Java 네트워크

네트워크 기본 개념

네트워크 기본 개념

☞ IP 주소(Internet Protocol Address)

- IP 주소란?

1 → 인터넷 상에서 장치간 통신을 위해 장치를 식별하는 주소

- IPv4 vs. IPv6

2 IPv4 : - 현재 일반적으로 사용되는 주소로 32bit 구성 (약 43억개)
- [0-255]. [0-255]. [0-255]. [0-255]와 같이 10진수 4개를 .으로 구분하여 표기 (예, 192.168.0.2)
- 특정 IP는 특별용도로 예약되어 사용 (예, 127.0.01 (로컬호스트(루프백) 주소))

3 IPv6 : - IPv4에서의 IP 부족현상을 해결하기 위한 방법으로 128bit(16byte)로 구성 (43억 * 43억 * 43억 * 43억)
- [0000-FFFF]:[0000-FFFF]:[0000-FFFF]:[0000-FFFF]:[0000-FFFF]:[0000-FFFF]:[0000-FFFF]:[0000-FFFF] 와 같이 4자리 16진수(16bit) 8개를 콜론(:)으로 연결하여 표기 (예. 200F:02AB:A038:FF09:200F:02AB:A038:FF09)
- 340,282,366,920,938,463,463,374,607,431,768,211,456 가지수

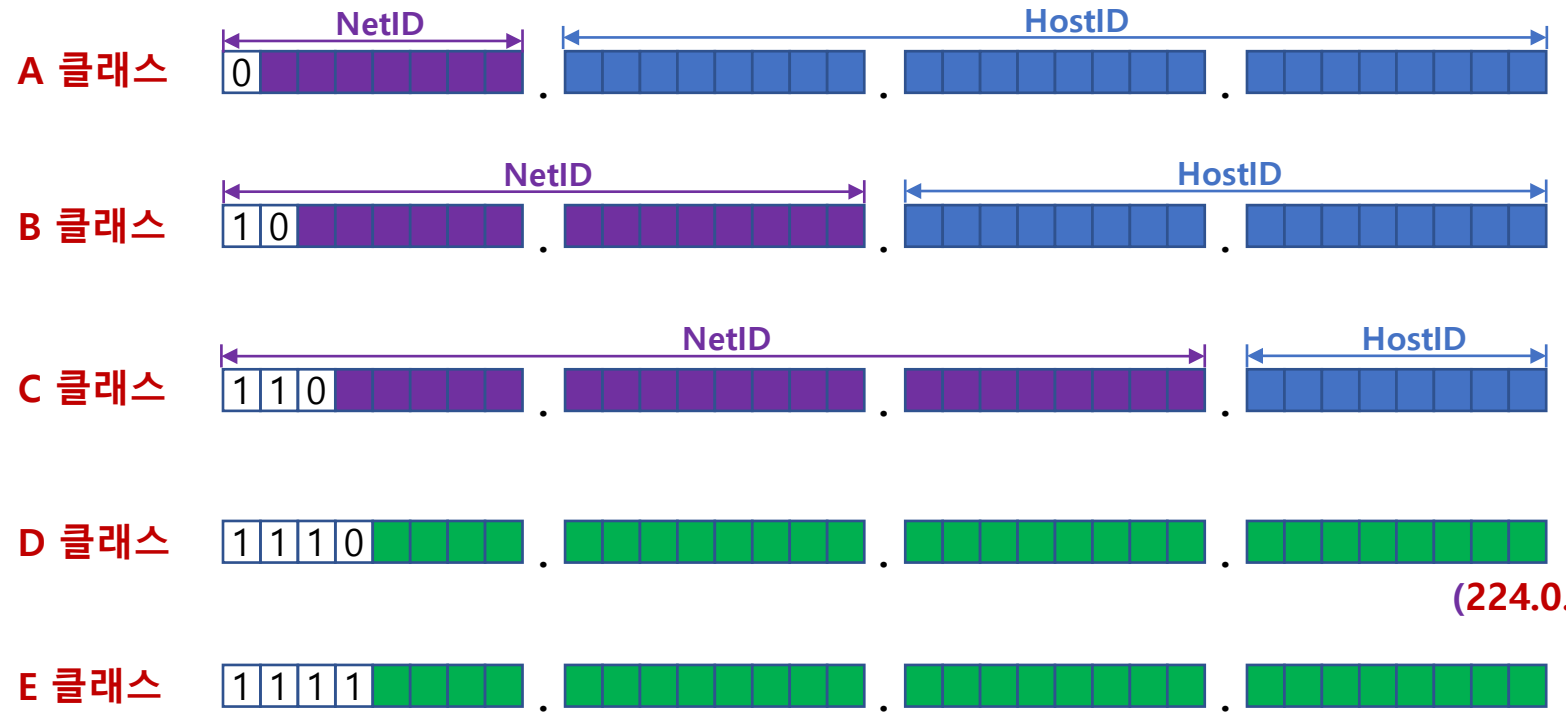
4 ipconfig →

```
이더넷 어댑터 이더넷:

   연결별 DNS 접미사. . . . : Davolink
   링크-로컬 IPv6 주소 . . . . : fe80::d06a:62bc:5dc0:ba96%3
   IPv4 주소 . . . . . . . . . . : 192.168.123.159
   서브넷 마스크 . . . . . . . . : 255.255.255.0
   기본 게이트웨이 . . . . . . . : 192.168.123.254
```

네트워크 기본 개념

☞ IP 주소(Internet Protocol Address)

- IP 주소의 분류



1

2

멀티캐스트 주소
(224.0.0.0 ~ 239.255.255.255)

향후 사용을 위해 예약

네트워크 기본 개념

☞ IP 주소(Internet Protocol Address)

- 클래스별 가용 네트워크 및 호스트 범위와 수

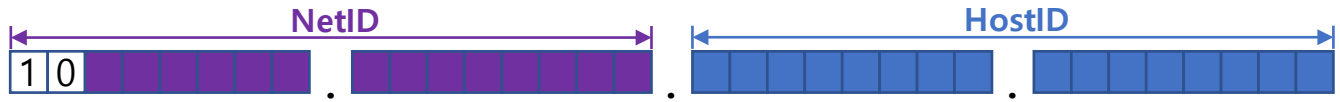
A 클래스

① → 네트워크 주소의 범위 : [0-127]
→ 할당 가능 네트워크 수: 128

→ 호스트 주소의 범위 : [0-255].[0-255].[0-255]
→ 할당 가능 호스트의 수: 256*256*256-2 = 16,777,214

② 가능범위의
첫번째 주소: network 주소
마지막 주소: broadcast 주소

B 클래스



③ → 네트워크 주소의 범위 : [128-191].[0-255]
→ 할당 가능 네트워크 수: 64*256=16,384

→ 호스트 주소의 범위 : [0-255].[0-255]
→ 할당 가능 호스트의 수: 256*256-2 = 65,534

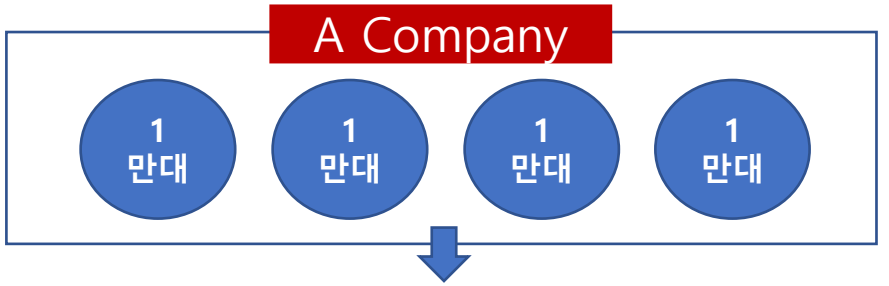
C 클래스

④ → 네트워크 주소의 범위 : [192-223].[0-255].[0-255]
→ 할당 가능 네트워크 수: 32*256*256=2,097,152

→ 호스트 주소의 범위 : [0-255]
→ 할당 가능 호스트의 수: 256-2 = 254

네트워크 기본 개념

2



☞ IP 주소(Internet Protocol Address)

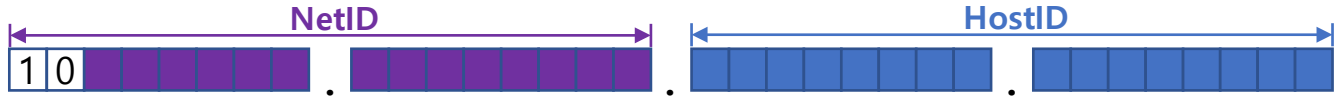
1 - 서브넷 마스크 → 네트워크 ID (NetID)와 호스트 ID(HostID)를 구분하는 마스크

→ IP주소 & 서브넷마스크 = 네트워크 ID

→ 예) IP(128.123.222.123), Subnet Mask(255.255.0.0) ➔ NetID=128.123.0.0

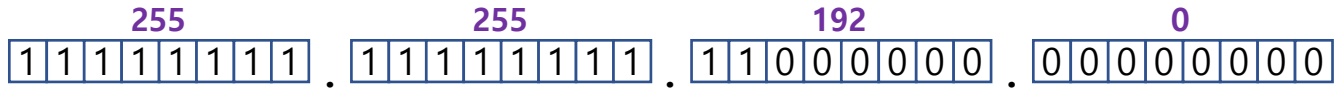
3 하나의 네트워크를 다시 여러 서브 네트워크로 변환하는 경우에도 사용

4 B 클래스 하나를 할당

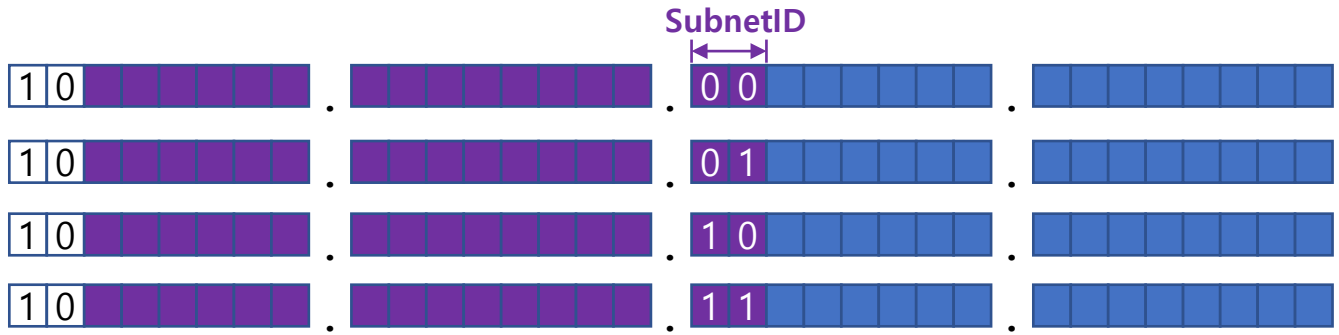


호스트: 65,536-2

4개의 서브네트워크를 구성



5 ¼의 호스트를 가질 수 있는 4개의 서브 네트워크 구성



호스트: 16,384-2

호스트: 16,384-2

호스트: 16,384-2

호스트: 16,384-2

네트워크 기본 개념

☞ IP 주소(Internet Protocol Address)

1 - 공인 IP vs. 사설 IP

공인 IP : - 인터넷 상에 서로 다른 장치들 간의 유일한 식별에 사용되는 IP
- 나라별 별도의 기관에서 관리하며 한국은 인터넷진흥원에서 관리

2

사설 IP : - 내부망 내에서만 한정적으로 사용되는 IP
- 내부에서만 사용되기 때문에 네트워크별로 중복 가능

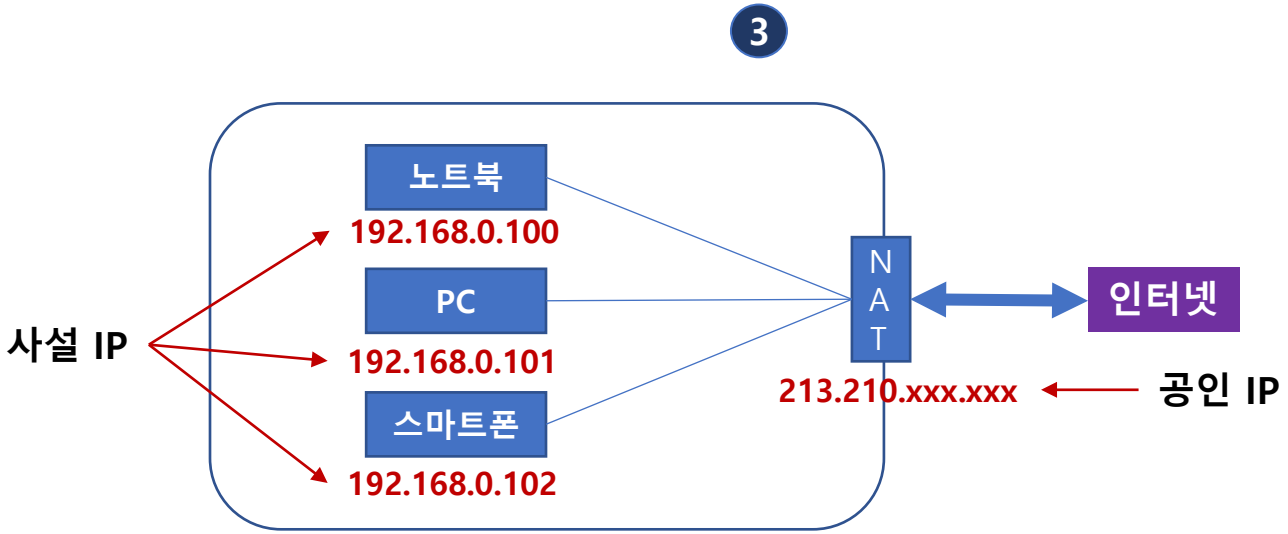
3

사설 IP

4 A 클래스 : **10**.[0-255].[0-255].[0-255]

B 클래스 : **172.[16-31]**.[0-255].[0-255]

C 클래스 : **192.168**.[0-255].[0-255]



The End

네트워크 기본 개념

☞ 포트(Port)

1 - 포트란? → 호스트 내에서 실행되고 있는 프로세스를 구분하기 위한 16비트의 논리적 할당(0~65535)
→ 호스트의 IP가 집주소 해당하는 개념이라면 Port는 방번호에 해당
→ 호스트의 IP가 컴퓨터를 찾기 위한 정보라면 Port는 프로그램에 해당 (어떤 프로그램이 사용하는 정보인지)

2 포트번호 : 0~1023
잘 알려진 포트(well-known port)로 대표적인 인터넷 프로그램이 미리 예약하여 사용하는 포트

3

포트	서비스	
20	FTP-Data	FTP
21	FTP-Control	FTP
23	Telnet	원격터미널
25	SMTP-mail	메일
53	DNS	도메인이름→IP주소
69	TFTP	간단한 FTP

포트	서비스	
80	**HTTP**	**Web**
110	POP3	메일
111	RPC	원격 프로시져 콜
138	NetBIOS	윈도우 파일 공유
143	IMAP	메일
161	SNMP	네크워크 관리

네트워크 기본 개념

1 ☞ **TCP(Transmission Control Protocol)와 UDP(User Datagram Protocol)**

- TCP vs. UDP

TCP : - 신뢰성이 높음 (오류시 재전송)
- 연결형 프로토콜 : 통신과정에서 연결 유지 필요 (통신상대가 많은 경우 시스템 부하가 높음)
2 - 전송데이터 크기는 제한이 없음
- 파일 전송과 같이 신뢰성이 필요한 서비스에 주로 사용

- 전화통신과 유사한 개념

UDP : - 신뢰성이 낮음 (오류시 또는 미전달시 데이터그램(전달데이터) 삭제)
- 비연결형 프로토콜 : 통신과정에서 연결 유지 불필요 (통신 상대가 많아도 시스템 부하가 낮음)
- 전송데이터의 크기는 65,536 바이트(헤더 포함)로 초과시 나누어 전송
3 - 실시간성과 같은 성능이 중요한 서비스에 주로 사용

- 우편(소포) 통신과 유사한 개념

네트워크 기본 개념

☞ TCP(Transmission Control Protocol)와 UDP(User Datagram Protocol)

- TCP vs. UDP

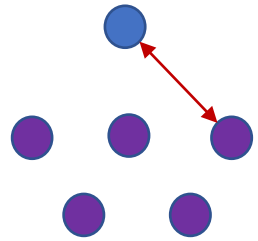
1

	TCP	UDP
신뢰성	높음	낮음
연결 방식	연결지향	비연결 지향
패킷 교환 방식	가상회선 방식	데이터그램 방식
전송 순서	전송 순서 보장	전송 순서 비보장
수신 여부 확인	수신 여부 확인	수신 여부 미확인
통신 방식	1:1 통신	1:1 또는 1:N 또는 N:N
속도	상대적으로 느림	상대적으로 빠름

