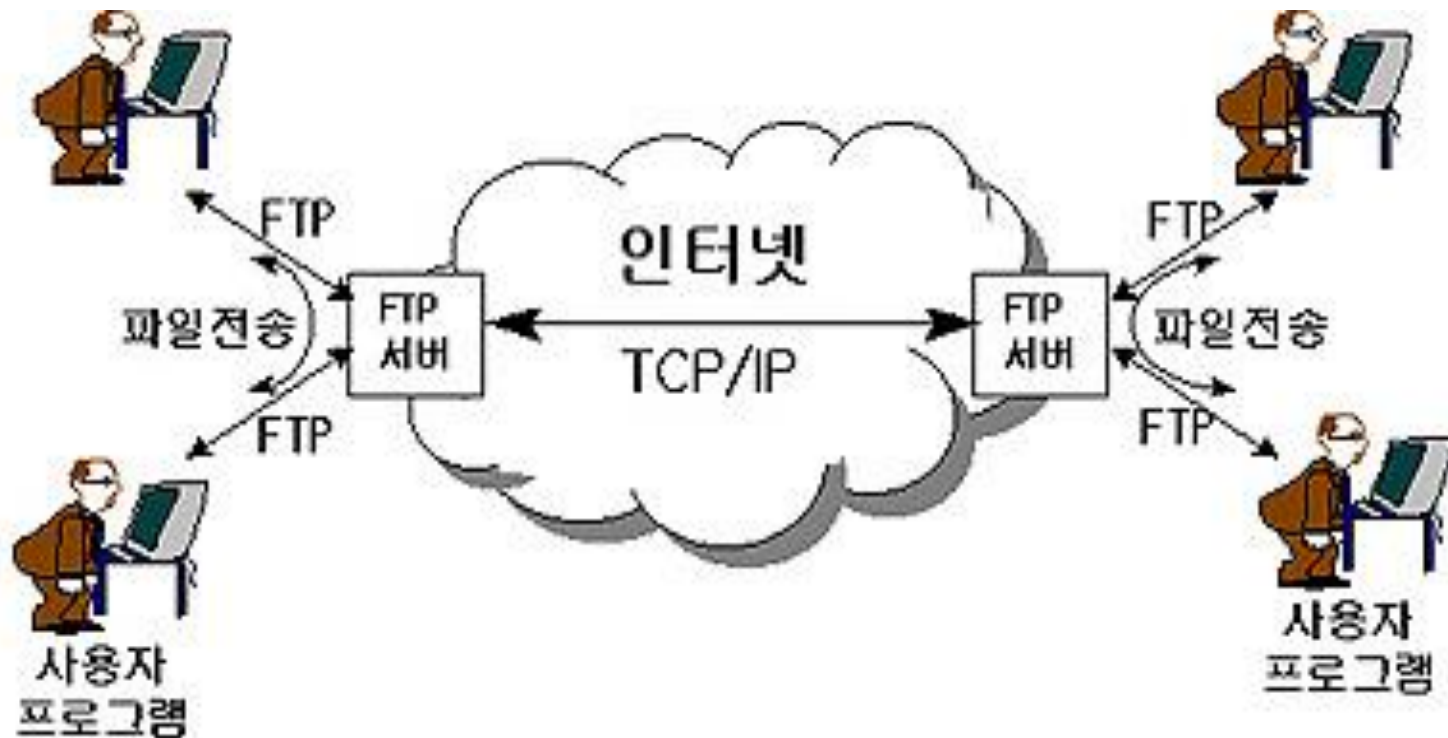
- Telnet 프로그램 종류 : Netterm, Winterm, CRT 등.
- 포트(port) : 일반적으로 23
- 접속법: %telnet domain-name (port) ,%telnet IP-address (port)
- Telnet 프로그램 종류 : Netterm, Winterm, CRT 등.



FTP(File Transfer Protocol)

- 텔넷의 최대 장점이자 단점 : 데이터의 저장과 원격지 호스트(Host)에서만 이루어진다.
- TCP 세션을 이용하는 단지 파일 전송만을 위한 프로토콜
- FTP(file transfer protocol)란 인터넷 망으로 연결되어 있는 멀리 떨어져 있는 서버로 파일을 올리거나 가져오기 위해 필요한 일종의 프로토콜이다.
- 우리가 흔히 말하는 ftp 프로그램이라는 것은 ftp 프로토콜을 내장하고 있는 파일 송수신 전용소프트웨어라고 할 수 있다. 이 ftp전용 소프트웨어에는 대표적인 것으로 ws_ftp, cute_ftp등이 있다.

FTP(File Transfer Protocol)

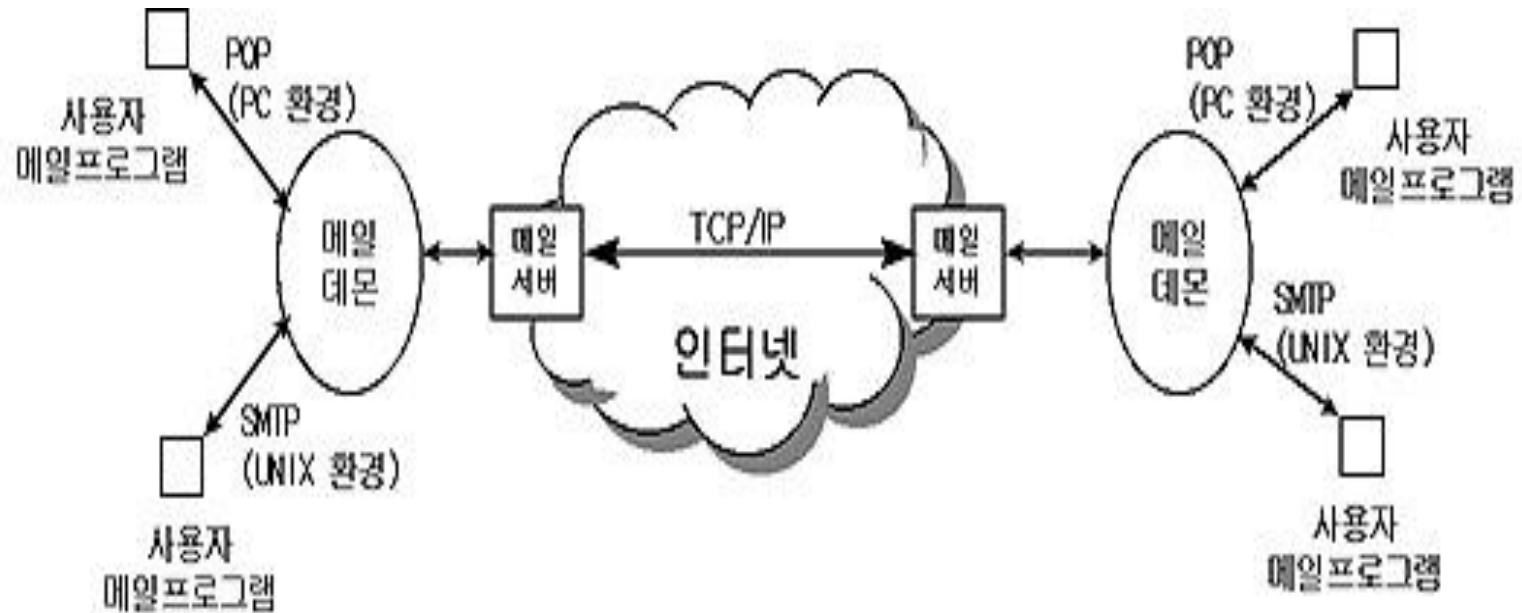


SMTP(Simple Mail Transfer Protocol)

- 단순한 메일 전송 프로토콜
- 보통 인터넷 서비스 업체에서 SMTP 서버를 메일 송신 서버로 POP(Post Office Protocol) 서버를 메일 수신 서버로 둔다. 하나의 호스트에 두 서비스를 두기도 하지만 보통 두 서버를 나눈다.
- SMTP는 안정적인 메일의 전송을 위해 TCP 세션을 이용한다.

POP(Post Office Protocol)

- 메일 수신 프로토콜
- POP(109) → POP2(109) → POP3(110)



Sun NFS(Network File System)

- Unix를 기반으로 하는 일련의 Network 호스트들이 자신의 파일 시스템을 여러 클라이언트들과 공유하기 위한 네트워크 자원
- NFS는 Sun의 Sparc 기종의 솔라리스나 Sun OS라는 네트워크 운영체제에서 많이 사용되는데 이 NFS로 말미암아 도스 기반의 시스템이나 노벨의 Netware, 윈도우 NT등에서 자신의 파일 시스템을 변경하지 않은 채 유닉스의 파일 시스템을 지역화 할 수 있다.
- 네트워크 상에서 서로 다른 파일 시스템을 연결 시켜 준다.

OSI Model

OSI(Open System Interconnection) Model

- 이질적인 컴퓨터 시스템의 연결을 위한 표준을 정의하는데 있어서 기준을 제공하였고, 오픈 시스템(Open System)간의 연결을 위한 토대가 되었다.
- ISO는 이를 7계층으로 구분하여 각 계층의 기능과 서비스를 정의하였고, 각 계층은 관리하기 적당하게 설계하였다

OSI 7 Layer

Layer 7	Application 계층 (사용자가 볼 수 있는 유일한 계층, TELNET, FTP, SMTP)
Layer 6	Presentation 계층 (데이터 변환, 프로토콜 변환)
Layer 5	Session 계층 (두 응용프로그램들 사이의 연결을 설정하고 관리)
Layer 4	Transport 계층 (전송될 메시지를 여러 조각으로 분할, 분할된 각 조각들은 작은 패킷으로 재구성, TCP/UDP는 트랜스포트 프로토콜의 예이다.)
Layer 3	Network 계층 (수신 컴퓨터까지의 데이터 전송경로를 설정하거나 교환하는 일을 담당한다. IP, X.25 등이 네트워크 계층 프로토콜의 예이다)
Layer 2	Data Link 계층 (데이터를 비트 단위의 전기적 신호로 바꾸어 전송)
Layer 1	Physical 계층 (OSI모델의 최하위 계층으로 시스템과 시스템간의 물리적인 접속을 제어하는 기능을 제공한다)

TCP/IP

TCP/IP

- 인터넷의 Network Protocol이 TCP/IP 이다.
- 개방형 프로토콜 이다.
- 확장성 : TCP/IP는 네트워크의 크기와 구성하는 Host의 수에 따라 A,B,C,D,E 등의 클래스로 나눌 수 있다. 또한 호스트가 꼭 차면 넷마스크를 변경하여 서브네트워크를 구성 할 수 있다.
- TCP/IP 계층 아키텍처는 4계층으로 추상화 할 수 있다.
- 비록 OSI 모델을 그대로 따른 것은 아니지만 안정성 있게 잘 동작하고 서로 다른 기종 컴퓨터 시스템을 연결하는데 가장 많이 이용되고 있다.

TCP/IP

- 인터넷에 연결된 모든 호스트는 IP(Internet Protocol)를 사용해야 한다. IP의 주요 임무는 호스트의 주소지정과 패킷의 전달이다. 그러나 종단간(End-to-End)에 전송되는 메시지의 안정성이나 흐름제어에 관해서는 책임이 없습니다. 단지 패킷을 다음 목적지로 전달하기 위해 최선을 다할 뿐 전달되었는지에 관해서는 보장해 주지 않습니다.
- TCP는 믿을 수 있는 종점간 (End-to-end) 통신에 필요한 에러검사를 한다.

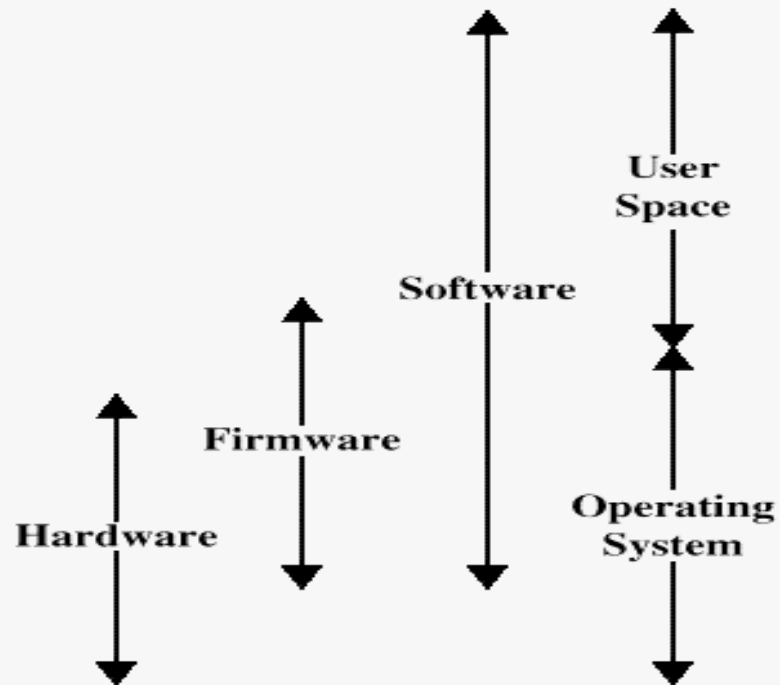
TCP/IP

- TCP는 데이터가 도착했는지 확인할 필요가 있고 전송한 데이터가 올바른 순서대로 도착해야만 하는 응용프로그램에 사용된다. TCP는 신뢰성, 연결지향, 바이트 스트림 프로토콜
- 연결형(Connection-Bases) 서비스 : 연결형은 데이터를 주고 받기 전에 미리 연결을 설정하고 연결된 전송로를 이용해서 데이터를 주고 받은 다음 연결을 해제하는 방식으로 스트림 서비스(Stream Service)라고도 한다. 이 방식은 TCP에서 사용하는 방식으로서 데이터를 신뢰성 있게 전송할 수 있다.

TCP/IP Layer

인터넷에서 사용하는 TCP/IP는 정확하게 보면, TCP는 OSI의 4번째 계층인 트랜스포트 계층에 포함 되고, IP는 Layer 3 네트워크 계층에 각각 해당된다. 하지만 TCP/IP라고 하면 일반적으로 인터넷에서의 사용하는 전체 프로토콜을 대변(인터넷 프로토콜)하고 있기 때문에 다시 세분하면 다음과 같이 4개의 계층으로 구분할 수 있다.

TCP/IP	OSI
Application	Application
	Presentation
	Session
Transport (host-to-host)	Transport
Internet	Network
Physical	Data Link
Physical	Physical



TCP/IP Layer

- 응용 계층(Application Layer) : 네트워크를 실제로 사용하는 응용 프로그램으로 이루어진다. 우리가 이미 알고 있는 파일 전송 프로그램 등이 이 계층에 해당되는 프로그램이다. OSI 모델에서 보면 애플리케이션 계층과 프리젠테이션 계층이 여기에 해당된다.
- 트랜스포트 계층(Transport Layer) : 보내는 쪽 <A>는 데이터를 적절히 분할해서 일련 번호를 붙여서 전달한다. 받는 쪽는 패킷에서 분리된 segment를 순차적으로 조합하여 원래의 데이터를 만들어서 이들을 응용 프로그램 계층에 전달한다.

TCP/IP Layer

TCP(Transmission Control Protocol) – 연결지향 (Connection Oriented) 전송. 에러 정정과 흐름 제어의 기능을 가지고 있어서 잘못되거나 유실된 패킷이 있을 경우 원래의 호스트에 재전송을 요청한다. 안전성과 신뢰성이 뛰어나서 사용자 데이터 전송, 자 데이터 전송, 대용량 전송에 이용된다.

UDP(User Datagram Protocol) – 비연결 (Connectionless) 전송. 에러 정정을 하지 않는다. 보통 시스템내부 메시지 전달과 데이터 전달, 소규모 데이터 전송에 이용된다.

TCP/IP Layer

- 인터넷 계층(Internet Layer) : 데이터그램을 정의하고 데이터그램을 routing하는 일을 담당한다. 즉 데이터를 정확한 곳에 보내기만 하면 되는 것이다. 데이터그램이라고 하는 것은 IP 프로토콜에서 다루는 패킷 데이터를 말한다. 데이터그램이 가지고 있는 자료는 보낸 주소(source address), 받을 주소(destination address), 그리고 보내는 데이터, 그외 몇 가지 조절 필드(control field)를 가지고 있다.

IP(Internet Protocol) – Data Segment를 Packet으로 만들어 이를 목적지로 전달하는 역할을 수행한다. 그러나 전달 여부를 보장하지 않는다.

- 물리 계층(Physical Layer) : 네트워크를 구성하는 물리적 장치 및 전송매체 그리고 채널상으로 데이터를 전송하는 기능을 한다. OSI의 Physical Layer와 Data Link Layer에 해당한다.

UDP(User Datagram Protocol)

- UDP는 응용프로그램에서 직접 데이터그램을 전송하기 위해 사용되며 IP가 제공하는 서비스와 유사하다. 그리고, 단순히 데이터를 보내기만 하기 때문에 프로토콜 오버헤드가 상당히 작다(즉, 신뢰적인 통신을 보장하지 않기 때문에 데이터그램이 잘 도착했는지 확인할 필요도 없으며 재전송할 필요도 없다).
- TCP의 대안이며, IP와 함께 쓰일 때에는 UDP/IP라고 표현하기도 한다. TCP와 마찬가지로 UDP도 한 컴퓨터에서 다른 컴퓨터로 데이터그램 이라고 불리는 실제 데이터 단위를 받기 위해 IP를 사용한다. 그러나 UDP는 TCP와는 달리, 메시지를 패킷(데이터그램)으로 나누고, 반대편에서 재조립하는 등의 서비스는 제공하지 않으며, 특히 도착하는 데이터 패킷들의 순서를 제공하지 않는다

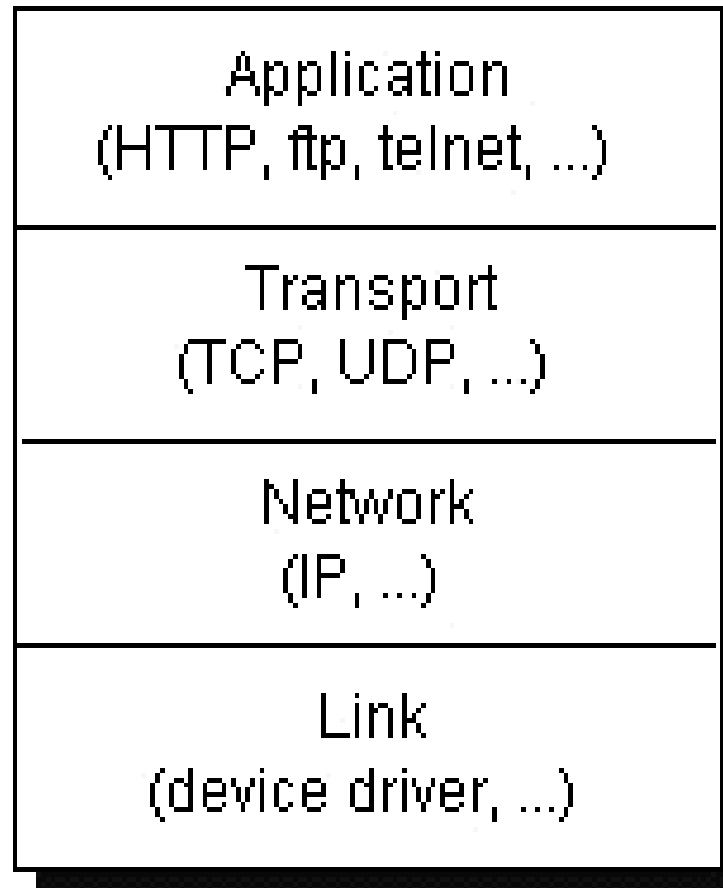
UDP(User Datagram Protocol)

- UDP는 신뢰할 수 있는 종점간 데이터 송수신을 보장하지 않으므로 파일 전송, 메일 서비스 등에는 적합하지 않다.
 - ▶ 도메인 네임(domain name) 등 한 패킷의 송수신으로 어떤 서비스가 이루어지는 경우에 많이 사용된다.
 - ▶ LAN과 같이 전송 오류가 거의 없고 패킷의 전달 순서가 바뀌지 않는 환경에서는 TCP보다 처리 속도가 빠른 UDP가 유리할 수 있다.
 - ▶ LAN에서 제공되는 NFS(Network File System)는 UDP를 사용한다.

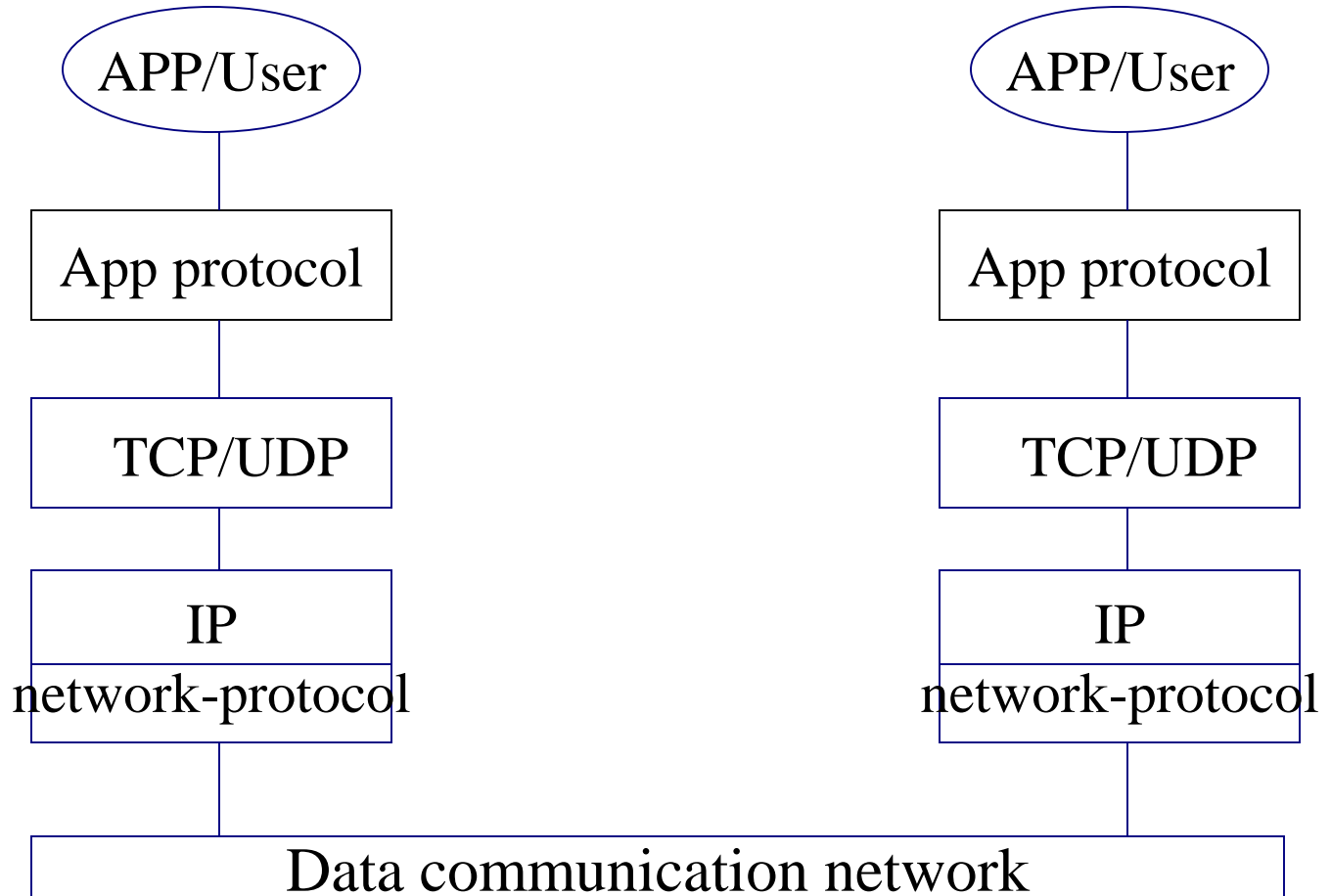
UDP(User Datagram Protocol)

- 연결을 설정하지 않고 데이터를 보내고 그 데이터가 제대로 갔는지 여부에 대해 전혀 신경 쓰지 않는 비연결(connection-less)

Internetworking Layer



Internetworking Layer



Internetworking Basics

- When you write Java programs that communicate over the network, *you are programming at the application layer*. Typically, you don't need to concern yourself with the TCP and UDP layers. Instead, you can use the classes in the java.net package. These classes provide system-independent network communication.

TCP

- When two applications want to communicate to each other reliably, they establish a connection and send data back and forth over that connection. *This is analogous to making a telephone call.*

TCP

- If you want to speak to Aunt Beatrice in Kentucky, a connection is established when you dial her phone number and she answers. You send data back and forth over the connection by speaking to one another over the phone lines. Like the phone company.

TCP

- TCP guarantees that data sent from one end of the connection actually gets to the other end and *in the same order it was sent*. Otherwise, an error is reported.

TCP

- *TCP provides a point-to-point channel for applications* that require reliable communications. The Hypertext Transfer Protocol (HTTP), File Transfer Protocol (FTP), and Telnet are all examples of applications that require a reliable communication channel.

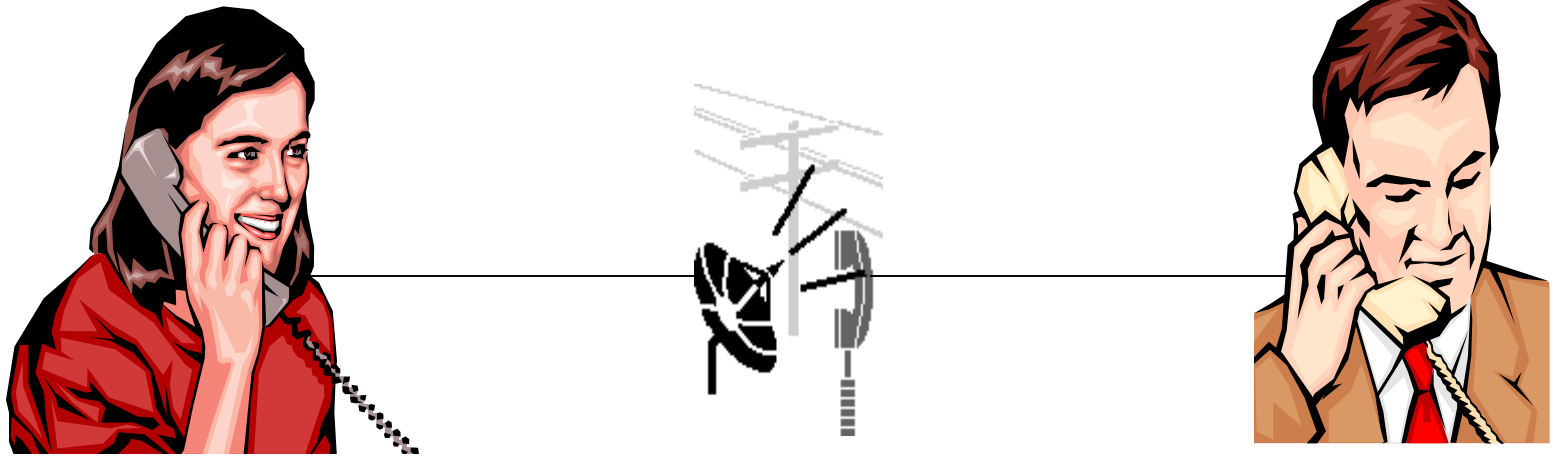
TCP

- The order in which the data is sent and received over the network is critical to the success of these applications. *When HTTP is used to read from a URL, the data must be received in the order in which it was sent.* Otherwise, you end up with a jumbled HTML file, a corrupt zip file, or some other invalid information.

TCP

- Definition
 - TCP (Transmission Control Protocol) is a connection-based protocol that provides a reliable flow of data between two computers.

TCP



Connection Oriented

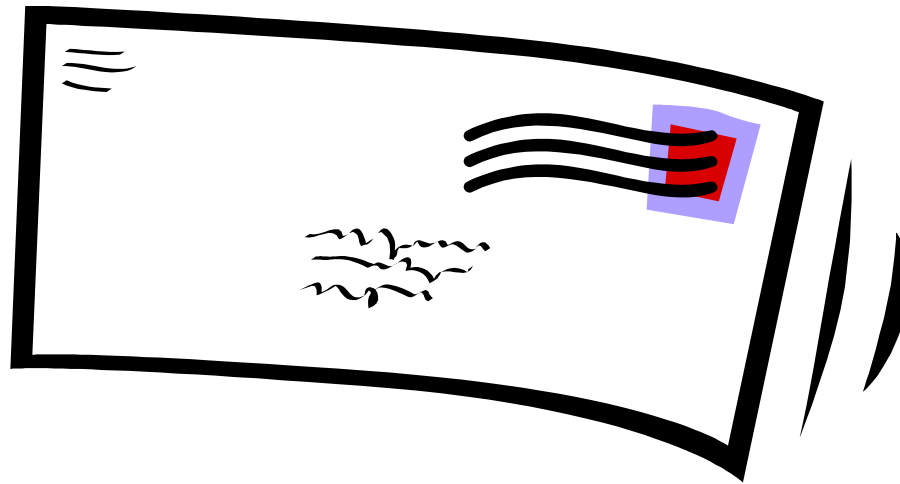
UDP

- The UDP protocol provides for communication that is not guaranteed between two applications on the network. UDP is not connection-based like TCP. Rather, it sends independent packets of data, called datagrams, from one application to another.

UDP

- Sending datagrams is much like sending a letter through the postal service
- The order of delivery is not important and is not guaranteed, and each message is independent of any other

UDP



Connectionless

UDP

- Definition
 - UDP (User Datagram Protocol) is a protocol that sends independent packets of data, called datagrams, from one computer to another with no guarantees about arrival. UDP is not connection-based like TCP.

Working with URLs

Java.net.URL Class

Working with URLs

- Java programs that interact with the Internet also may use URLs to find the resources on the Internet they wish to access.
- Java programs can use a class called URL

What Is a URL?

- Definition

- URL is an acronym for Uniform Resource Locator and is a reference (an address) to a resource on the Internet.

http://www.naver.com:80/index.html

Protocol Identifier

Host Name

port

File name

Creating a URL

- URL Constructors

- `new URL("http://www.abc.com:80/");`
- `new URL("http", "www.abc.com", "/index.html");`
- `new URL("http", "www.abc.com", 80, "pages/index.html");`

- Note

- URLs are "write-once" objects. Once you've created a URL object, you cannot change any of its attributes (protocol, host name, filename, or port number).

Parsing a URL

- The URL class provides several methods that let you query URL objects. You can get the protocol, host name, port number, and filename from a URL using these accessor methods:

Parsing a URL

- `public String getProtocol()`
 - Returns the protocol identifier component of the URL.
- `public String getHost()`
 - Returns the host name of the URL.
- `public int getPort()`
 - Returns the port number component of the URL.
- `public String getFile()`
 - Returns the filename of the URL.

Parsing a URL

- `public Object getContent()`
 - Gets the contents of this URL
- `public Object getContent(Class[] classes)`
 - Gets the contents of this URL

매개변수로 들어온 `class`의 인스턴스로 만들어진 자바의 URL 객체 콘텐츠를 반환한다. 클래스가 지원되지 않는 경우는 `null` 반환한다.

이 메소드를 호출한 쪽에서 어떻게 `content`를 인코딩 할지를 조절 할 수 있다. 예를들면 현재의 URL은 HTML 웹페이지의 인코딩을 `String`, HTML 객체, `InputStream` 타입으로 수행할 수 있도록 지원하고 있는데 이때 URL 리소스를 문자열로 인코딩하여 받고 싶다면 이 메소드에 `String.class`를 넘기면 되고, 스트림으로 인코딩 하여 받고 싶다면 `InputStream.class`를 넘기면 된다. 즉 이 메소드는 주어진 URL에서 여러가지 콘텐츠 인코딩을 지원하는 경우에 쓸모가 있다. `public Object getContent()`의 경우 반환되는 `content`의 `type`은 해당 콘텐츠에 달려 있다. 만약 URL이 어떤 이미지를 가리키고 있다면 반환되는 타입은 `Image`가 될것이다.

URL Connection Method

- `public final InputStream openStream()` throws `IOException`
 - Opens a connection to this URL and returns an `InputStream` for reading from that connection.
 - This method is a shorthand for `openConnection().getInputStream()`
- `public URLConnection openConnection()` throws `IOException`
 - Returns a `URLConnection` object that represents a connection to the remote object referred to by the URL.
 - A new connection is opened every time by calling the `openConnection` method of the protocol handler for this URL.