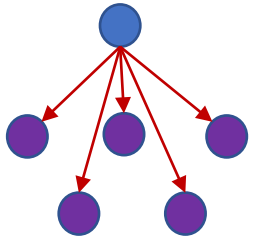
네트워크 기본 개념

☞ 1:1 통신과 1:N 통신

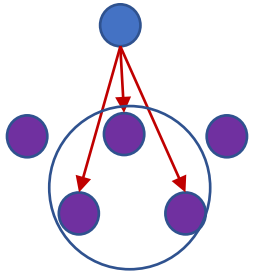
- 유니캐스팅(Unicasting) vs. 브로드캐스팅(Broadcasting) vs. 멀티캐스팅(Multicasting)



1 유니캐스팅 : - 두 장치간 1:1 통신 방식 (특정 장치의 주소를 지정하여 통신)
- 물리주소(MAC 주소)를 기반으로 통신 (자신의 물리주소가 아닌 패킷이 도착하면 프레임을 버림)
- 동일한 내용을 여러 사용자에게 보내고자 하는 경우 비효율적임



2 브로드캐스팅 : - UDP 기반 호스트가 속해 있는 네트워크 내의 모든 장치에 패킷을 전달하는 1:N 통신 방식
- 라우터의 설정에 따라 라우터를 경유하지 못할 수도 있으며 주로 내부 네트워크에만 한정
- 동일한 내용을 여러 사용자에게 보내고자 하는 경우 효율적임
- 브로드캐스트 주소
: (모든 HostID = 1): NetID + 11... 또는 NetID + SubnetID + 11....
→ 해당 네트워크 또는 서브넷의 모든 장치에 전달 (단, 라우터가 브로드캐스트를 지원해야 함)
: (모든 IP = 1): 255.255.255.255
→ 호스트가 속한 네트워크의 모든 장치에 전달



3 멀티캐스팅 : - 특정 장치들에 데이터를 전달하는 1:N 통신 방식
- 실제 호스트 주소가 아닌 가상 D 클래스 IP 주소에 가입된 호스트에게 데이터 전달
- D 클래스 IP 주소 [224-239].[0-255].[0-255].[0-255]
- 필요성: 예) 네트워크 내 자신을 포함하여 6개의 단말이 있고 이 중 3개의 단말에 동일 데이터를 전송하고자 하는 경우
유니캐스트로 보내는 경우 → 3번 반복해야 함
브로드캐스트로 보내는 경우 → 2 호스트가 필요없이 받음

The End

주소 저장 클래스

InetAddress

InetSocketAddress

주소 저장 클래스

1 ☞ **InetAddress** → IP주소(호스트이름) 저장 및 관리 클래스 (Port 번호 관리 못함)

2 - InetAddress 객체 생성 : 정적(static) 메서드를 사용하여 객체 생성

3

리턴타입	메서드 명	주요 내용
InetAddress	getByName(String host)	Host 이름과 해당 IP 주소 저장 객체 리턴
	getByAddress(byte[] addr)	입력 IP주소 저장 객체 리턴 (128 이상의 경우 (byte) 캐스팅 필요)
	getByAddress(String host, byte[] addr)	Host 이름과 입력 IP주소 저장 객체 리턴 (IP 주소가 128 이상의 경우 (byte) 타입으로 캐스팅 필요)
	getLocalHost()	로컬 호스트 IP저장 객체 리턴
	getLoopbackAddress()	루프백(loopback) IP (**127.0.0.1**) 저장 객체 리턴
InetAddress[]	getAllByName(String host)	**여러 개의 IP를 사용**하는 경우 모든 host IP 저장 객체 리턴

4

TIP

getByName(.)으로 생성하는 경우 : InetAdddress는호스트 이름 + IP주소 저장
((객체를 만드는 시점에) 실제 해당 호스트 이름이 DNS에 정확히 있어야 함 (없으면 예외발생))

getByAddress (byte[]) 로 생성하는 경우 : IP주소 저장 ((객체를 만드는 시점에) 실제 정확한 IP인지는 중요하지 않음)

getByAddress (String, byte[]) 로 생성하는 경우 : 호스트의 이름 + IP주소 저장
((객체를 만드는 시점에) 실제 정확한 호스트 이름인지는 중요하지 않음)

주소 저장 클래스

☞ **InetAddress** → IP주소(호스트이름) 저장 및 관리 클래스 (Port 번호 관리 못함)

- **InetAddress 주요 메서드 :**

리턴타입	메서드 명	주요 내용
byte[]	byte[] getAddress()	InetAddress 객체가 저장하고 있는 IP 주소를 byte[]로 리턴 (**-128~127** 사이의 값)
String	String getHostAddress()	InetAddress 객체가 저장하고 있는 IP 주소를 문자열로 리턴 (**0~255** 사이의 값)
String	String getHostName()	InetAddress 객체가 저장하고 있는 호스트의 이름을 문자열로 리턴
boolean	isLoopbackAddress()	IP가 루프백(loopback) 주소인지여부 확인
boolean	isMulticastAddress()	IP가 멀티캐스팅 영역의 주소인지여부 확인
boolean	isReachable(int timeout)	**Ping** 명령으로 리턴여부 확인 (cmd 창 : **ping** www.google.com)

1

2

```
C:₩Users₩dhkim>ping www.google.com

Ping www.google.com [216.58.221.228] 32바이트 데이터 사용:
216.58.221.228의 응답: 바이트=32 시간=58ms TTL=106
216.58.221.228의 응답: 바이트=32 시간=58ms TTL=106
216.58.221.228의 응답: 바이트=32 시간=58ms TTL=106
216.58.221.228의 응답: 바이트=32 시간=58ms TTL=106

216.58.221.228에 대한 Ping 통계:
    패킷: 보냄 = 4, 받음 = 4, 손실 = 0 (0% 손실),
왕복 시간(밀리초):
    최소 = 58ms, 최대 = 58ms, 평균 = 58ms
```

주소 저장 클래스

☞ **InetAddress** → IP주소(호스트이름) 저장 및 관리 클래스

2 다수의 서버를 이용하여 분산처리를 수행하기 때문에 실행 시마다 IP가 다르게 나올 수 있음

예시

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

  //#1. InetAddress 객체 생성
  //@1-1. 원격지 IP
  InetAddress ia1 = InetAddress.getByName("www.google.com");
  InetAddress ia2 = InetAddress.getByAddress(new byte[] {(byte)172,(byte)217,(byte)161,(byte)132});
  InetAddress ia3 = InetAddress.getByAddress("www.google.com", new byte[] {(byte)172,(byte)217,(byte)161,(byte)132});
  System.out.println(ia1);
  System.out.println(ia2);
  System.out.println(ia3);

  //@1-2. 로컬/루프백 IP
  InetAddress ia4 = InetAddress.getLocalHost();
  InetAddress ia5 = InetAddress.getLoopbackAddress(); //localhost/127.0.0.1
  System.out.println(ia4);
  System.out.println(ia5);

  //@1-3. 하나의 호스트가 여러 개의 IP를 가지고 있는 경우
  InetAddress[] ia6 = InetAddress.getAllByName("www.naver.com");
  System.out.println(Arrays.toString(ia6));
  System.out.println();
```

1 2 3

```
www.google.com/172.217.161.132
/172.217.161.132
www.google.com/172.217.161.132
DHKIM/192.168.123.159
localhost/127.0.0.1
[www.naver.com/210.89.164.90, www.naver.com/210.89.160.88]
```

주소 저장 클래스

☞ **InetAddress** → IP주소(호스트이름) 저장 및 관리 클래스

예시

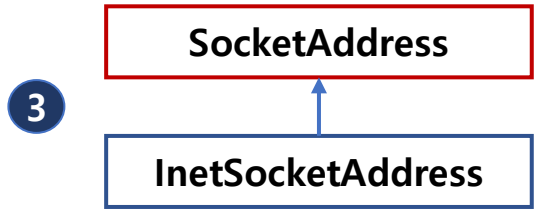
```
//#2. InetAddress 메서드

//@2-1. 호스트와 IP 알아내기
byte[] address = ia1.getAddress();
System.out.println(Arrays.toString(address));
System.out.println(ia1.getHostAddress());
System.out.println(ia1.getHostName());

//@2-2. 저장 주소의 특징 알아내기
System.out.println(ia1.isReachable(1000));
System.out.println(ia1.isLoopbackAddress());
System.out.println(ia1.isMulticastAddress());
System.out.println(InetAddress.getByAddress(new byte[] {127,0,0,1}).isLoopbackAddress());
System.out.println(InetAddress.getByAddress(new byte[] {(byte)225,(byte)225,(byte)225,(byte)225}).isMulticastAddress());

}
```

1

2

```
[-84, -39, -95, -124]
172.217.161.132
www.google.com
true
false
false
true
true
```

주소 저장 클래스



3

1 ☞ **SocketAddress** → IP주소(호스트이름) + Port 번호 관리하는 추상 클래스 = InetAddress+Port번호

2 **- SocketAddress 객체 생성 : 하위 클래스인 InetSocketAddress의 생성자로 객체 생성**

4

생성자	주요 내용
InetSocketAddress(int port)	IP주소 없이 내부의 포트 정보만 지정
InetSocketAddress(String hostname, int port)	매개변수의 호스트의 이름에 해당하는 IP와 포트번호를 지정
InetSocketAddress(InetAddress addr, int port)	매개변수의 InetAddress(IP 정보)와 포트 번호를 지정

- InetSocketAddress 주요 메서드

5

리턴타입	메서드 명	주요 내용
InetAddress	getAddress()	저장 IP 주소를 InetAddress 타입으로 리턴
int	getPort()	포트 번호을 정수형으로 리턴
String	getHostName()	호스트의 이름을 문자열로 리턴

주소 저장 클래스

☞ **SocketAddress** → **IP주소(호스트이름) + Port 번호 관리하는 추상 클래스 = InetAddress+Port번호**

예시

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

  //#1. InetAddress 객체 생성
  InetAddress ia = InetAddress.getByName("www.google.com");
  int port = 10000;

  InetSocketAddress isa1 = new InetSocketAddress(port);
  InetSocketAddress isa2 = new InetSocketAddress("www.google.com", port);
  InetSocketAddress isa3 = new InetSocketAddress(ia, port);

  System.out.println(isa1);
  System.out.println(isa2);
  System.out.println(isa3);
  System.out.println();

  //#2. InetAddress 메서드
  System.out.println(isa2.getAddress()); //InetAddress
  System.out.println(isa2.getHostName());
  System.out.println(isa2.getPort()); //10000

}
```

1

2

```
0.0.0.0/0.0.0.0:10000
www.google.com/172.217.174.196:10000
www.google.com/172.217.174.196:10000

www.google.com/172.217.174.196
www.google.com
10000
```

The End

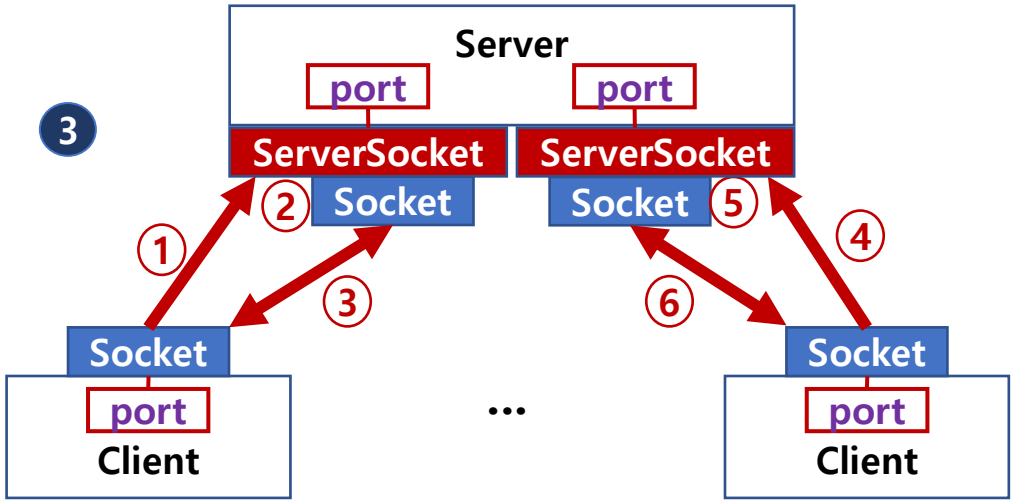
TCP(Transmission Control Protocol) 통신

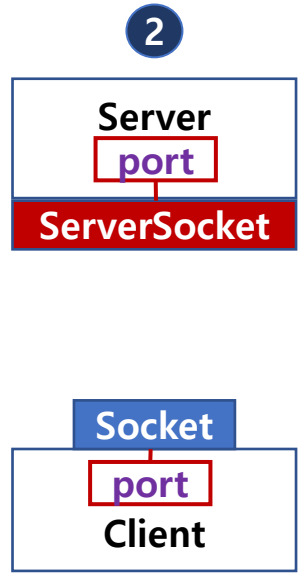
Socket

ServerSocket

TCP(Transmission Control Protocol) 통신

1 ☞ **TCP 통신 매커니즘** → 신뢰성이 높은 연결지향성 프로토콜





4

번호	동작
1, 4	Client의 Socket으로 Server의 ServerSocket에 접속
2, 5	Server는 각 Client와 통신할 수 있는 Socket 생성
3, 6	Server와 Client의 Socket간의 통신

* 두 호스트 간의 통신을 위해서는 두 호스트를 각각 server와 client로 구성하여 통신
* 다수 Client간의 통신을 위해서는 Server를 공유하여 client-server-client 간 통신

Socket

TCP(Transmission Control Protocol) 통신

☞ **Socket** → TCP 통신에서 두 호스트간 입출력 스트림을 제공(실제 통신 객체)하는 클래스

1 - Socket 객체 생성 : 객체의 생성과 동시에 연결 요청

UnknownHostException과 IOException 처리 필요

2

생성자	동작
Socket()	특정 주소에 연결 없이 통신을 위한 소켓 생성
Socket(String host, int port)	매개변수로 입력되는 원격지 host 주소(문자열)의 port에 연결하는 소켓 생성 (생성과 동시에 연결 요청)
Socket(String host, int port, InetAddress localAddr, int localPort)	매개변수로 입력되는 원격지 host 주소(문자열)의 port에 연결하는 소켓 생성 송신지의 InetAddress와 송신지 port정보를 포함하여 연결 요청
Socket(InetAddress address, int port)	매개변수로 입력되는 원격지 host 주소(InetAddress)의 port에 연결하는 소켓 생성 (생성과 동시에 연결 요청)
Socket(InetAddress address, int port, InetAddress localAddr, int localPort)	매개변수로 입력되는 원격지 host 주소(InetAddress)의 port에 연결하는 소켓 생성 송신지의 InetAddress와 송신지 port정보를 포함하여 연결 요청

TCP(Transmission Control Protocol) 통신

☞ **Socket** → **TCP 통신에서 두 호스트간 입출력 스트림을 제공(실제 통신 객체)하는 클래스**

- Socket 주요 메서드 1

반환타입	메서드 이름	동작
void void	connect(SocketAddress endpoint) connect(SocketAddress endpoint, int timeout)	**연결 정보가 없는 소켓**에 원격지 주소 정보(SocketAddress) 제공하여 timeout 시간동안 연결 요청 수행 (timeout=0인 경우 시간제약 없음) (IOException 처리 필요)
InetAddress int	getInetAddress() getPort()	**원격지 주소 정보**: Socket에 연결된 원격지 InetAddress와 port 리턴
InetAddress int SocketAddress	getLocalAddress() getLocalPort() getLocalSocketAddress()	**로컬 주소 정보**: 로컬의 InetAddress와 port 정보 및 두 가지 정보를 모두 포함한 SocketAddress 리턴
InputStream OutputStream	getInputStream() getOutputStream()	**입출력 스트림**: 원격지와의 통신을 위한 입출력 스트림 리턴
int void int void	getSendBufferSize() setSendBufferSize(int size) getReceiveBufferSize() setReceiveBufferSize(int size)	**송수신 버퍼사이즈**: 송수신 버퍼사이즈의 설정 및 읽기 (default: 65536) (SocketException 처리 필요)

TCP(Transmission Control Protocol) 통신

☞ **Socket** → TCP 통신에서 두 호스트간 입출력 스트림을 제공(실제 통신 객체)하는 클래스

예시

1

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

  //#1. Socket 객체 생성
  Socket socket1 = new Socket();
  Socket socket2 = null;
  Socket socket3 = null;
  Socket socket4 = null;
  Socket socket5 = null;

  try {
    socket2 = new Socket("www.naver.com", 80);
    socket3 = new Socket("www.naver.com", 80, InetAddress.getLocalHost(), 10000);
    socket4 = new Socket(InetAddress.getByName("www.naver.com"), 80);
    socket5 = new Socket(InetAddress.getByName("www.naver.com"), 80, InetAddress.getLocalHost(), 10000);
  }
  catch (UnknownHostException e) {}
  catch (IOException e) {}
```

TCP(Transmission Control Protocol) 통신

☞ **Socket** → TCP 통신에서 두 호스트간 입출력 스트림을 제공(실제 통신 객체)하는 클래스

예시

```
  //#2. Socket 메서드
  //@connect/원격지 주소정보
  System.out.println(socket1.getInetAddress()+":"+socket1.getPort()); //null:0
  try {
    socket1.connect(new InetSocketAddress("www.naver.com",80));
1 } catch (IOException e) {}
  System.out.println(socket1.getInetAddress()+":"+socket1.getPort());
  System.out.println(socket2.getInetAddress()+":"+socket2.getPort());
  System.out.println();

  //@로컬 주소 정보 (로컬주소 정보를 지정하지 않은 경우 + 지정한 경우)
  System.out.println(socket2.getLocalAddress()+":"+socket2.getLocalPort());
  System.out.println(socket2.getLocalSocketAddress());
2 System.out.println(socket3.getLocalAddress()+":"+socket3.getLocalPort());
  System.out.println(socket3.getLocalSocketAddress());
  System.out.println();

  //@send/receive 버퍼 사이즈
  try {
3   System.out.println(socket2.getSendBufferSize() + ", "+ socket2.getReceiveBufferSize()); //65536, 65536
  } catch (SocketException e) {}
}
```

```
null:0
www.naver.com/125.209.222.142:80
www.naver.com/125.209.222.142:80

/192.168.123.159:50901
/192.168.123.159:50901
/192.168.123.159:10000
/192.168.123.159:10000

65536, 65536
```

The End

ServerSocket

TCP(Transmission Control Protocol) 통신

2 **TIP** **Binding vs. Connect**
- **Binding**은 수신 호스트에서 연결요청이 들어온 경우 해당 데이터를 전달할 연결 포트를 지정
- **Connect**는 원격지 특정 주소로의 연결을 수행

☞ **ServerSocket** → TCP 통신에서 Socket으로 부터의 연결요청을 수락하는 서버역할의 클래스

- ServerSocket 객체 생성

IOException 처리 필요

1

생성자	동작
ServerSocket()	특정 포트 바인딩(bind) 없이 단순히 ServerSocket 객체만을 생성 (이후 binding 필요)
ServerSocket(int port)	매개변수로 입력된 port로 바인딩 된 SereverSocket 객체 생성 (이후 ServerSocket으로의 연결요청이 오면 바인딩 된 port로 전달)

- ServerSocket의 주요 메서드

3

반환타입	메서드 이름	동작
void	bind(SocketAddress endpoint)	**바인딩 정보가 없는 서버 소켓**에 바인딩 정보를 제공
boolean	isBound()	**바인딩 여부 확인**: ServerSocket이 바인딩 되어 있는지의 여부를 리턴
void int	setSoTimeout(int timeout) getSoTimeout()	**연결요청 리스닝 시간**: Socket으로 부터의 연결요청을 리스닝(listening)하는 시간의 설정 및 가져오기 (timeout=0 이면 무한 대기)
★ **Socket**	accept()	**연결요청 수락**: 연결요청이 수락 된 후 통신을 위한 Socket 객체 리턴 (연결수락까지 설정된 timeout 시간만큼 **blocking**)

TCP(Transmission Control Protocol) 통신

TIP **Binding vs. Connect**
- **Binding**은 수신 호스트에서 연결요청이 들어온 경우 해당 데이터를 전달할 연결 포트를 지정
- **Connect**는 원격지 특정 주소로의 연결을 수행

☞ **ServerSocket** → TCP 통신에서 Socket으로 부터의 연결요청을 수락하는 서버역할의 클래스

예시

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

  //#1. ServerSocket 객체 생성
  ServerSocket serverSocket1 = null;
  ServerSocket serverSocket2 = null;
  try {
    serverSocket1 = new ServerSocket();
    serverSocket2 = new ServerSocket(20000);
  } catch (IOException e) {}

  //#2. ServerSocket 메서드
  //@bind
  System.out.println(serverSocket1.isBound());//false
  System.out.println(serverSocket2.isBound());//true
  try {
    serverSocket1.bind(new InetSocketAddress("127.0.0.1", 10000));
  } catch (IOException e) {}
  System.out.println(serverSocket1.isBound());//true
  System.out.println(serverSocket2.isBound());//true

  System.out.println();
```

1

2

```
false
true
true
true
```

TCP(Transmission Control Protocol) 통신

☞ **ServerSocket** → TCP 통신에서 Socket으로 부터의 연결요청을 수락하는 서버역할의 클래스

예시

1

```
//@ 사용중인 TCP 포트 확인하기 (cmd: netstat -a)
for(int i=0; i<65536; i++) {
  try { ServerSocket serverSocket = new ServerSocket(i); }
  catch (IOException e) { System.err.println(i+"번째 포트 사용중 ..."); }
}
System.out.println();
```

2

```
명령 프롬프트
C:₩Users₩dhkim>netstat -a

활성 연결

  프로토콜  로컬 주소          외부 주소          상태
  TCP    0.0.0.0:135            DHKIM:0                LISTENING
  TCP    0.0.0.0:445            DHKIM:0                LISTENING
  TCP    0.0.0.0:2457           DHKIM:0                LISTENING
  TCP    0.0.0.0:5040           DHKIM:0                LISTENING
  TCP    0.0.0.0:12380          DHKIM:0                LISTENING
  TCP    0.0.0.0:12457          DHKIM:0                LISTENING
```

3

```
//@ accept() / setSoTimeout() / getSoTimeout()
//Client로 부터 TCP 접속 대기 (일반적으로 쓰레드 사용)
try {
  serverSocket1.setSoTimeout(2000);
} catch (SocketException e) {e.printStackTrace();}
try {
  Socket socket = serverSocket1.accept();
} catch (IOException e) {
  try { System.out.println(serverSocket1.getSoTimeout() + "ms 시간이 지나 listening을 종료"); } catch (IOException e1) {}
}
}
```

```
135번째 포트 사용중 ...
139번째 포트 사용중 ...
445번째 포트 사용중 ...
2457번째 포트 사용중 ...
4441번째 포트 사용중 ...
5040번째 포트 사용중 ...
8380번째 포트 사용중 ...
10000번째 포트 사용중 ...
```

```
57679번째 포트 사용중 ...
57680번째 포트 사용중 ...
63033번째 포트 사용중 ...

2000시간이 지나 listening을 종료
```

The End

TCP 통신 예제

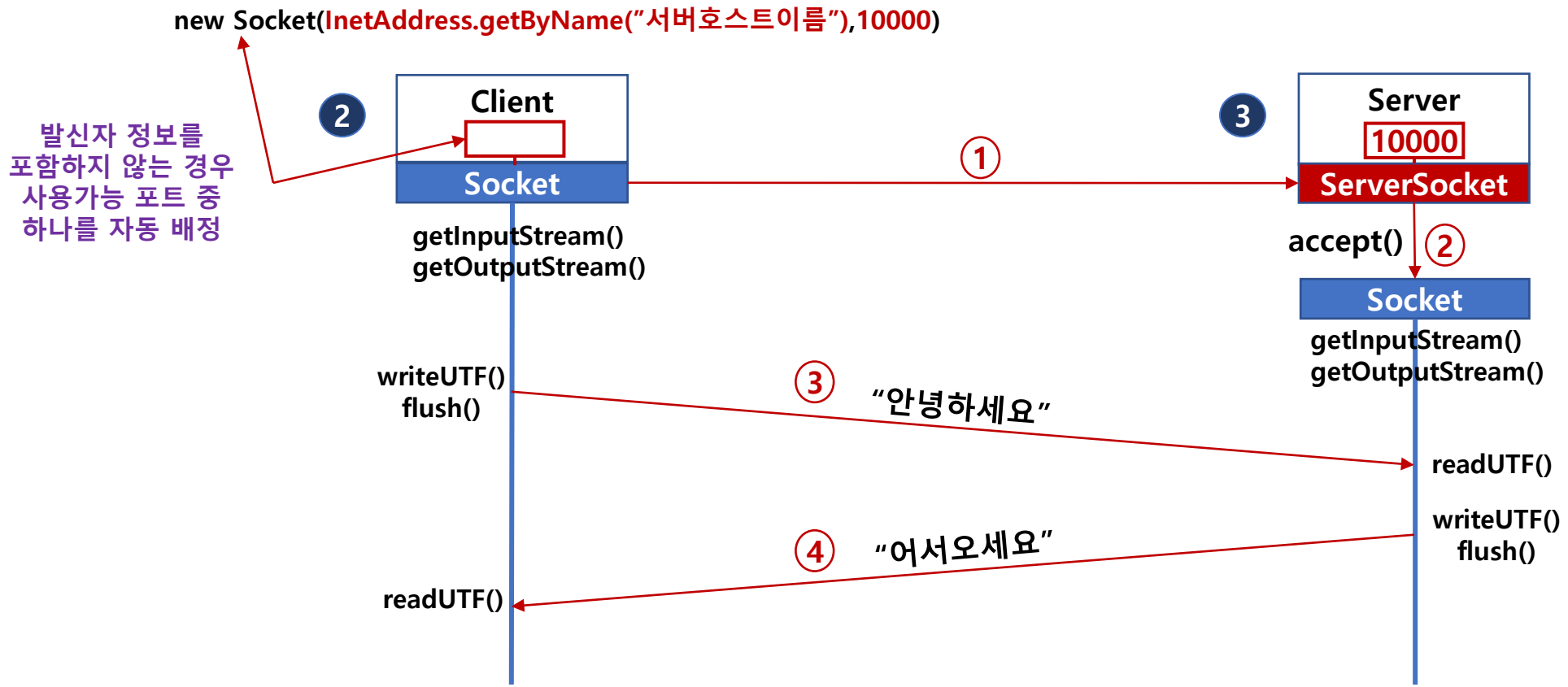
TCP 통신 예제1 : Client와 Server간의 Text 교환

TCP 통신 예제2 : Client와 Server간의 File 전송

TCP(Transmission Control Protocol) 통신

1 ☞ **TCP 통신 예제1** → Client와 Server간의 Text 교환

4 **실행방법**
Step#1. ServerSide 프로그램 실행
Step#2. ClientSide 프로그램 실행



TCP(Transmission Control Protocol) 통신

☞ **TCP 통신 예제1** → Client와 Server간의 Text 교환

CientSide

예시

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

  System.out.println("<<Client>>");
  Socket socket=null;
  try {
1   socket = new Socket(InetAddress.getByName("localhost"),10000);

    System.out.println("Server에 접속완료");
    System.out.println("접속 server 주소:"+socket.getInetAddress()+":"+socket.getPort());

2   DataInputStream dis=new DataInputStream(new BufferedInputStream(socket.getInputStream()));
    DataOutputStream dos=new DataOutputStream(new BufferedOutputStream(socket.getOutputStream()));

3   dos.writeUTF("안녕하세요");
    dos.flush();
    String str=dis.readUTF();

    System.out.println("server: "+str);
  }
  catch (UnknownHostException e) {}
  catch (IOException e) {}

}
```

4

원격지 연결정보(IP, Port) 포함 소켓 생성

원격 스트림생성

데이터 읽기 및 쓰기

```
<<Client>>
Server에 접속완료
접속 server 주소:localhost/127.0.0.1:10000
server: 어서오세요!
```

TCP(Transmission Control Protocol) 통신

☞ **TCP 통신 예제1** → Client와 Server간의 Text 교환

ServerSide

예시

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

  System.out.print("<<Server>>");
  ServerSocket serverSocket = null;
  try { serverSocket = new ServerSocket(10000); }
  catch (IOException e) { System.out.println("해당포트를 사용할 수 없습니다."); System.exit(0); }

  System.out.println(" - Client 접속 대기...");
  try {
1   Socket socket = serverSocket.accept();
    System.out.println("Client 연결 수락");
    System.out.println("접속 client 주소:"+socket.getInetAddress()+":"+socket.getPort());

2   DataInputStream dis=new DataInputStream(new BufferedInputStream(socket.getInputStream()));
    DataOutputStream dos=new DataOutputStream(new BufferedOutputStream(socket.getOutputStream()));

3   String str=dis.readUTF();
    System.out.println("client: " + str);
    dos.writeUTF("어서오세요!");
    dos.flush();

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

4

Client와 연결 수락과 함께 통신할 수 있는 Socket 객체 생성

원격 스트림생성

데이터 읽기 및 쓰기

```
<<Server>> - Client 접속 대기...
```

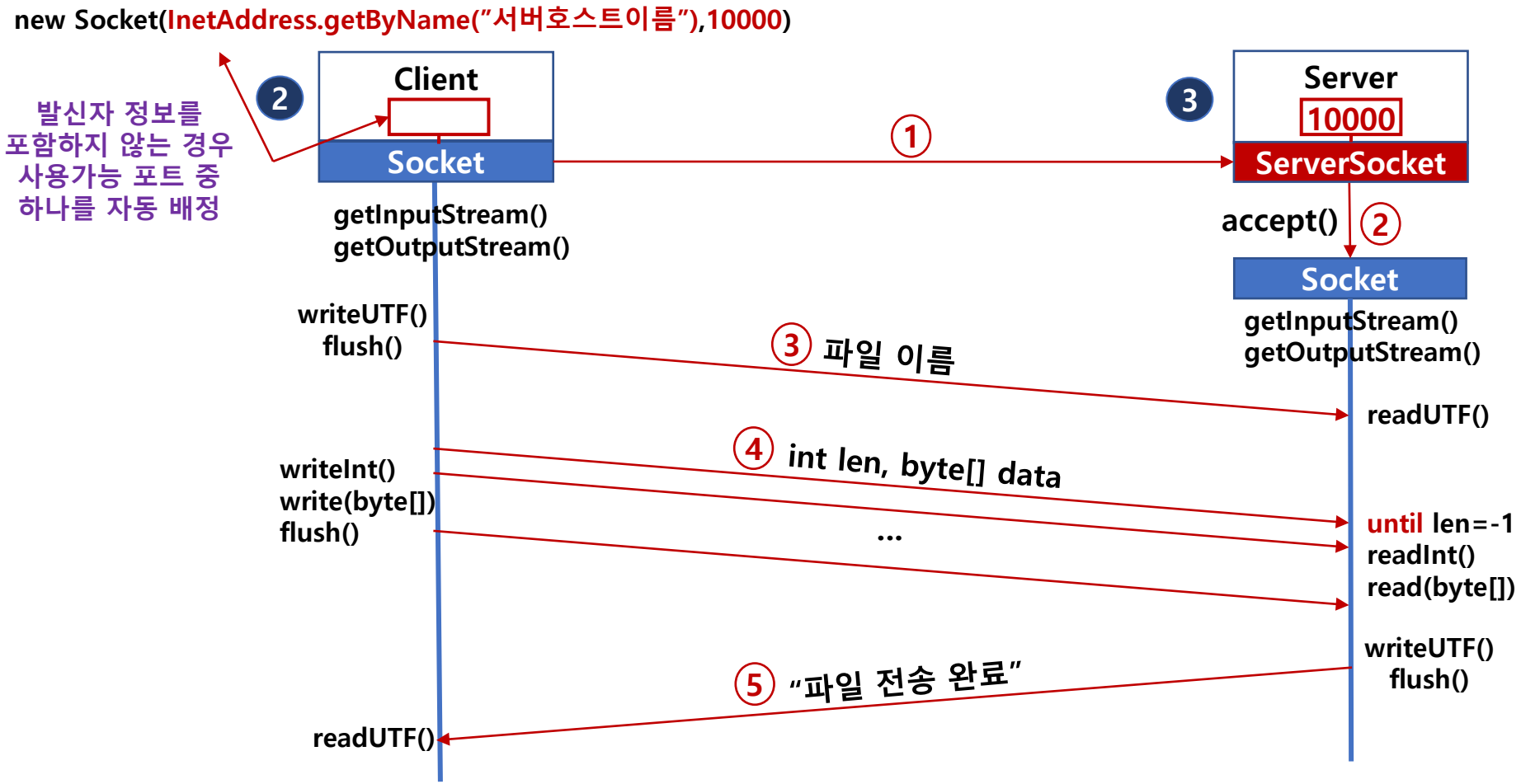
```
<<Server>> - Client 접속 대기...
Client 연결 수락
접속 client 주소:/127.0.0.1:57227
client: 안녕하세요
```

The End

TCP(Transmission Control Protocol) 통신

1 ☞ **TCP 통신 예제2** → Client와 Server간의 File 전송

4 **실행방법**
Step#1. ServerSide 프로그램 실행
Step#2. ClientSide 프로그램 실행



TCP(Transmission Control Protocol) 통신

☞ **TCP 통신 예제2** → Client와 Server간의 File 전송

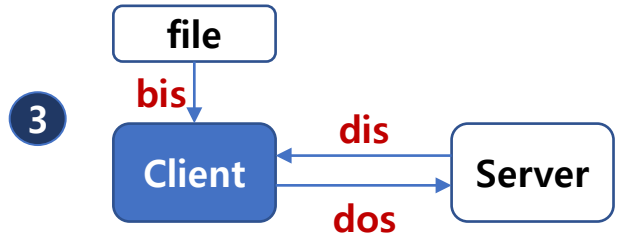
ClientSide(1/2)

예시

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

  System.out.println("<<Client>>");
  Socket socket=null;
  try {
    socket = new Socket(InetAddress.getByName("localhost"),10000);
    System.out.println("Server에 접속완료");
    System.out.println("접속 server 주소:"+socket.getInetAddress()+":"+socket.getPort());

1   DataInputStream dis=new DataInputStream(new BufferedInputStream(socket.getInputStream()));
    DataOutputStream dos=new DataOutputStream(new BufferedOutputStream(socket.getOutputStream()));

2   File file = new File("src/sec02_tcpcommunication/files_client/ImageFileUsingTCP.jpg");
    FileInputStream fis = new FileInputStream(file);
    BufferedInputStream bis = new BufferedInputStream(fis);
    System.out.println("파일전송: "+file.getName());

    dos.writeUTF(file.getName()); //파일이름전송
```