Example (URLInfoTest.java)

```
import java.io.*;
import java.net.*;
class URLInfoTest{
    public static void main(String[] args) throws Exception {
        try{
            //URL aURL = new URL("http://www.daum.net");
            URL aURL = new URL("http", "www.daum.net", 80 , "/index.html");
            System.out.println("protocol name:" + aURL.getProtocol());
            System.out.println("host name:" + aURL.getHost());
            System.out.println("file name:" + aURL.getFile());
            //생성자에서 port가 set되는 경우에 표시, 아니면 -1
            System.out.println("port name:" + aURL.getPort());
            //HTML 문서 내부의 위치를 지정하는 <a name="..."> 태그의 값을
            구한다.
            System.out.println("ref : "+aURL.getRef());
```


Example (URLInfoTest.java)

```
        BufferedReader br = new BufferedReader(new
        InputStreamReader(aURL.openStream()));
        String line;
        while((line = br.readLine()) != null) {
            System.out.println(line);
        }
    }
    catch(MalformedURLException e) {
        System.out.println("MalformedURLException : "+e);
    }
    catch(IOException e) {
        System.out.println("IOException :"+ e);
    }
}
```

Example (URLInfoTest2.java)

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

class URLInfoTest2{
    public static void main(String[] args) throws Exception {
        try{
            int ch;
            URL aURL = new URL("http", "www.oraclejava.co.kr", 80 ,
"/test.html");
            //
            Class[] classes = {Reader.class, InputStream.class};
            Object content = aURL.getContent(classes);
            System.out.println(content + "\n");
            while((ch=((InputStream)content).read()) != -1) {
                System.out.print((char) ch);
            }
        }
    }
}
```

Example (URLInfoTest2.java)

```
        System.out.println("-----");
        InputStream is = aURL.openStream();
        while((ch=is.read()) != -1) {
            System.out.print((char) ch);
        }
    }
    catch(MalformedURLException e) {
        System.out.println("MalformedURLException : "+e);
    }
    catch(IOException e) {
        System.out.println("IOException :"+ e);
    }
}
}
```

Example (PageReadTest.java)

```
import java.io.*;
import java.net.*;
public class PageReadTest{
    public static void main(String[] args) throws Exception {
        if(args.length<1)
            System.out.println("usage: java PageReadTest
http://www.oraclejava.co.kr");
        URL location = new URL(args[0].toString());
        BufferedReader in = new BufferedReader
            (new InputStreamReader(location.openStream()));
        String readLine;
        while ((readLine = in.readLine()) != null)
            System.out.println(readLine);
        in.close();
    }
}
```

java.net.URLConnection Class

- The abstract class URLConnection is the superclass of all classes that represent a communications link between the application and a URL.
- The URLConnection object is created by invoking the openConnection method on a URL and connect method on URLConnection.
- Instances of this class can be used both to read from and to write to the resource referenced by the URL.

java.net.URLConnection Class

- 자바 URL 객체에 대한 활성화된(프로토콜) 연결을 나타내는 클래스 이다.
- URL로 나타낸 자원에 대해 입력과 출력을 하는 것이 목적이다.
- 응용
 - HTTP 연결을 통한 이진 데이터 다운로드
 - 폼 데이터를 POST 방식으로 전송하기

URLConnectionTest.java

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

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

```
class URLConnectionTest{
```

```
    public static void main(String args[])
```

```
        throws MalformedURLException, IOException    {
```

```
        URL url = new URL( args[0] );
```

```
        URLConnection con = url.openConnection();
```

```
        System.out.println( con.getHeaderField("Content-Type") );
```

```
        System.out.println( con.getHeaderField("Content-Length") );
```

```
        System.out.println( con.getContentType() );
```

URLConnectionTest.java

```
System.out.println( con.getContentEncoding() );
System.out.println( con.getContentLength() );
System.out.println( new java.util.Date(con.getDate()) );
System.out.println( new java.util.Date(con.getExpiration()) );
System.out.println( new
java.util.Date(con.getLastModified()) );

System.out.println( con.getContent() );
InputStream in = con.getInputStream();
for(int ch; (ch = in.read()) != -1;)
    System.out.write( ch );
System.out.flush();
}
}
```


URLPostTest.java

//회원 가입된 임의의 사이트에 ID. PassWord를 POST 방식으로 넘겨
//실제 웹페이지에서 인증한것과 같은 결과를 받아 화면에 출력

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

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

```
class URLPostTest{
```

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

```
        URL url = new URL("http", "www.neonet.co.kr", 80,  
        "/rebank/common/login_after_new.neo?/rebank/index.neo");
```

```
        URLConnection con = url.openConnection();
```

```
        //"이 연결을 통해 데이터를 송신 하겠다" 를 알림
```

```
        //HTTP의 POST와 같은 효과
```

```
        con.setDoOutput(true);
```

```
        PrintWriter out = new PrintWriter(con.getOutputStream());
```

URLPostTest.java

```
StringBuffer sb = new StringBuffer("id=" +
    URLEncoder.encode("mylife68", "euc-kr"));
sb.append("&passwd=" + URLEncoder.encode("tatata", "euc-kr"));
out.print(sb.toString());
out.close();
//서버의 응답을 받아 내기 위해 getInputStream을 해야만 실제로
//프로토콜 핸들러가
//웹서버에 연결하고 POST명령을 보내고 출력스트림을 통해 쓴
//데이터를 전송한다.
BufferedReader br = new BufferedReader(new
    InputStreamReader(con.getInputStream(),"KSC5601"));
String line;
while((line = br.readLine()) != null) System.out.println(line);
br.close();
}
}
```

Java.net.HttpURLConnection

- public abstract class **HttpURLConnection**
extends **URLConnection**
- Constructor
`HttpURLConnection(URL u)`
Constructor for the HttpURLConnection.
- for Http Protocol
- Each HttpURLConnection instance is used to make a single request but the underlying network connection to the HTTP server may be transparently shared by other instances.

Java.net.HttpURLConnection

- Calling the close() methods on the InputStream or OutputStream of an HttpURLConnection after a request may free network resources associated with this instance but has no effect on any shared persistent connection. Calling the disconnect() method may close the underlying socket if a persistent connection is otherwise idle at that time.

HttpURLConnectionTest.java

```
import java.io.*;
import java.net.*;
class HttpURLConnectionTest{
    public static void main(String[] args) throws Exception {
        try{
            URL aURL = new URL("http", "www.oraclejava.co.kr", 80 ,
"/index.html");
            HttpURLConnection connection =
(HttpURLConnection)aURL.openConnection();
            int responseCode = connection.getResponseCode();

            if (responseCode == HttpURLConnection.HTTP_OK) {
                System.out.println("Request Method : "      +
connection.getRequestMethod());
                System.out.println("Request Code : "        +
connection.getResponseCode());
                System.out.println("Request Message : "      +
connection.getResponseMessage());
            }
        }
    }
}
```

HttpURLConnectionTest.java

```
        BufferedReader in = new BufferedReader(new
        InputStreamReader(connection.getInputStream()));
        //BufferedInputStream bi = new
        BufferedInputStream(connection.getInputStream());
        //int b;
        //while((b = bi.read()) != -1) {
            //System.out.write(b);

        //}
        String line;
        while ((line = in.readLine()) != null) {
            //System.out.println(line);
        }
    }
}
catch(IOException e) {
    System.out.println("IOException :" + e);
}
}
```

URL Encoding, Decoding

- *URLEncoder* class contains a utility method for converting a String into a MIME format called "*x-www-form-urlencoded*" format.
 - To convert a String, each character is examined in turn:
 - The ASCII characters 'a' through 'z', 'A' through 'Z', '0' through '9', and ".", "-", "*", "_" remain the same.
 - The space character ' ' is converted into a plus sign '+'.
 - All other characters are converted into the 3-character string "%xy", where xy is the two-digit hexadecimal representation of the lower 8-bits of the character.

URL Encoding, Decoding

- `URLEncoder` class
 - `public static String encode(String s)`
 - Translates a string into x-www-form-urlencoded format.
 - `public static String encode(String s, String enc)`
- `URLDecoder` class
 - `public static String decode(String s)`
 - Decodes a "x-www-form-urlencoded" to a String.
 - `public static String decode(String s, String enc)`

PostTest.java(Client)

//실행 : java PostTest sample.xml

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

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

```
import java.net.HttpURLConnection.*;
```

```
public class PostTest {
```

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

```
        String inputString = "";
```

```
        String input;
```

```
        BufferedReader in = new BufferedReader(new FileReader(args[0]));
```

```
        while((input = in.readLine()) != null)
```

```
            inputString = inputString + input;
```

```
        in.close();
```

```
        URL myURL = new URL("http", "www.oraclejava.co.kr", 8080,  
            "/servlet/PostTest");
```

```
        HttpURLConnection c = (HttpURLConnection)(myURL.openConnection()); 65
```

PostTest.java(Client)

```
c.setDoOutput(true);
    PrintWriter out = new PrintWriter(c.getOutputStream());
    // Here's whether the parameter is set.
    out.println("xmldoc=" + URLEncoder.encode(inputString, "euc-kr"));
    out.close();

    BufferedReader in2 = new BufferedReader(new
InputStreamReader(c.getInputStream(), "euc-kr"));

    String inputLine;
    while((inputLine = in2.readLine()) != null)
        System.out.println(inputLine);
    in2.close();
}
```

sample.xml(Client)

<여성부>

<자바실무></자바실무>

<시작일>2003.3.4</시작일>

<종료일>2003.4.24</종료일>

<원도우></원도우>

<시작일>2003.3.4</시작일>

<종료일>2003.4.24</종료일>

</여성부>

PostTest.java(Server)

```
import java.io.*;
import java.text.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class PostTest extends HttpServlet {
    public void doGet(HttpServletRequest request,
        HttpServletResponse response)
        throws IOException, ServletException {
        response.setContentType("text/plain");
        PrintWriter out = response.getWriter();
        HttpSession session = request.getSession();
        String xml = request.getParameter("xmldoc");
        out.println(xml);
    }
}
```

PostTest.java(Server)

```
String fileName = "input.xml";
PrintWriter fileWriter = new PrintWriter (new BufferedWriter(new
FileWriter(fileName)));
    fileWriter.write(xml);
    fileWriter.flush();
    fileWriter.close();
}
public void doPost(HttpServletRequest request,
                    HttpServletResponse response)
    throws IOException, ServletException {
    doGet(request, response);
}
}
```

Client Networking

- InetAddress Class
- Socket Class

TCP를 사용한 Client Network

- TCP는 IP를 지원하는 네트워크 위에 가상의 스트림 연결 상태를 만들어 놓은 프로토콜
- TCP 연결을 수행할 목적지의 주소를 지정할 때 사용하는 IP주소는 `InetAddress` 클래스로 처리하고, 실제로 서버에 TCP로 연결할 때는 `Socket` 클래스를 이용한다.
- 일단 연결이 된 후에는 이 연결된 소켓을 사용하여 `Remote Server`로의 스트림을 얻은 다음 이 스트림 인터페이스를 이용해 데이터를 보내고 받는다.

InetAddress 클래스

InetAddress 클래스

- represents an Internet Protocol (IP) address.
- TCP는 IP를 지원하는 네트워크 위에 가상의 스트림 연결 상태를 만들어 놓은 프로토콜
- TCP 연결을 수행할 목적지의 주소를 지정할 때 사용하는 IP주소는 InetAddress 클래스로 처리하고, 실제로 서버에 TCP로 연결할 때는 Socket 클래스를 이용한다.
- 일단 연결이 된 후에는 이 연결된 소켓을 사용하여 Remote Server로의 스트림을 얻은 다음 이 스트림 인터페이스를 이용해 데이터를 보내고 받는다.

InetAddress 클래스

- Address Types
 - unicast : 하나의 송신자가 다른 하나의 수신자로 데이터를 전송하는 방식, 일반적인 인터넷 응용프로그램이 모두 유니캐스트 방식을 사용하고 있다.
 - multicast : 하나 이상의 송신자들이 특정한 하나 이상의 수신자들에게 데이터를 전송하는 방식으로 인터넷 화상 회의 등의 응용에서 사용한다.
 - broadcast : 하나의 송신자가 같은 서브네트웍상의 모든 수신자에게 데이터를 전송하는 방식이다

InetAddress 클래스

- 그룹 통신을 위하여 다중 수신자들에게 동일한 데이터를 전송하고자 할 경우 유니캐스트 전송방식을 이용한다면 전송하고자 하는 데이터 패킷을 다수의 수신자에게 각각 여러 번 전송해야 하며, 이러한 동일한 패킷의 중복전송으로 인해 네트워크 효율이 저하된다. 또한 수신자 수가 증가할 경우 이러한 문제점은 더 커지게 된다.
- 반면 멀티캐스트 전송이 지원되면 송신자는 여러 수신자에게 한 번에 메시지가 전송되도록 하여, 데이터의 중복전송으로 인한 네트워크 자원의 낭비를 최소화할 수 있게 된다.

SimpleDNS.java

```
import java.net.InetAddress;

public class SimpleDNS {
    public static void main(String args[]) throws Exception
    {
        InetAddress address =
        InetAddress.getByName(args[0]);
        //the highest order byte of the address is in IP[0].
        byte IP[] = address.getAddress();
        System.out.println("IP.length : " + IP.length);
        for (int index = 0; index < IP.length; index++) {
```

SimpleDNS.java

```
if (index > 0) System.out.print(".");  
//System.out.print(Integer.toBinaryString((int)IP[index]));  
  
//System.out.print(new Byte(IP[index]).toString());  
//System.out.print(((int)IP[index]));  
System.out.print(((int)IP[index]) & 0xff);  
}  
System.out.println();  
}  
}
```

About Sockets

About Sockets

- In client-server applications, the server provides some service, such as processing database queries or sending out current stock prices. The client uses the service provided by the server, either displaying database query results to the user or making stock purchase recommendations to an investor. The communication that occurs between the client and the server must be reliable. That is, no data can be dropped and it must arrive on the client side in the same order in which the server sent it.

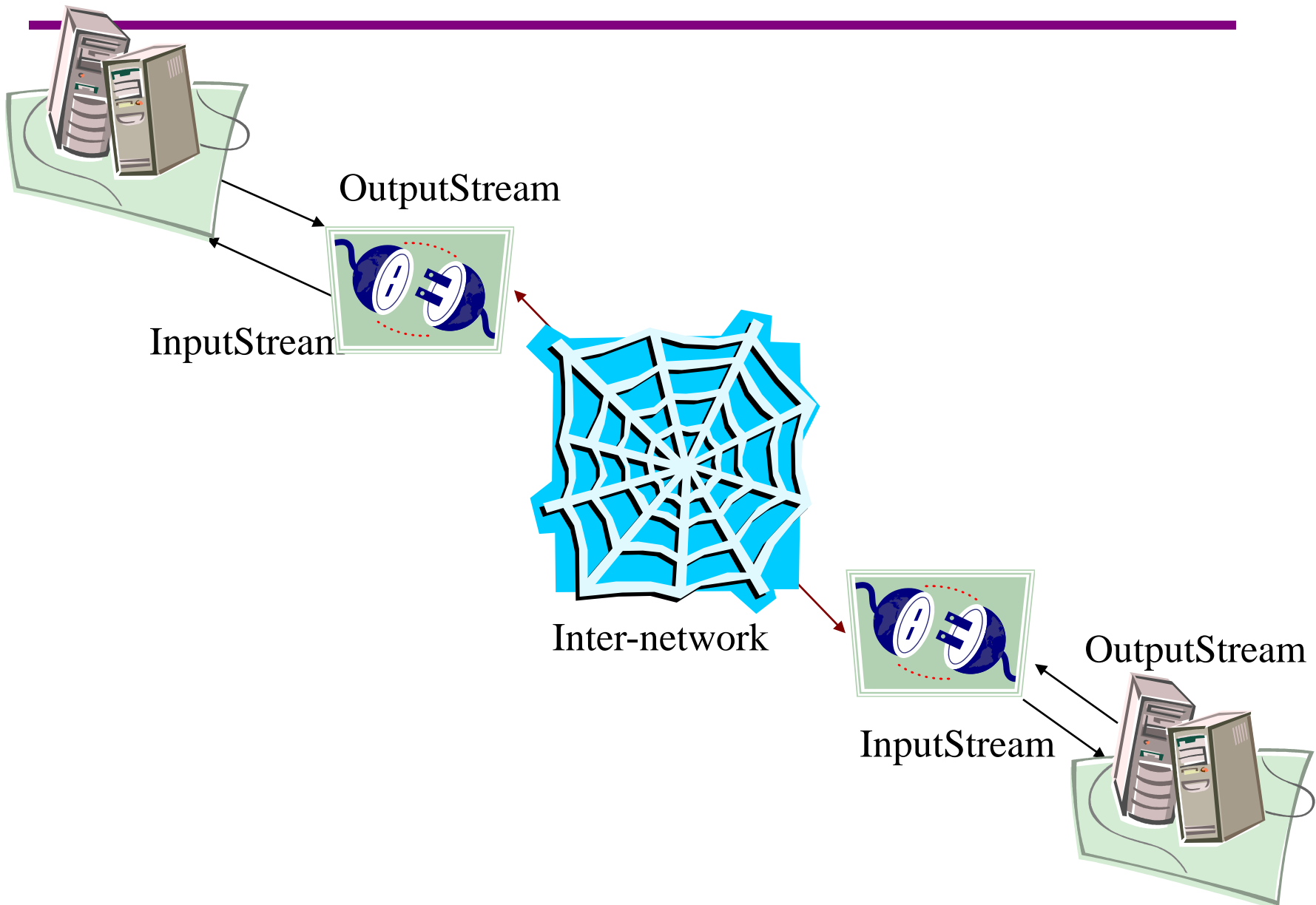
About Sockets

- TCP provides a reliable, point-to-point communication channel that client-server applications on the Internet use to communicate with each other. To communicate over TCP, a client program and a server program establish a connection to one another. Each program binds a socket to its end of the connection. To communicate, the client and the server each reads from and writes to the socket bound to the connection.

What Is a Socket?

- A socket is one end-point of a two-way communication link between two programs running on the network. Socket classes are used to represent the connection between a client program and a server program. The java.net package provides two classes--**Socket** and **ServerSocket**--that implement the client side of the connection and the server side of the connection, respectively.

What Is a Socket?



What Is a Socket?

- Constructor
 - `public Socket(InetAddress address, int port)`
 - Creates a stream socket and connects it to the specified port number at the specified IP address.
 - `public Socket(String host, int port)`
 - Creates a stream socket and connects it to the specified port number on the named host.

What Is a Socket?

- 소켓을 생성하면 연결은 자동으로 이루어진다. 연결 시 오류가 발생하면 `IOException`이 발생한다.
- TCP 연결에는 로컬IP 주소와 원격IP 주소, 로컬포트, 원격 포트 등이 사용된다. `Socket`이 생성되어 원격호스트에 접속 할 때 대개는 사용하지 않는 로컬 포트가 사용된다.

Error handling

Error handling

- Error handling is done by the class called *SocketException* which extends *IOException* class.
- One possible cause is that the local port you are using for is already in use. Another cause is that the user *cannot bind* to that particular port.

Error handling

- **BindException:**
 - *The local port is in use*, or the requested bind address couldn't be assigned locally.
- **ConnectException:**
 - This exception is raised when a *connection is refused at the remote host* (i.e., *no process is listening on that port*).
- **NoRouteToHostException:**
 - The connect attempt *timed out*, or the remote host is otherwise *unreachable*.

EchoLoopClient.java

//Echo server is a well-known service that clients can rendezvous with on port 7.

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

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

```
public class EchoLoopClient {
```

```
    public static void main(String [] args) {
```

```
        Socket mySocket;
```

```
        String hostName, aLine;
```

```
        BufferedReader fromSocket, userInput;
```

```
        PrintWriter toSocket;
```

```
        try {
```

```
            if(args.length > 0)    hostName = args[0];
```

```
            else                    hostName = "localhost";
```

```
            mySocket = new Socket(hostName, 7);
```

```
            fromSocket = new BufferedReader(new  
InputStreamReader(mySocket.getInputStream()));
```

```
            toSocket = new PrintWriter(mySocket.getOutputStream());
```

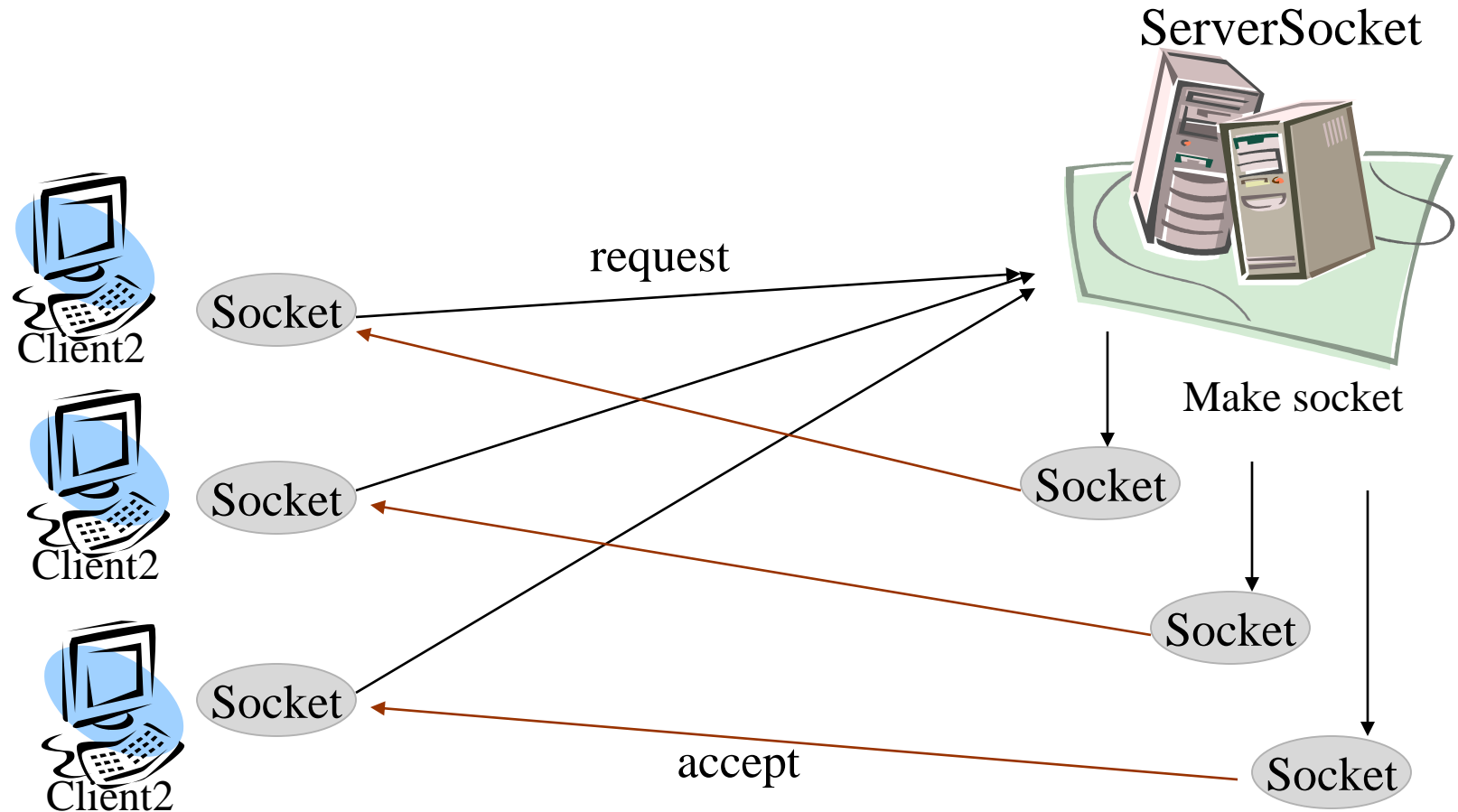

EchoLoopClient.java

```
        userInput = new BufferedReader(new
InputStreamReader(System.in));
        while(true) {
            aLine = userInput.readLine();
            if (aLine.equals(".")) break;
            toSocket.println(aLine);
            toSocket.flush();
            System.out.println(fromSocket.readLine());
        }
        fromSocket.close();
        toSocket.close();
    }
    catch(UnknownHostException e) { System.out.println(e); }
    catch(Exception e) {          System.out.println(e); }
}
}
```

Server Networking

- About ServerSocket

ServerSocket Class



ServerSocket Class

- This class implements server sockets.
- A server socket waits for requests to come in over the network. It performs some operation based on that request, and then possibly returns a result to the requester.

ServerSocket Class

- 서버 구현의 기본은 특정한 로컬 포트에 대해 ServerSocket 객체를 열고 클라이언트가 연결해 오기를 기다리는 것이다.
- 클라이언트가 연결해 올 때마다 ServerSocket은 각각의 클라이언트 연결에 대해 Socket 객체를 생성한다.
- ServerSocket은 대개 InputStream이나 OutputStream을 통해 클라이언트와 데이터를 주고 받는다.
- Port번호는 1~65535까지 사용 가능하며 이중 1~1023 까지는 시스템 서비스 용으로 예약 되어있다.

ServerSocket Class

- 소켓 생성시 포트 번호를 0으로 주는 것은 시스템에서 알아서 포트를 할당 하라는 의미이다.
- ServerSocket 객체가 생성되자마자 운영체제가 클라이언트로 부터 연결을 받을 준비를 한다. 클라이언트가 시도한 연결은 Queue에 들어가며 서버가 accept 메소드를 호출하면 하나씩 큐에서 빠져 나온다.
- ServerSocket 생성시 몇 개의 연결을 Queue에 넣을 것인지 지정 할 수 있다.

ServerSocket Class

- Constructor
 - `ServerSocket(int port)`
 - Creates a server socket on a specified port.
 - The maximum queue length for incoming connection indications (a request to connect) is set to 50. If a connection indication arrives when the queue is full, the connection is refused.

ServerSocket Class

- Constructor
 - `ServerSocket(int port, int backlog)`
 - Creates a server socket and binds it to the specified local port number, with the specified backlog.
 - The maximum queue length for incoming connection indications (a request to connect) is set to the backlog parameter. If a connection indication arrives when the queue is full, the connection is refused.

Look for Local Ports

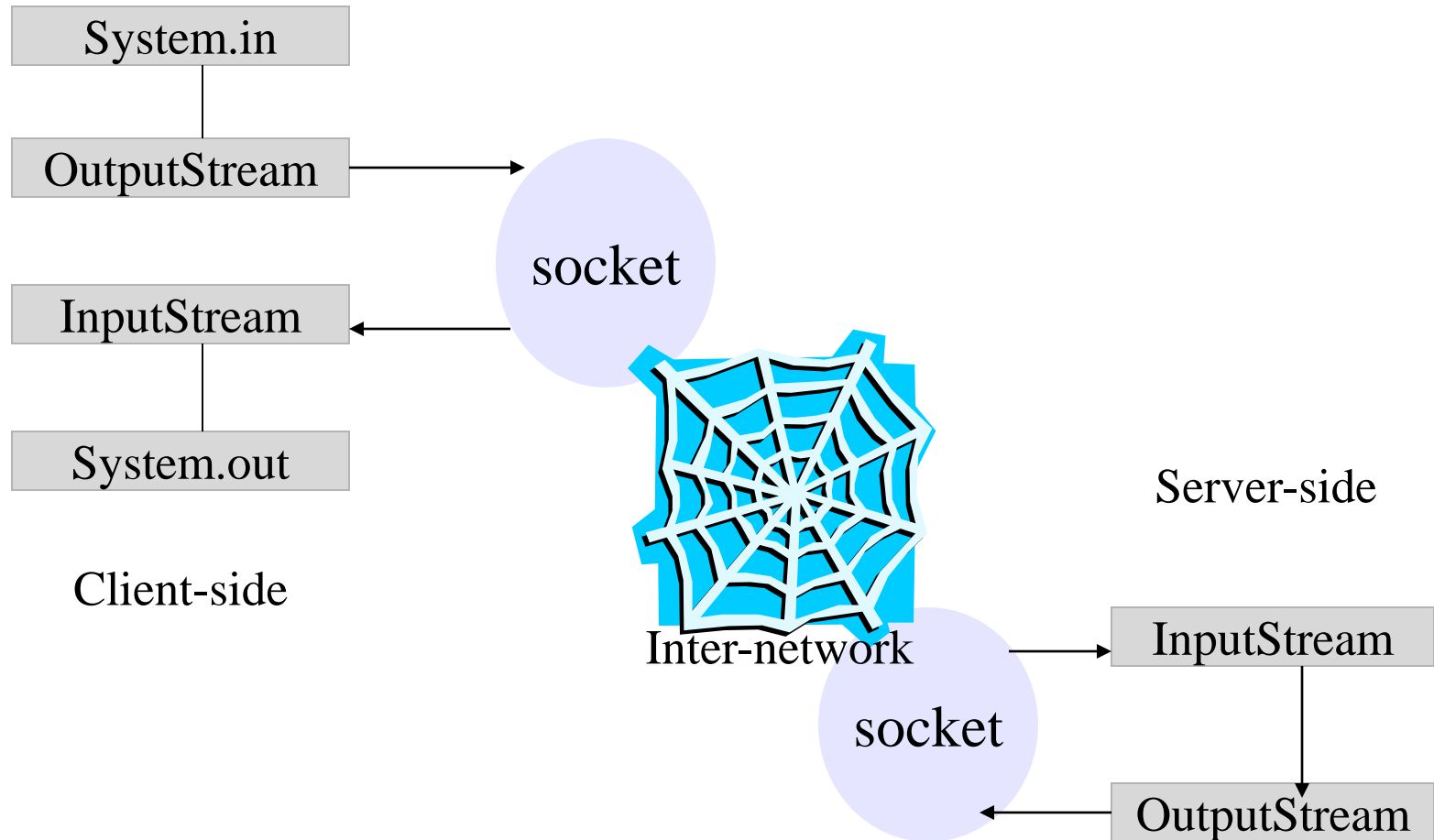
LocalPortScanner.java

```
import java.net.*;
import java.io.*;
public class LocalPortScanner {
    public static void main(String[] args) {
        for (int port = 1; port <= 65535; port++) {
            try {
                ServerSocket server = new ServerSocket(port);
            }
            catch (IOException e) {
                System.out.println("There is a server on port " +
port + ".");
            }
        }
    }
}
```

Custom Socket

Echo Server and Client

Echo Server and Client



EchoClient

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

class EchoClient
{
    public static void main( String[] args )
        throws IOException
    {
        Socket sock = null;
        try
        {
            sock = new Socket(args[0], Integer.parseInt(args[1]));
            System.out.println(sock + ": 연결 됨");
            OutputStream toServer = sock.getOutputStream();
            InputStream fromServer = sock.getInputStream();
        }
    }
}
```

EchoClient

```
byte[] buf = new byte[1024];
int count;
while( (count = System.in.read(buf)) != -1 )
{
    toServer.write( buf, 0, count );
    count = fromServer.read( buf );
    System.out.write( buf, 0, count );
}
toServer.close();
while( (count = fromServer.read(buf)) != -1 )
    System.out.write( buf, 0, count );
System.out.close();
System.out.println(sock + ": 연결 종료");
} catch( IOException ex )
```

EchoClient

```
{
    System.out.println("연결 종료 (" + ex + ")");
} finally
{
    try
    {
        if ( sock != null )
            sock.close();
    } catch( IOException ex ) {}
}
}
```


EchoServer

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

class EchoServer
{
    public static void main( String[] args )
        throws IOException
    {
        ServerSocket serverSock =
            new ServerSocket( Integer.parseInt(args[0]) );
        System.out.println(serverSock + ": 서버 소켓 생성");
        while(true)
        {
            Socket sock = null;
```

EchoServer

```
try    {
    sock = serverSock.accept();
    System.out.println(sock + ": 연결 됨");
    InputStream fromClient = sock.getInputStream();
    OutputStream toClient = sock.getOutputStream();
    byte[] buf = new byte[1024];
    int count;
    while( (count = fromClient.read(buf)) != -1 ) {
        toClient.write( buf, 0, count );
        System.out.write(buf, 0, count);
    }
    toClient.close();
    System.out.println(sock + ": 연결 종료");
} catch( IOException ex )
```

EchoServer

```
{
    System.out.println(sock + ": 연결 종료 (" + ex + ")");
} finally
{
    try
    {
        if ( sock != null )
            sock.close();
    } catch( IOException ex ) {}
}
}
}
```

Multi-Thread Echo Client, Server

Multi-Thread Echo Client, Server

- 이전 예제인 EchoServer의 경우 동시에 여러 개의 클라이언트를 처리하는데 있어서는 read 메소드의 Blocking으로 인해 어려움이 있다.
- 즉 동시에 여러 클라이언트의 요구를 처리 하지 못한다.
- 이문제를 해결하기 위해 다중 스레딩(Multi-Threading)을 구현한 서버를 사용하는 것이다.
- 다중 스레드 서버는 클라이언트가 접속 할때 마다 1개 이상의 스레드를 만들어 돌리기 때문에 블로킹 I/O 문제를 해결해 준다.
- 즉 메인 스레드는 클라이언트의 연결을 받기만 하고 클라이언트와 데이터를 주고 받는 일은 별도의 Thread에서 처리 하도록 구성한다.

MultiThreadEchoServer.java

```
//여러개의 클라이언트를 처리하기 위해
//멀티스레드로 구현한 MultiThreadEchoServer
import java.io.*;
import java.net.*;
class MultiThreadEchoServer extends Thread {
    protected Socket sock;
    //----- Constructor
    MultiThreadEchoServer (Socket sock) { this.sock = sock;}

    //-----
    public void run() {
        try {
            System.out.println(sock + ": 연결됨");
```

MultiThreadEchoServer.java

```
InputStream fromClient = sock.getInputStream();
OutputStream toClient = sock.getOutputStream();
byte[] buf = new byte[1024];    int count;
while( (count = fromClient.read(buf)) != -1 ) {
    toClient.write( buf, 0, count );
    System.out.write(buf, 0, count);
}
toClient.close();
System.out.println(sock + ": 연결 종료");
}
catch( IOException ex ) {
    System.out.println(sock + ": 연결 종료 (" + ex +
    ")");
}
```

MultiThreadEchoServer.java

```
        finally {
            try {
                if ( sock != null ) sock.close();
            }
            catch( IOException ex ) {}
        }
    }

//-----
public static void main( String[] args ) throws IOException {
    ServerSocket serverSock = new
    ServerSocket( Integer.parseInt(args[0]) );
    System.out.println(serverSock + ": 서버 소켓 생성");
}
```


MultiThreadEchoServer.java

```
while(true)    {
    Socket client = serverSock.accept();
    MultiThreadEchoServer myServer = new
MultiThreadEchoServer(client);
    myServer.start();
}
}
}
```

Thread pooling

Thread pooling Server

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

class PoolEchoServer
{
    public static void main( String[] args )
        throws IOException
    {
        ServerSocket serverSock =
            new ServerSocket( Integer.parseInt(args[0]) );
        for( int i = 0; i < 2; i++ )
            new Connection(serverSock);
    }
}
```

Thread pooling Server

```
static class Connection extends Thread
{
    private ServerSocket serverSock;

    public Connection( ServerSocket serverSock )
    {
        this.serverSock = serverSock;
        start();
    }

    public void run()
    {
        while(true)
```

Thread pooling Server

```
{
    Socket sock = null;
    byte[] buf = new byte[1024];
    int count;
    try
    {
        sock = serverSock.accept();
        System.out.println(sock + ": 연결 됨");
        InputStream fromClient = sock.getInputStream();
        OutputStream toClient = sock.getOutputStream();
        while( (count = fromClient.read(buf)) != -1 )
            toClient.write( buf, 0, count );
        toClient.close();
        System.out.println(sock + ": 연결 종료");
    }
}
```

Thread pooling Server

```
    } catch( IOException ex )
    {
        System.out.println(sock + ": 연결 종료 (" + ex + ")");
    } finally
    {
        try
        { sock.close();
          } catch( IOException ex ) {}
    }
}
}
}
```

Socket based Chatting Client and Server

Referenced from :

Java Network Programming, Second Edition

Merlin Hughes, Michael Shoffner, Derek Hamner

Manning Publications Company; ISBN 188477749X

Chatting Server

ChatServer.java

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

public class ChatServer {
    public static void main (String args[]) throws IOException {
        if (args.length != 1)
            throw new IllegalArgumentException ("Syntax: ChatServer
        <port>");

        int port = Integer.parseInt (args[0]);
        ServerSocket server = new ServerSocket (port);
```

ChatServer.java

```
System.out.println(server + " Server Started...");
while (true) {
    Socket client = server.accept ();
    System.out.println ("Accepted from " + client.getInetAddress ()
+ " 입장완료...");
    ChatHandler handler = new ChatHandler (client);
    handler.start ();
}
}
```

ChatHandler.java

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

public class ChatHandler implements Runnable {
    protected Socket socket;

    public ChatHandler (Socket socket) {
        this.socket = socket;
    }
    protected DataInputStream dataIn;
    protected DataOutputStream dataOut;
    protected Thread listener;
```

ChatHandler.java

```
public synchronized void start () {  
    if (listener == null) {  
        try {  
            dataIn = new DataInputStream  
                (new BufferedInputStream (socket.getInputStream ()));  
            dataOut = new DataOutputStream  
                (new BufferedOutputStream (socket.getOutputStream ()));  
            listener = new Thread (this);  
            listener.start ();  
        } catch (IOException ignored) {  
        }  
    }  
}
```

ChatHandler.java

```
public synchronized void stop () {  
    if (listener != null) {  
        try {  
            if (listener != Thread.currentThread ())  
                listener.interrupt ();  
            listener = null;  
            dataOut.close ();  
        } catch (IOException ignored) {  
        }  
    }  
}
```

ChatHandler.java

```
//protected static Vector handlers = new Vector ();
protected static ArrayList handlers = new ArrayList();
public void run () {
    try {
        //handlers.addElement (this);
        handlers.add(this);
        while (!Thread.interrupted ()) {
            String message = dataIn.readUTF ();
            broadcast (message);
        }
    }
    catch (EOFException ignored) {
    }
}
```

ChatHandler.java

```
catch (IOException ex) {  
    if (listener == Thread.currentThread ())  
        ex.printStackTrace ();  
}  
finally {  
    //handlers.removeElement (this);  
    handlers.remove(handlers.indexOf(this));  
}  
stop ();  
}
```

```
protected void broadcast (String message) {  
    System.out.println(message);  
}
```

ChatHandler.java

```
synchronized (handlers) {  
    //Enumeration enum = handlers.elements ();  
    Iterator enum = handlers.iterator();  
    //while (enum.hasMoreElements ()) {  
    while (enum.hasNext()) {  
        ChatHandler handler = (ChatHandler) enum.next();  
        try {  
            handler.dataOut.writeUTF (message);  
            handler.dataOut.flush ();  
        } catch (IOException ex) {  
            handler.stop ();  
        }  
    }  
}  
}
```

Chatting Client

ChatClient.java

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

```
public class ChatClient implements Runnable, WindowListener,
    ActionListener {
    protected String host;
    protected int port;
    protected Frame frame;
    protected TextArea output;
    protected TextField input;
```

ChatClient.java

```
public ChatClient (String host, int port) {  
    this.host = host;  
    this.port = port;  
    frame = new Frame ("ChatClient [" + host + ':' + port + "]);  
    frame.addWindowListener (this);  
    output = new TextArea ();  
    output.setEditable (false);  
    input = new TextField ();  
    input.addActionListener (this);  
    frame.add ("Center", output);  
    frame.add ("South", input);  
    frame.pack ();  
}
```

ChatClient.java

```
protected DataInputStream dataIn;
protected DataOutputStream dataOut;
protected Thread listener;

public synchronized void start () throws IOException {
    if (listener == null) {
        Socket socket = new Socket (host, port);
        try {
            dataIn = new DataInputStream
                (new BufferedInputStream (socket.getInputStream ()));
            dataOut = new DataOutputStream
                (new BufferedOutputStream (socket.getOutputStream ()));
        }
    }
}
```

ChatClient.java

```
catch (IOException ex) {  
    socket.close ();  
    throw ex;  
}  
listener = new Thread (this);  
listener.start ();  
frame.setVisible (true);  
}  
}  
  
public synchronized void stop () throws IOException {  
    frame.setVisible (false);
```

ChatClient.java

```
if (listener != null) {  
    listener.interrupt ();    listener = null;  
    dataOut.close ();  
}  
}  
public void run () {  
    try {  
        while (!Thread.interrupted ()) {  
            String line = dataIn.readUTF ();  
            output.append (line + "\n");  
        }  
    } catch (IOException ex) {    handleIOException (ex);    }  
}
```

ChatClient.java

```
protected synchronized void handleIOException (IOException ex)
{
    if (listener != null) {
        output.append (ex + "\n");
        input.setVisible (false);    frame.validate ();
        if (listener != Thread.currentThread ())
            listener.interrupt ();
        listener = null;
        try {
            dataOut.close ();
        }
        catch (IOException ignored) {    }
    }
}
```

ChatClient.java

```
public void windowOpened (WindowEvent event) {  
    input.requestFocus ();  
}  
public void windowClosing (WindowEvent event) {  
    try {  
        dataOut.writeUTF ("전 이만 나갑니다... Bye...");  
        stop ();  
    } catch (IOException ex) {  
        ex.printStackTrace ();  
    }  
}
```


ChatClient.java

```
public void windowClosed (WindowEvent event) {}
public void windowIconified (WindowEvent event) {}
public void windowDeiconified (WindowEvent event) {}
public void windowActivated (WindowEvent event) {}
public void windowDeactivated (WindowEvent event) {}
public void actionPerformed (ActionEvent event) {
    try {
        input.selectAll ();
        dataOut.writeUTF (event.getActionCommand ());
        dataOut.flush ();
    } catch (IOException ex) {
        handleIOException (ex);
    }
}
```

ChatClient.java

```
public static void main (String[] args) throws IOException {  
    if ((args.length != 1) || (args[0].indexOf(':') < 0))  
        throw new IllegalArgumentException ("Syntax: ChatClient  
        <host>:<port>");  
    int idx = args[0].indexOf(':');  
    String host = args[0].substring (0, idx);  
    int port = Integer.parseInt (args[0].substring (idx + 1));  
    ChatClient client = new ChatClient (host, port);  
    client.start ();  
}  
}
```

Socket을 이용한 SMTP Client

- SendMail.java
- SMTPException.java

SendMail.java

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

public class SendMail implements Serializable{
    private Socket smtp;    //SMTP서버에 접속하기 위한것
    private BufferedReader input;//서버의 응답을 저장하기 위한것
    private PrintStream output;// 서버와의 통신을 위한 PrintStream
    private String smtpServer ; //SMTP서버 주소
    private String serverReply;    // 서버응답
    private int port = 25;        // Default SMTP Port
    private String from = "mylife688@hanmail.net";
```

SendMail.java

```
private String to = "mylife688@hanmail.net";  
private String subject = "메일 연습입니다...";  
private String message = " 메일 본문 내용 입니다~";  
  
public static void main(String[] args) throws Exception{  
    SendMail mail = new SendMail();  
    mail.smtpServer = args[0];  
    mail.smtp = new Socket(args[0], mail.port);  
    mail.send();  
}
```

SendMail.java

// 메일을 보내기 위한 모든 메소드를 Call

```
public void send() throws SMTPException{  
    connect();  
    hail(from, to);  
    sendMessage(from, to, subject, message);  
    logout();  
}
```

// PrintStream을 서버에 보냄, 접속후 서버로 부터의 응답을
파악(200인 경우 서비스 준비OK, input을 위한
BufferedReader와 함께 새로운 서버 소켓을 생성

```
public void connect() throws SMTPException {  
    try {
```

SendMail.java

```
smtp = new Socket(smtpServer, port);
input = new BufferedReader(new
InputStreamReader(smtp.getInputStream()));
//접속된 메일서버에게 출력을 보내기 위한 스트림을 생성한다.
output = new PrintStream(smtp.getOutputStream());
serverReply = input.readLine();
if (serverReply.charAt(0) == '2' || serverReply.charAt(0) == '3') { }
else { throw new SMTPException("Error connecting to SMTP server
    " + smtpServer + "on port " + port);
    }
}
catch(Exception e) {
    throw new SMTPException(e.getMessage());
}
}
```

SendMail.java

//메일 보낸이, 받는이등을 메일서버에게 알림...