원격 스트림생성

파일읽기 객체생성



TCP(Transmission Control Protocol) 통신

☞ **TCP 통신 예제2** → Client와 Server간의 File 전송

ClientSide(2/2)

예시

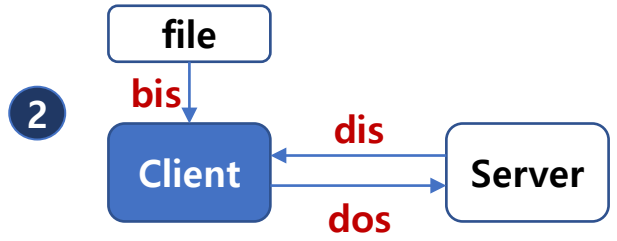
1

```
byte[] data = new byte[2048];
int len;
while((len = bis.read(data))!=-1) {
  dos.writeInt(len); //전송데이터의 길이
  dos.write(data,0,len); //전송데이터
  dos.flush();
}
dos.writeInt(-1); //데이터 전송완료 알림
dos.flush();
```

→ 파일전송

```
  String str = dis.readUTF();
  System.out.println(str);
 }
 catch (UnknownHostException e) {}
 catch (IOException e) {}

}
```

2




```
<<Client>>
Server에 접속완료
접속 server 주소:localhost/127.0.0.1:10000
파일전송: ImageFileUsingTCP.jpg
파일 전송 완료
```

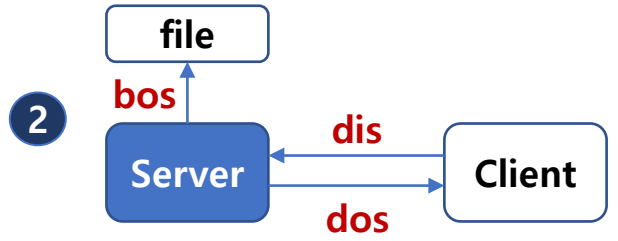
TCP(Transmission Control Protocol) 통신

☞ **TCP 통신 예제2** → Client와 Server간의 File 전송

ServerSide(1/2)

예시

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

  System.out.print("<<Server>>");
  ServerSocket serverSocket = null;
  try {
    serverSocket = new ServerSocket(10000);
  } catch (IOException e) {
    System.out.println("해당포트를 사용할 수 없습니다.");
    System.exit(0);
  }

  System.out.println(" - Client 접속 대기...");
  try {
    Socket socket = serverSocket.accept();

    System.out.println("Client 연결 수락");
    System.out.println("접속 client 주소:"+socket.getInetAddress()+":"+socket.getPort());

    DataInputStream dis=new DataInputStream(new BufferedInputStream(socket.getInputStream()));
    DataOutputStream dos=new DataOutputStream(new BufferedOutputStream(socket.getOutputStream()));
```



1

원격 스트림 생성

TCP(Transmission Control Protocol) 통신

☞ **TCP 통신 예제2** → Client와 Server간의 File 전송

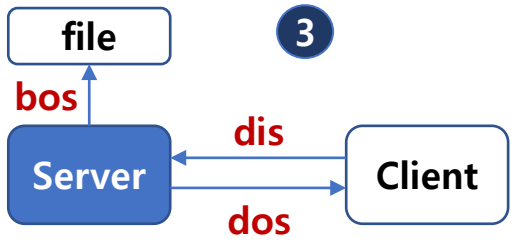
ServerSide(2/2)

예시

```
String receivedFileName=dis.readUTF(); //전송파일이름 읽기
System.out.println("파일수신: "+receivedFileName);

1  File file = new File("src/sec02_tcpcommunication/files_server/"+receivedFileName);
   FileOutputStream fos = new FileOutputStream(file);
   BufferedOutputStream bos = new BufferedOutputStream(fos);

2  byte[] data = new byte[2048];
   int len;
   while((len=dis.readInt())!=-1) {
     dis.read(data,0,len);
     bos.write(data,0,len);
     bos.flush();
   }

   System.out.println("파일 수신 완료");
   dos.writeUTF("파일 전송 완료");
   dos.flush();

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

파일쓰기 객체생성

파일 수신




```
<<Server>> - Client 접속 대기...
```

```
<<Server>> - Client 접속 대기...
Client 연결 수락
접속 client 주소:/127.0.0.1:54300
파일수신: ImageFileUsingTCP.jpg
파일 수신 완료
```

The End

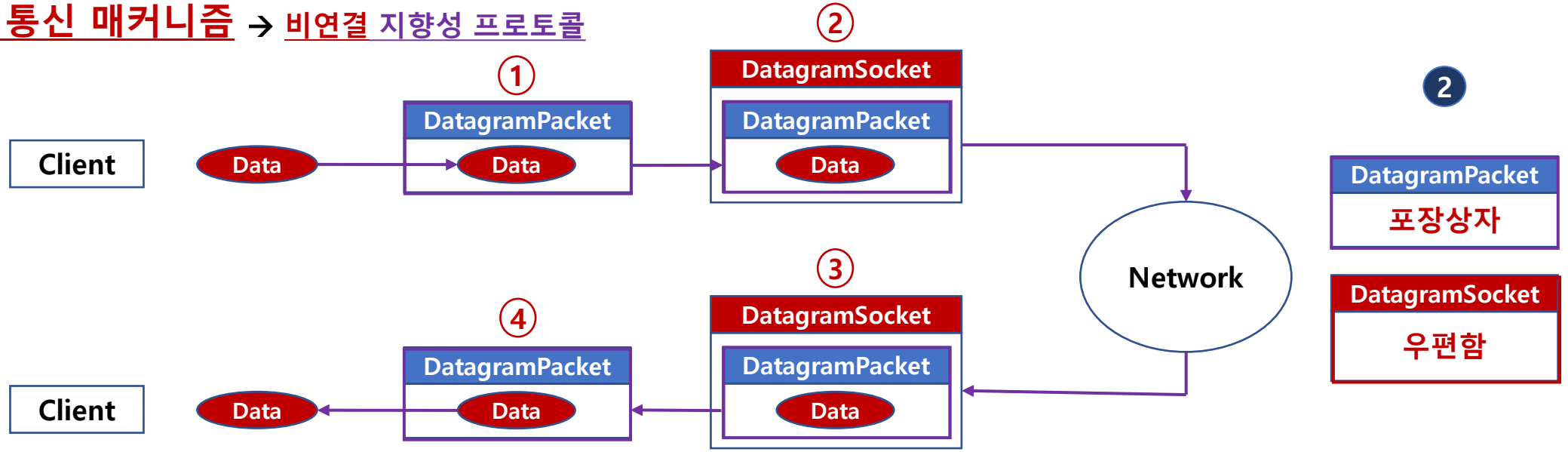
UDP(User Datagram Protocol) 통신

DatagramPacket

DatagramSocket

UDP(User Datagram Protocol) 통신

1 ☞ **UDP 통신 매커니즘** → 비연결 지향성 프로토콜



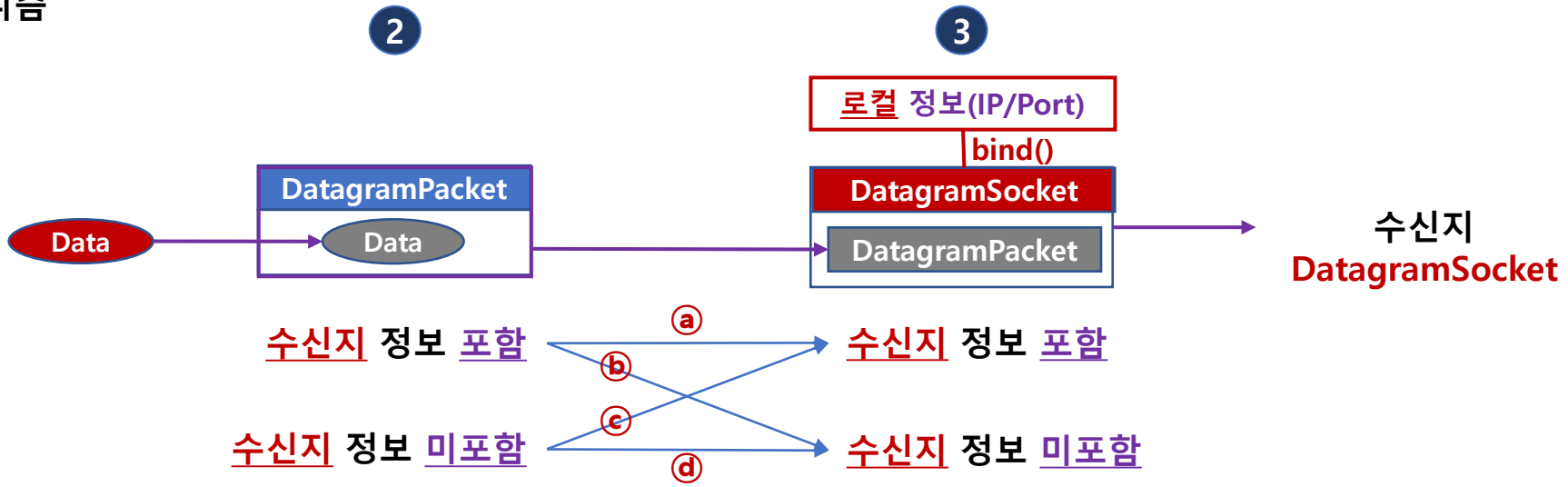
번호	동작
1	보낼 데이터를 **DatagramPacket**에 담기 (최대 65508 bytes=65536-20(IP header)-8(UDP header)
2	DatagreamPacket을 **DatagramSocket**으로 전송 (DatagramSocket의 **send()**)
3	DatagramSocket으로 부터 DatagramPcket 꺼내기 (DatagramSocket의 **receive()**) (수신시에는 비어있는 DatagramPacket을 하나 만들어 옮겨 담는 개념 : 수신한 DatagramSocket.receive(비어있는 DatagreamPacket)
4	DatagramPacket으로 부터 Data 꺼내기
* 보내고자 하는 데이터의 크기가 65508보다 큰 경우 여러 개의 DatagramPacket으로 나누어 전송	

4

UDP(User Datagram Protocol) 통신

☞ **UDP 통신 매커니즘** → 비연결 지향성 프로토콜

1 - 송신 매커니즘



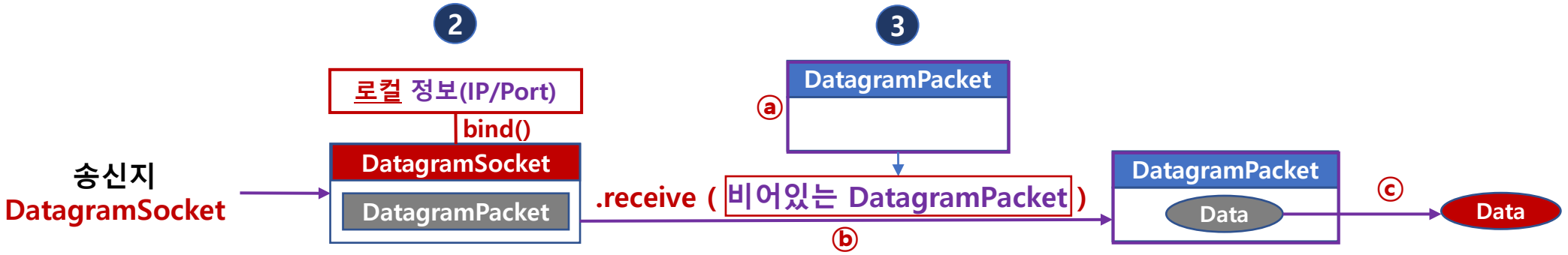
5

번호	동작
ⓐ	DatagramPacket 및 DatagramSocket에 연결된 수신지 주소로 전송 (이때 반드시 DatagramPacket의 주소 = DatagramSocket 연결주소) : 불일치시 IllegalArgumentException
ⓑ	DatagreamPacket에 포함된 주소로 전송 (**비연결 지향성**) (DatagramSocket의 **send()**)
ⓒ	DatagramSocket에 연결(connect)되어 있는 수신지 주소로 전송 (원격지 주소 연결: DatagramSocket의 **connect()** + DatagramSocket의 **send()**)
ⓓ	전송 불가 (NullPointerException)

UDP(User Datagram Protocol) 통신

☞ **UDP 통신 매커니즘** → 비연결 지향성 프로토콜

1 - 수신 매커니즘



4

번호	동작
ⓐ	비어있는 DatagramPacket 생성 (최대 **65508** bytes=65536-20(IP header)-8(UDP header)
ⓑ	수신한 DatagramPacket이 비어있는 DatagramPacket으로 복사 (DatagramSocket의 **receive(비어있는 DatagramPacket)**)
ⓒ	수신 DatagramPacket으로 부터 데이터 추출 (DatagramPacket의 **getData()**, **getOffset()**, **getLength()**)

The End

UDP(User Datagram Protocol) 통신

☞ **DatagramPacket** → UDP 통신에서 보내고자 하는 데이터를 포장(소포박스개념)하는 클래스 (하나의 패킷당 최대 65,508 byte)

1

- **DatagramPacket 객체 생성: byte[]의 데이터를 포함**

2

생성자	동작
DatagramPacket(byte[] buf, int length)	주소 없이 단순히 **데이터만 저장**하는 패킷 (일반적으로 수신측에서 **수신한 패킷을 저장**하는데 사용) buf[]: 데이터 바이트 배열, offset: 시작 오프셋, length: 데이터의 길이
DatagramPacket(byte[] buf, int offset, int length)	
DatagramPacket(byte[] buf, int length, InetAddress address, int port)	데이터의 정보에 추가하여 **수신측 주소 정보**를 **InetAddress**와 **port** 번호로 추가한 패킷 **(소포 포장지에 주소를 써 놓는 개념)** buf[]: 데이터 바이트 배열, offset: 시작 오프셋, length: 데이터의 길이
DatagramPacket(byte[] buf, int offset, int length, InetAddress address, int port)	
DatagramPacket(byte[] buf, int length, SocketAddress address)	데이터의 정보에 추가하여 **수신측 주소 정보**를 **SocketAddress (InetAddress+Port)**로 추가한 패킷 **(소포 포장지에 주소를 써 놓는 개념)** buf[]: 데이터 바이트 배열, offset: 시작 오프셋, length: 데이터의 길이
DatagramPacket(byte[] buf, int offset, int length, SocketAddress address)	

UDP(User Datagram Protocol) 통신

☞ **DatagramPacket** → UDP 통신에서 보내고자 하는 데이터를 포장(소포박스개념)하는 클래스 (하나의 패킷당 최대 65,508 byte)

1 - DatagramPacket 주요 메서드

2

반환타입	메서드 이름	동작
InetAddress void	getAddress() setAddress(InetAddress iaddr)	**원격지 IP주소 읽기 및 설정:** InetAddress 타입으로 설정된 수신지 주소 읽기 및 쓰기 (수신지 주소가 없는 경우 getAddress()는 null을 리턴)
int void	getPort() setPort(int iport)	**원격지 Port 읽기 및 설정:** int 타입으로 설정된 수신지 포트 읽기 및 쓰기 (수신지 포트가 없는 경우 getPort()는 -1을 리턴)
SocketAddress void	getSocketAddress() setSocketAddress(SocketAddress address)	**원격지 SocketAddress 읽기 및 설정:** SocketAddress 타입으로 설정된 수신지 IP와 Port 읽기 및 쓰기 (수신지 SocketAddress가 없는 경우 IllegalArgumentException)
byte[] void void	getData() setData (byte[] buf) setData (byte[] buf, int offset, int length)	**패킷에 포함되는 데이터 읽기 및 설정**: byte[] 타입으로 패킷에 포함된 데이터를 읽기 및 쓰기, 이때 **getData()의 리턴값은 offset, length와 관계 없이 buf값 리턴** (패킷당 전송가능한 최대 raw 데이터=65,508 bytes)
int int void	getLength() getOffset() setLength(int length)	**데이터 길이 읽기 및 설정 및 오프셋 읽기**: 패킷에 포함되는 데이터의 길이 읽기 및 쓰기와 오프셋 읽기