```
public void hail(String from, String to) throws
SMTPException {
    if (submitCommand("HELO " + smtpServer))
        throw new SMTPException("Error occurred
during HELO command.");
    if (submitCommand("MAIL FROM: " + from))
        throw new SMTPException("Error during MAIL
command.");
    if (submitCommand("RCPT TO: " + to))
        throw new SMTPException("Error during RCPT
command.");
}
```


SendMail.java

// 실제 메일을 전송, DATA 명령은 SMTP 서버에게 메일을 보낸다고 일리는 기능

public void sendMessage(String from, String to, String subject, String message) throws

SMTPException {

 Date ldate_today = new Date(System.currentTimeMillis());

 SimpleDateFormat lgmt_date = new

 SimpleDateFormat("d MMM yyyy HH:mm:ss 'GMT'");

 lgmt_date.setTimeZone(TimeZone.getTimeZone("GMT"));

 lgmt_date.format(ldate_today);

SendMail.java

```
try {
    if (submitCommand("DATA"))
        throw new SMTPException("Error during DATA
command.");
    String header = "From: " + from + "\r\n";
    header += "To: " + to + "\r\n";
    header += "Subject: " + subject + "\r\n";
    header += "Date: " + lgmt_date + "\r\n\r\n";
    if (submitCommand(header + message + "\r\n."))
        throw new SMTPException("Error during mail
transmission.");
}
catch (Exception e)    {    }
}
```

SendMail.java

//서버에 명령을 보낸 후 응답을 받다 Boolean값을 return, true인 경우에러

```
private boolean submitCommand(String command) throws
SMTPException {
    try {
        output.print(command + "\r\n");
        serverReply = input.readLine();
        if (serverReply.charAt(0) == '4' || serverReply.charAt(0) == '5')
//전송실패등..
            return true;
        else
            return false;
    }
    catch(Exception e) { throw new
SMTPException(e.getMessage()); }
}
```

SendMail.java

// SMTP 서버로 부터 LogOut

```
public void logout() throws SMTPException {
```

```
    try {
```

```
        if (submitCommand("Quit"))
```

```
            throw new SMTPException("Error during QUIT  
command");
```

```
        input.close();
```

```
        output.flush();
```

```
        output.close();
```

```
        smtp.close();
```

```
    }
```

```
    catch(Exception e) { }
```

```
}
```

```
}
```

SMTPException.java

//SendMail.java 에서 예외상황 발생시 에러를 Catch 하기위한 Class
//error내용을 String으로 저장한후 getMessage() 메소드에 돌려준다.

```
public class SMTPException extends Exception {  
    private String message;  
    public SMTPException(String message) {  
        this.message = message;  
    }  
    // 에러 메시지를 Return  
    public String getMessage() {  
        return message;  
    }  
}
```

Using Serialization with Sockets

Using Serialization with Sockets

- Shows how to use sockets and serialization for sending and receiving objects.
- In this example, the current date is sent from the client to the server. The file `Client.java` provides the code for sending the date and the file `Server.java` provides the code for receiving the date. The Server is the local host machine.

Example 1.

Simple Socket Communication with
Serialization.

Example 1. (Server.java)

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

public class Server {

    public static void main(String args[]) {

        ServerSocket ser = null;
        Socket soc = null;
        String str = null;
        Date d = null;
```

Example 1. (Server.java)

```
try {  
    ser = new ServerSocket(8020);  
  
    soc = ser.accept();  
    InputStream o = soc.getInputStream();  
    ObjectInput s = new ObjectInputStream(o);  
    str = (String) s.readObject();  
    d = (Date) s.readObject();  
    s.close();  
  
    // print out what we just received  
    System.out.println(str);  
    System.out.println(d);
```

Example 1. (Server.java)

```
    } catch (Exception e) {  
        System.out.println(e.getMessage());  
        System.out.println("Error during serialization");  
        System.exit(1);  
    }  
}  
}
```

Example1. (Client.java)

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

public class Client {

    public static void main(String args[]) {

        try {
            // Create a socket
            Socket soc = new Socket(InetAddress.getLocalHost(), 8020);
```

Example1. (Client.java)

```
// Serialize today's date to a outputstream associated to the socket
OutputStream o = soc.getOutputStream();
ObjectOutput s = new ObjectOutputStream(o);

s.writeObject("Today's date");
s.writeObject(new Date());
s.flush();
s.close();
    } catch (Exception e) {
System.out.println(e.getMessage());
System.out.println("Error during serialization");
System.exit(1);
    }
}
}
```

Example2.

NameCard Passing between Client
and Server.

Example2. (NameCard.java)

```
public class NameCard implements java.io.Serializable
{
    public static int SAVE = 0;
    public static int RETRIEVE = 1;
    public String name;
    public String address;
    public String phone;
    public int age;
    public String toString()
    {   return "[이름: " + name + ", 주소: " + address
        + ", 전화번호: " + phone + ", 나이: " + age + "];"
    }
}
```

Example2. (NameCardClient.java)

```
import java.awt.*;
```

```
import java.awt.event.*;
```

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

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

```
public class NameCardClient extends Frame {
```

```
    public static void main(String[] args) throws IOException {  
        new NameCardClient(args[0], Integer.parseInt(args[1]));  
    }
```

```
    Socket socket;
```

```
    ObjectOutputStream toServer;
```


Example2. (NameCardClient.java)

```
ObjectInputStream fromServer;
```

```
TextField name = new TextField(30);
```

```
TextField address = new TextField(30);
```

```
TextField phone = new TextField(30);
```

```
TextField age = new TextField(30);
```

```
Button send = new Button("저장...");
```

```
Button retrieve = new Button("검색...");
```

```
public NameCardClient(String host, int port) throws IOException {
```

```
    socket = new Socket(host, port);
```

```
    toServer = new
```

```
ObjectOutputStream(socket.getOutputStream());
```

```
    fromServer = new ObjectInputStream(socket.getInputStream());
```

Example2. (NameCardClient.java)

```
Panel p1 = new Panel(new GridLayout(0,1));
p1.add(new Label("이름:"));
p1.add(new Label("주소:"));
p1.add(new Label("전화번호:"));
p1.add(new Label("나이:"));
Panel p2 = new Panel(new GridLayout(0,1));
p2.add(name);
p2.add(address);
p2.add(phone);
p2.add(age);
Panel p3 = new Panel();
p3.add(send);
p3.add(retrieve);
```

Example2. (NameCardClient.java)

```
add(p1, BorderLayout.WEST);  
add(p2, BorderLayout.CENTER);  
add(p3, BorderLayout.SOUTH);
```

```
send.addActionListener(sendHandler);  
retrieve.addActionListener(retrieveHandler);  
addWindowListener(new ExitListener());
```

```
pack();  
setVisible(true);
```

```
}
```

```
ActionListener sendHandler = new ActionListener() {  
    public void actionPerformed(ActionEvent ev) {
```

Example2. (NameCardClient.java)

```
try {  
    NameCard nc = new NameCard();  
    nc.name = name.getText();  
    nc.address = address.getText();  
    nc.phone = phone.getText();  
    nc.age = Integer.parseInt(age.getText());  
  
    toServer.writeInt(NameCard.SAVE);  
    toServer.writeObject(nc);  
    toServer.flush();  
}  
catch(IOException e) { e.printStackTrace(); }  
};
```

Example2. (NameCardClient.java)

```

        ActionListener retrieveHandler = new ActionListener() {
        public void actionPerformed(ActionEvent ev) {
            try {
                toServer.writeInt(NameCard.RETRIEVE);
                toServer.writeObject(name.getText());
                toServer.flush();

                NameCard nc =
(NameCard)fromServer.readObject();
                if (nc == null) {
                    System.out.println("검색 실패~");
                    return;
                }
            }
        }
    }

```

Example2. (NameCardClient.java)

```
        address.setText(nc.address);
        phone.setText(nc.phone);
        age.setText(String.valueOf(nc.age));
    }
    catch(IOException e) { e.printStackTrace(); }
    catch(ClassNotFoundException e)
{ e.printStackTrace(); }
    }
};
```

```
class ExitListener extends WindowAdapter {
    public void windowClosing(WindowEvent e) {
        System.exit(0);
    }
}
}
```

Example2. (NameCardServer.java)

```
import java.util.*;
import java.io.*;
import java.net.*;
class NameCardServer extends Thread {
    public static void main(String[] args) throws IOException {
        ServerSocket server = new
        ServerSocket(Integer.parseInt(args[0]));
        System.out.println("Server Cocket Created..." + server);
        while(true) {
            Socket socket = server.accept();
            System.out.println("Client :::::" + socket);
            new NameCardServer(socket);
        }
    }
}
```

Example2. (NameCardServer.java)

```
static ArrayList database = new ArrayList();
```

```
Socket socket;
```

```
ObjectInputStream fromClient;
```

```
ObjectOutputStream toClient;
```

```
public NameCardServer(Socket socket) throws IOException {
```

```
    this.socket = socket;
```

```
    fromClient = new
```

```
ObjectInputStream(socket.getInputStream());
```

```
    toClient = new
```

```
ObjectOutputStream(socket.getOutputStream());
```

```
    start();
```

```
}
```


Example2. (NameCardServer.java)

```
public void run() {  
    try {  
        while(true) {  
            int command = fromClient.readInt();  
            if (command == NameCard.SAVE) save();  
            else if (command == NameCard.RETRIEVE)  
retrieve();  
        }  
    }  
    catch(IOException e) { e.printStackTrace(); }  
    catch(ClassNotFoundException e) { e.printStackTrace(); }
```

Example2. (NameCardServer.java)

```
        finally {  
            try {  
                if (socket != null) socket.close();  
            }  
            catch(IOException e) {}  
        }  
    }  
  
    void save() throws IOException, ClassNotFoundException {  
        NameCard data = (NameCard)fromClient.readObject();  
        database.add(data);  
        System.out.println("save complete :::::" + data);  
    }
```

Example2. (NameCardServer.java)

```
void retrieve() throws IOException, ClassNotFoundException {  
    String name = (String)fromClient.readObject();  
  
    synchronized(database) {  
        for(int i=0; i < database.size(); i++) {  
            NameCard storedNameCard = (NameCard)database.get(i);  
            if (storedNameCard.name.equals(name)) {  
                toClient.writeObject(storedNameCard);  
                System.out.println("retrieve complete :::::[\" + name + \"]");  
                toClient.flush();  
            }  
        }  
    }  
}
```

Example2. (NameCardServer.java)

```
        return;  
    }  
}  
}  
}  
toClient.writeObject(null);  
toClient.flush();  
}  
}
```

Example 3.

Chatting Client and Server using Object
Serialization.

ChatRequestHandler.java

```
public interface ChatRequestHandler
{
    void enter( String name );
    void sendChat( String message );
    void getUserList();
    void leave();
}
```

ChatRequest.java

```
public abstract class ChatRequest implements java.io.Serializable
{
    public abstract void execute(ChatRequestHandler req);
}
```

Enter.java

```
public class Enter extends ChatRequest
{
    String userName;
    public Enter(String userName)
    {   this.userName = userName;
    }
    public void execute(ChatRequestHandler req)
    {   req.enter(userName);
    }
}
```


GetUserList.java

```
public class GetUserList extends ChatRequest
{
    public void execute(ChatRequestHandler req)
    { req.getUserList();
    }
}
```

SendChat.java

```
public class SendChat extends ChatRequest
{
    String message;
    public SendChat(String message)
    {   this.message = message;
    }
    public void execute(ChatRequestHandler req)
    {   req.sendChat(message);
    }
}
```

Leave.java

```
public class Leave extends ChatRequest
{
    public void execute(ChatRequestHandler req)
    {   req.leave();
    }
}
```

ChatResponseHandler.java

```
public interface ChatResponseHandler
{
    void isLive();
    void enterAck( boolean success );
    void userEntered( String name );
    void chatReceived( String name, String message );
    void userListReceived( String[] userList );
    void userLeft( String name );
    void userDisconnected( String name );
}
```

ChatResponse.java

- `public abstract class ChatResponse implements java.io.Serializable`
- `{`
- `public abstract void execute(ChatResponseHandler res);`
- `}`

EnterAck.java

```
public class EnterAck extends ChatResponse
{
    boolean success;
    public EnterAck( boolean success )
    {   this.success = success;
    }
    public void execute(ChatResponseHandler res)
    {   res.enterAck( success );
    }
}
```

UserEntered.java

```
public class UserEntered extends ChatResponse
{
    String name;
    public UserEntered( String name )
    {   this.name = name;
    }
    public void execute(ChatResponseHandler res)
    {   res.userEntered( name );
    }
}
```

UserListReceived.java

```
public class UserListReceived extends ChatResponse
{
    String[] userList;
    public UserListReceived(String[] userList)
    {   this.userList = userList;
    }
    public void execute(ChatResponseHandler res)
    {   res.userListReceived( userList );
    }
}
```


CharReceived.java

```
public class ChatReceived extends ChatResponse
{
    String name, message;
    public ChatReceived( String name, String message )
    {   this.name = name;
        this.message = message;
    }
    public void execute(ChatResponseHandler res)
    {   res.chatReceived( name, message );
    }
}
```

IsLive.java

```
public class IsLive extends ChatResponse
{
    public void execute(ChatResponseHandler res)
    {    res.isLive();
    }
}
```

UserLeft.java

```
public class UserLeft extends ChatResponse
{
    String name;
    public UserLeft( String name )
    {   this.name = name;
    }
    public void execute(ChatResponseHandler res)
    {   res.userLeft( name );
    }
}
```

UserDisconnected.java

```
public class UserDisconnected extends ChatResponse
{
    String name;
    public UserDisconnected( String name )
    {   this.name = name;
    }
    public void execute(ChatResponseHandler res)
    {   res.userDisconnected( name );
    }
}
```

ChatClient.html

```
<html>
<head>
<META HTTP-EQUIV="Content-Type" CONTENT="text/html; charset=euc-kr">
</head>

<body>
<h1>채팅 애플릿</h1>
<applet code="ChatClient.class" width=80% height=80%>
</applet>
</body>
</html>
```

ChatClient.java

ChatClient.java

```
import java.util.*;
import java.io.*;
import java.net.*;
import java.awt.*;
import java.awt.List;
import java.awt.event.*;
import java.applet.*;

public class ChatClient extends Applet
    implements ActionListener, ChatResponseHandler, Runnable
{
    public static void main( String[] args )
    {
        final ChatClient client = new ChatClient();
```

ChatClient.java

```
client.userName.setText( System.getProperty( "user.name" ) );  
try  
{ client.hostName.setText(  
    InetAddress.getLocalHost().getHostName() );  
} catch( UnknownHostException e ) { }  
}
```

```
Socket socket;  
ObjectInputStream fromServer;  
ObjectOutputStream toServer;  
Thread receiver;
```

```
Frame frame = new Frame( "채팅 클라이언트" );  
TextArea view = new TextArea(30, 80);
```


ChatClient.java

```
TextField input = new TextField();
Panel cntlPanel = new Panel();
Button connectBtn = new Button( "연결" );
TextField userName = new TextField( "", 6 );
TextField hostName = new TextField( "", 20 );
TextField portNum = new TextField( "10000", 5 );
List userList = new List();

public ChatClient()
{
    view.setEditable( false );
    input.setEditable( false );

    // 이벤트 처리기 등록
```

ChatClient.java

```
connectBtn.addActionListener( this );
input.addActionListener( this );

frame.addWindowListener( new WindowAdapter()
{   public void windowClosing( WindowEvent ev )
    {
        try
        {
            send( new Leave() );
        } catch(IOException ex) {
        } catch(NullPointerException ex) {}
        frame.dispose();
        System.exit(0);
    }
});
```

ChatClient.java

// 컴포넌트 배치 및 화면 장식

```
Panel p;  
p = new Panel(new BorderLayout());  
p.add( cntlPanel, "North" );  
cntlPanel.add( connectBtn );  
cntlPanel.add( new Label( "사용자 이름: " ) );  
cntlPanel.add( userName );  
cntlPanel.add( new Label( "연결 호스트 이름: " ) );  
cntlPanel.add( hostName );  
cntlPanel.add( new Label( "연결 포트: " ) );  
cntlPanel.add( portNum );  
p.add( view, "Center" );  
frame.add( p, "Center" );
```

ChatClient.java

```
p = new Panel(new BorderLayout());  
p.add( new Label("사용자", Label.CENTER), "North" );  
p.add( userList, "Center" );  
frame.add( p, "East" );
```

```
frame.add( input, "South" );
```

```
frame.pack();  
frame.setVisible(true);
```

```
}
```

```
// 애플릿용
```

```
public void init()  
{
```

ChatClient.java

```
    hostName.setText( getCodeBase().getHost() );
    hostName.setEditable(false);
}

public void actionPerformed((ActionEvent ev)
{
    Object source = ev.getSource();
    if ( source == connectBtn )
    {
        if ( connectBtn.getLabel().equals( "연결" ) )
        {
            if ( receiver != null )
                return;
            connect();
        }
    }
}
```

ChatClient.java

```
    } else
    {
        try
        {
            send( new Leave() );
        } catch(IOException e)
        {
            disconnectError();
        }
    }
} else if ( source == input )
{
    try
    {
        send( new SendChat(input.getText()) );
    }
```

ChatClient.java

```
        input.setText("");
    } catch(IOException e)
    {
        disconnectError();
    }
}

void connect()
{
    String name = userName.getText();
    if ( name.equals("") )
    {   print( "*** 사용자 이름을 입력하십시오. ***" );
        userName.requestFocus();
    }
}
```

ChatClient.java

```
        return;  
    }  
  
    try  
    {  
        socket = new Socket( hostName.getText(),  
                             Integer.parseInt(portNum.getText()) );  
  
        OutputStream os = socket.getOutputStream();  
        toServer = new ObjectOutputStream( os );  
        send( new Enter(name) );  
        input.setText("");  
  
        InputStream is = socket.getInputStream();  
        fromServer = new ObjectInputStream( is );
```


ChatClient.java

```
print( "*** 서버와 연결되었습니다. ***" );

input.setEditable( true );
connectBtn.setLabel( "연결 해제" );
connectBtn.invalidate();
cntlPanel.validate();
input.requestFocus();

receiver = new Thread(this);
receiver.start();
} catch( IOException ex )
{
    print( "*** 연결 실패 *** " + ex);
    hostName.requestFocus();
}
```

ChatClient.java

```
    } catch( NumberFormatException ex )
    {   print( "*** 연결 실패: 포트 넘버가 잘못되었음 ***");
        portNum.requestFocus();
    }
}

void print( String line )
{
    view.append( line + '\n' );
}

public void run()
{   try
    {
        send( new GetUserList() );
```

ChatClient.java

```
// 서버로부터의 메시지 처리 루프.
while( receiver != null )
{
    ChatResponse mesg = (ChatResponse) fromServer.readObject();
    mesg.execute(this);
}
} catch( IOException e )
{
    disconnectError();
} catch( ClassNotFoundException e )
{
    print("*** 프로그램 버그: 클래스 화일이 누락되었습니다. ***");
    disconnectError();
}
}
```

ChatClient.java

// 메시지 처리 루틴 (ChatResponseHandler 인터페이스 구현 부분)

```
public void isLive()
{
}
```

```
public void enterAck( boolean success )
{
    if ( ! success )
    {
        print( "*** 오류: " + userName.getText()
            + "은 이미 사용중인 이름입니다. "
            + "다른 이름을 사용하십시오. ***" );
        close();
    }
}
```

ChatClient.java

```
        userName.requestFocus();
    }
}

public void userEntered(String name)
{
    print( "*** " + name + "님이 들어오셨습니다. ***" );
    userList.add( name );
}

public void chatReceived(String name, String message)
{
    print( "<" + name + "> " + message );
}
```

ChatClient.java

```
public void userListReceived(String[] userNames)
{
    userList.removeAll();
    for(int i = 0; i < userNames.length; i++)
        userList.add( userNames[i] );
}
```

```
public void userLeft( String name )
{
    print( "*** " + name + "님이 나가셨습니다. ***" );
    if ( userName.getText().equals(name) )
        close();
    else
        userList.remove( name );
}
```

ChatClient.java

```
public void userDisconnected( String name )  
{  
    print( "*** " + name + "님이 오류로 인해 연결이 끊어졌습니다. ***" );  
}
```

```
// 메시지 처리루틴 끝
```

```
void send(ChatRequest mesg) throws IOException  
{  
    toServer.writeObject(mesg);  
    toServer.flush();  
}
```

ChatClient.java

```
void disconnectError()
{
    print("*** 오류: 서버와의 연결이 끊어졌습니다. ***");
    close();
}
```

```
void close()
{
    userList.removeAll();
    try
    {
        fromServer.close();
        toServer.close();
        if ( socket != null )
```


ChatClient.java

```
        socket.close();
    } catch( IOException ex )
    {}

    receiver = null;

    connectBtn.setLabel( "연결" );
}
}
```

ChatServer.java

ChatServer.java

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

public class ChatServer extends Thread
{
    public static void main( String[] args )
        throws IOException
    {
        int port = (args.length == 0) ?
            10000 : Integer.parseInt(args[0]);
        new ChatServer( port ).start();
    }
}
```

ChatServer.java

```
ServerSocket socket;  
Hashtable nameConMap = new Hashtable();  
Vector conQ = new Vector();  
Vector errorQ = new Vector();  
  
public ChatServer( int port ) throws IOException  
{  
    socket = new ServerSocket( port );  
}  
  
public void run()  
{  
    while(true)  
    {
```

ChatServer.java

```
try
{
    Socket clientSock = socket.accept();
    ChatServerConnection con
        = new ChatServerConnection(this, clientSock);
    con.start();
} catch( IOException ex )
{
    ex.printStackTrace();
}
}
}
```

ChatServer.java

```
class ChatServerConnection extends Thread
    implements ChatRequestHandler
{
    ChatServer server;
    Socket socket;
    ObjectInputStream fromClient;
    ObjectOutputStream toClient;
    String userName;
    boolean shouldStop = false;

    public ChatServerConnection(ChatServer server, Socket socket)
        throws IOException
    {
        this.server = server;
```

ChatServer.java

```
this.socket = socket;
fromClient = new ObjectInputStream( socket.getInputStream() );
toClient = new ObjectOutputStream( socket.getOutputStream() );
}

public void run()
{   try
    {
        while( ! shouldStop )
        {
            ChatRequest mesg = (ChatRequest) fromClient.readObject();
            mesg.execute(this);
        }
    } catch( IOException ex )
```

ChatServer.java

```
{
    addError( this );
    broadcastError();
} catch( ClassNotFoundException e )
{
    System.out.println("치명적 오류: 클래스 화일이 누락되었습니다.");
    addError( this );
    broadcastError();
}
}

// 메시지 처리 루틴 (ChatRequestHandler 인터페이스 구현 부분)
```


ChatServer.java

```
public void enter( String name )
{
    synchronized (server)
    {
        ChatServerConnection otherCon
            = (ChatServerConnection) server.nameConMap.get(name);
        if ( otherCon != null )
        { // 이미 사용되고 있는 이름임
            try
            {
                // 연결이 여전히 살아있는 지 확인
                otherCon.send( new IsLive() );
            }
            try
            {
                send( new EnterAck(false) );
            }
        }
    }
}
```

ChatServer.java

```
        } catch( IOException ex2 )
        {
            close();
            return;
        } catch( IOException ex )
        {    // 연결이 살아있지 않음
            addError( otherCon );
            broadcastError();
        }
    }
}
```

```
// 새로운 사용자를 받아들임.
userName = name;
server.conQ.addElement(this);
```

ChatServer.java

```
        server.nameConMap.put(userName, this);
        broadcast( new UserEntered(userName) );
    }
}

public void sendChat( String message )
{
    broadcast(new ChatReceived(userName, message));
}

public void getUserList()
{
    synchronized (server)
    {
```

ChatServer.java

```
String[] names = new String[server.nameConMap.size()];
Enumeration enum = server.nameConMap.keys();
for( int i = 0; enum.hasMoreElements(); i++ )
    names[i] = (String) enum.nextElement();
try
{
    send( new UserListReceived(names) );
} catch( IOException ex )
{
    addError( this );
}
}
```

ChatServer.java

```
public void leave()
{
    synchronized (server)
    {
        server.conQ.removeElement(this);
        server.nameConMap.remove(userName);
        UserLeft mesg = new UserLeft(userName);
        broadcast( mesg );
        try
        {
            send( mesg );
        } catch( IOException ex )
        {}
        server.errorQ.removeAllElements();
    }
}
```

ChatServer.java

```
        close();
    }
}

// 메시지 처리 루틴 끝

synchronized void send(ChatResponse mesg) throws IOException
{
    toClient.writeObject(mesg);
    toClient.flush();
}

void close()
{
```

ChatServer.java

```
try
{
    fromClient.close();
    toClient.close();
    socket.close();
} catch( IOException ex )
{}
shouldStop = true;
}

void broadcast(ChatResponse mesg)
{
    broadcastInt(mesg);
    broadcastError();
}
```

ChatServer.java

```
void broadcastInt(ChatResponse mesg)
{
    for(int i = 0; i < server.conQ.size(); ++i)
    {   ChatServerConnection con
        = (ChatServerConnection) server.conQ.elementAt(i);
        try
        {
            con.send(mesg);
        } catch( IOException ex )
        {
            addError( con );
            i--;
        }
    }
}
```


ChatServer.java

```
void addError( ChatServerConnection con )
{
    server.conQ.removeElement(con);
    server.nameConMap.remove(con.userName);
    if ( ! server.errorQ.contains(con) )
        server.errorQ.addElement(con);
    con.close();
}
```

```
void broadcastError()
{
    for(int i = 0; i < server.errorQ.size(); ++i)
    {
        ChatServerConnection con
```

ChatServer.java

```
        = (ChatServerConnection) server.errorQ.elementAt(i);
        broadcastInt( new UserDisconnected(con.userName) );
    }
    server.errorQ.removeAllElements();
}
}
```

All About Datagrams

User Datagram Protocol

- UDP는 TCP와 달리 스트림 기반이 아니다. TCP와 같이 IP를 사용하여 데이터를 전송하며 매우 신속 하지만 신뢰성은 없다.
- TCP는 패킷을 직접 다루지 않으며 스트림을 통해 간접적으로 패킷의 전송을 다루므로 패킷의 전송에 거의 신경 쓸 일이 없으나 UDP는 패킷을 직접 다루는 프로토콜 이다.
- TCP는 가상적인 연결(Connection)이 있어야 데이터를 보낼 수 있지만 UDP의 경우 목적지를 명시한 패킷을 네트워크 상에 흘려 놓거나 언젠가 도착할 패킷을 마냥 기다리고 있을 뿐 이다.

User Datagram Protocol

- UDP 프로토콜은 데이터그램이라는 데이터 패킷을 사용하는데, 데이터그램 패킷 들은 서로 내용이 독립적이며 자체 내에 목적지에 대한 정보를 갖고 송신자로부터 수신자로 전송 된다
- UDP는 메시지가 제대로 도착했는지 확인하는 확인 응답을 사용하지 않을 뿐만 아니라, 수신된 메시지의 순서를 맞추지 않으며, 기계간의 정보 흐름 속도를 제어하기 위한 피드백을 제공하지 않기 때문에, UDP 메시지는 손실되거나, 중복되거나, 비순서적으로 도착될 수 있다.

User Datagram Protocol

- UDP는 IP상에 패킷을 흘려 놓은 후 TCP와 같이 확실히 도착 했는지의 여부를 검사하지 않는다.
- TCP의 경우 도중에 패킷이 유실되면 다시 전송하도록 하며 패킷의 순서가 뒤섞인 채 전송되면 패킷을 제대로 된 순서로 정리한다.
- TCP의 경우 라우터 처리용량에 여유가 없을 때는 패킷을 잘게 나누어 다시 재 시도 되기 때문에 어느 정도는 안전한 방법이나 UDP의 경우 라우터가 패킷을 거부하면 그 패킷은 버려지게 된다.
- UDP 패킷의 크기를 잘 조정 하는 것은 프로그래머가 해줘야 할 일 이다.

User Datagram Protocol

- UDP는 예전에 멀티미디어 스트리밍 서비스와 같이 소리나 화면이 잠깐 끊어져도 문제되지 않는 곳에 사용되었으나 근래에는 QoS(Quality of Service)의 문제로 인해 멀티미디어 스트리밍에도 잘 쓰이지 않는다.
- UDP는 작고 가벼우며 빠르기 때문에 router를 벗어나지 않는 내부 네트워크용으로 많이 사용된다.

All About Datagrams

- Some applications that you write to communicate over the network will not require the reliable, point-to-point channel provided by TCP. Rather, your applications might benefit from a mode of communication that delivers independent packages of information whose arrival and order of arrival are not guaranteed.

All About Datagrams

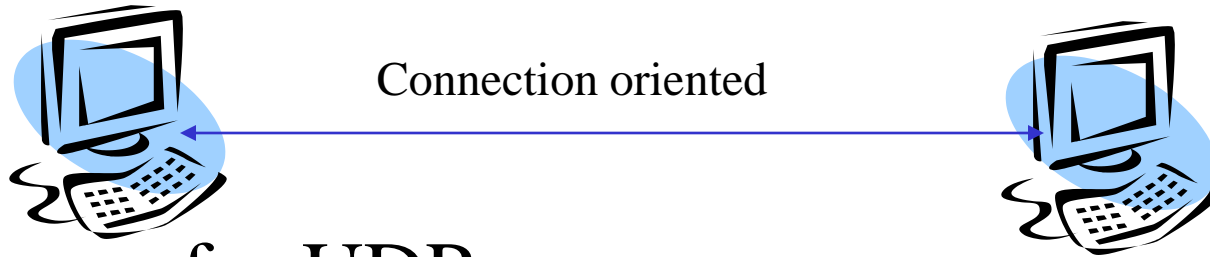
- The UDP protocol provides a mode of network communication whereby applications send packets of data, called datagrams, to one another.
- A datagram is an independent, self-contained message sent over the network whose arrival, arrival time, and content are not guaranteed.

What Is a Datagram?

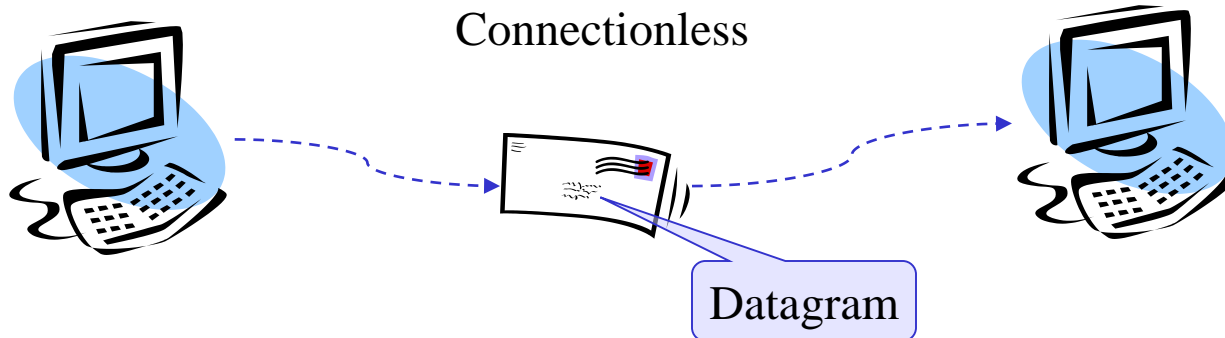
- Definition
 - A datagram is an independent, self-contained message sent over the network whose arrival, arrival time, and content are not guaranteed.

What Is a Datagram?

- Socket for TCP



- Datagram for UDP



What Is a Datagram?