UDP(User Datagram Protocol) 통신

☞ **DatagramPacket** → **UDP 통신에서 보내고자 하는 데이터를 포장(소포박스개념)하는 클래스 (하나의 패킷당 최대 65,508 byte)**

예시

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

  //#0. 송신데이터: 최대바이트수 65508 byte (64Kbyte) : ipv4 (65536-20(IP header)-8(UDP header)=65505)
  byte[] buf = "UDP-데이터그램패킷".getBytes();

  //#1. DatagramPacket 객체 생성
  //@1-1. 수신지 주소 미포함 DatagramPacket
1 DatagramPacket dp1 = new DatagramPacket(buf, buf.length); //UDP-데이터그램패킷
  DatagramPacket dp2 = new DatagramPacket(buf, 4, buf.length-4); //데이터그램패킷 (단, getData()는 전체 포함)

  //@1-2. 수신측 주소가 InetAddress와 port로 포함되어 있는 경우
  DatagramPacket dp3=null, dp4=null;
  try {
    dp3 = new DatagramPacket(buf, buf.length, InetAddress.getByName("localhost"), 10000); //data=UDP-데이터그램패킷
    dp4 = new DatagramPacket(buf, 4, buf.length-4, InetAddress.getByName("localhost"), 10000); //data=데이터그램패킷
2 } catch (UnknownHostException e) {}

  //@1-3. 수신측 주소가 SocketAddress로 포함되어 있는 경우
  DatagramPacket dp5=null, dp6=null;
  dp5 = new DatagramPacket(buf, buf.length, new InetSocketAddress("localhost", 10000));
  dp6 = new DatagramPacket(buf, 4, buf.length-4, new InetSocketAddress("localhost", 10000));
```

UDP(User Datagram Protocol) 통신

☞ **DatagramPacket** → **UDP 통신에서 보내고자 하는 데이터를 포장(소포박스개념)하는 클래스 (하나의 패킷당 최대 65,508 byte)**

예시

```
//#2. DatagramPacket 주요 메서드

//@2-1. getAddress(), getPort(), getSocketAddress()
System.out.println("원격지 IP    : " + dp1.getAddress()); //null
System.out.println("원격지 Port : " + dp1.getPort()); //-1
//System.out.println("원격지 SocketAddress : " + dp1.getSocketAddress()); //IllegalArgumentException 발생
System.out.println("원격지 IP    : " + dp3.getAddresst()); //localhost/127.0.0.1
System.out.println("원격지 Port : " + dp3.gePort()); //10000
System.out.println("원격지 SocketAddress : " + dp3.getSocketAddress()); //localhost/127.0.0.1:10000
System.out.println();

//@2-2. getData(). setData(), getOffset(), getLength()
System.out.println("포함데이터 : "+new String(dp1.getData())); //UDP-데이터그램패킷
System.out.println("포함데이터 : "+new String(dp2.getData())); //UDP-데이터그램패킷 (len와 상관없이 항상 전체데이터 리턴)
System.out.println("포함데이터 : "+new String(dp2.getData(),dp2.getOffset(),dp2.getLength())); //데이터그램패킷

dp1.setData("안녕하세요".getBytes());
System.out.println("포함데이터 : "+new String(dp1.getData())); //안녕하세요

}
```

1

2

```
원격지 IP    : null
원격지 Port : -1
원격지 IP    : localhost/127.0.0.1
원격지 Port : 10000
원격지 SocketAddress : localhost/127.0.0.1:10000

포함데이터 : UDP-데이터그램패킷
포함데이터 : UDP-데이터그램패킷
포함데이터 : 데이터그램패킷
포함데이터 : 안녕하세요
```

The End

UDP(User Datagram Protocol) 통신

☞ **DatagramSocket** → UDP 통신에서 DatagramPacket을 네트워크를 통해 전송하거나 수신하는 소켓(우편함 개념) 클래스

1

- **DatagramSocket 객체 생성: 소켓에 도착한 DatagramPacket을 바인딩 된 포트로 전달하는 기능**

4 **SocketException 처리 필요**

생성자	동작
2 DatagramSocket()	기본생성자로 객체를 생성하는 경우 객체 생성 호스트에 **가용한 Port에 바인딩** 된 DatagramSocket() 객체 생성
DatagramSocket(int port)	입력 매개변수로 전달된 **포트로 바인딩**된 DatagramSocket() 객체 생성 (DatagramSocket에 DatagramPacket이 도착하면 바인딩된 Port로 전달)
3 DatagramSocket(int port, InetAddress laddr)	매개변수로 전달된 **InetAddress**의 **port**에 바인딩 된 DatagramSocket 객체 생성
DatagramSocket(SocketAddress bindaddr)	매개변수로 전달된 SocketAddress에 바인딩 된 DatagramSocket 객체 생성

UDP(User Datagram Protocol) 통신

☞ **DatagramSocket** → UDP 통신에서 DatagramPacket을 네트워크를 통해 전송하거나 수신하는 소켓(우편함 개념) 클래스

1 - DatagramSocket 주요 메서드

2

반환타입	메서드 이름	동작
void void	send(DatagramPacket p) receive(DatagramPacket p)	**DatagramPacket의 전송과 수신: (IOException)** 데이터를 포함한 DatagramPacket을 전송하고 수신된 DatagramPacket을 비어 있는 DatagramPacket으로 담아서 수신
void void void	connect(InetAddress address, int port) connect(SocketAddress addr) disconnect()	**원격지 DatagramSocket 주소로 연결 및 연결해제:** 매개변수로 입력된 주소로 실제 연결 수행 및 연결 해제
InetAddress int	getInetAddress() getPort()	**원격지 주소 정보:** 원격지의 InetAddress와 Port값 읽기 (단, connect()를 통해 연결이 된 경우 유효한 값 리턴)
InetAddress int SocketAddress	getLocalAddress() getLocalPort() getLocalSocketAddress()	**로컬 주소 정보:** 바인딩 된 로컬 호스트의 InetAddress, Port 또는 이 둘의 정보를 포함한 SocketAddress의 정보를 리턴
int int void void	int getReceiveBufferSize() int getSendBufferSize() void setReceiveBufferSize(int size) void setSendBufferSize(int size)	**송수신 버퍼크기의 읽기 및 설정**: 송수신시 사용되는 버퍼 크기의 읽기 및 설정 (SocketException) (미설정시 송수신 버퍼크기는 65,536 byte)
int void	getSoTimeout() setSoTimeout(int timeout)	**수신 리스닝 시간**: DatagramPacket의 수신을 기다리는 시간 읽기 및 설정 (timeout=0 이면 무한 대기) (시간 완료시 SocketTimeoutException: 발생)

UDP(User Datagram Protocol) 통신

☞ **DatagramSocket** → UDP 통신에서 DatagramPacket을 네트워크를 통해 전송하거나 수신하는 소켓(우편함 개념) 클래스

예시

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

  //#1. DatagramSocket 객체 생성
  DatagramSocket ds1=null, ds2=null, ds3=null, ds4=null;
  try {
    //@1-1. port 미지정 / port만 지정
    ds1 = new DatagramSocket(); //현재 호스트의 비워져 있는 포트로 자동 바인딩
    ds2 = new DatagramSocket(10000); //10000 포트로 바인딩

    //@1-2. 바인딩 정보(IP, Port) 포함
    ds3 = new DatagramSocket(10001, InetAddress.getByName("localhost"));
    ds4 = new DatagramSocket(new InetSocketAddress("localhost", 10002));

  } catch (SocketException | UnknownHostException e) {e.printStackTrace();}

  //#2. DatagramSocket 메서드
  //@2-1. 소켓의 바인딩 주소 정보
  System.out.println("ds1 바인딩 정보 : "+ds1.getLocalAddress()+":"+ds1.getLocalPort());
  System.out.println("ds2 바인딩 정보 : "+ds2.getLocalAddress()+":"+ds2.getLocalPort());
  System.out.println("ds3 바인딩 정보 : "+ds3.getLocalAddress()+":"+ds3.getLocalPort());
  System.out.println("ds4 바인딩 정보 : "+ds4.getLocalAddress()+":"+ds4.getLocalPort());
  System.out.println();
```

1

2

```
ds1 바인딩 정보 : 0.0.0.0/0.0.0.0:58895
ds2 바인딩 정보 : 0.0.0.0/0.0.0.0:10000
ds3 바인딩 정보 : /127.0.0.1:10001
ds4 바인딩 정보 : /127.0.0.1:10002
```

UDP(User Datagram Protocol) 통신

☞ **DatagramSocket** → **UDP 통신에서 DatagramPacket을 네트워크를 통해 전송하거나 수신하는 소켓(우편함 개념) 클래스**

예시

1

```
//@2-2. 원격지 주소 정보 (connect()된 경우에만 연결된 원격지 주소 정보)
System.out.println("원격지 주소 정보: "+ds4.getInetAddress()+":"+ds4.getPort());

try {
  ds4.connect(new InetSocketAddress("localhost", 10003));
} catch (SocketException e2) {}

System.out.println("원격지 주소 정보: "+ds4.getInetAddress()+":"+ds4.getPort());
ds4.disconnect();
System.out.println();
```

```
원격지 주소 정보: null:-1
원격지 주소 정보: localhost/127.0.0.1:10003
```

2

```
//@2-3. send(), connect(), disconnect();

//@2-3-0. 전송패킷 2개 (수신지 주소가 없는 패킷 + 수신지 주소가 있는 패킷)
byte[] data1 = "수신지 주소가 없는 데이터그램 패킷".getBytes();
byte[] data2 = "수신지 주소가 있는 데이터그램 패킷".getBytes();
DatagramPacket dp1 = new DatagramPacket(data1, data1.length);
DatagramPacket dp2 = new DatagramPacket(data2, data2.length, new InetSocketAddress("localhost", 10002));
```

UDP(User Datagram Protocol) 통신

☞ **DatagramSocket** → **UDP 통신에서 DatagramPacket을 네트워크를 통해 전송하거나 수신하는 소켓(우편함 개념) 클래스**

예시

```
try {
  //@2-3-1. 수신지 주소가 없는 패킷 전송 = connect() + send() + disconnect();
  //ds1.send(dp1); //불가능: 소켓이 connect된 곳 없음 + 패킷에 수신지 주소 없음 (NullPointerException)
1 //ds2.send(dp1); //불가능: 소켓이 connect된 곳 없음 + 패킷에 수신지 주소 없음 (NullPointerException)
  //ds3.send(dp1); //불가능: 소켓이 connect된 곳 없음 + 패킷에 수신지 주소 없음 (NullPointerException)
  ds1.connect(new InetSocketAddress("localhost", 10002));
  ds2.connect(new InetSocketAddress("localhost", 10002));
  ds3.connect(new InetSocketAddress("localhost", 10002));
  ds1.send(dp1); //가능 : 소켓이 connect된 곳 있음 + 패킷에 수신지 주소 없음
  ds2.send(dp1); //가능 : 소켓이 connect된 곳 있음 + 패킷에 수신지 주소 없음
2 ds3.send(dp1); //가능 : 소켓이 connect된 곳 있음 + 패킷에 수신지 주소 없음
  ds1.disconnect();
  ds2.disconnect();
  ds3.disconnect();
  //@2-3-2. 수신지 주소가 있는 패킷 전송 = send();
  ds1.send(dp2); //가능 : 소켓이 connect된 곳 없음 + 패킷에 수신지 주소 있음
3 ds2.send(dp2); //가능 : 소켓이 connect된 곳 없음 + 패킷에 수신지 주소 있음
  ds3.send(dp2); //가능 : 소켓이 connect된 곳 없음 + 패킷에 수신지 주소 있음
  //@2-3-2. connect 된 소켓이용 + 수신지 주소가 있는 패킷 전송 : 반드시 주소 일치;
  ds3.connect(new InetSocketAddress("localhost", 10002));
4 ds3.send(dp2); //소켓이 connect()된 경우 패킷 연결된 주소와 패킷의 수신지가 같아야 함( IllegalArgumentException )
  ds3.disconnect();
} catch (IOException e2) {}
```

UDP(User Datagram Protocol) 통신

☞ **DatagramSocket** → UDP 통신에서 DatagramPacket을 네트워크를 통해 전송하거나 수신하는 소켓(우편함 개념) 클래스

예시

```
//@2-4. receive() : 데이터수신
byte[] receivedData = new byte[65508];
DatagramPacket dp = new DatagramPacket(receivedData, receivedData.length);
try {
  for(int i=0; i<7; i++) {
    ds4.receive(dp);
    System.out.println("송신자 정보 "+dp.getAddress()+":"+dp.getPort()+" -> "+new String(dp.getData()).trim());
  }
} catch (IOException e2) {}
System.out.println();

//@2-5. 송신 및 수신 버퍼 크기
try {
  System.out.println("송신버퍼크기"+ds1.getSendBufferSize()+", 수신버퍼크기"+ds1.getReceiveBufferSize());
} catch (SocketException e) {}

}
```

1

2

3

```
송신자 정보 /127.0.0.1:65170 -> 수신지 주소가 없는 데이터그램 패킷
송신자 정보 /127.0.0.1:10000 -> 수신지 주소가 없는 데이터그램 패킷
송신자 정보 /127.0.0.1:10001 -> 수신지 주소가 없는 데이터그램 패킷
송신자 정보 /127.0.0.1:65170 -> 수신지 주소가 있는 데이터그램 패킷
송신자 정보 /127.0.0.1:10000 -> 수신지 주소가 있는 데이터그램 패킷
송신자 정보 /127.0.0.1:10001 -> 수신지 주소가 있는 데이터그램 패킷
송신자 정보 /127.0.0.1:10001 -> 수신지 주소가 있는 데이터그램 패킷

송신버퍼크기65536, 수신버퍼크기65536
```

The End

UDP 통신 예제

UDP 통신 예제1 : Client간 Text 교환 (Nonconnected-DatagramSocket)

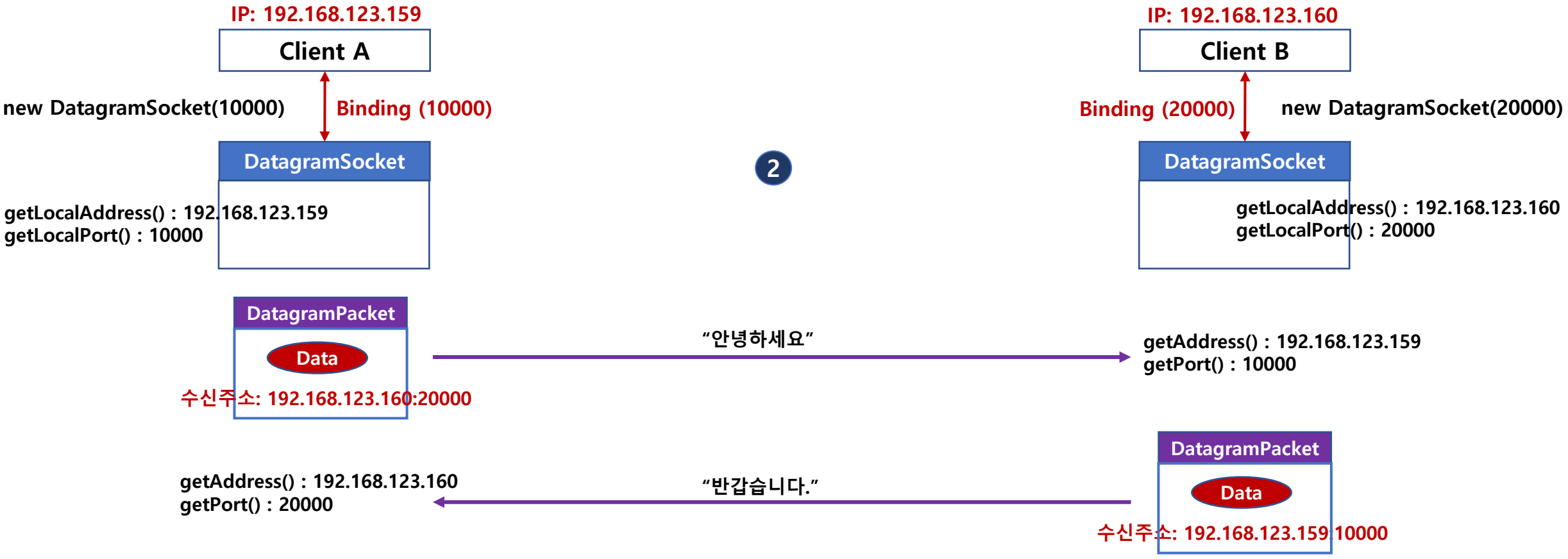
UDP 통신 예제2 : Client간 File 전송 (connected-DatagramSocket)

UDP(User Datagram Protocol) 통신

실행방법
Step#1. Clinet B 프로그램 실행
Step#2. Client A 프로그램 실행

3

1 ☞ **UDP 통신 예제1** → Client간 Text 전송
각 클라이언트는 DatagramSocket을 바인딩하고 연결없이 DatagramPacket()에 수신주소를 넣어 전송



4 비연결성인 경우 데이터의 도착이 전송 순서와 상이할 수 있음 (순서가 중요한 경우에는 TCP 통신이 적절)

UDP(User Datagram Protocol) 통신

☞ **UDP 통신 예제1** → **Client간 Text 전송**

각 클라이언트는 DatagramSocket을 바인딩하고 연결없이 DatagramPacket에 수신주소를 넣어 전송

예시

CientA

```
public static void main(String[] args) {
  System.out.println("<<ClientA>> - Text");
  //#1. DatagramSocket 생성(binding 포함)
1 DatagramSocket ds = null;
  try { ds = new DatagramSocket(10000); } catch (SocketException e) {}

  //#2. 전송데이터 생성 + Datagrampacket 생성(수신지 주소 포함)
2 byte[] sendData = "안녕하세요".getBytes();
  DatagramPacket sendPacket = new DatagramPacket(sendData, sendData.length, new InetSocketAddress("127.0.0.1", 20000));

  //#3. 데이터그램패킷 전송
  try {
    System.out.println("송신데이터 : " + new String(sendPacket.getData()).trim());
    ds.send(sendPacket);
  } catch (IOException e) {}

  //#4. 데이터그램 패킷 수신
3 byte[] receivedData = new byte[65508];
  DatagramPacket receivedPacket = new DatagramPacket(receivedData, receivedData.length);
  try {
    ds.receive(receivedPacket);
  } catch (IOException e) {}
  System.out.println("수신데이터 : " + new String(receivedPacket.getData()).trim());
}
```

10000(port)에 바인딩 된 DatagramSocket 객체 생성

원격지 정보 포함 DatagramPacket 생성

수신을 위한 빈 DatagramPacket 생성

```
<<ClientA>> - Text
송신데이터 : 안녕하세요
수신데이터 : 반갑습니다.
```

```
<<ClientB>> - Text
수신데이터 : 안녕하세요
송신데이터 : 반갑습니다.
```

UDP(User Datagram Protocol) 통신

☞ **UDP 통신 예제1** → **Client간 Text 전송**

각 클라이언트는 DatagramSocket을 바인딩하고 연결없이 DatagramPacket()에 수신주소를 넣어 전송

예시

CientB

```
public static void main(String[] args) {
  System.out.println("<<ClientB>> - Text");

  //#1. DatagramSocket 생성(binding 포함)
1 DatagramSocket ds = null;
  try { ds = new DatagramSocket(20000); } catch (SocketException e) {}

  //#2. 데이터그램 패킷 수신
2 byte[] receivedData = new byte[65508];
  DatagramPacket receivedPacket = new DatagramPacket(receivedData, receivedData.length);
  try { ds.receive(receivedPacket); } catch (IOException e) {}
  System.out.println("수신데이터 : " + new String(receivedPacket.getData()).trim());

  //#3. 전송데이터 생성 + Datagrampacket 생성(수신지 주소 = 수신 패킷의 원격지 정보 추출)
3 byte[] sendData = "반갑습니다.".getBytes();
  DatagramPacket sendPacket = new DatagramPacket(sendData, sendData.length, receivedPacket.getSocketAddress());

  //#4. 데이터그램패킷 전송
  try {
    System.out.println("송신데이터 : " + new String(sendPacket.getData()).trim());
    ds.send(sendPacket);
  } catch (IOException e) {}
}
```

→ 10000(port)에 바인딩 된 DatagramSocket 객체 생성

→ 수신을 위한 빈 DatagramPacket 생성

→ 원격지 정보 포함 DatagramPacket 생성

```
<<ClientA>> - Text
송신데이터 : 안녕하세요
수신데이터 : 반갑습니다.
```

```
<<ClientB>> - Text
수신데이터 : 안녕하세요
송신데이터 : 반갑습니다.
```

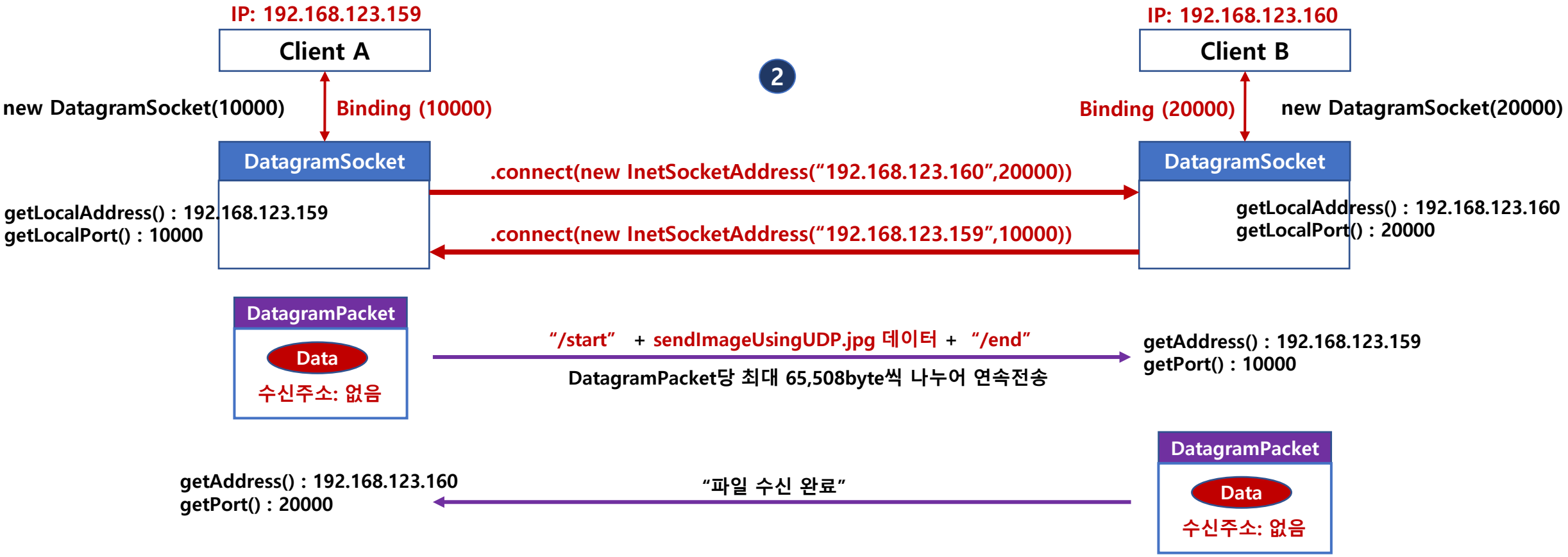
The End

UDP(User Datagram Protocol) 통신

실행방법
Step#1. Clinet B 프로그램 실행
Step#2. Client A 프로그램 실행

3

1 ☞ **UDP 통신 예제2** → Client간 File 전송
두 개의 DatagramSocket을 connect()하고 통신 (DatagramPacket 주소 필요 없음)



4 단, DatagramPacket에 수신주소가 있는 경우 반드시 connect된 Socket의 바인딩값과 동일하여야 함
(불일치시 IllegalArgumentException)

UDP(User Datagram Protocol) 통신

☞ **UDP 통신 예제2** → **Client간 File 전송**

두 개의 DatagramSocket을 connect()하고 통신 (DatagramPacket 주소 필요 없음)

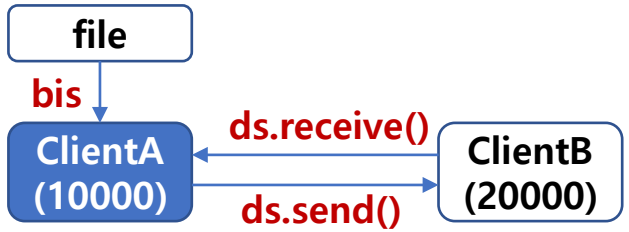
예시

CientA (1/3)

```
public static void main(String[] args) {
  System.out.println("<<ClientA>> - File");

  //#1. DatagramSocket 생성(binding 포함) + 소켓간 연결
  DatagramSocket ds = null;
  try {
    ds = new DatagramSocket(10000);
    ds.connect(new InetSocketAddress("127.0.0.1",20000));
  } catch (SocketException e) {}

  //#2. 파일 로딩
  File file = new File("src/sec02_udpcommunication/files_clientA/ImageFileUsingUDP.jpg");
  BufferedInputStream fis=null;
  try {
    bis = new BufferedInputStream(new FileInputStream(file));
  } catch (FileNotFoundException e1) {}

  //#3.  데이터그램패킷 전송
  DatagramPacket sendPacket=null;
```

1

2



→ **전송파일 입력스트림 연결**

UDP(User Datagram Protocol) 통신

☞ **UDP 통신 예제2** → **Client간 File 전송**

두 개의 DatagramSocket을 connect()하고 통신 (DatagramPacket 주소 필요 없음)

예시

CientA (2/3)

1

```
//@3-1. 파일이름 전송
String fileName = file.getName();
sendPacket = new DatagramPacket(fileName.getBytes(), fileName.length());
try {
  ds.send(sendPacket);
} catch(IOException e) {}
```

파일이름 저장 DatagramPacket 생성 (원격지 주소 미포함)

2

```
//@3-2. 파일전송 시작 사인 전송 (/start)
String startSign = "/start";
sendPacket = new DatagramPacket(startSign.getBytes(), startSign.length());
try {
  ds.send(sendPacket);
} catch(IOException e) {}
```

파일데이터 시작 사인저장 DatagramPacket 생성 (원격지 주소 미포함)

2048크기로 파일 읽어와 데이터패킷 생성 및 전송

3

```
//@3-3. 2048 사이즈로 나누어 실제 파일 데이터 전송
int len;
byte[] filedata = new byte[2048]; //최대는 65508byte이지만 안정적 네트워크 통신을 위해서 2048byte씩 잘라서 전송
try {
  while((len = bis.read(filedata)) != -1) {
    sendPacket = new DatagramPacket(filedata, len);
    ds.send(sendPacket);
}
} catch (IOException e) {}
```

UDP(User Datagram Protocol) 통신

☞ **UDP 통신 예제2** → **Client간 File 전송**

두 개의 DatagramSocket을 connect()하고 통신 (DatagramPacket 주소 필요 없음)

예시

CientA (3/3)

```
//@3-4. 파일전송 종료 사인 전송
String endSign = "/end";
sendPacket = new DatagramPacket(endSign.getBytes(), endSign.length());
try {
  ds.send(sendPacket);
} catch (IOException e1) {}
```

파일데이터 종료 사인저장 DatagramPacket 생성 (원격지 주소 미포함)

1

```
//#4. 데이터그램 패킷 수신
byte[] receivedData = new byte[65508];
DatagramPacket receivedPacket = new DatagramPacket(receivedData, receivedData.length);
try {
  ds.receive(receivedPacket);
} catch (IOException e) {}
System.out.println("수신데이터 : " + new String(receivedPacket.getData()).trim());
```

수신을 위한 빈 DatagramPacket 생성

2

```
<<ClientA>> - File
수신데이터 : 파일 수신 완료
```

```
<<ClientB>> - File
수신 파일이름 : ImageFileUsingUDP.jpg
```

UDP(User Datagram Protocol) 통신

☞ **UDP 통신 예제2** → **Client간 File 전송**

두 개의 DatagramSocket을 connect()하고 통신 (DatagramPacket 주소 필요 없음)

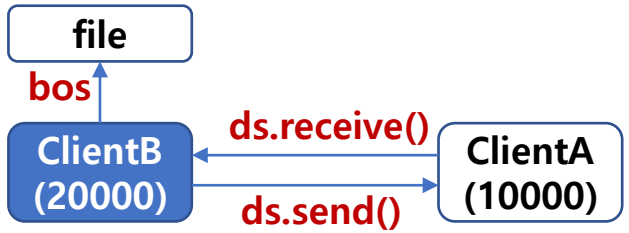
예시

CientB (1/3)

```
public static void main(String[] args) {
  System.out.println("<<ClientB>> - File");

  //#1. DatagramSocket 생성(binding 포함) + 소켓간 연결
  DatagramSocket ds = null;
  try {
    ds = new DatagramSocket(20000);
    ds.connect(new InetSocketAddress("127.0.0.1",10000));
  } catch (SocketException e) {}

  //#2. 데이터그램패킷 수신
  byte[] receivedData = null;
  DatagramPacket receivedPacket = null;

  //@2-1. 파일이름 수신
  receivedData = new byte[65508];
  receivedPacket = new DatagramPacket(receivedData, receivedData.length);

  try {
    ds.receive(receivedPacket);
  } catch(IOException e) {}
```

1

2



수신을 위한 빈 DatagramPacket 생성

UDP(User Datagram Protocol) 통신

☞ **UDP 통신 예제2** → **Client간 File 전송**

두 개의 DatagramSocket을 connect()하고 통신 (DatagramPacket 주소 필요 없음)

예시

CientB (2/3)

1

```
String fileName = new String(receivedPacket.getData(), 0, receivedPacket.getLength());
File file = new File("src/sec02_udpcommunication/files_clientA/"+fileName);
BufferedOutputStream bos=null;
try {
  bos = new BufferedOutputStream(new FileOutputStream(file));
} catch (FileNotFoundException e2) {
  e2.printStackTrace();
}
System.out.println("수신 파일이름 : "+fileName);

//@2-2. 시작태그와 끝태그를 기준으로 파일 수신
String startSign = "/start";
String endSign = "/end";
```

→ **파일이름 수신 후 파일 출력스트림 연결**

2

```
receivedData = new byte[65508];
receivedPacket = new DatagramPacket(receivedData, receivedData.length);
```

→ **수신을 위한 빈 DatagramPacket 생성**

UDP(User Datagram Protocol) 통신

☞ **UDP 통신 예제2** → **Client간 File 전송**

두 개의 DatagramSocket을 connect()하고 통신 (DatagramPacket 주소 필요 없음)

예시

CientB (3/3)

1

```
try {
  ds.receive(receivedPacket);
  if(new String(receivedPacket.getData(),0,receivedPacket.getLength()).equals(startSign)) {
    while(true) {
      ds.receive(receivedPacket);
      if(new String(receivedPacket.getData(),0,receivedPacket.getLength()).equals(endSign))
        break;
      bos.write(receivedPacket.getData(), 0, receivedPacket.getLength());
      bos.flush();
    }
  }
  bos.close();
} catch (IOException e) {}
```

→ **"/start"부터 "/end"까지 반복적으로 데이터 수신 및 파일출력스트림에 기록**

2

```
//#4. 전송데이터 생성 + Datagrampacket 생성(수신지 주소 포함)
byte[] sendData = "파일 수신 완료".getBytes();
DatagramPacket sendPacket = new DatagramPacket(sendData, sendData.length);
```

→ **파일 수신완료 메시지 전송**

3

```
try {
  ds.send(sendPacket);
} catch (IOException e) {}
```

```
<<ClientA>> - File
수신데이터 : 파일 수신 완료
```

```
<<ClientB>> - File
수신 파일이름 : ImageFileUsingUDP.jpg
```

The End

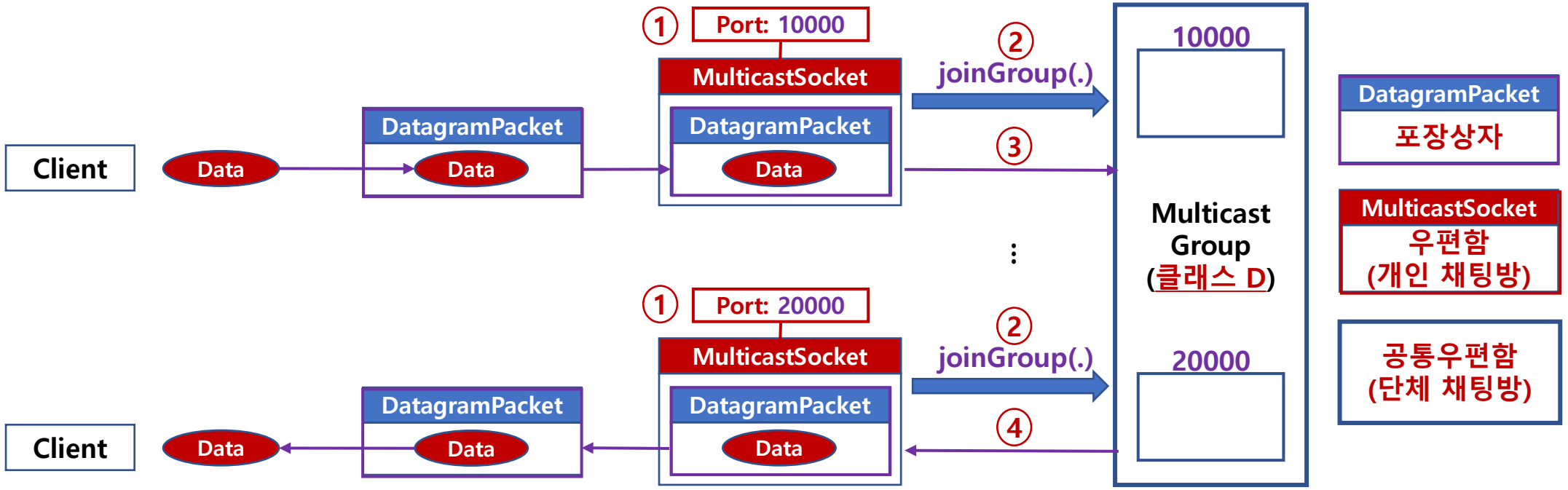
Multicast 통신

MulticastSocket

Multicast 통신

3 클래스 D
224.0.0.0 ~ 239.255.255.255

1 ☞ **Multicast 통신 매커니즘** → 그룹 Join을 통한 데이터그램패킷 전송 및 수신



4

번호	동작
1	각 클라이언트는 Multicast를 지원하는 MulticastSocket 생성 (바인딩 정보 필수 포함 (바인딩 port 미지정시 비어있는 port 자동 바인딩))
2	생성 MulticastSocket을 멀티캐스팅 IP(클래스 D)를 가지는 Multicast Group에 조인 (**joinGroup(InetAddress)**)
3	(송신) 보낼 데이터를 **DatagramPacket**에 담아 MulticastSocket을 통해 보내기 (**send(.)**)
4	(수신) Multicast Group에 패킷이 전달되면 패킷의 수신 포트로 바인딩 된 조인된 모든 Client에게 패킷 전송 (**receive(.)**)