- The java.net package contains two classes to help you write Java programs that use datagrams to send and receive packets over the network
 - DatagramSocket, DatagramPacket
 - An application can send and receive DatagramPackets through a DatagramSocket.

Class DatagramSocket

- DatagramPacket 객체를 전송하고 수신하는데 사용된다. 즉 DatagramPacket을 사용하기 위해서는 DatagramSocket을 생성해야 한다.
- DatagramSocket 객체 하나만 가지고도 여러 목적지로 패킷을 보낼 수 있으며 여러 발신지로 부터 패킷을 받을 수 있다. 어느 호스트로 부터 패킷을 받을지 정하는 방법은 없다.
- 대개 모든 DatagramSocket은 하나의 Local port에서 돌아간다. 이 port에서 Socket은 Host로 들어오는 데이터를 기다리고 Network으로 나가는 Datagram 에 헤더를 붙인다.

Class DatagramSocket

- DatagramSocket은 로컬 호스트의 어떤 UDP Port 에서 패킷이 날아 오기만 기다리고 있다. 대개의 경우 서버 프로그램은 포트를 지정해 주며 클라이언트 프로그램의 경우 UDP 포트를 시스템에서 임의로 지정하게 하는 방식을 사용한다.
- TCP 처럼 Server와 Client 별개의 소켓을 가지고 있지 않다.

Class DatagramSocket


- DatagramSocket 은 TCP 방식과는 달리 서버와 클라이언트가 동일하게 사용하며, 먼저 서버측에서 소켓을 생성하여 대기하면 클라이언트가 소켓을 생성하여 데이터를 전송한다.
- UDP서버와 TCP 서버는 같은 포트를 사용 할 수 있다.

Class DatagramSocket

- This class represents a socket for sending and receiving datagram packets
- A datagram socket is the sending or receiving point for a packet delivery service. Each packet sent or received on a datagram socket is individually addressed and routed. Multiple packets sent from one machine to another may be routed differently, and may arrive in any order.

Class DatagramSocket

- Constructor
 - DatagramSocket()
 - Constructs a datagram socket and binds it to any available port on the local host machine.
 - DatagramSocket(int port)
 - Constructs a datagram socket and binds it to the specified port on the local host machine.
 - DatagramSocket(int port, InetAddress laddr)
 - Creates a datagram socket, bound to the specified local address.
 - 보통 여러 개의 NIC(Network Interface Card)가 꽂힌 컴퓨터의 로컬 주소와 DatagramSocket을 바인딩



Port 번호가
0인 경우는
임의의 포
트와 바인
딩

Class DatagramPacket

- 하나의 데이터그램 패킷을 캡슐화 한다
- 메시지 몸체와 목적지 주소로 구성
- DatagramPacket 클래스는 데이터그램을 사용할 수 있도록 기능을 제공해 주며, 데이터그램 패킷은 비연결 패킷 전송 서비스(connectionless packet delivery service)를 구현하기 위해 사용된다. 메시지는 패킷 내에 포함되어 있는 정보에 기반 하여 하나의 호스트에서 다른 호스트로 라우팅 된다.
- 한 호스트에서 다른 호스트로 전송된 여러 개의 패킷(multiple packet)은 서로 다르게 라우팅 될 것이며, 원래의 순서와 관계없이 도착될 수 있다.

Class DatagramPacket

- Datagram packets are used to implement a connectionless packet delivery service. Each message is routed from one machine to another based solely on information contained within that packet. Multiple packets sent from one machine to another might be routed differently, and might arrive in any order.

Class DatagramPacket

- Constructor
 - DatagramPacket(byte[] buf, int length)
 - Constructs a DatagramPacket **for receiving** packets of length length.
 - buf는 수신되는 UDP 패킷을 받아 낼 바이트 배열
 - DatagramPacket(byte[] buf, int length, InetAddress address, int port)
 - Constructs a datagram packet **for sending** packets of length length to the specified port number on the specified host.

Class DatagramPacket

- Constructor
 - DatagramPacket(byte[] buf, int offset, int length)
 - Constructs a DatagramPacket **for receiving** packets of length length, specifying an offset into the buffer.
 - DatagramPacket(byte[] buf, int offset, int length, InetAddress address, int port)
 - Constructs a datagram packet **for sending** packets of length length with offset ioffsetto the specified port number on the specified host.

데이터 송수신 순서

- ✓ 서버는 클라이언트로부터 요청을 받고 그에 대해 응답을 하기 위한 데이터그램 소켓을 생성하여 특정 포트에 연결한다.
- ✓ 서버는 클라이언트로부터 오는 요청 내용을 저장할 입력 데이터그램 패킷을 생성한 뒤 클라이언트로부터 요청이 오기를 기다리며 대기한다.
- ✓ 클라이언트 프로그램을 실행하면 서버에서와 마찬가지로 데이터그램 소켓을 하여 특정 포트에 연결한다.
- ✓ 클라이언트는 서버에 요청을 보내기 위한 출력 데이터그램 패킷을 생성한 뒤 서버에게 송신한다.

데이터 송수신 순서

- ✓ 서버는 클라이언트가 보낸 데이터그램 패킷을 받아들인다. 여기서, 서버는 입력 데이터그램 패킷을 생성하고 대기하였는데, 이 입력 데이터그램 패킷은 아무런 내용이 없는 빈 패킷으로서 이 패킷이 클라이언트가 보낸 데이터그램 패킷으로 대체된다. 따라서 서버 자신이 생성한 입력 데이터그램 패킷을 받는 것과 곧 클라이언트가 보낸 데이터그램 패킷을 받는 것과 동일한 효과를 나타낸다.
- ✓ 클라이언트도 서버가 보낸 응답을 받기 위한 입력 데이터그램 패킷을 생성하고 대기한다.

Example 1.

Writing a Datagram Client and Server

Quote Client and Server

- The server continuously receives datagram packets over a datagram socket. Each datagram packet received by the server indicates a client request for a quotation. When the server receives a datagram, it replies by sending a datagram packet that contains a one-line "quote of the moment" back to the client.

Writing a Datagram Server

The QuoteServer

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

```
public class QuoteServer {  
    public static void main(String[] args) throws  
        IOException {  
        new QuoteServerThread().start();  
    }  
}
```

The QuoteServer

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

```
public class QuoteServerThread extends Thread {  
    protected DatagramSocket socket = null;  
    protected BufferedReader in = null;  
    protected boolean moreQuotes = true;  
  
    public QuoteServerThread() throws IOException {  
        this("QuoteServerThread");  
    }  
}
```

The QuoteServer

```
public QuoteServerThread(String name) throws IOException {  
    super(name);  
    socket = new DatagramSocket(4445);  
  
    try {  
        in = new BufferedReader(new FileReader("one-liners.txt"));  
    } catch (FileNotFoundException e) {  
        System.err.println("Could not open quote file. Serving time  
instead.");  
    }  
}  
  
public void run() {
```

The QuoteServer

```
while (moreQuotes) {  
    try {  
        byte[] buf = new byte[256];  
        // receive request  
        DatagramPacket packet = new DatagramPacket(buf,  
buf.length);  
        socket.receive(packet);  
  
        // figure out response  
        String dString = null;  
        if (in == null)  
            dString = new Date().toString();  
        else
```

The QuoteServer

```
dString = getNextQuote();
    buf = dString.getBytes();
    // send the response to the client at "address" and "port"
    InetAddress address = packet.getAddress();
    int port = packet.getPort();
    packet = new DatagramPacket(buf, buf.length, address,
port);
    socket.send(packet);
} catch (IOException e) {
    e.printStackTrace();
moreQuotes = false;
}
}
socket.close();
```

The QuoteServer

```
}  
protected String getNextQuote() {  
    String returnValue = null;  
    try {  
        if ((returnValue = in.readLine()) == null) {  
            in.close();  
            moreQuotes = false;  
            returnValue = "No more quotes. Goodbye.";  
        }  
    } catch (IOException e) { returnValue = "IOException..." }  
    return returnValue;  
}  
}
```

Writing a Datagram Client

The QuoteClient

```
import java.net.*;
import java.util.*;

public class QuoteClient {
    public static void main(String[] args) throws IOException {
        // get a datagram socket
        DatagramSocket socket = new DatagramSocket();
        // send request
        byte[] buf = new byte[256];
        InetAddress address = InetAddress.getByName(args[0]);
        DatagramPacket packet = new DatagramPacket(buf,
            buf.length, address, 4445);
        socket.send(packet);
    }
}
```

The QuoteClient

```
// get response
```

```
packet = new DatagramPacket(buf, buf.length);
```

```
socket.receive(packet);
```

```
// display response
```

```
String received = new String(packet.getData());
```

```
System.out.println("Quote of the Moment: " + received);
```

```
socket.close();
```

```
}
```

```
}
```

Example 2.

Daytime Server and Client

Daytime Server

UDPDatimeServer.java

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

public class UDPDatimeServer {

    public static final int DAYTIME_PORT = 13;

    public static void respond(DatagramSocket ds, DatagramPacket dp) {

        DatagramPacket outgoing;

        Date now = new Date();
        String s = now.toString();
```

UDPDatimeServer.java

```
byte[] data = s.getBytes();
```

```
try {
```

```
    outgoing = new DatagramPacket(data, data.length, dp.getAddress(),  
    dp.getPort());
```

```
    ds.send(outgoing);
```

```
}
```

```
catch (IOException e) {
```

```
    System.err.println(e);
```

```
}
```

```
}
```

```
public static void main(String args[]) {
```

```
try {
```

```
    DatagramSocket ds = new DatagramSocket(DAYTIME_PORT);
```

```
    DatagramPacket incoming;
```

UDPDatimeServer.java

```
byte[] buffer = new byte[2048];
```

```
while(true) {
```

```
    incoming = new DatagramPacket(buffer, buffer.length);
```

```
    ds.receive(incoming);
```

```
    respond(ds, incoming);
```

```
}
```

```
} catch(IOException e) {
```

```
    System.out.println(e);
```

```
}
```

```
}
```

```
}
```

Daytime Client

Daytime.java

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

public class Daytime {
    public static final int DAYTIME_PORT = 13;

    public static final String getTime(String hostname) throws IOException {
        InetAddress host;
        DatagramSocket socket = new DatagramSocket();
        byte[] dummyData = new byte[1];
        byte[] timeData = new byte[256];
        DatagramPacket sendPacket, receivePacket;

        host = InetAddress.getByName(hostname);
```

Daytime.java

```
sendPacket =  
    new DatagramPacket(dummyData, dummyData.length, host,  
        DAYTIME_PORT);  
receivePacket = new DatagramPacket(timeData, timeData.length);  
  
socket.send(sendPacket);  
socket.receive(receivePacket);  
  
return  
    new String(receivePacket.getData(), 0, receivePacket.getLength());  
  
}  
  
public static void main(String[] args) {  
    String server = "tock.usno.navy.mil";
```

Daytime.java

```
if(args.length == 1)
    server = args[0];
else if(args.length > 1) {
    System.err.println("Usage: Daytime [hostname]");
    return;
}

try {
    // Time should include newline, so we don't use println().
    System.out.print(getTime(server));
    System.out.flush();
} catch(IOException e) {
    e.printStackTrace();
}
```

Daytime.java

```
    return;  
}  
}  
}
```

Example 3.

- Datagram Client and Server

DatagramClient

DatagramClient.java

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

public class DatagramClient {
    public static void main(String[] args) {
        byte[] b = new byte[256];
        String send = null;
        DatagramPacket packet = null;
        try {
            DatagramSocket udpSocket = new DatagramSocket();
            InetAddress serverAddress =
                InetAddress.getByName(args[0]);
```


DatagramClient.java

```
BufferedReader in = new BufferedReader (new
    InputStreamReader(System.in));
    System.out.print("send... ");
    while ((send = in.readLine()) != null) {
        // 문자열을 바이트배열로 전환한 후 포트 4444로 전송한다.
        b = send.getBytes();
        packet = new DatagramPacket(b, b.length, serverAddress,
4444);
        udpSocket.send(packet);
        // 패킷을 수신하여 문자열로 전환한 후 화면에 출력한다.
        packet = new DatagramPacket(b, b.length);
```

DatagramClient.java

```
        udpSocket.receive(packet);
        String received = new String(packet.getData());
        System.out.println("receive... " + received);
        System.out.print("send... ");
    }
    udpSocket.close();
} catch(IOException ex) {
    ex.printStackTrace();
}
}
```

DatagramServer

DatagramServer.java

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

public class DatagramServer2 {
    public static void main(String[] args) {
        DatagramPacket packet, packet2 = null;
        byte[] b = new byte[256];
        try {
            DatagramSocket socket = new DatagramSocket(4444);
            packet = new DatagramPacket(b, 256);
```

DatagramServer.java

```
while(true) {  
    packet.setLength(256);  
    socket.receive(packet);  
    //실제 클라이언트가 보내는 데이터 만큼 가져옴  
    String received = new String(packet.getData(), 0,  
packet.getLength());  
    //버퍼의 모든 사이즈만큼 가져옴 jcleer->jalee, ja --  
>jcleer, j --> jcleer Return  
    //String received = new String(packet.getData());  
    System.out.println(received); //서버 콘솔에 출력  
    // 데이터를 바이트배열로 전환하여 패킷으로 다시 보냄  
    InetAddress address = packet.getAddress();
```

DatagramServer.java

```
    int port = packet.getPort();
    b = received.getBytes();
    packet2 = new DatagramPacket(b, b.length, address, port);
    socket.send(packet2);
}
} catch(IOException ex) {
    ex.printStackTrace();
}
}
```

DatagramServer.java

```
    } catch(IOException ex) {  
        ex.printStackTrace();  
    }  
}  
}
```

Multicasting Through Java

Broadcasting

- 하나의 네트워크 단위 내에 있는 호스트들이 데이터를 받을 수 있다. 네트워크 내의 특정한 호스트에 보내는 것이 아니라 일괄적으로 보낸다.(텔레비전 송신기)
- 브로드캐스트 주소란 주어진 네트워크 안의 모든 호스트들이 받을 수 있는 주소를 말하며 만약 UDP 패킷이 브로드 캐스트 주소로 전송되면 그 네트워크안의 모든 호스트들은 패킷을 받을 수 있다.
- 로컬 브로드 캐스트 주소로 사용되는 값은 보통 로컬 서브넷의 가장 끝 주소이다.
- 브로드캐스트 패킷은 주어진 네트워크안의 모든 호스트가 잡지만 멀티캐스트 패킷은 원하는 호스트들만 잡는다.

Multicasting

- 패킷이 라우터를 통해 이동, 여러 네트워크를 거치며 따라서 목적지 호스트가 어디 있는지 데이터를 받을 수 있다. 멀티캐스트용 호스트와 라우터는 IGMP(Internet Group Management Protocol)을 지원해야 하며 멀티캐스트 주소로 보내어지는 패킷을 처리할 줄 알아야 한다.(화상회의)

Multicasting

- 멀티캐스팅은 한 호스트에서 많은 호스트로 데이터를 전송하는 것이며 전체 모든 호스트로 일괄적으로 전송하지 않는다. 데이터를 특정 멀티캐스트 그룹에 가입한 호스트에만 전달한다.
- 애플리케이션은 데이터그램을 멀티캐스트 주소로 전송한다. 멀티캐스트 주소는 일반주소와 크게 다르지 않으며 라우터는 데이터가 멀티캐스트 그룹에 속한 호스트에 모두 전달되었는지 확인한다. 물론 이 경우 라우터가 멀티캐스팅을 지원해야 가능하다.
- 애플리케이션에서도 데이터그램의 헤더에 TTL(Time to Live)이라는 필드를 추가해야 한다. TTL은 패킷이 지나갈 수 있는 라우터 최대 통과 횟수 이다. 0이 되면 패킷은 폐기된다. 한번 지나갈대 마다 1씩 감소한다.(1보다 큰 수가 감소 되기도 한다.)

Multicasting

- 멀티캐스트 주소는 멀티캐스트 그룹 호스트의 주소이며 멀티캐스트 주소는 224.0.0.0~239.255.255.255 이다. (D 클래스)
- 멀티캐스트 그룹이란 하나의 멀티캐스트 주소를 공유하는 인터넷 호스트의 집합이다. 멀티캐스트 주소로 전달된 임의의 데이터는 그룹의 모든 멤버에 전달된다.
- 멀티캐스트 그룹에 가입할 수 있는 권한은 아무 호스트에게 있으며 자유롭게 가입/탈퇴가 가능하다.
- 새롭게 멀티캐스트 그룹을 생성하기 위해서는 멀티캐스트 주소 중 임의의 주소를 골라낸 후 그 주소에 대한 `InetAddress` 객체를 생성하여 데이터를 전송 하는 일 이다.

Multicasting

- 멀티캐스트와 UDP의 차이는 패킷이 도달 할 목적지가 멀티캐스트 주소인지 하나의 호스트 인지의 차이 이다.
- “어떤 멀티캐스트 주소를 사용 할 까?” 는 정해진 것이 없으며 239로 시작하는 모든 멀티캐스트 주소는 내부 지정용으로 예약되어 있으므로 시험해 보기 위해서는 이 주소를 사용 한다. (224.0.0.1은 로컬 서브넷에서 멀티캐스팅을 지원하는 모든 시스템을 포함하는 멀티캐스트 그룹이다.)

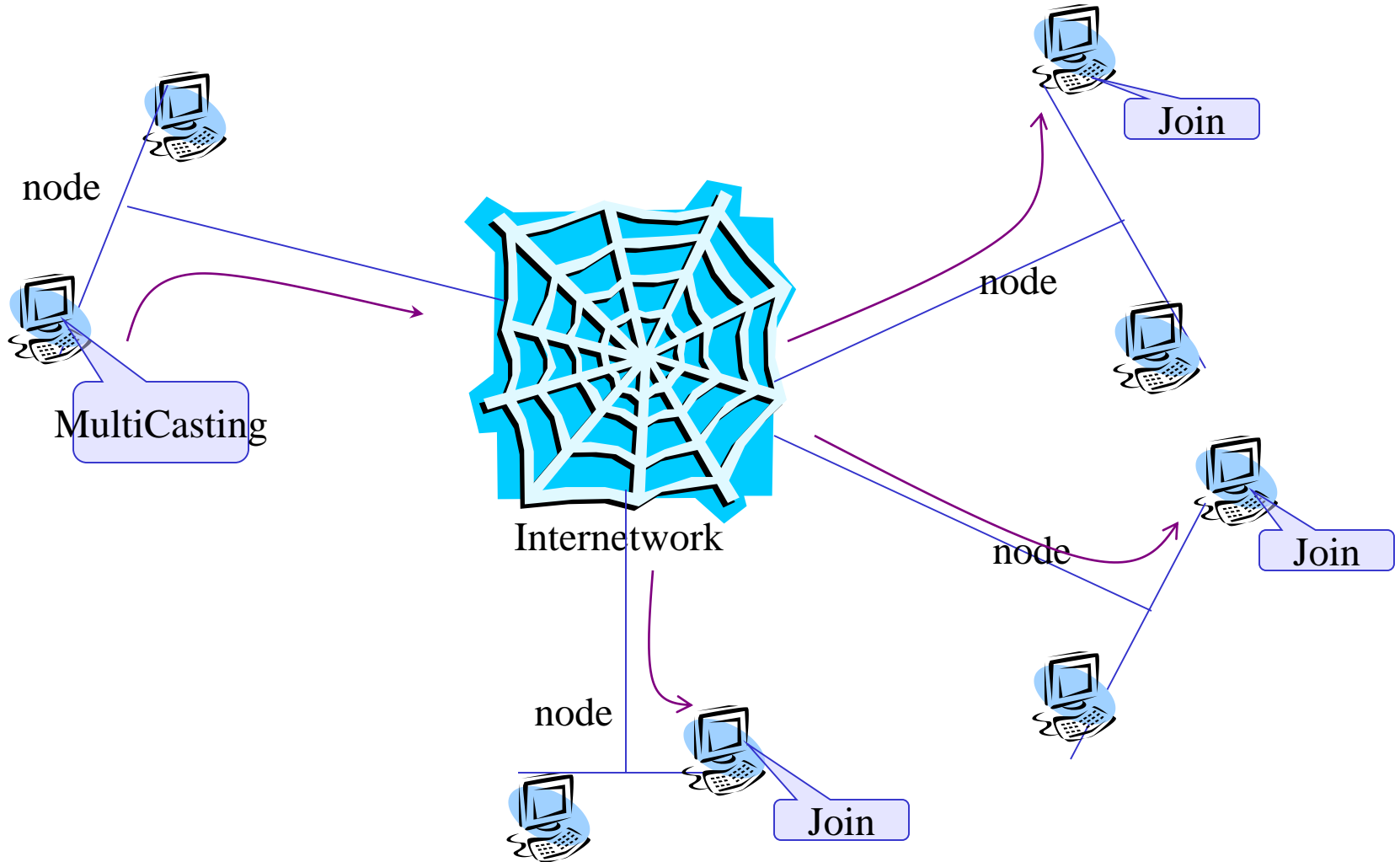
Multicasting

- Multicasting, is the *Internet's version of broadcasting*. A site that multicasts information is similar in many ways to a television or radio station that broadcasts its signal.
- The signal originates from one source, but it can reach everyone in the station's signal area. The information passes on by those who don't want to catch the signal or don't have the right equipment.

Multicasting

- Most high-level network protocols only provide a unicast transmission service. That is, nodes of the network only have the ability to send to one other node at a time. All transmission with a unicast service is inherently point-to-point. *If a node wants to send the same information to many destinations using a unicast transport service, it must perform a replicated unicast, and send N copies of the data to each destination in turn.*

Multicasting



Multicasting

- Broadcasting is generally suited to any applications that requires a number of machine in a distributed group to receive the same data; for example conferencing, group mail and news distribution, and network management.

Multicasting

- Multicast Groups
 - The notion of "group" is essential to the concept of multicasting.
 - By definition a multicast message is sent from a source to a group of destination hosts.
 - In IP multicasting, multicast groups have an ID called multicast group ID.

Multicasting

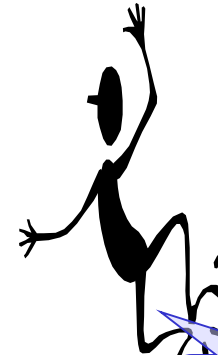
- Multicast Groups
 - Whenever a multicast message is sent out, a multicast group ID specifies the destination group. These group ID's are essentially a set of IP addresses called "*Class D*". Therefore, if a host (a process in a host) wants to receive a multicast message sent to a particular group, it needs to somehow listens to all messages sent to that particular group.

Multicasting

- Multicast Groups



Multicast Group



I wanna listen!
Let's join!

Multicasting

- IP Addresses of Class D - Multicasting
 - The range of Class D addresses are in dotted decimal notation from 224.h.h.h.h to 239.h.h.h, where h is a number from 0 to 255. Address 224.0.0.0 is reserved and can not be used, while address 224.0.0.1 is used to address all hosts that take part in IP multicasting.
(224.0.0.1은 IP 멀티캐스트에 참여하는 모든 호스트와 라우터를 포함하는 모든 호스트 그룹에 영구히 할당되어 있으므로 사용할 수 없다.)

Multicasting

- Life of Multicast Packets (TTL's)
 - Broadcast packets need to have a finite life in order to avoid bouncing of the packets around the network forever. Each packet has a time to live (TTL) value, a counter that is decremented every time the packet passes through an hop i.e a router between the network. Because of TTLs, each multicast packet is a ticking time bomb.

Multicasting

- Life of Multicast Packets (TTL's)
 - Take for example, a TV station where TTLs would be the station's signal area -- the limitation of how far the information can travel. As the packet moved around the company's internal network, its TTL would be notched down every time it passed through an router. When the packet's TTL reached 0, the packet would die and not be passed further.
 - Generally multicast with long TTLs -- perhaps 200 - to guarantee that the information will reach around the world

Class MulticastSocket

- Constructor

MulticastSocket()

Create a multicast socket.

MulticastSocket(int port)

Create a multicast socket and bind it to a specific port.

MulticastSocket(SocketAddress bindaddr)

Create a MulticastSocket bound to the specified socket address.

Example 1.

A Simple Example that explains
multicasting

Server Program

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

```
public class BroadcastServer{
```

```
    public static final int PORT = 1200;
```

```
    final static int TTL=16;
```

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

```
        MulticastSocket socket;
```

```
        DatagramPacket packet;
```

```
        InetAddress address;
```

Server Program

```
address = InetAddress.getByName(args[0]);  
socket = new MulticastSocket();  
  
// join a Multicast group and send the group salutations  
  
socket.joinGroup(address);  
byte[] data = null;  
  
for(;;){  
  
    Thread.sleep(1000);  
    System.out.println("Sending ");  
    String str = (new Date()).toString();  
    data = str.getBytes();
```

Server Program

```
packet = new DatagramPacket  
        (data,str.length(),address,PORT);
```

```
// Sends the packet  
socket.send(packet, (byte)TTL);  
}
```

```
}
```

```
}
```

Client Program

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

public class BroadcastClient{
    public static final int PORT = 1200;
    public static void main(String args[]) throws Exception{

        MulticastSocket socket;
        DatagramPacket packet;
        InetAddress address;

        address = InetAddress.getByName(args[0]);
        socket = new MulticastSocket(PORT);
```

Client Program

//join a Multicast group and send the group salutations

```
socket.joinGroup(address);
```

```
byte[] data = null;
```

```
packet = new DatagramPacket(data,data.length);
```

```
for(;;){
```

```
    // receive the packets
```

```
    socket.receive(packet);
```

```
    String str = new String(packet.getData());
```

```
    System.out.println(" Time signal received from"+  
        packet.getAddress() + " Time is : " +str);
```

```
    }
```

```
}
```

```
}
```

Example 2.

Broadcasting over Internet!

From console input.

Server code

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

public class MulticastSender
{
    final static int TTL=16;
    final static int BUFFER_SIZE=512;

    public static void main(String[] args) throws Exception
    {
        InetAddress maddr = InetAddress.getByName(args[0]);
        int receiverPort = Integer.parseInt(args[1]);

        MulticastSocket msocket = new MulticastSocket();
```


Server code

```
byte[] buf = new byte[BUFFER_SIZE];
DatagramPacket outPacket = new DatagramPacket(
    buf, buf.length, maddr, receiverPort );
int sentCount = 0;
int count;
try
{
    while( (count = System.in.read(buf)) != -1 )
    {
        outPacket.setLength(count);
        msocket.send(outPacket, (byte)TTL);
        sentCount++;
    }
    System.err.println(sentCount + "개 패킷 송신 완료");
}
```

Server code

```
    } catch( IOException ex )
    {
        System.out.println( ex );
    } finally
    {
        msocket.close();
    }
}
```

Client code

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

class MulticastReceiver
{
    final static int BUFFER_SIZE=512;

    public static void main(String[] args) throws Exception
    {
        InetAddress maddr = InetAddress.getByName(args[0]);
        int port = Integer.parseInt(args[1]);

        MulticastSocket msocket = new MulticastSocket(port);
        byte[] buf = new byte[BUFFER_SIZE];
```

Client code

```
DatagramPacket inPacket = new DatagramPacket(buf, buf.length );
msocket.joinGroup(maddr);
int receivedCount = 0;
try
{
    while( true )
    {
        inPacket.setLength(buf.length);
        msocket.receive(inPacket);
        System.err.println( inPacket.getAddress()
                               + ":" + inPacket.getPort()
                               + " (" + ++receivedCount
                               + "개 패킷 수신 완료" + ")" );
        System.out.write(buf, 0, inPacket.getLength());
    }
}
```

Client code

```
        System.out.println();
        System.out.flush();
    }
} catch( IOException ex )
{
    System.out.println( ex );
} finally
{
    msocket.leaveGroup(InetAddress.getByName(args[0]));
}
}
```

Example 3.

Peer to Peer Chatting Program

Referenced from Java Network programming 2nd Edition, Manning

MulticastChat.java

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

public class MulticastChat implements Runnable, WindowListener,
    ActionListener {
    protected InetAddress group;
    protected int port;

    public MulticastChat (InetAddress group, int port) {
        this.group = group;
        this.port = port;
        initAWT ();
    }
}
```

MulticastChat.java

```
protected Frame frame;  
protected TextArea output;  
protected TextField input;  
  
protected void initAWT () {  
    frame = new Frame  
        ("MulticastChat [" + group.getHostAddress () + ":" + port + "]");  
    frame.addWindowListener (this);  
    output = new TextArea ();  
    output.setEditable (false);  
    input = new TextField ();  
    input.addActionListener (this);  
    frame.setLayout (new BorderLayout ());  
    frame.add (output, "Center");
```


MulticastChat.java

```
frame.add (input, "South");  
frame.pack ();  
}
```

```
protected Thread listener;
```

```
public synchronized void start () throws IOException {  
    if (listener == null) {  
        initNet ();  
        listener = new Thread (this);  
        listener.start ();  
        frame.setVisible (true);  
    }  
}
```

MulticastChat.java

```
protected MulticastSocket socket;  
protected DatagramPacket outgoing, incoming;  
  
protected void initNet () throws IOException {  
    socket = new MulticastSocket (port);  
    socket.setTimeToLive (1);  
    socket.joinGroup (group);  
    outgoing = new DatagramPacket (new byte[1], 1, group, port);  
    incoming = new DatagramPacket (new byte[65508], 65508);  
}  
  
public synchronized void stop () throws IOException {  
    frame.setVisible (false);
```

MulticastChat.java

```
if (listener != null) {  
    listener.interrupt ();  
    listener = null;  
    try {  
        socket.leaveGroup (group);  
    } finally {  
        socket.close ();  
    }  
}  
  
public void windowOpened (WindowEvent event) {  
    input.requestFocus ();  
}
```

MulticastChat.java

```
public void windowClosing (WindowEvent event) {  
    try {  
        stop ();  
    } catch (IOException ex) {  
        ex.printStackTrace ();  
    }  
}
```

```
public void windowClosed (WindowEvent event) {}  
public void windowIconified (WindowEvent event) {}  
public void windowDeiconified (WindowEvent event) {}  
public void windowActivated (WindowEvent event) {}  
public void windowDeactivated (WindowEvent event) {}
```

MulticastChat.java

```
public void actionPerformed (ActionEvent event) {  
    try {  
        byte[] utf = event.getActionCommand ().getBytes ("UTF8");  
        outgoing.setData (utf);  
        outgoing.setLength (utf.length);  
        socket.send (outgoing);  
        input.setText ("");  
    } catch (IOException ex) {  
        handleIOException (ex);  
    }  
}
```

MulticastChat.java

```
protected synchronized void handleIOException (IOException ex) {  
    if (listener != null) {  
        output.append (ex + "\n");  
        input.setVisible (false);  
        frame.validate ();  
        if (listener != Thread.currentThread ())  
            listener.interrupt ();  
        listener = null;  
        try {  
            socket.leaveGroup (group);  
        } catch (IOException ignored) {  
        }  
        socket.close ();  
    }  
}
```

MulticastChat.java

```
public void run () {  
    try {  
        while (!Thread.interrupted ()) {  
            incoming.setLength (incoming.getData ().length);  
            socket.receive (incoming);  
            String message = new String  
                (incoming.getData (), 0, incoming.getLength (), "UTF8");  
            output.append (message + "\n");  
        }  
    } catch (IOException ex) {  
        handleIOException (ex);  
    }  
}
```

MulticastChat.java

```
public static void main (String[] args) throws IOException {
    if ((args.length != 1) || (args[0].indexOf (":") < 0))
        throw new IllegalArgumentException
            ("Syntax: MulticastChat <group>:<port>");

    int idx = args[0].indexOf (":");
    InetAddress group = InetAddress.getByName (args[0].substring (0, idx));
    int port = Integer.parseInt (args[0].substring (idx + 1));

    MulticastChat chat = new MulticastChat (group, port);
    chat.start ();
}
```

Java Remote Method Invocation

Distributed Computing

분산 컴퓨팅 이란?

- 네트워크에서 서로 다른 시스템 간에 응용 프로그램을 분산해서 처리하는 환경을 말한다.
- 즉 하나의 컴퓨터에 존재하는 Application이나 프로세스에서 스스로 처리하거나 수행하기 어려운 작업을 다중 프로세서나 컴퓨터에 분산시키는 것이다.
- 분산 컴퓨팅을 적용한 Application을 Distributed Application 이라고 한다.

분산 객체(Distributed Object)

- 분산 컴퓨팅 기술이 객체 지향과 접목되어 하나의 프로세서나 컴퓨터에서 실행되는 객체가 다른 프로세서나 컴퓨터에서 객체와 통신이 가능 하도록 하는 기술이 분산 객체 기술이다.
- RMI는 java-to-java, CORBA는 language independent
- 분산 객체란 자신이 존재하는 런타임 환경과는 다른 런타임에 있는 객체와 통신이 가능한 객체이다.

Component Technologies

Interoperation	COM	CORBA	RMI
Vendor	MicroSoft	OMG(Object Management Group)	Sun Microsystems
Source-code	COM Library Co API	CORBA API's vendor extns	Java Beans, 100% Java
Type	Type library COM IDL	CORBA IDL platform neutral	Java class files, Java types
Over the wire	DCOM DCE RPC	variety of protocols IIOP (Internet Inter ORB Protocol), CDR	RMI RMI/IIOP

Distributed computing in java

– Using Sockets

- Design the distributed application using Sockets over TCP/IP coding with `java.net.*`
- Communication through application-level protocol
 - Data Streams
 - Object through Serialization

– Using RMI

- Support for RPC (Remote Procedure Call) between client and server
- Communication of objects over the network through serialization.
- Integrates distributed object model into Java in a natural way. (Library mostly `java.rmi.*`)

Distributed computing in java

– Object Serialization

- Save objects to disk and read them later (checkpointing)
- Send Objects over the network using Sockets
- Every object that needs to be Serialized needs to implement `java.io.Serializable`
- This interface has no methods!
- Primarily a way to inform the JVM that you want a certain Object to be serialized.
- SignedObjects that encapsulate the data, a digital signature, and a PublicKey

Getting Started Using RMI

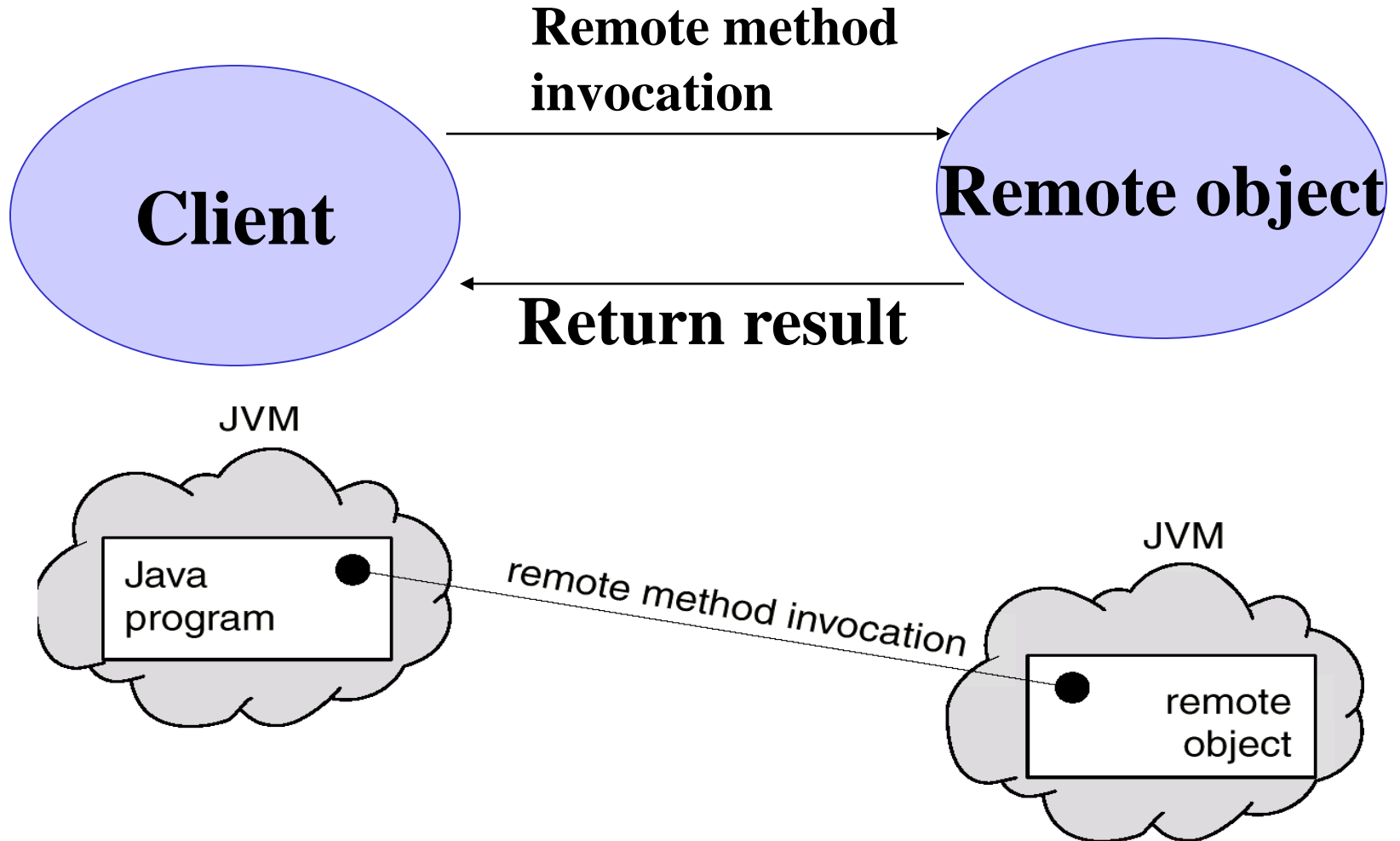
RMI(Remote Method Invocation)

- Sun Microsystems에서는 JDK1.1 부터 분산 객체 기술을 위한 RMI를 발표함
- 기존 자바언어의 장점과 풍부한 API를 분산 객체 기술에 이용이 가능하게 됨
- 다른 실행환경에 있는 객체의 메소드를 로컬에서 생성한 객체의 메소드와 다름 없이 호출할 수 있도록 하는 자바의 분산 객체 기술이다.