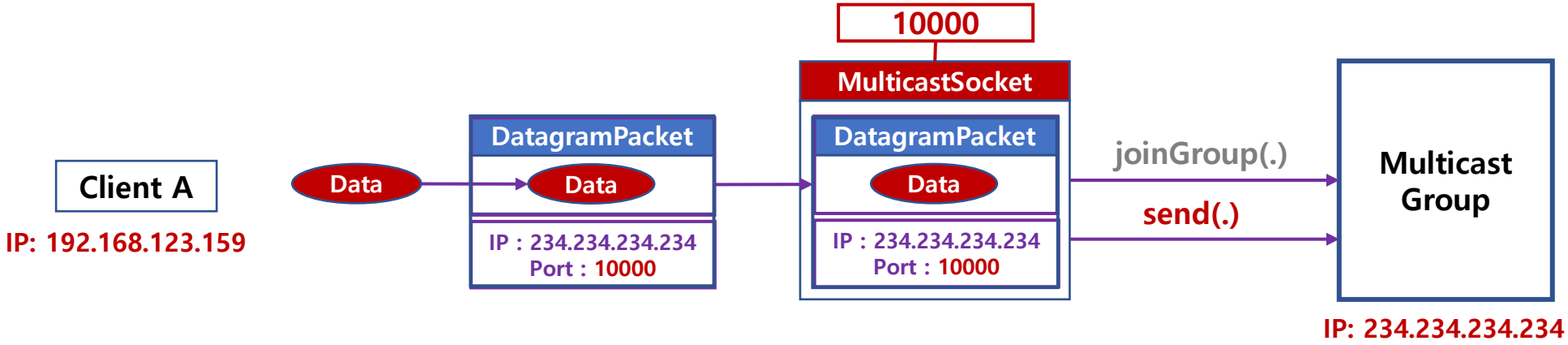
Multicast 통신

클래스 D
224.0.0.0 ~ 239.255.255.255

☞ **Multicast 통신 매커니즘** → 그룹 Join을 통한 UDP 패킷 전송 및 수신

1 - 송신 매커니즘

2



3

send(.) : Multicast Group에 등록(joinGroup())이 안되어 있어도 전송 가능

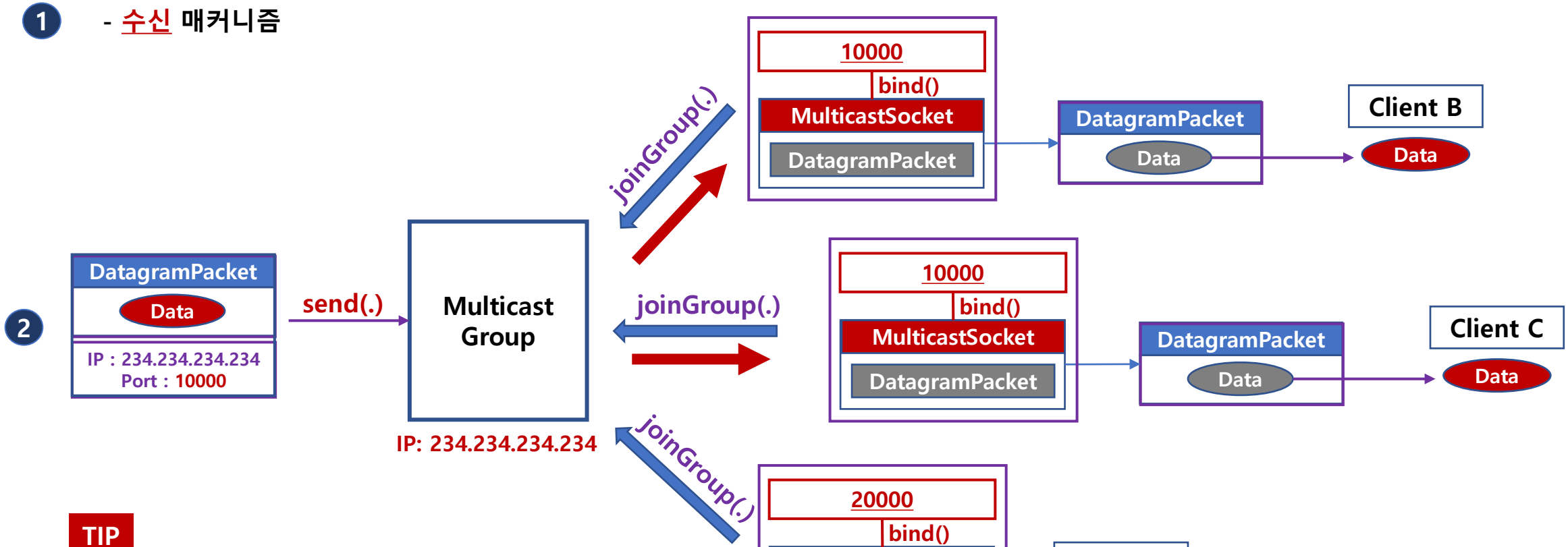
joinGroup(.) → send(.) : Multicast Group에 등록되어 있는 경우
DatagramPacket의 수신지 port = MulticastSokcet 바인딩 port 이면
자신이 보낸 데이터를 자신도 받음

Multicast 통신

클래스 D
224.0.0.0 ~ 239.255.255.255

☞ **Multicast 통신 매커니즘** → 그룹 Join을 통한 UDP 패킷 전송 및 수신

1 - 수신 매커니즘



2

3

TIP

- MulticastSocket은 하나의 포트에 여러 개의 MulticastSocket 바인딩 가능
- 외부IP에서 보내는 데이터 수신을 위해서는 localhost가 아닌 실제 IP 입력

Multicast 통신

☞ **MulticastSocket** → 가상의 멀티캐스트 그룹에 가입(join)할 수 있는 소켓으로 멀티캐스트을 지원

1 - MulticastSocket 객체 생성

2

생성자	동작
MulticastSocket()	포트를 지정하지 않는 경우 비워져 있는 **포트로 자동 바인딩** (주로 **송신하는 경우에 사용**)
MulticastSocket(int port)	매개변수로 전달된 port로 가상의 **Multicast Group에서 수신할 포트를 바인딩** (즉, Multicast Group에 도착하는 패킷 중 포트가 port인 데이터를 수신)
MulticastSocket(SocketAddress bindaddr)	매개변수로 전달된 SocketAddress로 바인딩 (이때, 데이터수신을 위해서는 127.0.0.1 또는 localhost가 아닌 실제 IP를 정보를 포함하는 **InetAddress을 지정**)

Multicast 통신

☞ **MulticastSocket** → 가상의 멀티캐스트 그룹에 가입(join)할 수 있는 소켓으로 멀티캐스트을 지원

1 - **MulticastSocket의 주요메서드**

반환타입	메서드 이름	동작
int void	getTimeToLive() setTimeToLive(int ttl)	**패킷의 생존 기간 읽기 및 설정:** 0~255까지 가능 (이외 IllegalArgumentException 발생) : 단말(스위치, 라우터 등)을 통과할 때 마다 숫자 감소 (1(**default**)인 경우 내부 네트워크에서만 사용)
void void	joinGroup(InetAddress mcastaddr) leaveGroup(InetAddress mcastaddr)	**Multicast Group에 가입 및 해제:** 그룹에 가입한 경우 이후의 바인딩 된 포트로 들어오는 모든 패킷 수신 가능 (해제(leaveGroup())시 수신 불가)
void void	send(DatagramPacket p) receive(DatagramPacket p)	**데이터그램 패킷의 전송과 수신:** 상위 클래스인 DatagramSocket의 메서드로 데이터그램의 패킷 전송과 수신 수행

2

3 **TIP** **브로드캐스팅**

멀티캐스팅 주소가 아니라 브로드캐스팅 주소(255.255.255.255)로 데이터그램 패킷을 전송하는 경우 내부 네트워크의 모든 단말에 전달 (join 등의 과정 없음)

Multicast 통신

☞ **MulticastSocket** → 가상의 멀티캐스트 그룹에 가입(join)할 수 있는 소켓으로 멀티캐스트을 지원

예시

```
public static void main(String[] args) {
  //# 멀티캐스트 (224.0.0.0~239.255.255.255) : multicastSocket 클래스
  //#1. MulticastSocket 객체 생성
  MulticastSocket mcs1 = null, mcs2 = null, mcs3 = null;
  try {
    mcs1 = new MulticastSocket(); //비어있는 포트로 자동 바인딩
    mcs2 = new MulticastSocket(10000);
    mcs3 = new MulticastSocket(new InetSocketAddress(InetAddress.getLocalHost(),10000)); //일반적으로 포트만 지정
  } catch (IOException e) { }
  System.out.println(mcs1.getLocalSocketAddress()); //0.0.0.0/0.0.0.0:55477
  System.out.println(mcs2.getLocalSocketAddress()); //0.0.0.0/0.0.0.0:10000
  System.out.println(mcs3.getLocalSocketAddress()); //local IP:10000
  System.out.println();

  //#2. MulticastSocket 주요 메서드
  //@2-1. getTimeToLive(), setTimeToLive(); 0~255까지 가능 (이외 IllegalArgumentException 발생)
  try {
    System.out.println("TimeToLive: " + mcs1.getTimeToLive()); //1
    mcs1.setTimeToLive(50);
    System.out.println("TimeToLive: " + mcs1.getTimeToLive()); //50
  } catch (IOException e1) {}
  System.out.println();
```

1

2

임의 포트(mcs1)와 10000번 포트(mcs2, mcs3)에 바인딩 된 MulticastSocket 객체 생성

```
0.0.0.0/0.0.0.0:55477
0.0.0.0/0.0.0.0:10000
/192.168.0.5:10000
```

```
TimeToLive: 1
TimeToLive: 50
```

Multicast 통신

☞ **MulticastSocket** → 가상의 멀티캐스트 그룹에 가입(join)할 수 있는 소켓으로 멀티캐스트을 지원

예시

```
//@2-2. joinGroup(InetAddress), leaveGroup(InetAddress), send(DatagramPacket), receive(DatagramPacket)
try {
```

1

```
mcs1.joinGroup(InetAddress.getByName("234.234.234.234"));
mcs2.joinGroup(InetAddress.getByName("234.234.234.234"));
mcs3.joinGroup(InetAddress.getByName("234.234.234.234"));
```

→ **234.234.234.234 멀티캐스트 그룹에 가입**

234.234.234.234:10000 원격지 주소로 문자열 전송

```
byte[] sendData = "안녕하세요".getBytes();
DatagramPacket sendPacket =
                        new DatagramPacket(sendData, sendData.length, InetAddress.getByName("234.234.234.234"), 10000);
mcs1.send(sendPacket);
```

2

```
byte[] receivedData;
DatagramPacket receivedPacket;
```

3

```
mcs2가 수신한 데이터 : 안녕하세요 송신지 : /192.168.0.5:55477
```

```
receivedData = new byte[65508];
receivedPacket = new DatagramPacket(receivedData, receivedData.length);
mcs2.receive(receivedPacket);
```

→ **비어있는 DatagramPacket을 이용하여 데이터 수신**

```
System.out.print("mcs2가 수신한 데이터 : " + new String(receivedPacket.getData()).trim());
System.out.println("  송신지 : "+receivedPacket.getSocketAddress());
```

Multicast 통신

☞ **MulticastSocket** → 가상의 멀티캐스트 그룹에 가입(join)할 수 있는 소켓으로 멀티캐스트을 지원

예시

1

```
    receivedData = new byte[65508];
    receivedPacket = new DatagramPacket(receivedData, receivedData.length);
    mcs3.receive(receivedPacket);

    System.out.print("mcs3가 수신한 데이터 : " + new String(receivedPacket.getData()).trim());
    System.out.println("  송신지 : "+receivedPacket.getSocketAddress());

  } catch (UnknownHostException e) {
    e.printStackTrace();
  } catch (IOException e) {
    e.printStackTrace();
  }
}
```

2

```
mcs3가 수신한 데이터 : 안녕하세요 송신지 : /192.168.0.5:55477
```

The End

Multicast 통신 예제

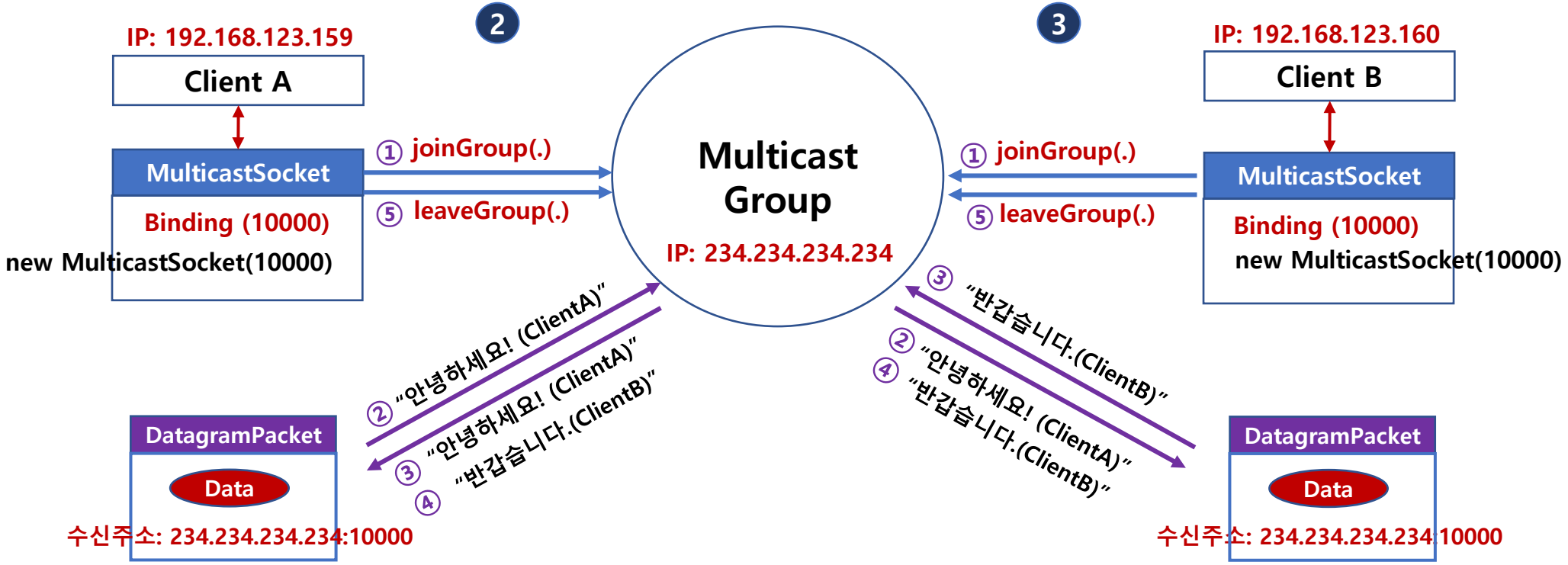
Multicast 통신 예제1 : Client간 Text 교환

Multicast 통신 예제2 : Client간 File 전송

Multicast 통신

실행방법
Step#1. Clinet B 프로그램 실행
Step#2. Client A 프로그램 실행

1 ☞ **Multicast 통신 예제1** → Client간 Text 전송 (두 개의 클라이언트가 Multicast Group에 조인하고 데이터 송수신)



4 Multicast Group에 조인된 경우 같은 포트에 바인딩 되어 있으면 자신이 보낸 패킷도 다시 자신이 수신함
두 클라이언트가 다른 포트로 바인딩되어 있으면 자신이 보낸 데이터를 자신은 받지 않음

Multicast 통신

☞ **Multicast 통신 예제1** → **Client간 Text 전송 (두 개의 클라이언트가 Multicast Group에 조인하고 데이터 송수신)**

예시

CientA(1/2)