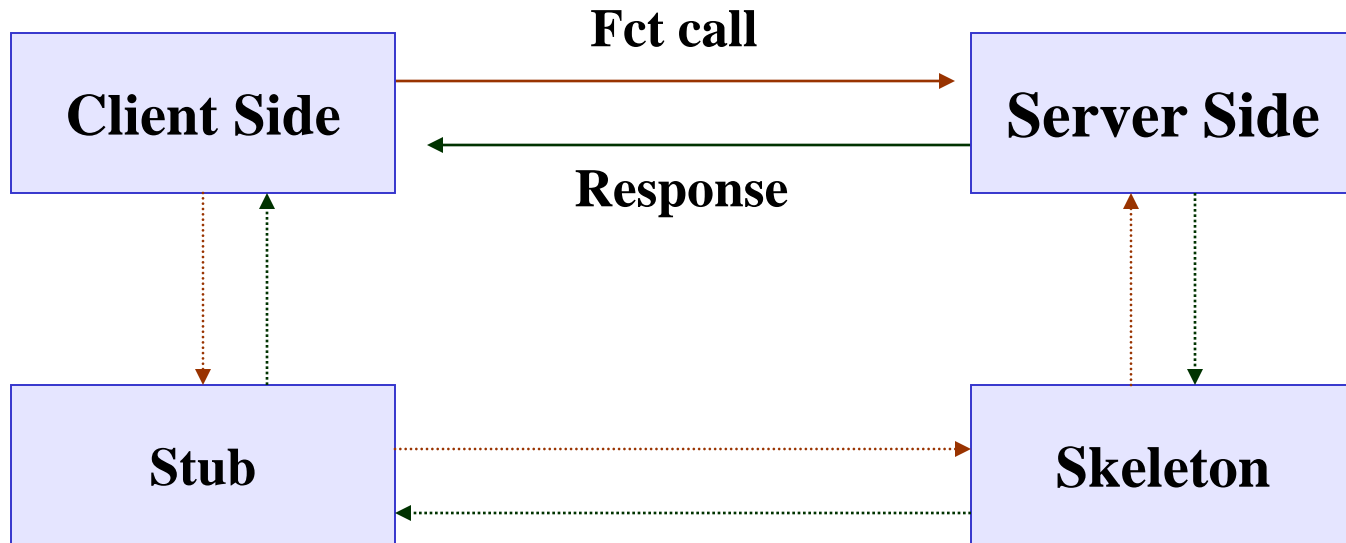
RMI(Remote Method Invocation)

- RMI is a specification.
(하나의 JVM에 존재하는 객체가 어떻게 다른 JVM에 존재하는 객체의 메소드를 부르느냐...)
- Sun Microsystems의 이 스펙에 대한 구현은 JRMP (Java Remote Method Protocol) 이다.
- Distributed computing model을 구현하기 위한 자바 Object 간의 통신을 구현한 도구이다.

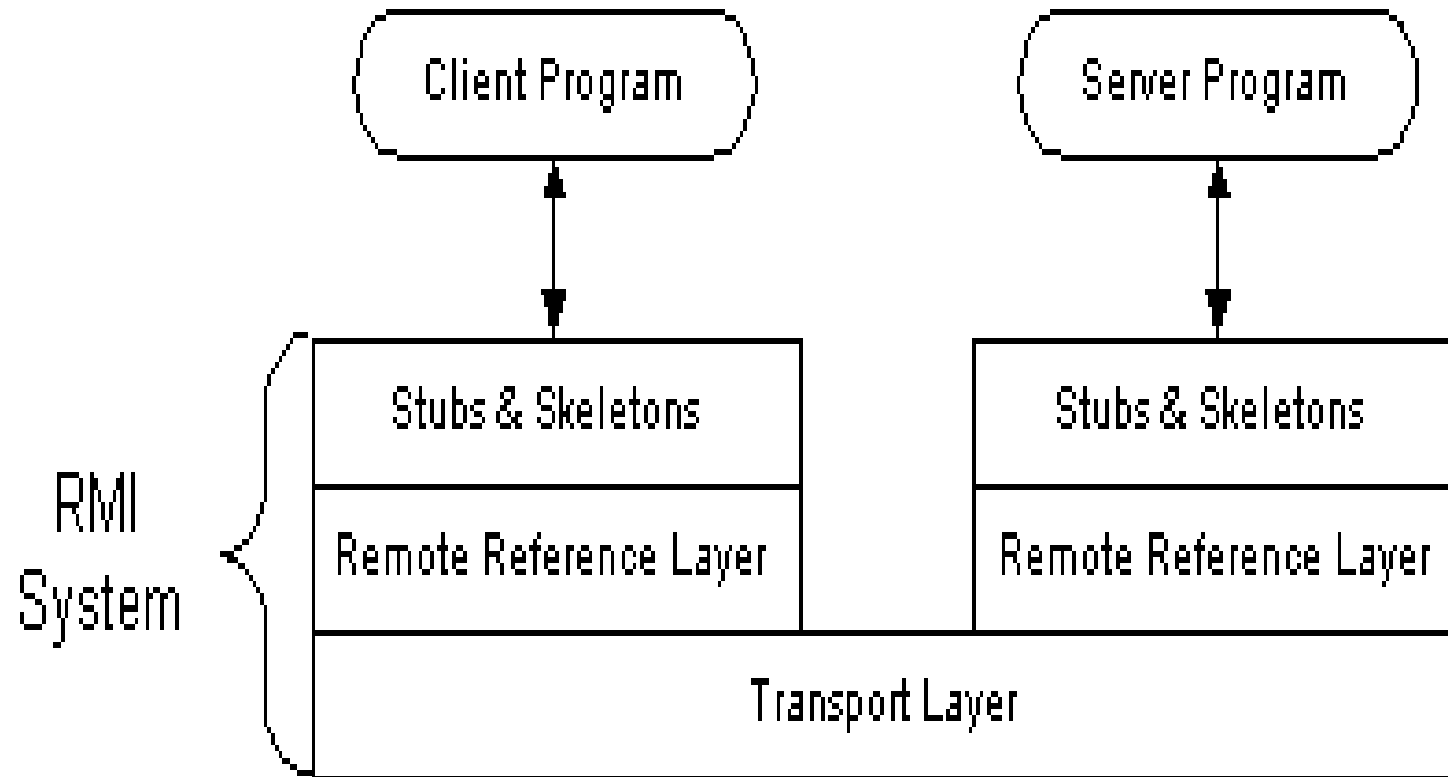
RMI Concept



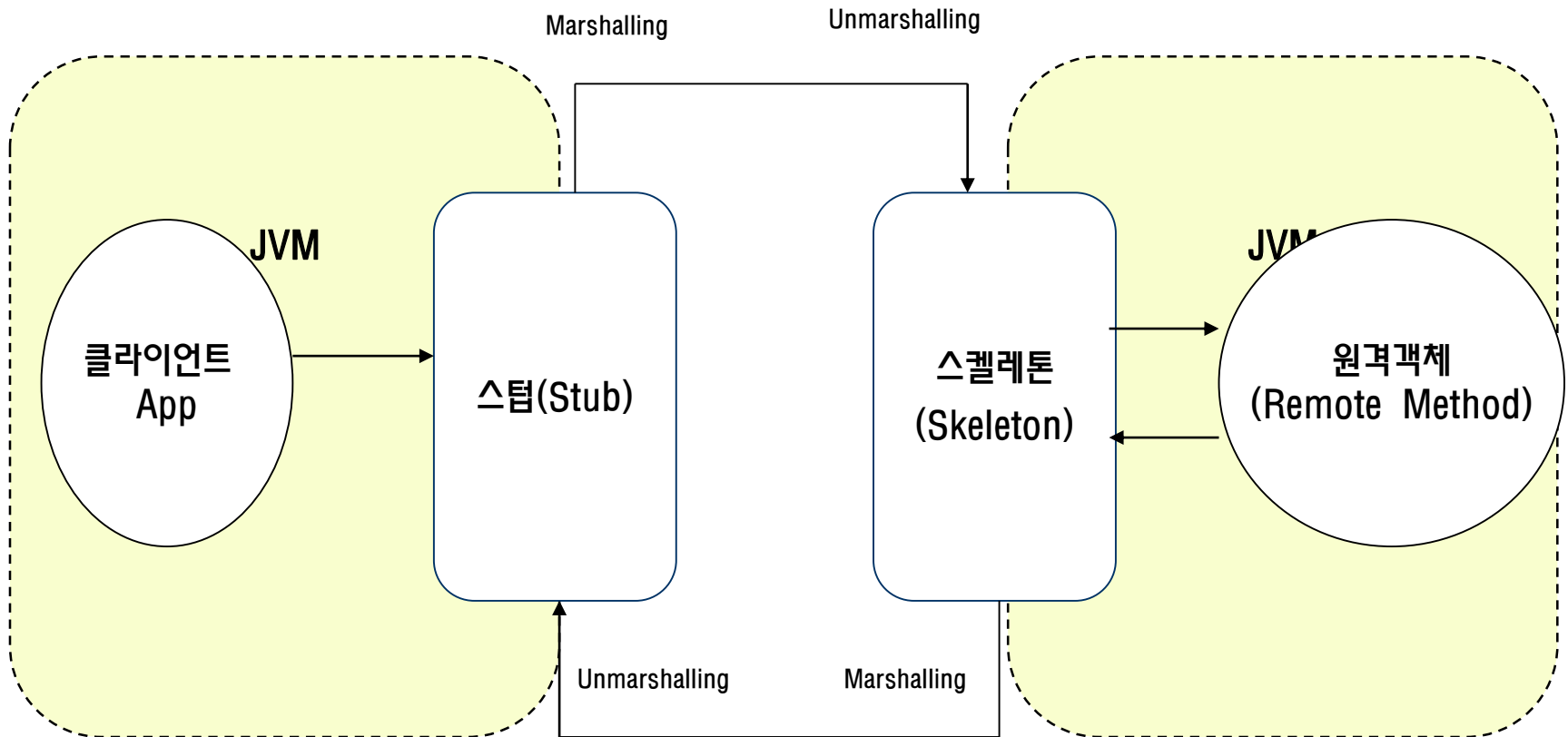
RMI Concept



RMI Concept



RMI Concept



RMI 작성 과정 – 원격 인터페이스 정의

- 원격 객체 사이의 메시지 전송을 위한 것
- 자바 RMI에서는 RMI 클라이언트와 RMI 서버 간의 메시지 전송을 위한 방법으로 자바의 인터페이스를 이용한다. 즉 인터페이스에서 정의한 메소드로 클라이언트와 서버간의 통신이 이루어진다.

RMI 작성 과정 – 원격 인터페이스 구현 클래스 정의(servant)

- 원격 인터페이스에서 정의한 메소드를 구현 한 클래스

RMI 작성 과정 – RMI 서버 App 작성

- 원격 인터페이스를 구현한 클래스의 인스턴스를 생성하고 이를 원격의 **RMI** 클라이언트가 접근하여 구현된 비즈니스 메소드를 원격 호출할 수 있도록 하는 프로그램 이다.
- 즉 원격 인터페이스를 구현한 클래스의 인스턴스를 생성하고 이를 원격의 클라이언트가 접근할 수 있도록 이름으로 등록하는 일을 담당하는 프로그램

RMI 작성 과정 – RMI 클라이언트 App 작성

- RMI 서버 Application이 생성한 원격 객체를 접근함으로써 원격 인터페이스에 정의된 비즈니스 메소드를 필요에 따라 호출하는 프로그램

“Hello World” RMI Application

“Hello World” RMI Application

- 하나의 RMI Application을 작성 하기 위해서는 3~4개 정도의 자바 파일이 필요 하다.
- 필요한 파일
 - 원격 인터페이스 : 자바 인터페이스
 - 원격 인터페이스를 구현한 클래스 : 클래스
 - RMI 서버 Application : 클래스
 - RMI 클라이언트 Application : 클래스

“Hello World” – 원격 인터페이스

- 원격 인터페이스는 원격 객체에 대하여 클라이언트가 호출 할 수 있는 비즈니스 메소드를 정의 한다.
- 클라이언트는 원격 인터페이스에서 정의된 메소드만 서로 다른 실행환경에서 원격으로 호출 할 수 있다.
- 이렇게 원격에서 호출 가능한 메소드를 “원격 메소드” 라고 한다.
- Hello.java에서는 sayHello라는 원격 메소드를 정의 하고 있다.

“Hello World” – 원격 인터페이스

```
//Hello.java
```

```
public interface Hello extends java.rmi.Remote {  
    public String sayHello(String name) throws  
        java.rmi.RemoteException;  
}
```

“Hello World” – 원격 인터페이스를 구현한 클래스

- HelloImpl.java는 원격 인터페이스 Hello를 구현한 클래스이며 HelloImpl 클래스의 인스턴스가 원격객체로서 RMI 서버에 의해 이름으로 등록된다.
- “Hello World” Application 에서는 `java.rmi.UnicastRemoteObject` 를 상속하며 원격 인터페이스를 구현하고 있다.

“Hello World” – 원격 인터페이스를 구현한 클래스

```
//HelloImpl.java
```

```
import java.rmi.server.*;
```

```
import java.rmi.*;
```

```
public class HelloImpl extends UnicastRemoteObject implements
```

```
    Hello {
```

```
    public HelloImpl() throws RemoteException {
```

```
        super();
```

```
    }
```

```
//원격 메소드 구현
```

```
    public String sayHello(String name) {
```

```
        return "Hello World ... " + name + "!";
```

```
    }
```

```
}
```


“Hello World” – RMI 서버 Application

- HelloServer.java의 경우 RMI 서버 Application의 기능만을 정의한 단순한 클래스 이다.
- HelloImpl 객체를 생성하고 “HelloRemote”라는 이름으로 등록하며 이후 RMI 클라이언트에서의 메소드 호출을 기다린다.

“Hello World” – RMI 서버 Application

```
//HelloServer.java
```

```
public class HelloServer {  
    public static void main(String[] args) {  
        try {  
            HelloImpl remoteObj = new HelloImpl();  
            java.rmi.Naming.rebind("rmi://localhost:1099/HelloRemote", remoteObj);  
            System.out.println("Hello Remote Object bound to the registry and  
ready to service incoming client calls...");  
        } catch(java.rmi.RemoteException e) {  
            System.err.println("Exception occurred during processing incoming  
method call");  
        } catch(java.net.MalformedURLException e) {  
            System.err.println("Check the url String...");  
        }  
    }  
}
```

“Hello World” – RMI 클라이언트 Application

- RMI 서버 Application 에서 등록한 원격객체 “HelloRemote” 에 대한 reference를 얻고 있는 부분을 제외하면 실제 로컬의 런타임 환경에서 생성한 객체와 전혀 다를 바 없는 레퍼런스를 사용한 메소드 호출을 하고 있다.

“Hello World” – RMI 클라이언트 Application

//HelloClient.java, 실행시 이름을 인자로...

```
import java.rmi.Naming;

public class HelloClient {
    public static void main(String[] args) {
        try {
            Object obj =
Naming.lookup("rmi://localhost:1099/HelloRemote");
            Hello remoteObj = (Hello)obj;
            String msg = remoteObj.sayHello(args[0]);
            System.out.println(msg);
        }
    }
}
```

“Hello World” – RMI 클라이언트 Application

```
        catch(java.rmi.RemoteException e) {  
            System.out.println("Something has gone  
wrong during remote method call...");  
        }  
        catch(java.rmi.NotBoundException e) {  
            System.out.println("Could't bound...");  
        }  
        catch(java.net.MalformedURLException e) {  
            System.out.println("Check url stirng...");  
        }  
    }  
}
```

“Hello World” 실행 방법

- RMI 소스 파일을 컴파일 한다.
- 원격 인터페이스 구현 클래스에 대한 스텝 생성 (JAVA_HOME/bin/rmic 사용)
 - rmic HelloImpl
- 네이밍 서버 데몬 시작
 - rmiregistry 1099 & (Unix)
 - start rmiregistry 1099 (Windows)
- RMI 서버 실행
 - java HelloServer
- RMI 클라이언트 실행
 - java HelloClient jclee

클라이언트와 서버로
분리하는 경우

서버

HelloServer.java
HelloImpl.java
Hello.java

클라이언트(rmic까지)

Hello.java
HelloImpl.java
HelloClient.java

C:\WINNT\System32\cmd.exe

E:\강의자료\여성부\자바네트웍\예제\HelloWorld_rmi>javac Hello.java

E:\강의자료\여성부\자바네트웍\예제\HelloWorld_rmi>javac HelloImpl.java

E:\강의자료\여성부\자바네트웍\예제\HelloWorld_rmi>javac HelloServer.java

E:\강의자료\여성부\자바네트웍\예제\HelloWorld_rmi>javac HelloClient.java

E:\강의자료\여성부\자바네트웍\예제\HelloWorld_rmi>rmic HelloImpl

E:\강의자료\여성부\자바네트웍\예제\HelloWorld_rmi>dir

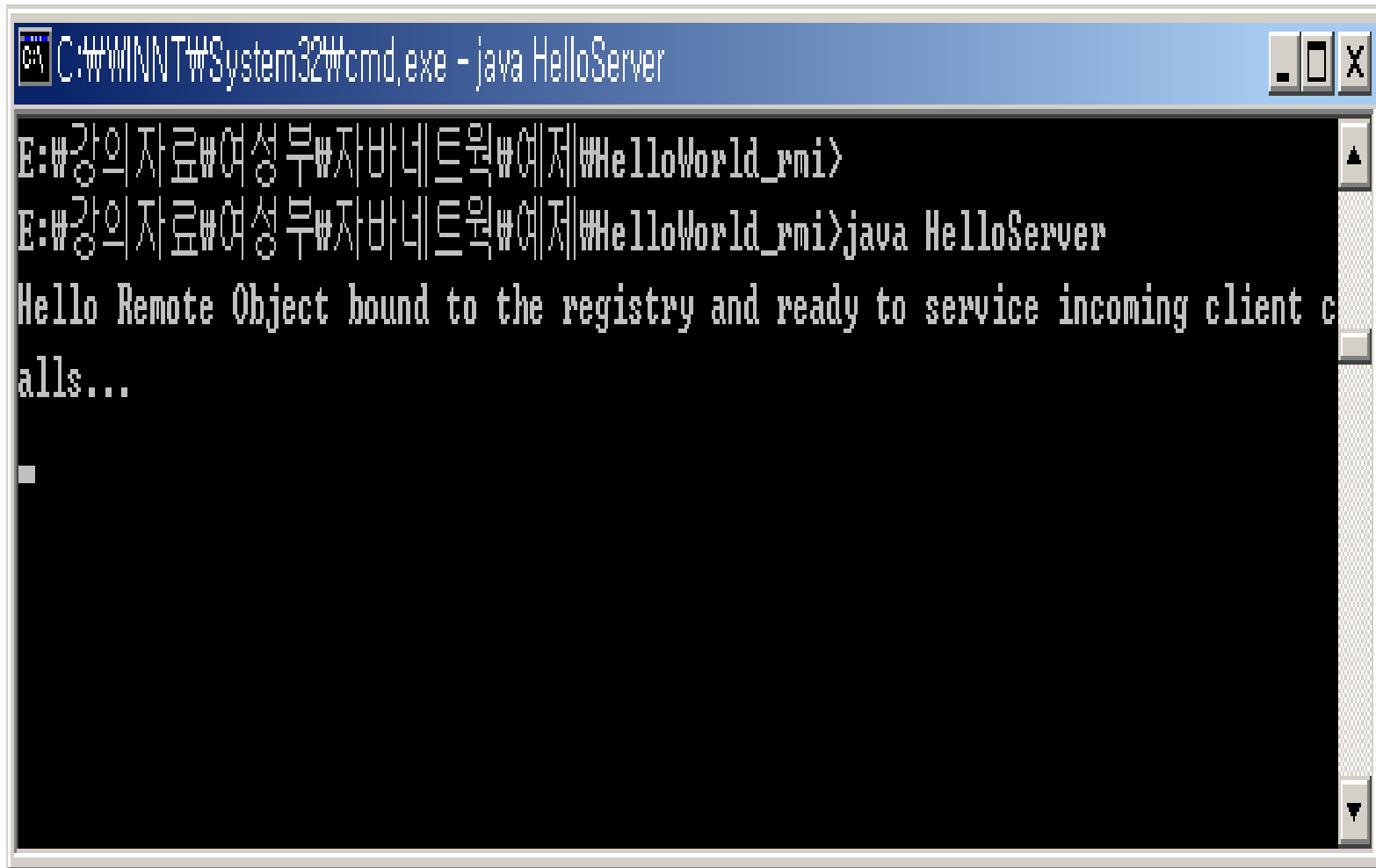
E 드라이브의 볼륨: data2
볼륨 일련 번호: 0C4C-6A0A

E:\강의자료\여성부\자바네트웍\예제\HelloWorld_rmi 디렉터리

2003-04-02	10:21a	<DIR>	.
2003-04-02	10:21a	<DIR>	..
2003-04-02	10:20a		227 Hello.class
2003-04-02	10:16a		122 Hello.java
2003-04-02	10:21a		1,183 HelloClient.class
2003-04-02	10:16a		659 HelloClient.java
2003-04-02	10:21a		568 HelloImpl.class
2003-04-02	10:16a		295 HelloImpl.java
2003-04-02	10:21a		1,748 HelloImpl_Skel.class
2003-04-02	10:21a		3,270 HelloImpl_Stub.class
2003-04-02	10:21a		1,135 HelloServer.class
2003-04-02	10:16a		577 HelloServer.java
	10개 파일		9,784 바이트
	2 디렉터리		1,623,781,376 바이트 남음

E:\강의자료\여성부\자바네트웍\예제\HelloWorld_rmi>

HelloServer 실행 모습



```
C:\WINNT\System32\cmd.exe - java HelloServer

E:\강의자료\여성부\자바네트웍\예제\HelloWorld_rmi>
E:\강의자료\여성부\자바네트웍\예제\HelloWorld_rmi>java HelloServer
Hello Remote Object bound to the registry and ready to service incoming client calls...

■
```


RMI

- RMI에서 원격객체를 생성하고 이를 이름으로 등록한 후 원격 객체에 대한 클라이언트의 서비스 요청을 기다리는 쪽이 서버 이다.
- 원격 객체를 이름으로 찾아 해당 객체가 제공하는 원격 메소드를 호출하는 프로그램이 RMI 클라이언트 이다.

RMI

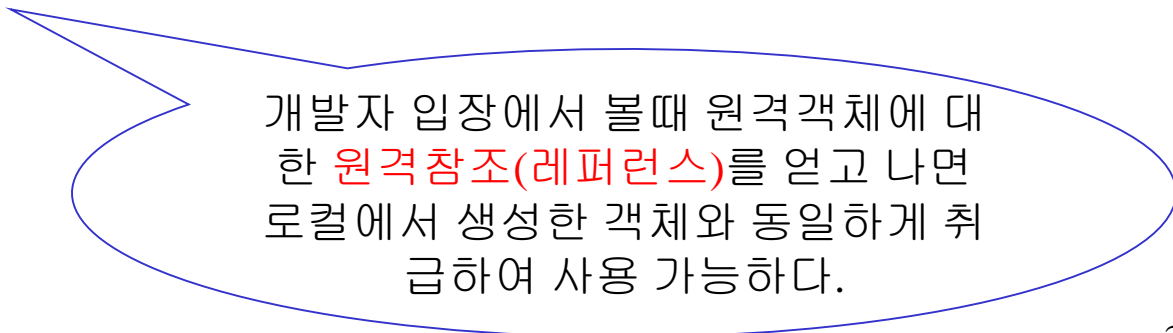
- 원격 객체에 대한 레퍼런스(원격 레퍼런스)를 이름을 통해 얻어 낼 수 있도록 rmiregstry 라는 네이밍 서버 데몬을 제공한다.
- 원격 참조(레퍼런스)를 통해 서로 다른 JVM 환경의 원격 메소드를 자신의 로컬 JVM 환경에서 생성한 객체와 다름없이 호출 할 수 있는 메커니즘(stub, skeleton)을 제공한다.
- 원격 객체의 메소드 호출 시 객체를 주고 받을 수 있는 메커니즘을 제공한다. (원격 객체 메소드 호출시 사용되는 파라미터나 리턴 타입이 Serializable을 구현했을 경우 객체 직렬화를 이용 가능)

RMI

- RMI에서 pass-by-value이 경우는 원격 메소드 호출 시 사용하는 파라미터나 리턴 타입에 적용되며 pass-by-reference인 경우는 원격 객체에 대한 레퍼런스이다. pass-by-reference는 stub과 skeleton의 통신시에 사용된다.
- 다음 코드에서 remoteObj가 Hello 타입의 원격 레퍼런스다.

//HelloClient.java

```
Object obj = Naming.lookup("rmi://localhost:1099/HelloRemote");  
Hello remoteObj = (Hello)obj;
```



개발자 입장에서 볼때 원격객체에 대한 **원격참조(레퍼런스)**를 얻고 나면 로컬에서 생성한 객체와 동일하게 취급하여 사용 가능하다.

RMI

- RMI는 원격 객체에 대한 참조로 stub을 사용한다. 즉 RMI 클라이언트 Application에서 사용하는 원격 객체의 reference는 실제로 stub에 대한 reference 이다.
- 하나의 JVM에서 생성한 객체를 다른 JVM에서 그대로 실행 한다는 것은 불가능 하다.(주소 공간이 틀리기 때문), 즉 이것이 RMI에서 원격 객체에 대한 참조를 원격 객체를 직접 가리키지 않고 stub을 사용하는 이유이다.
- 예를들면 RMI 클라이언트에서 RMI 서버의 메소드를 호출 한다고 했을 때 먼저 로컬 stub 객체에 먼저 전달된 후에 원격의 skeleton 객체에 전달되고 skeleton이 실제 객체의 비즈니스 메소드를 호출 하는것이다.

RMI

- 원격 메소드 호출시에 사용하는 실매개변수와 리턴되는 데이터의 상태를 원격의 클라이언트/서버 간에 유지할 필요가 없을때는 직렬화(Serialization)를 통해 Pass-by-value로 복사본을 주고 받는다.(객체나 자바 내장 데이터 타입인 경우 객체 직렬화를 사용한다.)
- 자바 RMI의 경우 서버에서 이름으로 등록한 원격 객체를 클라이언트에서 개발자가 직접 저수준의 네트웍 코드를 직접 작성하지 않고 등록된 이름으로 원격에 존재하는 비즈니스 객체를 참조 할 수 있도록 rmiregistry라는 Naming Server Daemon을 제공한다.

RMI

- 아래의 코드는 rmiregistry를 통해 원격 객체 remoteObj를 “HelloRemote”라는 이름으로 등록하는 RMI 서버 Application의 일부분 이다.

//원격 객체 생성

```
HelloImpl remoteObj = new HelloImpl();
```

//생성한 원격 객체를 “HelloRemote”라는 이름으로 rmiregistry에 등록 한다.

```
java.rmi.Naming.rebind("rmi://localhost:1099/HelloRemote", remoteObj);
```

RMI

- 앞의 코드에서 처럼 rmiregistry에 등록된 원격 객체에 대한 reference를 얻어내어 원격 메소드를 호출하는 RMI 클라이언트의 부분은 아래와 같다.

//rmiregistry에서 “HelloRemote”로 등록된 원격객체의 reference를 Object type으로 얻는다.

Object obj =

Naming.lookup("rmi://localhost:1099/HelloRemote");

//원격 메소드의 호출을 위해 적절한 원격 인터페이스 타입으로 Casting 한다.

Hello remoteObj = (Hello)obj;

//로컬객체에 대한 메소드 호출과 같이 원격메소드를 호출한다.

String msg = remoteObj.sayHello(args[0]);

RMI

- 앞의 코드에서 처럼 자바는 `rmiregistry`라는 네이밍 서비스를 통해 개발자가 원격 객체에 대해 접근하는 저수준의 메트릭 코드를 사용하지 않고도 이름으로 투명하게 접근 할 수 있는 방법을 제공한다.

“Hello World” 다시보기-원격 인터페이스

- 분산 객체를 사용하는 것은 이 분산객체(원격으로 서비스를 제공하는 객체)를 통해 원격의 클라이언트에 제공 하고자 하는 서비스(메소드)가 있을 것이다.
- 원격 인터페이스는 클라이언트가 원격으로 호출할 메소드 즉 원격 메소드를 정의하는 것이다.
- 원격 객체가 구현 할 인터페이스 이다.
- `java.rmi.Remote` 인터페이스를 직/간접적으로 상속해야 한다.(보통 직접 상속 한다.)

“Hello World” 다시보기-원격 인터페이스

- 원격 인터페이스에서 정의한 모든 메소드는 `java.rmi.RemoteException`이나 부모클래스 (`java.io.IOException`, `java.lang.Exception`)을 throws 해야 한다.(RMI 스펙에는 `RemoteException`을 throws 하도록 권고 하고 있다.)

```
public interface Hello extends java.rmi.Remote {  
    public String sayHello(String name) throws  
    java.rmi.RemoteException;  
}
```

“Hello World” 다시보기-원격 인터페이스 구현 클래스

- RMI 서버 Application은 원격 인터페이스 구현 클래스의 인스턴스를 생성하여 이를 rmiregistry에 이름으로 등록 한다.
- 원격 객체는 자신을 원격의 클라이언트가 접근 할 수 있도록 등록 할 수 있어야 하며 클라이언트에서 원격 메소드 요청이 발생 했는지 지속적으로 모니터링 할 수 있어야 한다. 아쉽게도 원격 인터페이스에는 이러한 기능은 정의되지 않았다.
- HelloImpl 객체가 원격 객체로 작동 할 수 있는것은 원격메소드를 구현해서가 아니라 java.rmi.server.UnicastRemoteObject를 상속했기 때문이다.

“Hello World” 다시보기-원격 인터페이스 구현 클래스

- java.rmi.server.RemoteObject,
java.rmi.server.RemoteServer 클래스에서 정의하는 기능을 가져야 한다.
 - 원격객체는 RMI 스펙에서 제안하는 규격에 맞게 java.lang.Object에서 상속한 hashCode, equals, toString 메소드를 재정의 할 필요가 있으며 원격객체임을 RMI 시스템에 알리고 자신을 원격 객체로 rmiregistry에 등록 할 수 있어야 한다.

“Hello World” 다시보기-원격 인터페이스 구현 클래스

- RemoteObject 클래스는 hashCode, equals, toString을 RMI에 맞게 재정의 하고 있으며 RemoteServer 클래스는 원격 클라이언트에서 오는 원격 메소드 호출을 처리하는 기능을 정의 하고 있다.
- 이 두개의 클래스 RemoteObject와 RemoteServer의 기능을 동시에 가지고 있는 클래스가 UnicastRemoteObject 클래스이며 일반적으로 원격 인터페이스를 구현한 클래스(RMI 원격 객체 클래스)는 UnicastRemoteObject를 상속한다. 그러나 반드시 상속해야 하는 것은 아니며 기능을 구현한 다른 클래스를 상속해도 무방하다.

“Hello World” 다시보기-원격 인터페이스 구현 클래스

- 원격 객체 기능 구현을 위해
java.rmi.server.RemoteObject 클래스를 상속한 경우에는 default 생성자를 반드시 정의 해야 한다.
 - UnicastRemoteObject 클래스는 자신의 생성자에서 원격 객체를 등록하는 기능을 정의하고 있다.
 - 왜 반드시 해야할까? (생성자에서 throws 하는 이유) → 안하면 컴파일시 에러가 발생한다. 물론 이것은 이유가 아니며 궁극적인 이유는 상속받는 UnicastRemoteObject의 생성자가 RemoteException을 throws 하고 있으므로 하위 클래스인 “원격 인터페이스 구현 클래스”의 생성자에서도 RemoteException을 throws 해야 한다.

“Hello World” 다시보기-원격 인터페이스 구현 클래스

- 원격 인터페이스에서 정의하는 원격 메소드 이외의 메소드도 정의가 가능하다.
 - 원격 메소드 이외의 메소드는 오로지 Local의 JVM에서만 호출이 가능 하다.

“Hello World” 다시보기-서버/클라이언트 Application

- RMI 서버 Application은 원격 객체를 생성하고 이를 rmiregistry에 이름으로 등록하는 프로그램 이다.
- RMI에서는 원격 객체를 이름으로 등록하거나 원격 객체에 대한 원격 reference를 얻어내기 위한 용도로 java.rmi.Naming이라는 유틸리티 클래스를 제공하고 있다.

“Hello World” 다시보기-서버/클라이언트 Application

- 원격 객체를 생성/등록 하기 위해서는 Naming 클래스의 static 메소드인 bind(...)나 rebind(...)를 사용하고 등록된 원격 객체에 접근을 위해서는, 즉 원격객체에 대한 원격 reference를 얻기 위해서는 lookup(...) 메소드를 사용 한다.
- RMI 클라이언트 프로그램에서 하는 일은 lookup해서 원격 메소드를 호출하는 일 이다.

RMI 기본 구조

JRMP(Java Remote Method Protocol)

- 자바 RMI의 기반이 되는 전송 프로토콜
- 클라이언트/서버가 순수 자바 환경으로 이루어진 분산 객체를 위해 설계된 프로토콜
- 썬에서는 RMI 클라이언트나 서버가 CORBA 클라이언트나 서버와 통신이 가능 하도록 RMI/IIOP를 지원하고 있다.

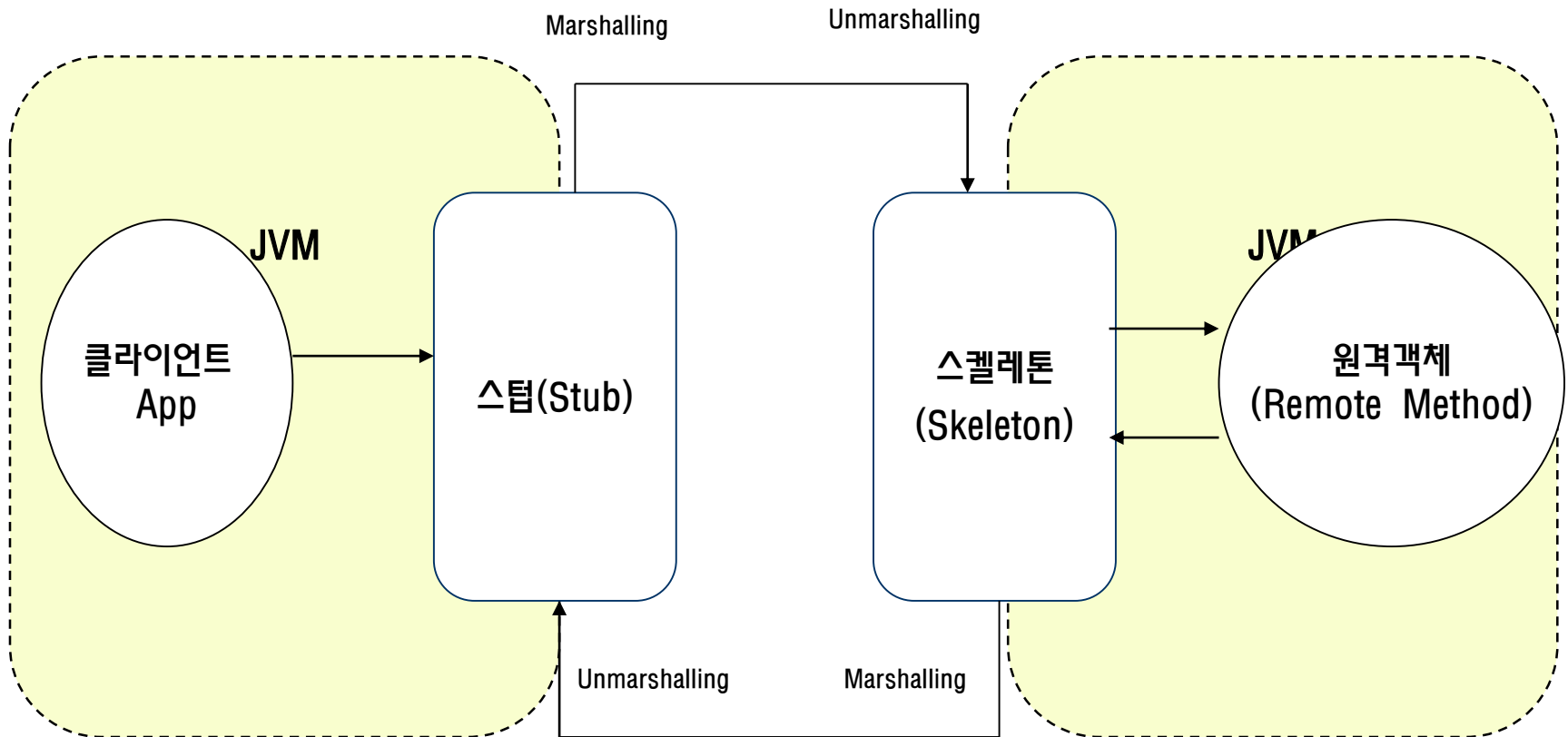
Stub & Skeleton

- 자바 RMI에서 원격 객체의 메소드를 호출해서 사용하기 위해 다른 메모리 공간(JVM)에 존재하는 원격 객체에 대한 클라이언트쪽 참조(레퍼런스)로서 원격 객체 자체에 대한 레퍼런스를 사용하지 않고 원격 객체에 대한 stub을 참조를 사용한다.
- 이는 자신이 실행되는 JVM과 다른 메모리 공간에 존재하는 객체의 참조(주소, 포인터)는 의미가 없기 때문이다.

Stub & Skeleton

- RMI 클라이언트는 stub 참조를 통해 원격 메소드를 호출하고 stub 은 호출시에 넘어오는 메소드 명이나 파라미터등을 적절한 형태로 packaging 하여 네트워크를 통해 RMI 서버 Application의 원격 객체에 대한 Skeleton에 전달한다.
- RMI 서버에서는 이를 다시 unpackaging 하여 원격메소드 명과 파라미터등을 알아내어 실제 원격 객체의 메소드에 전달 한다. 원격메소드의 실행이 완료되어 return 하면 Skeleton은 다시 이 결과를 packaging하여 네트워크를 통해 클라이언트의 stub에게 전송 한다. 이를 packaging하는것을 marshalling, 반대로 unpackaging하는것을 unmarshalling이라 한다.

Stub & Skeleton



원격 객체와 Stub

- RMI 클라이언트는 원격 객체의 원격 메소드만을 호출할 수 있으며 이 호출을 위해 실제의 원격객체에 대한 참조 대신 원격 객체의 stub reference를 이용한다고 하였다. 이는 곧 원격객체가 구현한 모든 원격 메소드를 원격객체에 대한 stub 객체도 함께 가지고 있다는 것을 의미 한다.(물론 동일 한 것은 아니다.)
- 앞서 작성한 “Hello World”의 경우 HelloImpl은 원격 인터페이스 Hello를 구현한 원격 객체 클래스이며 이를 통해 rmic를 통해 작성된 HelloImpl_stub 역시 HelloImpl과 동일한 Hello를 구현 한 클래스 이다.

원격 객체와 Stub

- 앞서 작성한 “Hello World”의 클라이언트 Application 인 HelloClient.java를 살펴보면 lookup 메소드를 통해 Return되는 실제 객체는 HelloRemote라는 이름으로 등록된 원격객체 HelloImpl 객체에 대한 stub인 HelloImpl_stub 객체가 Return 될 것이다.
- 또한 stub reference인 remoteObj를 Hello로 캐스팅 가능한 이유는 remoteObj가 참조하는 stub 객체인 HelloImpl_stub이 Hello 인터페이스를 구현 했기 때문이다.
- `javap HelloImpl_stub` 명령을 통해 stub 객체의 메소드를 확인하는 것은 가능하다.

객체 직렬화

- 원격 객체에 대한 reference가 실제로 원격 객체에 대한 stub 객체를 통해 클라이언트에 참조로 전달된다면(원격객체에 대한 참조가 stub 참조로 전달되는 경우)참조로 전달되는 원격 메소드의 파라미터나 반환 데이터는 복사본이 전송된다(pass-by-value) 이때 RMI는 객체 직렬화를 이용 한다.
- 결국 marshalling과 unmarshalling은 객체 직렬화와 환원이다.
- 그러므로 기본적으로 원격 메소드의 파라미터나 리턴 타입은 객체 직렬화가 가능 해야 한다.

객체 직렬화

- 자바에서 직렬화가 가능한 조건은 다음과 같다.
 - java.io.Serializable 인터페이스를 직/간접적으로 구현해야 한다.
 - 내장 데이터 타입은 기본적으로 직렬화 가능하다.
 - 멤버변수에 transient로 선언하면 직렬화에서 제외된다.(멤버 변수중 직렬화가 불가능한 변수등이 있을 경우 객체 자체가 직렬화 되지 못하는 오류가 발생할 수 있으므로 이를 해결하기 위해 “이 멤버는 직렬화에서 제외 하세요” 라는 뜻이다.)
 - static 멤버 변수는 직렬화 되지 않는다.

객체 직렬화

- “Hello World” RMI Application에서 Hello.java의 원격 메소드인 sayHello를 살펴보자. RMI 클라이언트에서 stub을 통해 sayHello 메소드를 호출시 name pass-by-value방식으로 복사되어 원격 서버로 전송된다. 이때 파라미터 name은 형이 String이므로 직렬화 되어 전송된다.
- 만약 직렬화 불가능한 파라미터등이 전달된다면 java.rmi.MarshalException 이 발생 한다.
- sayHello의 반환 값 역시 Skeleton을 통해 클라이언트의 stub으로 직렬화 되어 전송된다. 만약 sayHello의 return 형이 Serializable을 구현하지 않았다면 java.rmi.MarshalException 이 발생 한다.

객체 직렬화

- 조금만 더 깊이 들어가 보자... 자바 RMI에서 원격 메소드의 호출은 stub/skeleton을 통해 이루어지며 데이터 전송을 위해서는 marshalling과 unmarshalling을 한다고 했으며 이는 직렬화 및 환원 과정 이라고 했다.
- 결국 stub과 skeleton에서는 직렬화 및 환원을 위해 ObjectOutputStream과 ObjectInputStream이 필요하다는 것에 대해 감을 잡았을 것이다. 다시 말하면 marshalling을 위해 ObjectOutputStream.writeObject(...) , unmarshalling을 위해 ObjectInputStream.readObject(...)를 사용한다.

RMI Example 2

확장 가능한 RMI Application

- 원격 인터페이스 : Workable.java
- 원격 인터페이스 구현 : GenericServer.java
- RMI 클라이언트 : QueryClient.java
- 서버/클라이언트 공유 I/F : Anything.java
QueryObject.java

원격 인터페이스

```
//Workable.java
```

```
package pejb.rmi.base;
```

```
import java.rmi.Remote;
```

```
import java.rmi.RemoteException;
```

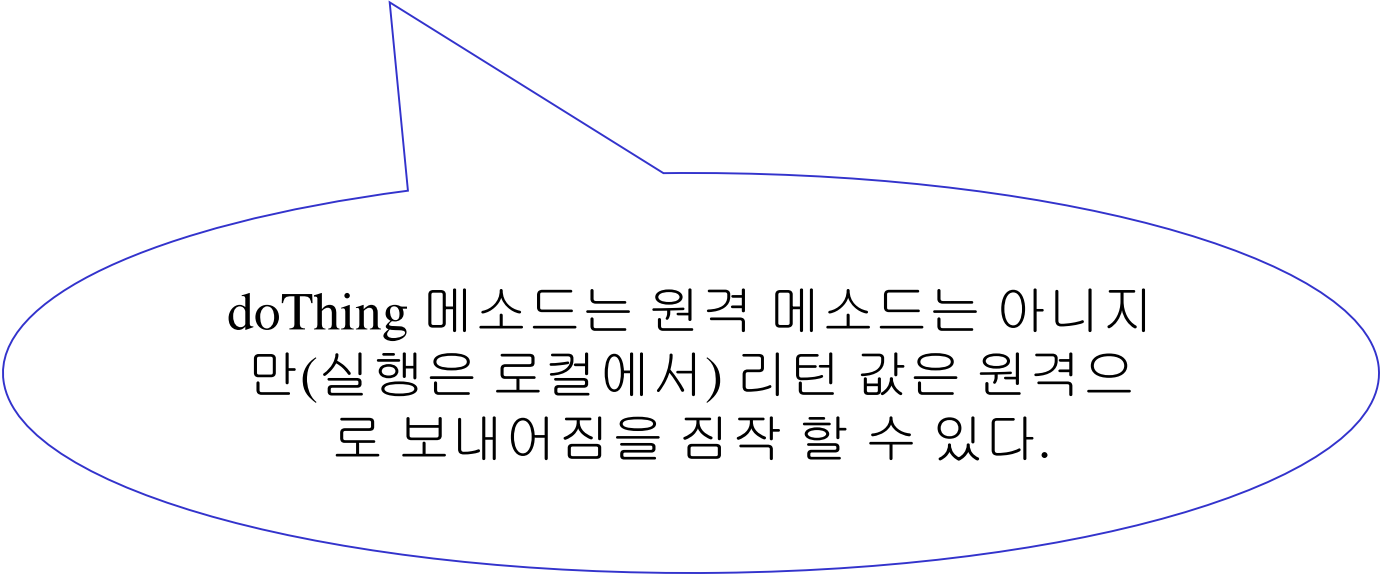
```
import java.io.Serializable;
```

```
public interface Workable extends Remote {
```

```
    public Serializable doJob(Anything any) throws RemoteException;
```

```
}
```

```
//Anything.java
package pejb.rmi.base;
import java.io.Serializable;
public interface Anything extends Serializable {
    public java.io.Serializable doThing();
}
```



doThing 메소드는 원격 메소드는 아니지만(실행은 로컬에서) 리턴 값은 원격으로 보내어짐을 짐작 할 수 있다.

원격 인터페이스 구현 & RMI 서버

```
//GenericServer.java
package pejb.rmi.server;
import java.rmi.server.*;
import java.rmi.*;
import java.rmi.registry.Registry;
import pejb.rmi.base.*;
public class GenericServer extends UnicastRemoteObject implements
    Workable {
    public GenericServer() throws RemoteException { super(); }
    public java.io.Serializable doJob(Anything any) {
        return any.doThing();
    }
}
```

원격 인터페이스 구현 & RMI 서버

```
public static void main(String[] args) {  
    String rmiString = null, host=null;  
    try {  
        Workable remoteObj = new GenericServer();  
        System.out.println(rmiString);  
        Naming.rebind("rmi://localhost:1099/Workable", remoteObj);  
        System.out.println("Remote Object registered and ready to  
servie..." );  
    }  
    catch(Exception e) {  
        System.err.println("Could't successfully register remote object" + e);  
    }  
}
```

QueryClient.java/QueryObject.java

- QueryClient.java는 RMI 서버 Application에서 생성한 원격객체(GenericServer 객체)를 “Workable”이라는 이름으로 찾아 원격 메소드 doJob을 호출하는 일을 하고 있다.
- QueryClient.java 클래스는 원격 레퍼런스 workable을 통해 원격 메소드 doJob()을 호출하기 전에 QueryObject 객체를 생성하고 이 객체를 doJob 호출 시에 매개변수(파라미터)로 사용하고 있다.

QueryClient.java/QueryObject.java

- Anything 인터페이스를 구현한 QueryObject의 객체가 클라이언트에서 원격 메소드 doJob()에 복사되어 전달되면, 서버측의 GenericServer 객체 doJob() 메소드에서는 파라미터로 전달되어 온 QueryObject의 객체 qryObj의 doThing() 메소드를 호출한 결과를 다시 QueryClient 클라이언트로 반환한다.
- 여기서 중요한 것은 QueryObject 객체에 비즈니스 로직이(doThing) 정의 되어 있고 이것이 RMI 클라이언트로 부터 RMI 서버 Application으로 전달되어 실제로는 서버측의 원격 객체 doJob() 내에서 실행된 결과값이 리턴 되는 것이다.

QueryClient.java/QueryObject.java

- 원격 객체 클래스 GenericServer에는 비즈니스 메소드 doJob에 대해 로직이 정의 되어 있지 않다. 단지 파라미터로 전달된 Anything 객체의 doThing을 호출하고 이를 Return하는 기능만 정의 되어 있다. 이는 클라이언트에서 복사되어 전달되는 Anything 구현 객체가 정의한 doThing 메소드에 따라 서버측의 원격 객체 비즈니스 로직이 변경 될 수 있음을 나타낸다. “결국 확장 가능한 RMI Application” 이다.

RMI 클라이언트

```
//QueryClient.java
```

```
package pejb.rmi.client;
```

```
import java.rmi.Naming;
```

```
import java.rmi.registry.Registry;
```

```
import java.util.ArrayList;
```

```
import pejb.rmi.base.Workable;
```

```
public class QueryClient {
```

```
    public static void main(String[] args) {
```

```
        String rmiString=null;
```

```
        String sql="select id from users";
```

```
        try {
```

RMI 클라이언트

```
Workable workable =
(Workable)Naming.lookup("rmi://211.55.52.254:1099/Workable");
QueryObject qryObj = new QueryObject(sql);
System.out.println("Result for Query " +sql );
ArrayList arrayList = (ArrayList)workable.doJob(qryObj);
for(int i=0; i<arrayList.size(); i++) {
    System.out.println(arrayList.get(i));
}
}
catch(Exception e) {
    System.out.println("Could't successfully register remote object..." +e);
}
}
}
```

QueryObject.java

- “Hello World” RMI Application에서 Hello.java의 원격 메소드인 sayHello를 살펴보자. RMI 클라이언트에서 stub을 통해 sayHello 메소드를 호출시 name pass-by-value방식으로 복사되어 원격 서버로 전송된다. 이때 파라미터 name은 형이 String이므로 직렬화 되어 전송된다.
- 만약 직렬화 불가능한 파라미터등이 전달된다면 java.rmi.MarshalException 이 발생 한다.
- sayHello의 반환 값 역시 Skeleton을 통해 클라이언트의 stub으로 직렬화 되어 전송된다. 만약 sayHello의 return 형이 Serializable을 구현하지 않았다면 java.rmi.MarshalException 이 발생 한다.

RMI 클라이언트

```
//QueryObject.java
```

```
package pejb.rmi.client;
```

```
import java.util.ArrayList;
```

```
import pejb.rmi.base.AnyThing;
```

```
import java.io.Serializable;
```

```
import java.sql.*;
```

```
public class QueryObject implements AnyThing {
```

```
    private String sql;
```

```
    private Connection con;
```

```
    private ArrayList arrayList;
```

RMI 클라이언트

```
public QueryObject(String sql) {    this.sql = sql;    }  
public Serializable doThing() {  
    try {  
        Class.forName("oracle.jdbc.driver.OracleDriver");  
        con =  
DriverManager.getConnection("jdbc:oracle:thin:@211.55.52.254:1  
521:WINK", "test", "test");  
        Statement stmt =  
con.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,  
  
        ResultSet.CONCUR_READ_ONLY);  
        ResultSet rs = stmt.executeQuery("select id from  
users");
```

RMI 클라이언트

```
rs.last();
arrayList = new ArrayList(rs.getRow());
rs.beforeFirst();
while(rs.next()) {
    arrayList.add(rs.getString("id"));
}
}
catch(Exception e) {
    System.out.println("Exception caught: " +
e.getMessage());
}
```

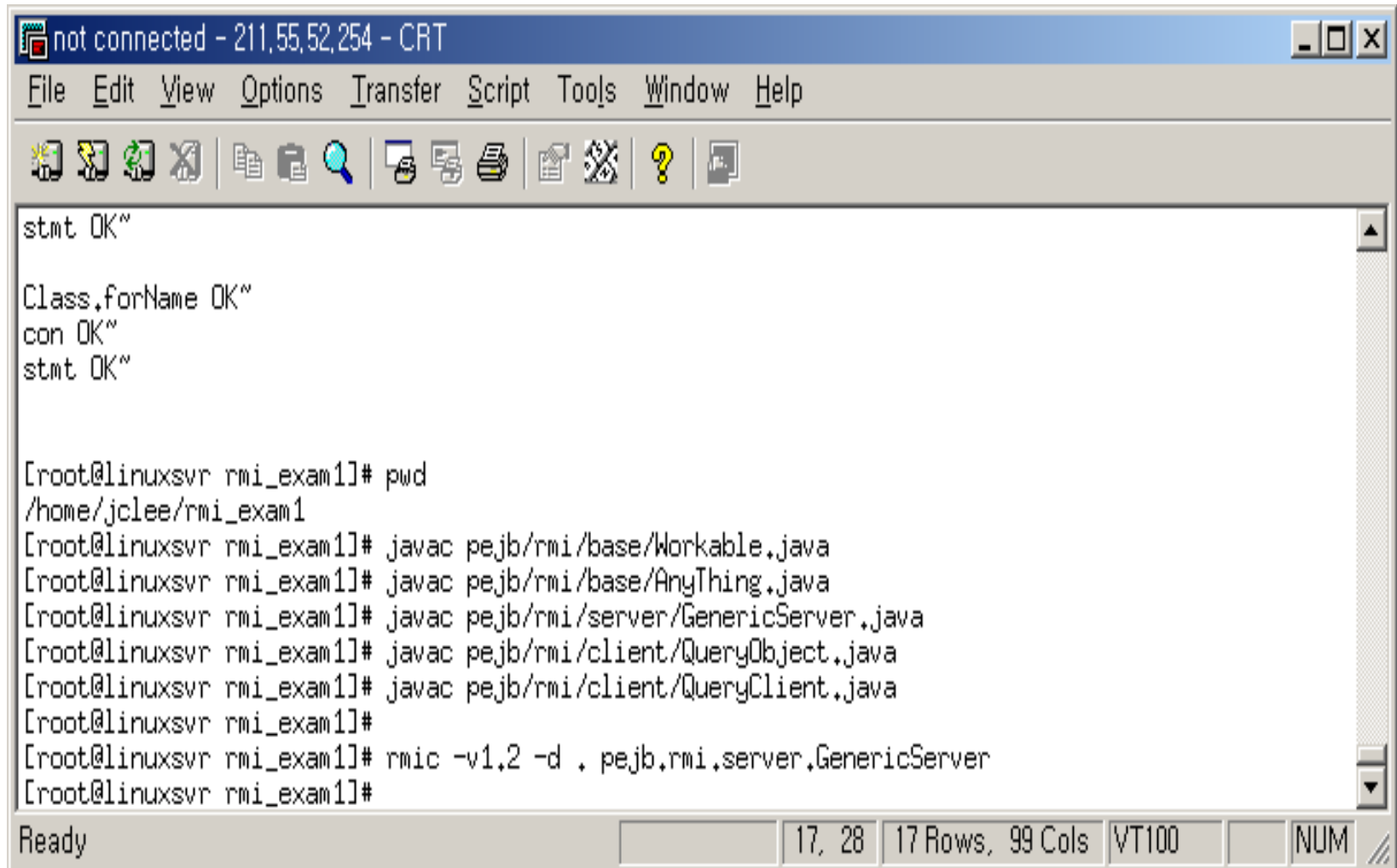
RMI 클라이언트

```
finally {  
    try {  
        if (con != null) con.close();  
    }  
    catch(Exception e) {}  
}  
return arrayList;  
}  
}
```

컴파일 후 stub 생성(윈도우)

```
C:\WINNT\System32\cmd.exe
E:\강의자료\여성부\자바네트웍\예제\rmi_exam1>
E:\강의자료\여성부\자바네트웍\예제\rmi_exam1>
E:\강의자료\여성부\자바네트웍\예제\rmi_exam1>javac pejb/rmi/base/Anything.java
E:\강의자료\여성부\자바네트웍\예제\rmi_exam1>javac pejb/rmi/base/Workable.java
E:\강의자료\여성부\자바네트웍\예제\rmi_exam1>javac pejb/rmi/server/GenericServer
.java
E:\강의자료\여성부\자바네트웍\예제\rmi_exam1>javac pejb/rmi/client/QueryObject.j
ava
E:\강의자료\여성부\자바네트웍\예제\rmi_exam1>javac pejb/rmi/client/QueryClient.j
ava
E:\강의자료\여성부\자바네트웍\예제\rmi_exam1>rmic pejb.rmi.server.GenericServer
E:\강의자료\여성부\자바네트웍\예제\rmi_exam1>
```

컴파일 후 stub 생성(리눅스)



The screenshot shows a terminal window titled "not connected - 211,55,52,254 - CRT". The window has a menu bar with "File", "Edit", "View", "Options", "Transfer", "Script", "Tools", "Window", and "Help". Below the menu bar is a toolbar with various icons. The terminal content shows the following commands and output:

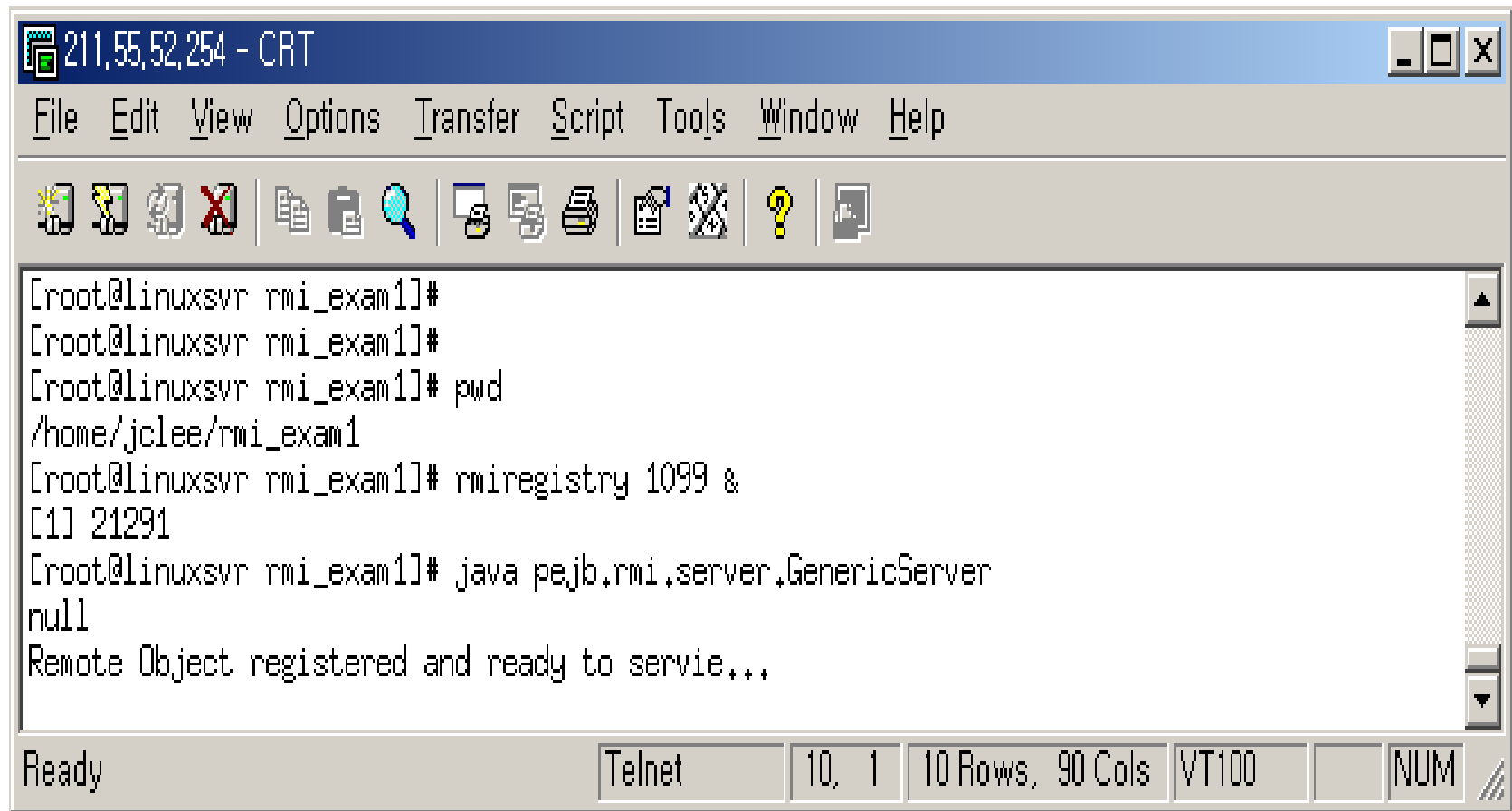
```
stmt OK~  
  
Class.forName OK~  
con OK~  
stmt OK~  
  
[root@linuxsvr rmi_exam1]# pwd  
/home/jclee/rmi_exam1  
[root@linuxsvr rmi_exam1]# javac pejb/rmi/base/Workable.java  
[root@linuxsvr rmi_exam1]# javac pejb/rmi/base/Anything.java  
[root@linuxsvr rmi_exam1]# javac pejb/rmi/server/GenericServer.java  
[root@linuxsvr rmi_exam1]# javac pejb/rmi/client/QueryObject.java  
[root@linuxsvr rmi_exam1]# javac pejb/rmi/client/QueryClient.java  
[root@linuxsvr rmi_exam1]#  
[root@linuxsvr rmi_exam1]# rmic -v1.2 -d . pejb,rmi,server,GenericServer  
[root@linuxsvr rmi_exam1]#
```

The status bar at the bottom of the terminal window displays "Ready", a cursor icon, "17, 28", "17 Rows, 99 Cols", "VT100", and "NUM".

JDBC Driver 설치

- Oracle 다운로드 사이트
(http://technet.oracle.com/software/tech/java/sqlj_jdbc/content.html)에서 JDBC Driver를 다운 받은 후 로컬 PC에 복사한다.
- JDBC Driver를 포함해서 절대경로를 클래스패스(classpath) 환경 변수에 추가 한다.

서버 실행

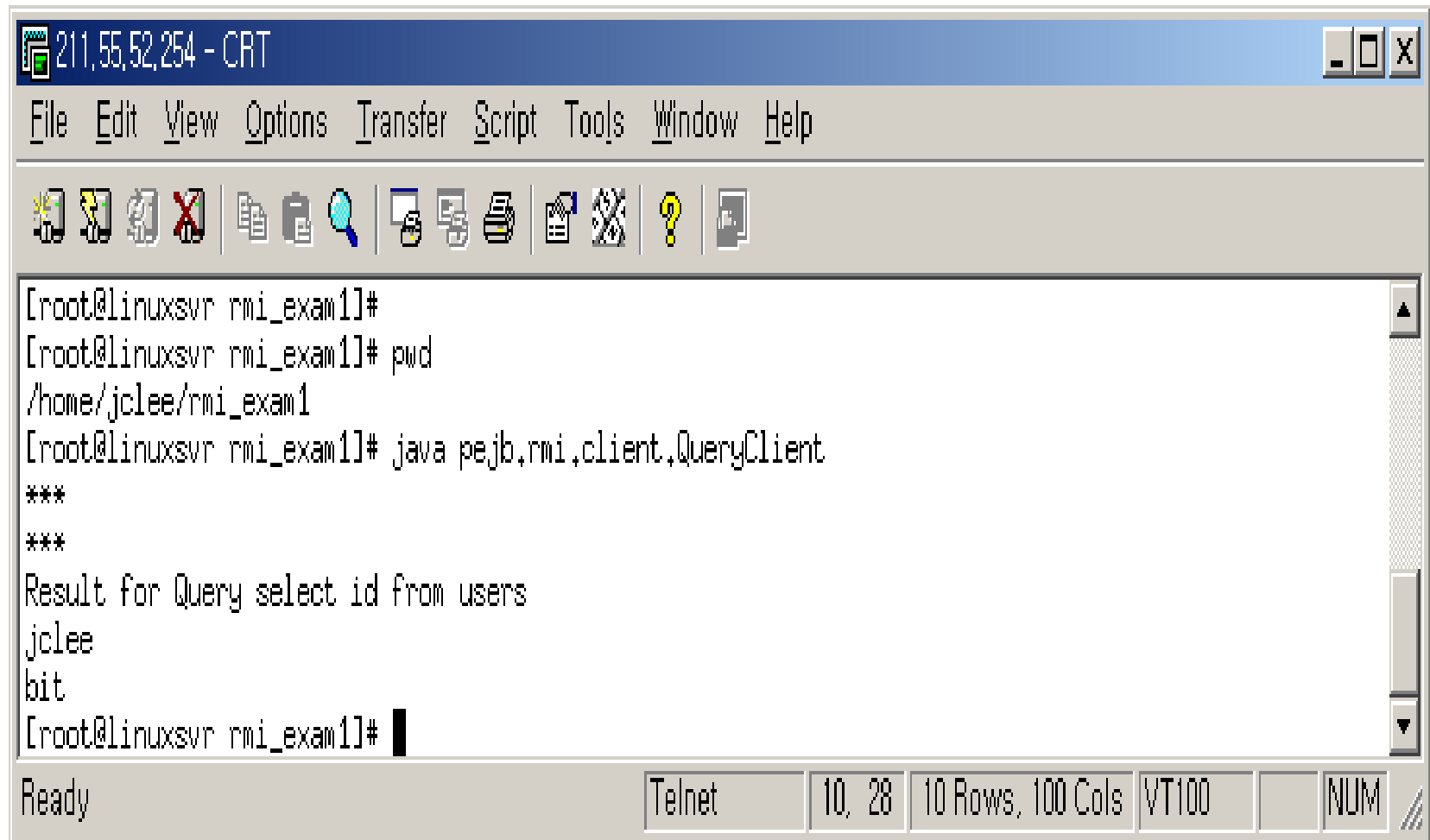


```
211.55.52.254 - CRT
File Edit View Options Transfer Script Tools Window Help

[root@linuxsvr rmi_exam1]#
[root@linuxsvr rmi_exam1]#
[root@linuxsvr rmi_exam1]# pwd
/home/jclee/rmi_exam1
[root@linuxsvr rmi_exam1]# rmiregistry 1099 &
[1] 21291
[root@linuxsvr rmi_exam1]# java pejb.rmi.server.GenericServer
null
Remote Object registered and ready to servie...
```

Ready Telnet 10, 1 10 Rows, 90 Cols VT100 NUM

클라이언트 실행



```
211.55.52.254 - CRT
File Edit View Options Transfer Script Tools Window Help
[Icons]

[root@linuxsvr rmi_exam1]#
[root@linuxsvr rmi_exam1]# pwd
/home/jclee/rmi_exam1
[root@linuxsvr rmi_exam1]# java pejb,rmi,client,QueryClient
***
***
Result for Query select id from users
jclee
bit
[root@linuxsvr rmi_exam1]#

Ready Telnet 10, 28 10 Rows, 100 Cols VT100 NUM
```

클라이언트 서버간 Callback

참조 전달

- 자바 RMI에서 원격 메소드의 파라미터나 반환 데이터는 `java.io.Serializable` 또는 `java.rmi.Remote` 객체 둘 중의 하나 이다.
- `Serializable` 객체가 사용되는 경우는 `pass-by-value` 의 경우이고 `Remote` 객체가 사용되는 경우는 참조(원격 레퍼런스)가 전달 된다.
- 자바 RMI 에서 의미하는 참조의 의미는 원격 객체에 대한 stub 레퍼런스이며 이를 원격 레퍼런스, 원격 객체에 대한 참조 등으로 표현 한다.

콜백(CallBack)

- 일반적인 클라이언트/서버 환경에서는 클라이언트는 단순히 서비스를 요청하며 서버는 요청된 서비스를 제공하는 기능을 정의 한다.
- 서버가 클라이언트의 요청에 근거하여 서버가 클라이언트에게 어떤 반응을 보이게 하는 것은 불가능 하다. (서버는 클라이언트에게 자신이 원하는 반응을 보이도록 할 방법이 존재하지 않는다.)
- **Callback**은 바로 클라이언트의 서비스 요청에 근거하여 서버측에서 의도적으로 클라이언트에게 원하는 반응을 보일 수 있도록 하는 메커니즘 이다.

콜백(CallBack)

- RMI 콜백은 RMI 클라이언트 서버 간 상호 원격 메소드 호출이 가능하도록 하는 메커니즘 이다.
- 단순히 상호간 원격 메소드 호출이 가능하다고 해서 Callback이라고 의미 하지는 않는다. 상대방에서 메소드 호출 요청을 받으면 메소드 호출을 받는 쪽에서 상대방의 메소드를 다시 호출하는 것을 말한다.

참조전달을 통한 클라이언트 서버간 콜백 구현

RMI Callback Chatting Program

RMI Callback 구현

- 서로다른 JVM 환경에서는 클라이언트 자신에 대한 로컬의 레퍼런스(메모리 주소)는 원격의 다른 메모리 공간에서는 의미가 없다.
- RMI Callback 구현시에는 자신에 대한 로컬 레퍼런스 대신에 자신에 대한 참조를 상대방에 전달한다.
- RMI 에서 참조는 원격 객체에 대한 stub 레퍼런스이다. 즉 RMI 클라이언트는 원격 객체에 대한 참조를 통하여 원격의 JVM에 존재하는 원격 메소드를 호출 할 수 있었던 것이다.

RMI Callback 구현

- RMI에서는 원격의 JVM에 존재하는 객체의 메소드를 호출하려면 해당 객체가 원격 객체로 정의되어야 하며 원격 객체에는 원격 메소드가 정의되어야 한다.
- Callback이 클라이언트 서버가 공히 상대방의 메소드를 호출 한다고 하면 RMI Callback의 구현을 위해서 클라이언트와 서버가 원격 객체가 되어야 하며 동시에 원격 메소드를 정의해야 한다는 것은 당연한 것이다.
- RMI Callback을 이용하면 Socket을 이용한 것 보다 효율적이면서 쉽게 채팅 프로그램을 제작 할 수 있다.

서버 측

- 원격 인터페이스를 정의 한다. (Server.java)
 - 클라이언트가 호출 할 원격 메소드를 정의 한다.
 - 이때 원격 메소드의 파라미터중 하나는 Callback 으로 호출 할 클라이언트의 원격 메소드를 정의하는 원격 인터페이스 타입으로 선언 한다.
- 원격 인터페이스를 구현하고 원격 객체의 기능을 갖는 원격 객체 클래스를 정의 한다.
(ChatServer.java)
 - java.rmi.server.UnicastRemoteObject를 상속
- RMI 서버 Application을 제작(ChatServer.java)

클라이언트 측

- 서버측의 원격 객체가 콜백으로 호출 할 원격 메소드를 정의하는 원격 인터페이스를 작성한다.
(Receiver.java)
- 원격 인터페이스를 구현 하고 원격 객체의 기능을 갖는 원격 객체 클래스를 제작(ChatClient.java)
- RMI 클라이언트 Application을 작성 한다.
 - 서버측의 원격 객체에 대한 원격 레퍼런스를 얻고 원격 메소드를 호출 한다. 이때 원격 서버가 다시 클라이언트 자신의 원격 메소드를 호출 할 수 있도록 클라이언트 자신의 레퍼런스를 호출시 사용하는 파라미터로 넘겨 준다. 이때 실제 로컬 클라이언트의 레퍼런스(주소)가 넘어 가지 않고 클라이언트 객체에 대한 stub 객체가 서버측에 참조로 전달된다.

서버 측(원격 인터페이스)

- **Server.java**에서는 RMI registry에 등록하기 위한 원격 객체의 이름을 상수로서 정의
- 클라이언트에서 호출 할 메소드를 정의, 콜백을 위해 클라이언트가 서버의 원격 메소드를 호출 하면 호출된 원격 메소드에서 다시 클라이언트의 원격 메소드를 호출 하는 구조이다.
- 원격 메소드의 파라미터나 리턴 데이터는 `java.io.Serializable`(복사본) 이나 `java.rmi.Remote`(참조) 객체 이어야 한다고 했다.

서버 측(원격 인터페이스)

- 클라이언트가 `addReceiver(...)`, `removeReceiver(...)` 메소드를 호출 할 때 넘겨주는 데이터는 복사되어 전달되지 않고 참조로 전달된다.
- `Broadcast(...)` 메소드가 실제로 클라이언트에서 서버로 호출된 뒤 클라이언트의 원격 메소드를 다시 호출 할 메소드로 사용된다. 물론 파라미터로 전달된 `String` 타입의 `msg`는 복사되어(`pass-by-value`) 전달된다.

서버 측(Server.java)

```
package pejb.rmi.chat.server;
import java.rmi.*;
import pejb.rmi.chat.client.Receiver;

public interface Server extends Remote {
    public String CHAT_SERVER = "chatserver";
    public void broadcast(String msg) throws RemoteException;
    public void addReceiver(Receiver client) throws
        RemoteException;
    public void removeReceiver(Receiver client) throws
        RemoteException;
}
```

서버 측(인터페이스 구현, RMI 서버)

- **ChatServer.java**에서는 멤버 변수로 allClients를 ArrayList로 선언하고 있는데 여기에 콜백을 위한 모든 클라이언트의 원격 레퍼런스를 참조로 저장한다.
- 또한 원격 메소드 broadcast(...)에서는 실제로 Callback 기능을 정의하고 있다. broadcast(...) 메소드에서는 모든 클라이언트의 원격 레퍼런스를 뽑아내고 이를 통해 Callback으로 모든 클라이언트의 원격 메소드인 printMsg(...)를 호출한다.

서버 측(ChatServer.java)

```
//ChatServer.java
package pejb.rmi.chat.server;
import java.rmi.*;
import java.rmi.server.*;
import java.rmi.registry.*;
import java.util.*;
import pejb.rmi.chat.client.*;
public class ChatServer extends UnicastRemoteObject implements
    Server {
    private ArrayList allClients = new ArrayList(100);
    //default 생성자 UnicastRemoteObject의 생성자가 예외를 던
    진다.
```

서버 측(ChatServer.java)

```
public ChatServer() throws RemoteException {  
    super();  
}  
//-----원격인터페이스 Server의 원격 메소드 구현  
public void addReceiver(Receiver client) throws  
RemoteException {  
    allClients.add(client);  
    System.out.println("Added a client...");  
}  
synchronized public void broadcast(String msg) throws  
RemoteException {  
    int count = allClients.size();
```


서버 측(ChatServer.java)

```
System.out.println("현재 접속중인 클라이언트 ::: " +  
count);  
Receiver client = null;  
for(int i=0; i<count; i++) {  
    client = (Receiver)allClients.get(i);  
    client.printMsg(msg);  
}  
}  
  
synchronized public void removeReceiver(Receiver client)  
throws RemoteException {  
    allClients.remove(allClients.indexOf(client));  
}
```

서버 측(ChatServer.java)

```
//----- RMI 서버 Application
public static void main(String[] args) {
    try{
        Server server = new ChatServer();
        Naming.rebind("rmi://127.0.0.1:2000/ChatServer", server);
        System.out.println("Remote Object [" +
            Server.CHAT_SERVER + "] bound... on port 2000");
    }
    catch (Exception e)
    {
        System.err.println("RMI 서버 생성시 오류 발생 ::: " + e);
        System.exit(1);
    }
}
```

클라이언트 측(원격 인터페이스)

- **Receiver.java**에서는 원격 서버의 원격 객체가 Callback으로 호출 한 메소드 `printMsg(String msg)`를 정의 하고 있다.
- 파라미터 `msg`는 채팅시 주고 받는 메시지 이다.

클라이언트 측(원격 인터페이스)

```
package pejb.rmi.chat.client;

import java.rmi.*;

public interface Receiver extends Remote {

    public void printMsg(String msg) throws
        RemoteException;

}
```

클라이언트 측(인터페이스 구현, RMI 클라이언트 기능)

- **ChatClient.java**에서는 원격 인터페이스 Receiver를 구현 하고 있다. printMsg 메소드를 구현한다는 의미이다. 이렇게 함으로서 원격 서버에서 Callback을 통해 클라이언트의 printMsg를 원격 메소드로서 호출 할 수 있게 된다. (물론 이곳만 가지고는 원격 객체의 구실을 할 수 없다... 그래서 UnicastRemoteObject는 당연히 상속해야 한다.)
- 그런데 ChatClient.java에서는 javax.swing.JApplet을 상속 했기 때문에 불가능하다. 그럼, 어떻게???

클라이언트 측(인터페이스 구현, RMI 클라이언트 기능)

- ChatClient는 원격 객체의 기능을 갖기 위해 `java.rmi.server.UnicastRemoteObject`를 상속하는 대신 생성자 내에서 아래와 같이 `UnicastRemoteObject`의 static 메소드인 `exportObject`를 호출 하도록 했다.

```
public ChatClient() throws RemoteException {  
    UnicastRemoteObejct.exportObject(this);  
}
```

- `exportObject` 메소드는 파라미터로 전달된 Remote 객체를 RMI 시스템에 원격 객체로 알리는 기능을 수행한다.

클라이언트 측(인터페이스 구현, RMI 클라이언트 기능)

- `printMsg(...)` 메소드가 하는 일은 단지 서버가 복사본으로 넘겨 주는 채팅 문자열을 받아 채팅창(`jts_chatDisplat`)에 뿌리는 기능을 수행한다.
- `ChatClient`에서 원격 메소드 호출과 관련된 메소드는 `printMsg()`, `openConnection()`, `closeConnection()`, `send()` 등이 있다. 그외의 메소드는 화면 UI 조작과 관련된 메소드 등이다.
- `openConnection` 메소드는 서버측의 원격 객체에 대한 원격 레퍼런스를 얻고 이를 통해 클라이언트 자신을 `addReceiver(this)`를 통해 서버의 `ArrayList`에 증록한다. 파라미터 `this`는 `Receiver` 타입임을 기억하자.

클라이언트 측(인터페이스 구현, RMI 클라이언트 기능)

- `this` 자체가 원격 서버로 복사되어 넘어간다고 생각하면 오산이다. 이 `this`는 `java.rmi.Remote`를 상속한 객체 이므로 stub 객체(`ChatClient_Stub`)의 참조가 `ChatServer`의 원격메소드 `addReceiver`로 전송된다.
- 이렇게 해서 `ChatServer`에서는 원격의 클라이언트에 대한 참조를 `ArrayList`인 `allClients` 객체 참조 변수에 저장한다.

클라이언트 측(인터페이스 구현, RMI 클라이언트 기능)

- send() 메소드에서는 ChatServer에 대한 원격 레퍼런스를 통해 원격메소드인 broadcast(...)를 호출 한다. 이렇게 함으로서 ChatServer의 ArrayList에 저장된 모든 클라이언트의 참조를 통해 모든 채팅 참여자에게 Callback 으로 printMsg를 다시 호출 하는 것이다.

클라이언트 측(ChatClient.java)

```
package pejb.rmi.chat.client;

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import java.rmi.*;
import java.rmi.server.*;
import pejb.rmi.chat.server.Server;

public class ChatClient extends JApplet implements Receiver,
    ActionListener {
    private String login_name="";
    private boolean isApplet, loggedIn;
```

클라이언트 측(ChatClient.java)

```
private static final int LOGIN=1, LOGOUT=2, CLEAR=3, OK=4,
CANCEL=5;

private JButton jb_logout, jb_login, jb_clear, jb_ok, jb_cancel;
private JTextArea jta_chatDisplay;
private JTextField jtf_chat_string, jtf_avatar_name;
private JDialog jd_chat_name;
private JScrollPane scrollPane;
private Server server = null;

public ChatClient() throws RemoteException {
    UnicastRemoteObject.exportObject(this);
}
```

클라이언트 측(ChatClient.java)

```
public void init() {  
    isApplet = true;  
    initUI();  
    addListeners();  
}
```

//채팅 종료시

```
public void destroy() {  
    try {  
        if (server != null) server.removeReceiver(this);  
    }  
    catch(RemoteException e) {}  
}
```

클라이언트 측(ChatClient.java)

```
private void createComponents() {  
    jb_logout= new JButton("Log Out");  
    jb_login = new JButton("Log In");  
    jb_clear = new JButton("Clear Messages");  
    jb_ok = new JButton("OK");  
    jb_cancel = new JButton("Cancel");  
    jtf_avatar_name = new JTextField(10);  
    jtf_chat_string = new JTextField();  
    jd_chat_name= new JDialog();  
    jta_chatDisplay = new JTextArea();  
}
```

클라이언트 측(ChatClient.java)

```
private void initUI() {  
    createComponents();  
    jb_logout.setActionCommand(LOGOUT + "");  
    jb_login.setActionCommand(LOGIN + "");  
    jb_clear.setActionCommand(CLEAR + "");  
    jb_ok.setActionCommand(OK + "");  
    jb_cancel.setActionCommand(CANCEL + "");  
    jb_login.setToolTipText("login with new chat name");  
    jb_logout.setToolTipText("logout chatRoom");  
    jb_clear.setToolTipText("clear chat messages since started...");  
}
```

클라이언트 측(ChatClient.java)

```
jd_chat_name.setTitle("Input Avatar Name");
JPanel dummy = new JPanel ();
dummy.add(jb_ok);
dummy.add(jb_cancel);
JPanel p = new JPanel ();

p.add(new Label("Avatar Name: "));
p.add(jtf_avatar_name);
jd_chat_name.getContentPane().add(p, BorderLayout.NORTH);
jd_chat_name.getContentPane().add(dummy,
BorderLayout.SOUTH);
jd_chat_name.pack();
```

클라이언트 측(ChatClient.java)

```
jta_chatDisplay.setBackground(Color.black);
jta_chatDisplay.setForeground(Color.green);
jta_chatDisplay.append("Welcome ! RMI Callback Based
ChatClient....\n");
jta_chatDisplay.append("Click 'Log In' to enter chat,'Log Out' to
Exit Chat Room.\n");
jta_chatDisplay.setEditable(false);
jb_logout.setEnabled(false);

JPanel tmpPanel = new JPanel();
tmpPanel.add(jb_login);
tmpPanel.add(jb_logout);
tmpPanel.add(jb_clear);
scrollPane = new JScrollPane(jta_chatDisplay);
```


클라이언트 측(ChatClient.java)

```
Container contentPane = getContentPane();
contentPane.setLayout(new BorderLayout());
    contentPane.add(tmpPanel , BorderLayout.NORTH);
    contentPane.add(scrollPane, BorderLayout.CENTER);
    contentPane.add(jtf_chat_string, BorderLayout.SOUTH);
}
private void addListeners() {
    ActionListener tf_listener = new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            if(!loggedIn) {
                jtf_chat_string.setText("Please Login
First...");
                return;
            }
        }
    }
}
```

클라이언트 측(ChatClient.java)

```
        try {
            send(jtf_chat_string.getText());
            jtf_chat_string.setText("");
        } catch (RemoteException ex) {
            ex.printStackTrace();
        }
    }

};

jb_logout.addActionListener(this);
jb_login.addActionListener(this);
jb_clear.addActionListener(this);
jb_ok.addActionListener(this);
jb_cancel.addActionListener(this);
jtf_chat_string.addActionListener(tf_listener);
}
```

클라이언트 측(ChatClient.java)

```
private void openConnection() throws RemoteException {  
    if(server == null) {  
        try {  
            server =  
(Server)Naming.lookup("rmi://211.55.52.254:2000/ChatServer");  
            System.out.println("Looked Up Remote  
Object...");  
            server.addReceiver(this);  
            send(login_name + " entered ...\n");  
            loggedIn = true;  
        } catch( Exception e) {  
            throw new RemoteException("Couldn't get a  
valid remote refrence ...["+ e + "]");  
        }  
    }  
}
```

클라이언트 측(ChatClient.java)

```
jb_login.setEnabled(false);
    jb_logout.setEnabled(true);
    jtf_chat_string.requestFocus();
    jtf_chat_string.setText("");
}
private void closeConnection() throws RemoteException {
    server.broadcast(" " + login_name + " 님이 퇴실 했습니다...");

    server.removeReceiver(this);
    loggedIn = false;
    server = null;
    jb_logout.setEnabled(false);
    jb_login.setEnabled(true);
}
```

클라이언트 측(ChatClient.java)

```
private void go() {  
    JFrame jf = new JFrame ();  
    jf.getContentPane().add(this);  
    init();  
    isApplet = false;  
    jf.setSize(350, 400);  
    jf.setVisible(true);  
    jf.addWindowListener(new WindowCloser());  
}  
  
private void send(String msg) throws RemoteException {  
    server.broadcast "[" + login_name + "]" + msg);  
    jtf_chat_string.setText("");  
}
```

클라이언트 측(ChatClient.java)

```
public void printMsg(String msg) throws RemoteException {  
    jta_chatDisplay.append(msg+ "\n");  
    JScrollBar vScroll = null;  
    vScroll = scrollPane.getVerticalScrollBar();  
    vScroll.setValue(vScroll.getMaximum());  
}
```

```
public static void main(String [] args) {  
    try {  
        new ChatClient().go();  
    } catch(RemoteException ex) {  
        ex.printStackTrace();  
    }  
}
```

클라이언트 측(ChatClient.java)

//Inner Class Definition

```
class WindowCloser extends WindowAdapter {  
    public void windowClosing(WindowEvent we) {  
        if(server != null) {  
            try {  
                closeConnection();  
            } catch (RemoteException e) {  
                e.printStackTrace();  
            }  
        }  
        System.exit(0);  
    }  
}
```

클라이언트 측(ChatClient.java)

// ActionListener event method...

```
public void actionPerformed(ActionEvent ae) {  
    int buttonId = Integer.parseInt(ae.getActionCommand());  
    switch(buttonId) {  
        case LOGOUT:  
            try {        closeConnection();  
            } catch (RemoteException e) {  
                e.printStackTrace();  
            }  
            break;  
        case LOGIN:  
  
            jd_chat_name.setLocation(ChatClient.this.getLocationOnScreen(  
            ).x + 50,
```


클라이언트 측(ChatClient.java)

```
        ChatClient.this.getLocationOnScreen().y + 50);
        jd_chat_name.setVisible(true);
        break;
    case OK:
        login_name = jtf_avatar_name.getText();
        try {
            openConnection();
            jta_chatDisplay.setText("Your Avatar
Name Is " + login_name + ".\n");
            jd_chat_name.setVisible(false);
        } catch (RemoteException e) {
            e.printStackTrace();
        }
    }
```

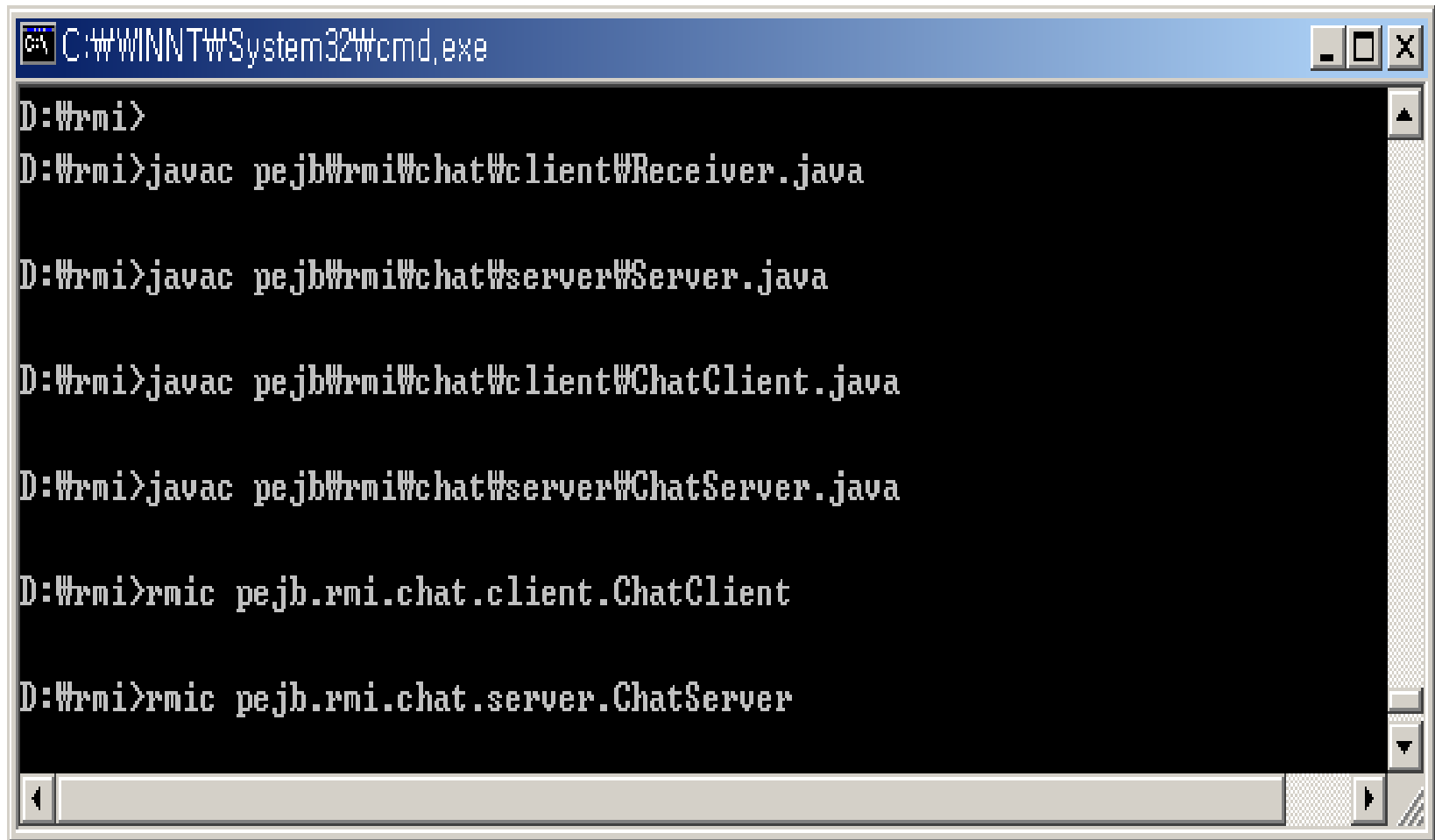
클라이언트 측(ChatClient.java)

```
        break;
    case CANCEL:
        jd_chat_name.setVisible(false);
        break;
    case CLEAR:
        jta_chatDisplay.setText("Cleard the diplay
area.\n");
        break;
    }
}
}
```

적절한 경로에 소스 저장

- 패키지별로 소스를 저장한다.
 - 현재 디렉토리가 d:\rmi라고 한다면... 그아래에 pejb\rmi\chat\server, pejb\rmi\chat\client 디렉토리를 만든후 소스들을 적절한 위치에 복사한다. 예를 들어 Server.java인 경우 package pejb.rmi.chat.server 이므로 d:\rmi\pejb\rmi\chat\server에 저장한다.

소스파일 컴파일 및 stub 생성(윈도우)



```
C:\WINNT\System32\cmd.exe

D:\rmi>
D:\rmi>javac pejb\rmi\chat\client\Receiver.java

D:\rmi>javac pejb\rmi\chat\server\Server.java

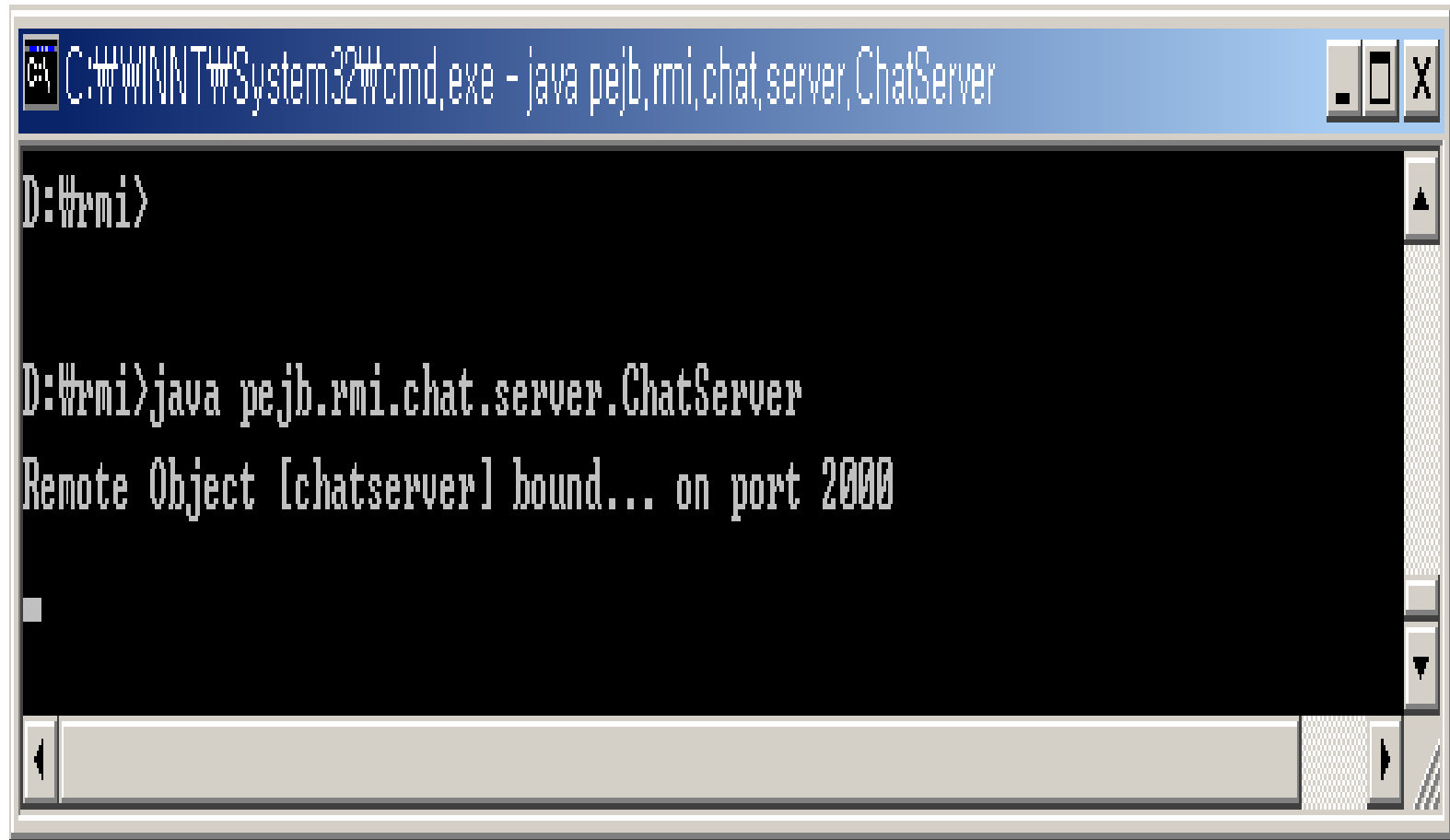
D:\rmi>javac pejb\rmi\chat\client\ChatClient.java

D:\rmi>javac pejb\rmi\chat\server\ChatServer.java

D:\rmi>rmic pejb.rmi.chat.client.ChatClient

D:\rmi>rmic pejb.rmi.chat.server.ChatServer
```

채팅 서버 실행(윈도우)



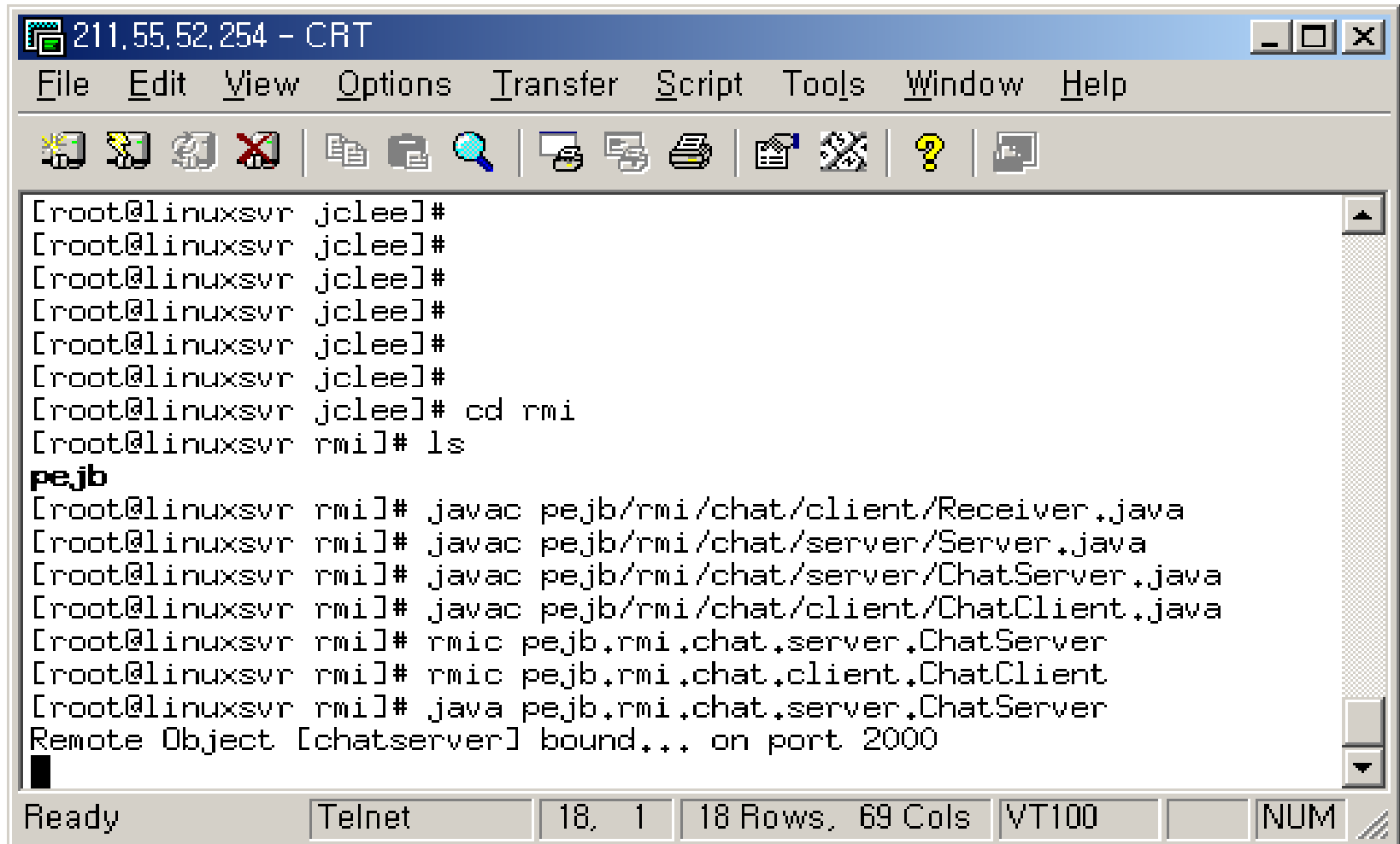
A screenshot of a Windows command prompt window. The title bar at the top is blue and contains the text "C:\WINNT\System32\cmd.exe - java pejb.rmi.chat.server.ChatServer". The main area of the window is black with white text. The prompt "D:\rmi>" is visible. The command "java pejb.rmi.chat.server.ChatServer" has been entered and executed. The output is "Remote Object [chatserver] bound... on port 2000". The window has standard Windows controls (minimize, maximize, close) in the top right corner and a scrollbar on the right side.

```
C:\WINNT\System32\cmd.exe - java pejb.rmi.chat.server.ChatServer

D:\rmi>

D:\rmi>java pejb.rmi.chat.server.ChatServer
Remote Object [chatserver] bound... on port 2000
```

컴파일 & stub 생성 & 서버 실행 (리눅스)

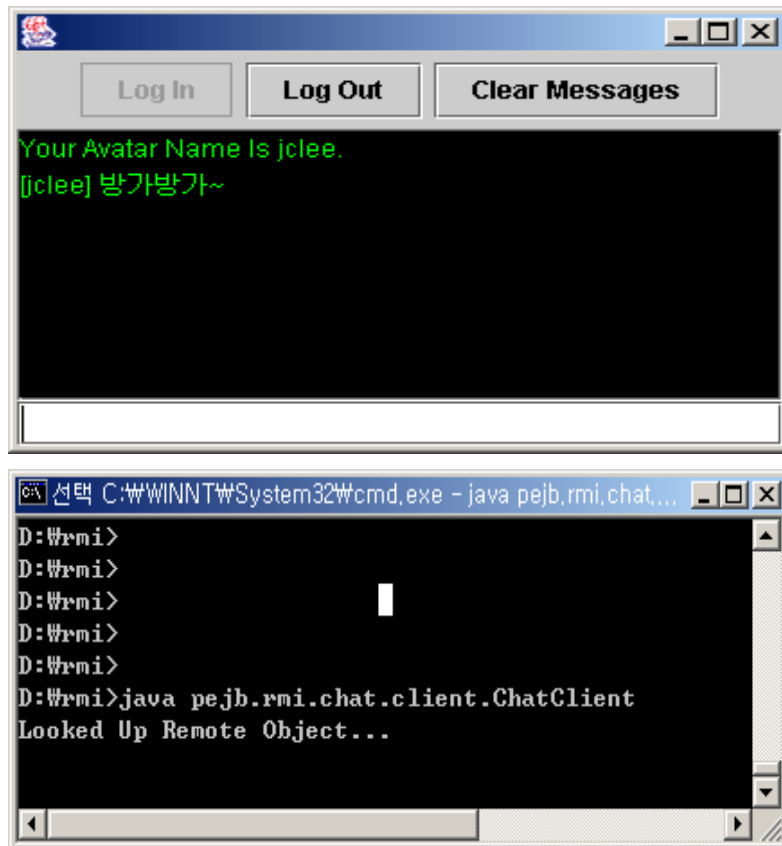


The screenshot shows a Telnet window titled "211.55.52.254 - CRT". The menu bar includes File, Edit, View, Options, Transfer, Script, Tools, Window, and Help. The terminal text is as follows:

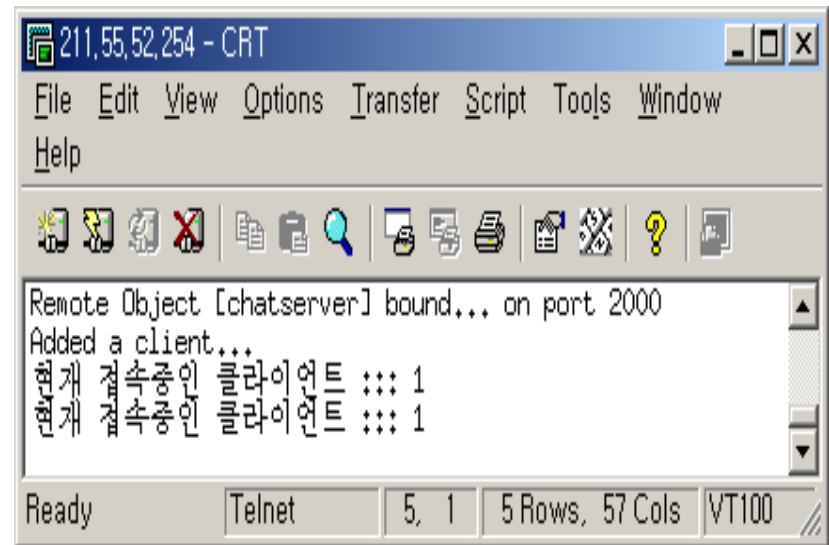
```
[root@linuxsvr jclee]#  
[root@linuxsvr jclee]#  
[root@linuxsvr jclee]#  
[root@linuxsvr jclee]#  
[root@linuxsvr jclee]#  
[root@linuxsvr jclee]#  
[root@linuxsvr jclee]#  
[root@linuxsvr jclee]# cd rmi  
[root@linuxsvr rmi]# ls  
pejb  
[root@linuxsvr rmi]# javac pejb/rmi/chat/client/Receiver.java  
[root@linuxsvr rmi]# javac pejb/rmi/chat/server/Server.java  
[root@linuxsvr rmi]# javac pejb/rmi/chat/server/ChatServer.java  
[root@linuxsvr rmi]# javac pejb/rmi/chat/client/ChatClient.java  
[root@linuxsvr rmi]# rmic pejb.rmi.chat.server.ChatServer  
[root@linuxsvr rmi]# rmic pejb.rmi.chat.client.ChatClient  
[root@linuxsvr rmi]# java pejb.rmi.chat.server.ChatServer  
Remote Object [chatserver] bound... on port 2000  
█
```

The status bar at the bottom shows "Ready", "Telnet", "18, 1", "18 Rows, 69 Cols", "VT100", and "NUM".

채팅 클라이언트 실행



클라이언트



서버

Linux 에서 RMI 서버 운영시 Tip

- **Connection refused to host: 127.0.0.1** The error
java.rmi.ConnectException: Connection refused
to host: 127.0.0.1; nested exception is:
java.net.ConnectException: Connection refused
has something to do with the /etc/hosts file. On one
Redhat6.1 machine there was a /etc/hosts file that
looked like
- 127.0.0.1 localhost.localdomain localhost
[Qserver] xx.xx.xx.xx Qserver [localhost]
where [] means optional. When "Qserver" is present on
the first line and the rmi server is running on this host,
the client gets the `rmi connection' refused error.

RMI 동적 클래스 로딩 (Dynamic Class Loading)

동적 클래스 다운로드

- 다른 실행 환경(JVM)에 존재하는 클래스를 동적으로 로컬의 JVM에 적재하는 기능을 “동적 클래스 로딩(Dynamic Class Loading)이라 한다.
- 클라이언트의 비즈니스 로직이 변경 되었더라도 서버의 로직에 영향을 주지 않는 Application의 개발이 가능하다.
- RMI 클라이언트가 자신의 비즈니스 로직이 담긴 클래스 파일을 서버에 넘겨주고 이를 서버에서 요청 시 처리 한다면 가능 하다.

동적 클래스 다운로드

- 일반적으로 클라이언트 측에서는 자신이 요구 할 비즈니스 로직이 담긴 클래스 파일을 넘겨주며, 서버측에서는 원격 객체 클래스에 대한 스텝 클래스 파일을 주로 클라이언트로 로딩 되도록 한다.
- RMI에서는 동적으로 클래스를 다운 받을 위치를 “java.rmi.server.codebase” 라는 시스템 프로퍼티를 통해 참조 한다.
- 만약 클라이언트에서 서버측으로 클래스 파일이 동적으로 다운되길 원하는 경우라면 클라이언트 JVM 환경에서 “java.rmi.server.codebase” System Property를 설정해야 한다.

동적 클래스 다운로드

- JVM에 java.rmi.server.codebase System property 설정 방법

- **java 실행 명령어에 “-D” 옵션 사용**

java -Djava.rmi.server.codebase

=http://www.oraclejava.co.kr/codebase/ GenericServer (jar 파일이 아닌경우 마지막에 '/')

- **실행 코드 내에서 직접 시스템 프로퍼티 지정**

```
Properties props = System.getProperties();  
props.put("java.rmi.server.codebase",  
"http://www.oraclejava.co.kr/codebase/");  
//( jar 파일이 아닌경우 마지막에 '/')
```

RMI와 보안관리자

- 기본적으로 RMI는 검증되지 않은 클래스 파일이 다운로드 되어 로컬의 JVM에서 실행되는 것을 허락하지 않는다.
- 동적으로 다운 로드 되어 로컬의 JVM에 로드 되는 코드는 반드시 보안관리자 (`java.rmi.RMISecurityManager`)의 검증을 거쳐야 한다.
- Application에서 보안관리자로 `RMISecurityManager`를 등록하는 방법
 - `System.setSecurityManager(new RMISecurityManager());`

RMI와 보안관리자

- 보안관리자를 설치하면 해당 보안관리자로 부터 접근 가능한 자원과 불가능한 자원에 관한 설정을 해야 한다.
- Java2 플랫폼에 맞는 보안정책파일(Policy)을 작성하고 이를 보안관리자의 보안 정책으로 지정 해야 한다. 등록된 보안관리자에게 보안 정책 파일을 지정하기 위해서는 “java.security.policy” 라는 시스템 프로퍼티에 지정 한다.

```
Properties props = System.getProperties();  
props.put(“jva.security.policy”, “permit.policy”);  
System.setProperties(props);
```

Example

Dynamic code downloading using RMI

개 요

- 이전의 예제인 `GenericServer`와 `QueryClient`를 실행하는 경우 클라이언트에서 서버로 `QueryObject`가 직렬화 되어 복사 되어 가는 경우 서버측에서는 클라이언트에서 전달 받는 객체의 실제의 타입정보 즉, `QueryObject.class`를 가지고 있어야 했다.
- 이번에 할 예제를 통해 우리는 서버측에는 `QueryObject`와 관련된 파일을 가지고 있지 않고 실행시에 동적으로 클래스를 다운 받을 수 있는 구성을 익히며 또한 클라이언트에서도 `GenericServer`와 관련된 일체의 것을 가지고 있지 않으며 실행시 `GenericServer_Stub.class`를 다운 받는 `Application`을 작성 할 것이다.(만약 `QueryObject`를 각각의 클라이언트가 다르게 바꾸어서 서버로 보낸다면 어떻게 될 것 인가를 생각해 보자.)

파일 구성

- 서버 측
 - pejb.rmi.server : GenericServer.java
 - pejb.rmi.base : Anything.java, Workable.java
- 클라이언트 측
 - pejb.rmi.client : QueryObject.java, ClientWrapper.java, QueryClient.java
 - pejb.rmi.base : Anything.java, Workable.java

서버 측 파일 소스

- 서버 측의 소스 파일은 기존의 것을 그대로 사용토록 한다. (GenericServer.java 파일에 몇 줄만 추가 하자. 동적으로 파일을 다운 받기 위해서는 보안관리자가 필요하다.)

GenericServer.java에서 수정 할 곳

```
try {
```

```
Properties props = System.getProperties();
```

```
props.put("java.security.policy", "permit");
```

//서버가 클라이언트의 프로그램을 실행시 클래스를 찾다가 못찾으면 아래
http://www.oraclejava.co.kr/fromClient_toServer/client.jar에 가보라는 의미, 즉 이 프
로그램에서는 QueryObject. ClientWrapper등이 있는 곳을 서버에 알려 주는 것이다.

```
props.put("java.rmi.server.codebase", "http://www.oraclejava.co.kr/fro  
mClient_toServer/client.jar");
```

```
System.setProperties(props);
```

```
System.setSecurityManager(new RMISecurityManager()); //보안관리자
```

```
Workable workable =
```

```
(Workable)Naming.lookup("rmi://211.55.52.254:2020/Workable");
```

```
ClientWrapper wrapper = new ClientWrapper(sql);
```

```
System.out.println("Result for Query " +sql ); .....
```

```
}
```

클라이언트 측 파일 소스

- QueryObject.java, QueryClient.java, ClientWrapper.java 파일에서 필요한 부분만 수정하도록 하자.

QueryObject 에서 수정 할 곳

```
public class QueryObject implements Serializable{
```

```
    private String sql;
```

```
    private Connection con;
```

```
    private ArrayList arrayList;
```

```
    public QueryObject(String sql) {
```

```
        this.sql = sql;
```

```
    }
```

```
    public Serializable query() {
```

```
        try {
```

```
            Class.forName("oracle.jdbc.driver.OracleDriver");
```

```
            .....
```

ClientWrapper.java

```
package pejb.rmi.client;
```

```
import pejb.rmi.base.AnyThing;
```

```
import java.io.Serializable;
```

```
public class ClientWrapper extends QueryObject implements AnyThing{  
    public ClientWrapper(String sql) {  
        super(sql);  
    }  
    public Serializable doThing() {  
        return query();  
    }  
}
```

ClientClient.java 에서 수정 할 곳

```
package pejb.rmi.client;

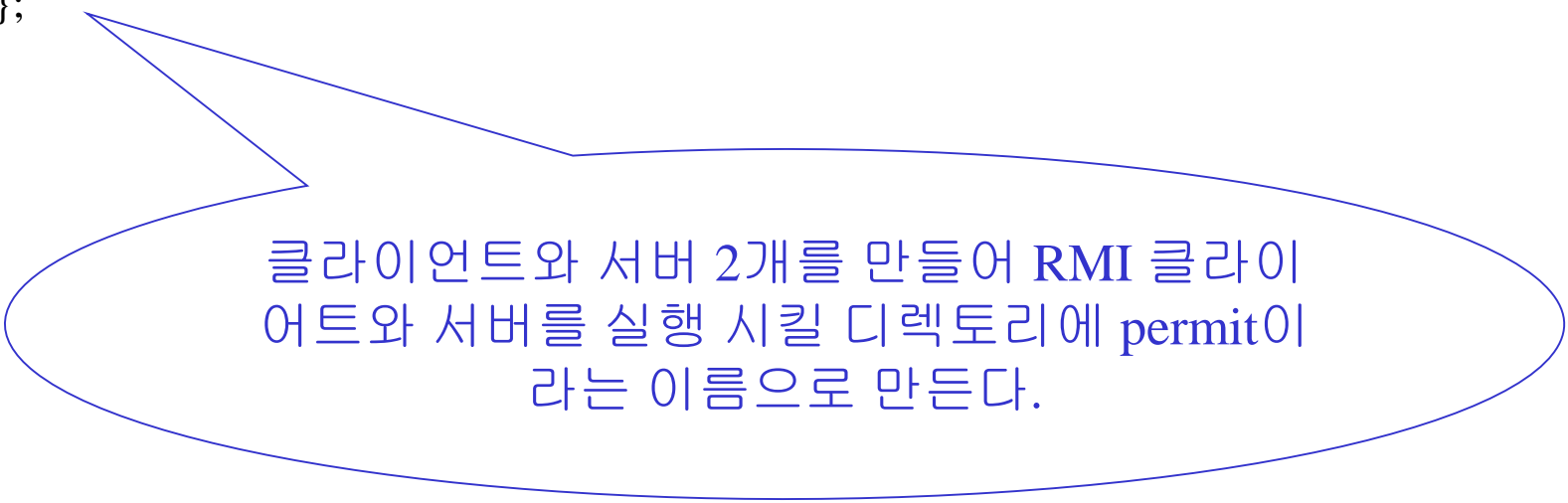
.....

import java.util.Properties;
import java.rmi.RMISecurityManager;

public class QueryClient {
    public static void main(String[] args) {
        String sql="select id from users";
        try {
            Properties props = System.getProperties();
            props.put("java.security.policy", "permit");
            props.put("java.rmi.server.codebase","http://www.oraclejava.co.kr/fromClient_to
Server/client.jar");
            System.setProperties(props);
            System.setSecurityManager(new RMISecurityManager());
            Workable workable =
            (Workable)Naming.lookup("rmi://211.55.52.254:2020/Workable");
            ClientWrapper qryObj = new ClientWrapper(sql);
            //QueryObject qryObj = new QueryObject(sql);
            System.out.println("Result for Query " +sql );
        }
    }
}
```

permit(보안 설정 파일)

```
grant {  
    permission java.net.SocketPermission "*:1024-65535", "connect,accept";  
    permission java.net.SocketPermission "*:80", "connect,accept";  
};
```



클라이언트와 서버 2개를 만들어 RMI 클라이언트와 서버를 실행 시킬 디렉토리에 permit이라는 이름으로 만든다.

컴파일 및 실행

	클라이언트	서버
필요파일 (컴파일)	<p>공통 : Workable.java, Anything.java</p> <p>Client측:QueryClient.java, ClientWrapper.java, QueryObject.java</p>	<p>공통 : Workable.java, Anything.java</p> <p>Server측:GenericServer.java</p>
실행시 필요파일 (다운되는것)	GenericServer_Stub.class	<p>QueryObject.java</p> <p>ClientWrapper.java</p>
보안 관련 파일	permit(클라이언트 실행 디렉토리에 위치)	permit(서버 실행 디렉토 리에 위치)

RMI 서버측 컴파일 및 실행

```
211.55.52.254 - CRT
File Edit View Options Transfer Script Tools Window Help

[root@linuxsvr dynamic_rmi]#
[root@linuxsvr dynamic_rmi]#
[root@linuxsvr dynamic_rmi]# javac pejb/rmi/base/Anything.java
[root@linuxsvr dynamic_rmi]# javac pejb/rmi/base/Workable.java
[root@linuxsvr dynamic_rmi]# javac pejb/rmi/server/GenericServer.java
[root@linuxsvr dynamic_rmi]# rmic -v1.2 pejb.rmi.server.GenericServer
[root@linuxsvr dynamic_rmi]# jar cvf stub.jar pejb/rmi/server/GenericServer_Stub.class pejb/rmi/base/Anything.class pejb/rmi/base/Workable.class
added manifest
adding: pejb/rmi/server/GenericServer_Stub.class(in = 1895) (out= 954)(deflated 49%)
adding: pejb/rmi/base/Anything.class(in = 209) (out= 167)(deflated 20%)
adding: pejb/rmi/base/Workable.class(in = 279) (out= 210)(deflated 24%)
[root@linuxsvr dynamic_rmi]# mv stub.jar /home/httpd/oraclejava/htdocs/fromServer_toClient/
mv: `/home/httpd/oraclejava/htdocs/fromServer_toClient/stub.jar' and `/home/httpd/oraclejava/htdocs/fromServer_toClient/stub.jar' are the same file
[root@linuxsvr dynamic_rmi]# pwd
/home/jclee/dynamic_rmi
[root@linuxsvr dynamic_rmi]# cd ..
[root@linuxsvr jclee]# pwd
/home/jclee
[root@linuxsvr jclee]# rmiregistry 2030 &
[2] 3056
[root@linuxsvr jclee]# cd dynamic_rmi/
[root@linuxsvr dynamic_rmi]# java -Djava.security.policy=permit -Djava.rmi.server.codebase=http://www.oraclejava.co.kr/fromServer_toClient/stub.jar pejb.rmi.server.GenericServer
Remote Object registered and ready to servie...
```

주: 만약 package root 즉 pejb가 있는 곳에서 rmiregistry를 실행하면 나중에 클라이언트에서 다음과 같은 에러가 발생한다 .java.lang.ClassNotFoundException: access to class loader denied

RMI 서버 실행 디렉토리에 permit이라는 보안 파일 만들것

RMI 클라이언트측 컴파일/실행

```
C:\WINNT\System32\cmd.exe
E:\강의자료\여성부\자바네트웍\예제\dynamic_rmi>javac pejb\rmi\base\Anything.java

E:\강의자료\여성부\자바네트웍\예제\dynamic_rmi>javac pejb\rmi\base\Workable.java

E:\강의자료\여성부\자바네트웍\예제\dynamic_rmi>javac pejb\rmi\client\QueryObject
.java

E:\강의자료\여성부\자바네트웍\예제\dynamic_rmi>javac pejb\rmi\client\ClientWrapp
er.java

E:\강의자료\여성부\자바네트웍\예제\dynamic_rmi>javac pejb\rmi\client\ClientWrapp
er.java

E:\강의자료\여성부\자바네트웍\예제\dynamic_rmi>jar cvf client.jar pejb/rmi/clien
t/ClientWrapper.class pejb/rmi/client/QueryObject.class
추가된 manifest
추가 중: pejb/rmi/client/ClientWrapper.class<내부 = 378> <외부= 273><27%가 감소
되었습니다.>
추가 중: pejb/rmi/client/QueryObject.class<내부 = 1900> <외부= 1090><42%가 감소
되었습니다.>

E:\강의자료\여성부\자바네트웍\예제\dynamic_rmi>client.jar를 java.rmi.server.code
base 값으로 지정할 디렉토리를 이동하자. 본예제에서는 이파일을 www.oraclejava.co.
kr/fromClient_toServer 디렉토리로 옮긴것으로 간주하자.
'client.jar'은<는> 내부 또는 외부 명령, 실행할 수 있는 프로그램, 또는
배치 파일이 아닙니다.

E:\강의자료\여성부\자바네트웍\예제\dynamic_rmi>java -classpath . pejb.rmi.client
.QueryClient
Result for Query select id from users
jclee
bit

E:\강의자료\여성부\자바네트웍\예제\dynamic_rmi>_
```

Example

NameCard Server and Client

Used LocateRegistry class to create
Register

NameCard.java

```
public class NameCard implements java.io.Serializable
{
    public String name;
    public String address;
    public String phone;
    public int age;
    public String toString()
    { return "[이름: " + name + ", 주소: " + address
        + ", 전화번호: " + phone + ", 나이: " + age + "];"
    }
}
```

NameCardInterface.java

```
public interface NameCardInterface extends java.rmi.Remote
{
    void save(NameCard card)
        throws java.rmi.RemoteException, AlreadyNameExistException;
    NameCard retrieve(String name) throws java.rmi.RemoteException;
}
```

AlreadyNameExistException.java

```
public class AlreadyNameExistException extends Exception
{
    public AlreadyNameExistException(String s)
    {    super(s);
    }
}
```

NameCardClient.java

```
import java.awt.*;
import java.awt.event.*;
import java.rmi.*;

public class NameCardClient extends Frame
{
    public static void main(final String[] args)
    {
        new NameCardClient(args[0]);
    }

    NameCardInterface namecardServer;
    TextField name = new TextField(70);
    TextField address = new TextField(70);
```


NameCardClient.java

```
TextField phone = new TextField(70);
TextField age = new TextField(70);
Button send = new Button("저장");
Button retrieve = new Button("검색");

public NameCardClient(String host)
{

    if ( System.getSecurityManager() == null )
        System.setSecurityManager(new RMISecurityManager());
    try
    {
        namecardServer = (NameCardInterface) Naming.lookup(
            "rmi://" + host + "/namecard"); // host-> localhost
    }
}
```

NameCardClient.java

```
} catch (Exception e)
{   e.printStackTrace();
}
```

```
Panel p1 = new Panel( new GridLayout(0, 1) );
p1.add( new Label("이름: ") );
p1.add( new Label("주소: ") );
p1.add( new Label("전화번호: ") );
p1.add( new Label("나이: ") );
```

```
Panel p2 = new Panel( new GridLayout(0, 1) );
p2.add( name );
p2.add( address );
p2.add( phone );
```

NameCardClient.java

```
p2.add( age );

Panel p3 = new Panel();
p3.add( send );
p3.add( retrieve );

add( p1, "West" );
add( p2, "Center" );
add( p3, "South" );

send.addActionListener( sendHandler );
retrieve.addActionListener( retrieveHandler );
pack();
setVisible(true);
}
```

NameCardClient.java

```
ActionListener sendHandler = new ActionListener()
{
    public void actionPerformed( ActionEvent ev )
    {
        try
        {
            NameCard nc = new NameCard();
            nc.name = name.getText();
            nc.address = address.getText();
            nc.phone = phone.getText();
            nc.age = Integer.parseInt( age.getText() );
            namecardServer.save( nc );
        } catch( AlreadyNameExistException ex )
        {
            System.err.println( "오류: 이미 저장된 이름");
        } catch( RemoteException ex )
        {
```

NameCardClient.java

```
        { ex.printStackTrace();  
        }  
    }  
};
```

```
ActionListener retrieveHandler = new ActionListener()  
{ public void actionPerformed((ActionEvent ev )  
    {  
        try  
        {  
            NameCard nc = namecardServer.retrieve(name.getText());  
            if ( nc == null )  
            {  
                System.err.println( "검색 실패" );  
            }  
        }  
    }  
};
```

NameCardClient.java

```
        return;  
    }  
    address.setText( nc.address );  
    phone.setText( nc.phone );  
    age.setText( String.valueOf(nc.age) );  
} catch (RemoteException ex)  
{    ex.printStackTrace();  
}  
}  
};  
}
```

NameCardServer.java

```
import java.util.*;
import java.io.*;
import java.rmi.*;
import java.rmi.registry.LocateRegistry;
import java.rmi.server.UnicastRemoteObject;

class NameCardServer extends UnicastRemoteObject
    implements NameCardInterface
{
    public static void main(final String[] args)
    {
        try
        {
            if ( System.getSecurityManager() == null )
```

NameCardServer.java

```
        System.setSecurityManager(new RMISecurityManager());
        LocateRegistry.createRegistry(1099);
        NameCardServer namecard = new NameCardServer();
        Naming.rebind("namecard", namecard);
    } catch (Exception e)
    {   e.printStackTrace();
    }
}

Vector database = new Vector();

public NameCardServer() throws RemoteException
{
}
```


NameCardServer.java

```
public synchronized void save(NameCard card)
    throws java.rmi.RemoteException, AlreadyNameExistException
{
    for( int i = 0; i < database.size(); i++ )
    {
        NameCard c = (NameCard) database.elementAt(i);
        if (c.name.equals(card.name))
            throw new AlreadyNameExistException(card.name);
    }
    database.addElement( card );
}
```

```
public synchronized NameCard retrieve(String name)
    throws java.rmi.RemoteException
```

NameCardServer.java

```
{
    for ( int i = 0; i < database.size(); i++ )
    {
        NameCard storedCard = (NameCard) database.elementAt(i);
        if ( storedCard.name.equals(name) )
        {
            return storedCard;
        }
    }
    return null;
}
```

Example

Dynamic code downloading using RMI

Compute.java

```
import java.rmi.Remote;  
import java.rmi.RemoteException;  
  
public interface Compute extends Remote {  
    Object executeTask(Task t) throws RemoteException;  
}
```

ComputeEngine.java

```
import java.rmi.*;
import java.rmi.server.*;

public class ComputeEngine extends UnicastRemoteObject
    implements Compute
{
    public ComputeEngine() throws RemoteException {
        super();
    }

    public Object executeTask(Task t) {
        return t.execute();
    }
}
```

ComputeEngine.java

```
public static void main(String[] args) {
    if (System.getSecurityManager() == null) {
        System.setSecurityManager(new RMISecurityManager());
    }
    String name = "//localhost/Compute";
    try {
        Compute engine = new ComputeEngine();
        Naming.rebind(name, engine);
        System.out.println("ComputeEngine bound");
    } catch (Exception e) {
        System.err.println("ComputeEngine exception: " + e.getMessage());
        e.printStackTrace();
    }
}
```

Task.java

```
import java.io.Serializable;
```

```
public interface Task extends Serializable {  
    Object execute();  
}
```

Pi.java

```
import java.math.*;
```

```
public class Pi implements Task {
```

```
    /** constants used in pi computation */
```

```
    private static final BigDecimal ZERO =  
        BigDecimal.valueOf(0);
```

```
    private static final BigDecimal ONE =  
        BigDecimal.valueOf(1);
```

```
    private static final BigDecimal FOUR =  
        BigDecimal.valueOf(4);
```

```
    /** rounding mode to use during pi computation */
```

```
    private static final int roundingMode = BigDecimal.ROUND_HALF_EVEN;
```


Pi.java

```
/** digits of precision after the decimal point */  
private int digits;
```

```
/**  
 * Construct a task to calculate pi to the specified  
 * precision.  
 */
```

```
public Pi(int digits) {  
    this.digits = digits;  
}
```

```
/**  
 * Calculate pi.  
 */
```

Pi.java

```
public Object execute() {  
    return computePi(digits);  
}  
  
public static BigDecimal computePi(int digits) {  
    int scale = digits + 5;  
    BigDecimal arctan1_5 = arctan(5, scale);  
    BigDecimal arctan1_239 = arctan(239, scale);  
    BigDecimal pi = arctan1_5.multiply(FOUR).subtract(  
        arctan1_239).multiply(FOUR);  
    return pi.setScale(digits,  
        BigDecimal.ROUND_HALF_UP);  
}
```

Pi.java

```
public static BigDecimal arctan(int inverseX,  
                                int scale)  
{  
    BigDecimal result, numer, term;  
    BigDecimal invX = BigDecimal.valueOf(inverseX);  
    BigDecimal invX2 =  
        BigDecimal.valueOf(inverseX * inverseX);  
  
    numer = ONE.divide(invX, scale, roundingMode);  
  
    result = numer;  
    int i = 1;  
    do {  
        numer = numer.divide(invX2, scale, roundingMode);
```

Pi.java

```
    int denom = 2 * i + 1;
    term = numer.divide(BigDecimal.valueOf(denom), scale, roundingMode);
    if ((i % 2) != 0) {
        result = result.subtract(term);
    } else {
        result = result.add(term);
    }
    i++;
} while (term.compareTo(ZERO) != 0);
return result;
}
}
```

ComputePi.java

```
import java.rmi.*;
import java.math.*;

public class ComputePi {
    public static void main(String args[]) {
        if (System.getSecurityManager() == null) {
            System.setSecurityManager(new RMISecurityManager());
        }
        try {
            String name = "//localhost/Compute";
            Compute comp = (Compute) Naming.lookup(name);
            Pi task = new Pi(Integer.parseInt(args[0]));
            BigDecimal pi = (BigDecimal) (comp.executeTask(task));
            System.out.println(pi);
        }
    }
}
```

ComputePi.java

```
    } catch (Exception e) {  
        System.err.println("ComputePi exception: " + e.getMessage());  
        e.printStackTrace();  
    }  
}  
}
```

Policy.all

```
grant {  
    permission java.net.SocketPermission "*:1024-65535", "connect,accept";  
    permission java.net.SocketPermission "*:80", "connect";  
};
```

Running Example

1. Start rmiregistry

- rmiregistry

2. Start Server

- java ComputeEngine

3. Start Client

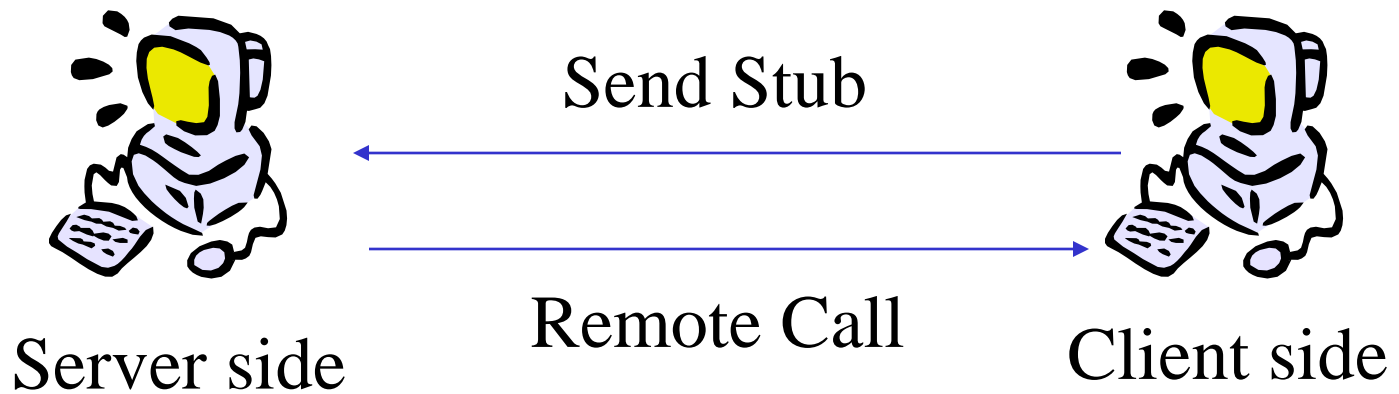
- java -Djava.rmi.server.codebase=http://localhost:8080/ -
Djava.security.policy=policy.all ComputePi 10

RMI Callback

RMI Callback

- Send a Remote Object Reference(Stub Object) to receipt side
- In this case, Don't need register to a RMI Registration Server.

RMI Callback



RMI Callback Example 1.

Simple Registration Message Sending

Registry.java

```
public interface Registry extends java.rmi.Remote
{
    void register(String name, RegistryNotify client)
        throws java.rmi.RemoteException;

    void deregister(String name)
        throws java.rmi.RemoteException;
}
```

RegistryServer.java

```
import java.util.*;
import java.rmi.*;
import java.rmi.registry.LocateRegistry;
import java.rmi.server.UnicastRemoteObject;

public class RegistryServer extends UnicastRemoteObject
    implements Registry
{
    public static void main(String args[])
    {
        try
        {
            LocateRegistry.createRegistry(1099);
            RegistryServer regserver1 = new RegistryServer();
        }
    }
}
```

RegistryServer.java

```
        Naming.rebind("regserver1", regserver1);
    } catch (Exception e)
    {   e.printStackTrace();
    }
}
```

```
Vector names = new Vector();
// RegistryNotify의 벡터
Vector clients = new Vector();
```

```
public RegistryServer() throws RemoteException
{
}
}
```

RegistryServer.java

```
public synchronized void register(String name, RegistryNotify client)
    throws java.rmi.RemoteException
{
    int ind = names.indexOf(name);
    if ( ind != -1 )
        throw new RemoteException(
            "오류: 이미 등록되어있는 이름 :" + name);
    names.addElement( name );
    clients.addElement( client );
    for( int i = 0; i < clients.size(); i++ )
    {
        try
        {
```


RegistryServer.java

```
        RegistryNotify aClient
            = (RegistryNotify) clients.elementAt(i);
        aClient.notify( "새로 등록된 이름: " + name );
    } catch (Exception e)
    {
        e.printStackTrace();
    }
}

public synchronized void deregister(String name)
    throws java.rmi.RemoteException
{
    int ind = names.indexOf(name);
    if ( ind == -1 )
```

RegistryServer.java

```
        throw new RemoteException(
            "오류: 등록되지 않은 이름 :" + name);
    for( int i = 0; i < clients.size(); i++ )
    {
        try
        {
            RegistryNotify aClient
                = (RegistryNotify) clients.elementAt(i);
            aClient.notify( "등록 취소된 이름: " + name );
        } catch (Exception e)
        {
            e.printStackTrace();
        }
    }
    names.removeElementAt(ind);
```

RegistryServer.java

```
    clients.removeElementAt(ind);  
}  
}
```

RegistryNotify.java

```
public interface RegistryNotify extends java.rmi.Remote
{
    void notify(String message) throws java.rmi.RemoteException;
}
```

RegistryClient.java

```
import java.applet.*;
import java.awt.*;
import java.awt.event.*;
import java.rmi.*;
import java.rmi.server.*;
import java.lang.reflect.Method;

public class RegistryClient extends Applet
    implements RegistryNotify, ActionListener
{
    public static void main(String[] args)
    {
        Frame frame = new Frame();
        RegistryClient client = new RegistryClient();
```

RegistryClient.java

```
client.rmiRegistryHost = args[0];  
frame.add(client, "Center");  
client.init();  
frame.setSize( 500, 500 );  
frame.setVisible( true );  
}
```

```
String rmiRegistryHost = null;  
Registry regServer;
```

```
TextField nameField = new TextField(10);  
Button registerButton = new Button("등록");  
Button deregisterButton = new Button("등록 취소");  
TextArea displayArea = new TextArea();
```

RegistryClient.java

```
public void init()
{
    if ( System.getSecurityManager() == null )
        System.setSecurityManager(new RMISecurityManager());
    try
    {
        if (rmiRegistryHost == null )
            rmiRegistryHost = getCodeBase().getHost();
        regServer = (Registry) Naming.lookup(
            "rmi://" + rmiRegistryHost + "/regserver1");
        UnicastRemoteObject.exportObject(this);
    } catch (Exception e)
    {
        e.printStackTrace();
    }
}
```

RegistryClient.java

```
setLayout( new BorderLayout() );
Panel p = new Panel();
p.add( new Label("이름: ") );
p.add( nameField );
p.add( registerButton );
p.add( deregisterButton );
add( p, "North" );
add(displayArea, "Center");

registerButton.addActionListener( this );
deregisterButton.addActionListener( this );
}
```


RegistryClient.java

```
public void actionPerformed((ActionEvent ev)
{
    try
    {
        if ( ev.getActionCommand().equals("등록") )
            regServer.register( nameField.getText(), this );
        else if ( ev.getActionCommand().equals("등록 취소") )
            regServer.deregister(nameField.getText());
    } catch( RemoteException ex )
    {
        displayArea.append( "예외: " + ex + "\n" );
    }
}
```

RegistryClient.java

```
public void notify(String message)
    throws java.rmi.RemoteException
{
    displayArea.append( message + "\n" );
}
}
```

RegistryClient.html

```
<HTML><body>  
<h1> Registry 클라이언트 애플릿 </h1>  
<applet code="RegistryClient" width=500 height=500></applet>  
</body></HTML>
```

RMI Callback Example 2.

RMI Callback Chatting

referenced from Java Network Programming 2nd edition
MAMMING pub.

RMICallbackServer.java

```
import java.rmi.*;
```

```
public interface RMICallbackServer extends Remote {  
    public static final String REGISTRY_NAME = "Callback Server";  
    public abstract void register (RMICallbackClient client) throws RemoteException;  
    public abstract void deregister (RMICallbackClient client) throws RemoteException;  
    public abstract void say (String message) throws RemoteException;  
}
```

RMICallbackClient.java

```
import java.rmi.*;
```

```
public interface RMICallbackClient extends Remote {  
    public abstract void said (String message) throws RemoteException;  
}
```

RMICallbackServerImpl.java

```
import java.rmi.*;
import java.util.*;
import java.rmi.server.*;
import java.rmi.registry.*;

public class RMICallbackServerImpl extends UnicastRemoteObject
    implements RMICallbackServer {
    protected Vector clients;

    public RMICallbackServerImpl () throws RemoteException {
        clients = new Vector ();
    }

    public void register (RMICallbackClient client) {
```

RMICallbackServerImpl.java

```
try {
    say (getClientHost () + " has joined.");
} catch (ServerNotActiveException ignored) {
}
clients.addElement (client);
}

public void deregister (RMICallbackClient client) {
    clients.removeElement (client);
    try {
        say (getClientHost () + " has left.");
    } catch (ServerNotActiveException ignored) {
    }
}
```


RMICallbackServerImpl.java

```
public void say (String message) {  
    Vector clients = (Vector) this.clients.clone ();  
    for (int i = 0; i < clients.size (); ++ i) {  
        RMICallbackClient client = (RMICallbackClient) clients.elementAt (i);  
        try {  
            client.said (message);  
        } catch (RemoteException ex) {  
            this.clients.removeElement (client);  
        }  
    }  
}
```

RMICallbackServerImpl.java

```
public static void main (String[] args) throws RemoteException {  
    RMICallbackServerImpl callbackServer = new RMICallbackServerImpl ();  
    Registry registry = LocateRegistry.getRegistry ();  
    registry.rebind (REGISTRY_NAME, callbackServer);  
}  
}
```

RMICallbackClientImpl.java

```
import java.awt.*;  
import java.rmi.*;  
import java.awt.event.*;  
import java.rmi.server.*;  
import java.rmi.registry.*;
```

```
public class RMICallbackClientImpl extends UnicastRemoteObject implements  
    RMICallbackClient, ActionListener {  
    protected String host;  
    protected Frame frame;  
    protected TextField input;  
    protected TextArea output;
```

RMICallbackClientImpl.java

```
public RMICallbackClientImpl (String host) throws RemoteException {
    this.host = host;
    frame = new Frame ("RMICallbackClientImpl [" + host + "]");
    frame.add (output = new TextArea (), "Center");
    output.setEditable (false);
    frame.add (input = new TextField (), "South");
    input.addActionListener (this);
    frame.addWindowListener (new WindowAdapter () {
        public void windowOpened (WindowEvent ev) {
            input.requestFocus ();
        }

        public void windowClosing (WindowEvent ev) {
```

RMICallbackClientImpl.java

```
try {  
    stop ();  
} catch (RemoteException ex) {  
    ex.printStackTrace ();  
}  
}  
});  
frame.pack ();  
}
```

```
protected RMICallbackServer server;
```

```
public synchronized void start () throws RemoteException, NotBoundException  
{  
    if (server == null) {
```

RMICallbackClientImpl.java

```
Registry registry = LocateRegistry.getRegistry (host);
server = (RMICallbackServer) registry.lookup
(RMICallbackServer.REGISTRY_NAME);
server.register (this);
frame.setVisible (true);
}
}
```

```
public synchronized void stop () throws RemoteException {
    frame.setVisible (false);
    RMICallbackServer server = this.server;
    this.server = null;
    if (server != null)
        server.deregister (this);
}
```

RMICallbackClientImpl.java

```
public void said (String message) {  
    output.append (message + "\n");  
}  
  
public void actionPerformed (ActionEvent ev) {  
    try {  
        RMICallbackServer server = this.server;  
        if (server != null) {  
            server.say (ev.getActionCommand ());  
            input.setText ("");  
        }  
    } catch (RemoteException ex) {  
        input.setVisible (false);  
        frame.validate ();  
    }  
}
```

RMICallbackClientImpl.java

```
    ex.printStackTrace ();
}
}

public static void main (String[] args) throws RemoteException,
    NotBoundException {
    if (args.length != 1)
        throw new IllegalArgumentException
            ("Syntax: RMICallbackClientImpl <host>");
    RMICallbackClientImpl callbackClient =
        new RMICallbackClientImpl (args[0]);
    callbackClient.start ();
}
}
```

RMI & Distribute Server

Making Distribute Server
with RMI & Servlet

Counter.java

```
public interface Counter extends java.rmi.Remote
{
    int increment() throws java.rmi.RemoteException;
}
```

CounterServer.java

```
import java.rmi.*;
import java.rmi.server.*;

public class CounterServer extends UnicastRemoteObject
    implements Counter
{
    int i;

    public CounterServer(int ival) throws RemoteException
    {
        i = ival;
    }
}
```

CounterServer.java

```
public synchronized int increment() throws RemoteException
{
    try
    {   System.out.println(RemoteServer.getClientHost());
    } catch ( ServerNotActiveException ex )
    {   ex.printStackTrace();
    }
    return ++i;
}
```

```
public static void main(String args[])
{
    try{
        CounterServer counter = new CounterServer(0);
```

CounterServer.java

```
        Naming.rebind("counter", counter);
    } catch (Exception e){
        e.printStackTrace();
    }
}
}
```

CounterServletServer.java

```
import java.rmi.*;
import java.io.*;
import java.net.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class CounterServletServer extends HttpServlet
{
    Counter counter=null;

    public void init() throws ServletException {

        try {
            Counter counter = (Counter) Naming.lookup("rmi://localhost/counter");
```

CounterServletServer.java

```
    } catch (NotBoundException e){
        System.out.println("NotBoundException Occured!!");
    } catch (MalformedURLException e) {
        System.out.println("Check your destination!!");
    } catch (RemoteException e){
        System.out.println("Check our Server and else..");
    }
}

public void doGet(HttpServletRequest req, HttpServletResponse res)
                throws ServletException, IOException
{
    if(System.getSecurityManager()==null)
        System.setSecurityManager(new RMISecurityManager());
}
```

CounterServletServer.java

```
int connectNumber=0;
connectNumber = counter.increment();

res.setContentType("text/html");
PrintWriter out = res.getWriter();

out.println("<html><head><title> Connect_Number </title></head>" +
    "<body>" +
    "Yon are " + connectNumber+ " th visitor!!" +
    "</body></html>"
    );
out.close();
}
}
```


THE END