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

  //#1. 멀티캐스팅 주소지 생성
  InetAddress multicastAddress = null;
  try {
    multicastAddress = InetAddress.getByName("234.234.234.234");
  } catch (UnknownHostException e) {
    e.printStackTrace();
  }

  //#2. 멀티캐스트소켓 생성
  int multicastPort = 10000;
  MulticastSocket mcs = null;
  try {
    mcs = new MulticastSocket(multicastPort);
  } catch (IOException e) {
    e.printStackTrace();
  }

  //#3. 멀티캐스트 그룹에 조인
  try {
    mcs.joinGroup(multicastAddress);
  } catch (IOException e) {
    e.printStackTrace();
  }
```

1

2

3

MulticastSocket에 수신된 DatagramPacket으로 부터 데이터 정보 출력

4

```
static void receiveMessage(MulticastSocket mcs) {
  byte[] receivedData;
  DatagramPacket receivedPacket;

  receivedData = new byte[65508];
  receivedPacket = new DatagramPacket(receivedData, receivedData.length);
  try {
    mcs.receive(receivedPacket);
  } catch (IOException e) {
    e.printStackTrace();
  }

  System.out.println("보내온 주소 : " + receivedPacket.getSocketAddress());
  System.out.println("보내온 내용 : " + new String(receivedPacket.getData()).trim());
}
```

Multicast 통신

☞ **Multicast 통신 예제1** → Client간 Text 전송 (두 개의 클라이언트가 Multicast Group에 조인하고 데이터 송수신)

예시

CientA(2/2)

```
//#4. 전송 데이터그램 패킷 생성 + 전송
byte[] sendData = "안녕하세요!(ClientA)".getBytes();

DatagramPacket sendPacket = new DatagramPacket(sendData, sendData.length, multicastAddress, multicastPort);
try {
  mcs.send(sendPacket);
} catch (IOException e) {
  e.printStackTrace();
}

//#5. 데이터그램 패킷 수신
receiveMessage(mcs); //자기자신(ClientA)이 보낸 데이터 수신
receiveMessage(mcs); //상대편(ClientB)이 보낸 데이터 수신

//#6. 멀티캐스트 그룹 나가기
try {
  mcs.leaveGroup(multicastAddress);
} catch (IOException e) {
  e.printStackTrace();
}

//#7. 소켓 닫기
mcs.close();
}
```

1

2

3

4

5

```
<<ClientA>> - Text
보내온 주소 : /192.168.0.5:10000
보내온 내용 : 안녕하세요!(ClientA)
보내온 주소 : /192.168.0.5:10000
보내온 내용 : 반갑습니다!(ClientB)
```

```
<<ClientB>> - Text
보내온 주소 : /192.168.0.5:10000
보내온 내용 : 안녕하세요!(ClientA)
보내온 주소 : /192.168.0.5:10000
보내온 내용 : 반갑습니다!(ClientB)
```

Multicast 통신

☞ **Multicast 통신 예제1** → Client간 Text 전송 (두 개의 클라이언트가 Multicast Group에 조인하고 데이터 송수신)

예시

CientB(1/2)

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

  System.out.println("<<ClientB>> - Text");

  //#1. 멀티캐스팅 주소지 생성
  InetAddress multicastAddress = null;
  try {
    multicastAddress = InetAddress.getByName("234.234.234.234");
  }
  catch (UnknownHostException e) {
  e.printStackTrace();
  }

  //#2. 멀티캐스트소켓 생성
  int multicastPort = 10000;
  MulticastSocket mcs = null;
  try {
    mcs = new MulticastSocket(multicastPort);
  } catch (IOException e) { e.printStackTrace();  }

  //#3. 멀티캐스트 그룹에 조인
  try {
    mcs.joinGroup(multicastAddress);
  } catch (IOException e) { e.printStackTrace();  }
```

1

2

3

4

```
static void receiveMessage(MulticastSocket mcs) {
  byte[] receivedData;
  DatagramPacket receivedPacket;

  receivedData = new byte[65508];
  receivedPacket = new DatagramPacket(receivedData, receivedData.length);
  try {
    mcs.receive(receivedPacket);
  } catch (IOException e) {
    e.printStackTrace();
  }

  System.out.println("보내온 주소 : " + receivedPacket.getSocketAddress());
  System.out.println("보내온 내용 : " + new String(receivedPacket.getData()).trim());
}
```

Multicast 통신

☞ **Multicast 통신 예제1** → Client간 Text 전송 (두 개의 클라이언트가 Multicast Group에 조인하고 데이터 송수신)

예시

CientB(2/2)

```
//#4. 데이터그램 패킷 수신 대기
receiveMessage(mcs); //상대편(ClientA)가 보낸 메시지 수신

//#5. 전송 데이터그램 패킷 생성 + 전송
byte[] sendData = "반갑습니다!(ClientB)".getBytes();

DatagramPacket sendPacket = new DatagramPacket(sendData, sendData.length, multicastAddress, multicastPort);
try {
  mcs.send(sendPacket);
} catch (IOException e) {  e.printStackTrace();  }

//#6. 데이터그램 패킷 수신 대기
receiveMessage(mcs); //자기자신(ClientB)이 보낸 메시지 수신

//#7. 멀티캐스트 그룹 나가기
try {
  mcs.leaveGroup(multicastAddress);
} catch (IOException e) {
  e.printStackTrace();
}

//#8. 소켓 닫기
  mcs.close();
}
```

1 2 3 4

5

```
<<ClientA>> - Text
보내온 주소 : /192.168.0.5:10000
보내온 내용 : 안녕하세요!(ClientA)
보내온 주소 : /192.168.0.5:10000
보내온 내용 : 반갑습니다!(ClientB)
```

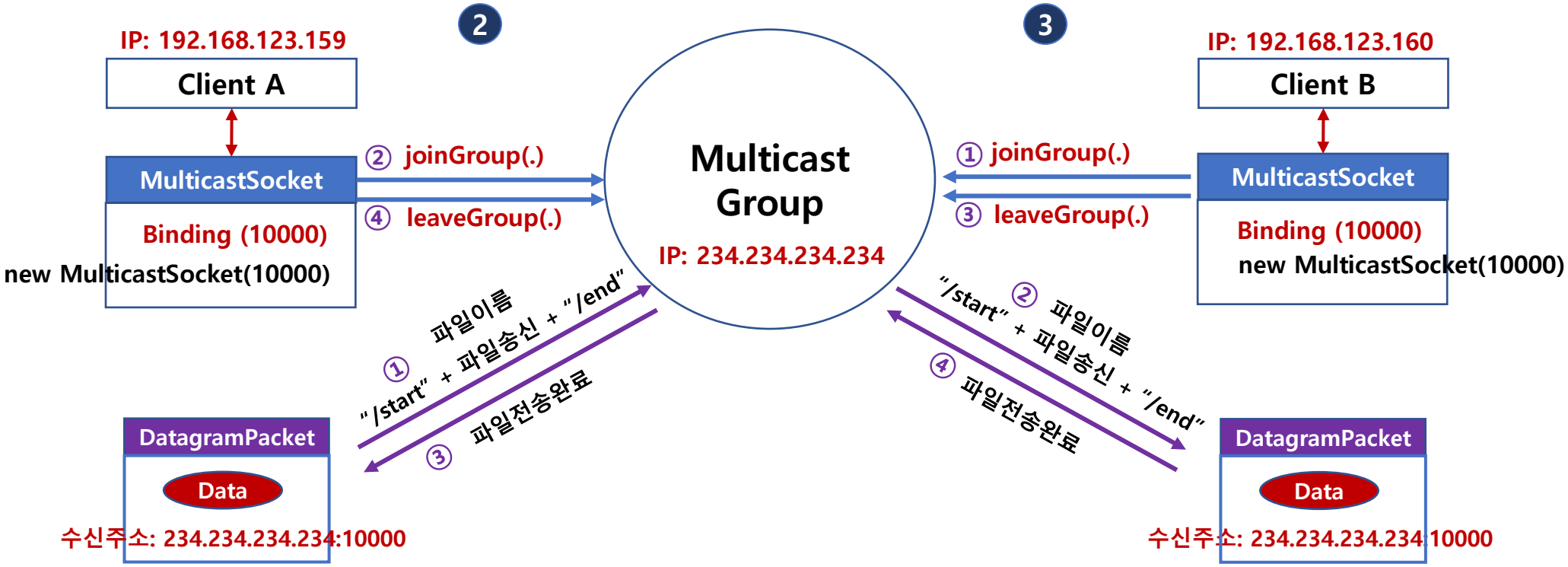
```
<<ClientB>> - Text
보내온 주소 : /192.168.0.5:10000
보내온 내용 : 안녕하세요!(ClientA)
보내온 주소 : /192.168.0.5:10000
보내온 내용 : 반갑습니다!(ClientB)
```

The End

Multicast 통신

실행방법
Step#1. Clinet B 프로그램 실행
Step#2. Client A 프로그램 실행

1 ☞ **Multicast 통신 예제2** → Client간 File 전송 (두 개의 클라이언트가 Multicast Group에 조인하고 File 송수신)



4 Multicast Group에 조인된 경우 같은 포트에 바인딩 되어 있으면 자신이 보낸 패킷도 다시 자신이 수신함
두 클라이언트가 다른 포트로 바인딩되어 있으면 자신이 보낸 데이터를 자신은 받지 않음

Multicast 통신

☞ **Multicast 통신 예제2** → Client간 File 전송 (두 개의 클라이언트가 Multicast Group에 조인하고 File 송수신)

예시

CientA(1/4)

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

  System.out.println("<<ClientA>> - File");

  //#1. 멀티캐스팅 주소지 생성
  InetAddress multicastAddress = null;
  try {
    multicastAddress = InetAddress.getByName("234.234.234.234");
  } catch (UnknownHostException e) {  e.printStackTrace();  }
  int multicastPort = 10000;

  //#2. 멀티캐스트소켓 생성
  MulticastSocket mcs = null;
  try {
    mcs = new MulticastSocket(multicastPort);
  } catch (IOException e) {  e.printStackTrace();  }

  //#3. 파일 로딩
  File file = new File("src₩₩sec04_multicastcommunication₩₩files_clientA₩₩ImageFileUsingMulticast.jpg");
  BufferedInputStream bis = null;
  try {
    bis = new BufferedInputStream(new FileInputStream(file));
  } catch (FileNotFoundException e) {  e.printStackTrace();  }
```

1

2

3

Multicast 통신

☞ **Multicast 통신 예제2** → Client간 File 전송 (두 개의 클라이언트가 Multicast Group에 조인하고 File 송수신)

예시

CientA(2/4)

```
//#4. 파일데이터 전송
DatagramPacket sendPacket = null;

//@4-0. 파일이름 전송
String fileName = file.getName();
sendPacket = new DatagramPacket(fileName.getBytes(), fileName.length(), multicastAddress, multicastPort);
try {
  mcs.send(sendPacket);
} catch (IOException e) {  e.printStackTrace(); }
System.out.println(fileName + " 파일 전송 시작");

//@4-1. 파일 시작 태그를 전송 (/start)
String startSign = "/start";
sendPacket = new DatagramPacket(startSign.getBytes(), startSign.length(), multicastAddress, multicastPort);
try {
  mcs.send(sendPacket);
} catch (IOException e) {  e.printStackTrace();  }
```

1

2

Multicast 통신

☞ **Multicast 통신 예제2** → **Client간 File 전송 (두 개의 클라이언트가 Multicast Group에 조인하고 File 송수신)**

예시

CientA(3/4)

```
//@4-2. 실제 파일 데이터 전송 (2048사이즈로 나누어서 파일 전송)
int len;
byte[] filedata = new byte[2048];
try {
  while((len=bis.read(filedata))!=-1) {
    sendPacket = new DatagramPacket(filedata, len, multicastAddress, multicastPort);
    mcs.send(sendPacket);
  }
} catch (IOException e) {  e.printStackTrace();  }

//@4-3. 파일 시작 태그를 전송 (/start)
String endSign = "/end";
sendPacket = new DatagramPacket(endSign.getBytes(), endSign.length(), multicastAddress, multicastPort);
try {
  mcs.send(sendPacket);
} catch (IOException e) {
  e.printStackTrace();
}

//#5. 멀티캐스트 그룹에 조인
try {
  mcs.joinGroup(multicastAddress);
} catch (IOException e) {
  e.printStackTrace();
}
```

1

2

3

Multicast 통신

☞ **Multicast 통신 예제2** → Client간 File 전송 (두 개의 클라이언트가 Multicast Group에 조인하고 File 송수신)

예시

CientA(4/4)

1

```
//#6. 데이터 수신 대기
receiveMessage(mcs);

//#7. 멀티캐스트 그룹 나가기
try {
  mcs.leaveGroup(multicastAddress);
} catch (IOException e) {
  e.printStackTrace();
}

//#8. 소켓 닫기
mcs.close();
}
```

2

3

```
static void receiveMessage(MulticastSocket mcs) {
  byte[] receivedData;
  DatagramPacket receivedPacket;

  receivedData = new byte[65508];
  receivedPacket = new DatagramPacket(receivedData, receivedData.length);
  try {
    mcs.receive(receivedPacket);
  } catch (IOException e) {
    e.printStackTrace();
  }

  System.out.println("보내온 주소 : " + receivedPacket.getSocketAddress());
  System.out.println("보내온 내용 : " + new String(receivedPacket.getData()).trim());
}
```

4

```
<<ClientA>> - File ImageFileUsingMulticast.jpg 파일 전송 시작
보내온 주소 : /192.168.0.5:10000
보내온 내용 : (ClientB) 파일 수신 완료
```

```
<<ClientB>> - File ImageFileUsingMulticast.jpg 파일 수신 시작
파일 수신 완료
```

Multicast 통신

☞ **Multicast 통신 예제2** → **Client간 File 전송 (두 개의 클라이언트가 Multicast Group에 조인하고 File 송수신)**

예시

CientB(1/4)

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

  System.out.println("<<ClientB>> - File");

  //#1. 멀티캐스팅 주소지 생성
  InetAddress multicastAddress = null;
  try {
    multicastAddress = InetAddress.getByName("234.234.234.234");
  } catch (UnknownHostException e) {  e.printStackTrace();  }
  int multicastPort = 10000;

  //#2. 멀티캐스트소켓 생성
  MulticastSocket mcs = null;
  try {
    mcs = new MulticastSocket(multicastPort);
  } catch (IOException e) {  e.printStackTrace();  }

  //#3. 멀티캐스트 그룹에 조인
  try {
    mcs.joinGroup(multicastAddress);
  } catch (IOException e) {  e.printStackTrace();  }
```

1

2

3

Multicast 통신

☞ **Multicast 통신 예제2** → Client간 File 전송 (두 개의 클라이언트가 Multicast Group에 조인하고 File 송수신)

예시

CientB(2/4)

```
//#4. 파일 데이터 수신
byte[] receivedData;
DatagramPacket receivedPacket;

//@4-1. 파일 이름 수신
receivedData = new byte[65508];
receivedPacket = new DatagramPacket(receivedData, receivedData.length);
try {
  mcs.receive(receivedPacket);
} catch (IOException e) {
  e.printStackTrace();
}
String fileName = new String(receivedPacket.getData()).trim();
System.out.println(fileName + " 파일 수신 시작");

//@4-2. 파일 저장을 위한 파일 출력 스트림 생성
File file = new File("src₩₩sec04_multicastcommunication₩₩files_clientB₩₩"+fileName);
BufferedOutputStream bos=null;
try {
  bos = new BufferedOutputStream(new FileOutputStream(file));
} catch (FileNotFoundException e) {
  e.printStackTrace();
}
```

1

2

Multicast 통신

☞ **Multicast 통신 예제2** → Client간 File 전송 (두 개의 클라이언트가 Multicast Group에 조인하고 File 송수신)

예시

CientB(3/4)

```
//@4-3. 시작태그부터 끝태그까지 모든 데이터패킷의 내용을 파일에 기록
String startSign = "/start";
String endSign = "/end";
receivedData = new byte[65508];
receivedPacket = new DatagramPacket(receivedData, receivedData.length);

try {
  mcs.receive(receivedPacket);
  if(new String(receivedPacket.getData(), 0, receivedPacket.getLength()).equals(startSign)) {
    while(true) {
      mcs.receive(receivedPacket);
      if(new String(receivedPacket.getData(), 0, receivedPacket.getLength()).equals(endSign))
        break;
      bos.write(receivedPacket.getData(), 0, receivedPacket.getLength());
      bos.flush();
    }
  }
} catch (IOException e) {  e.printStackTrace();  }

try {
  bos.close();
} catch (IOException e) {  e.printStackTrace();  }
System.out.println("파일 수신 완료");
```

1

2

Multicast 통신

☞ **Multicast 통신 예제2** → Client간 File 전송 (두 개의 클라이언트가 Multicast Group에 조인하고 File 송수신)

예시

CientB(4/4)

```
//#5. 멀티캐스트 그룹 나가기
try {
  mcs.leaveGroup(multicastAddress);
} catch (IOException e) {  e.printStackTrace();  }

//#6. 전송 데이터그램 생성 + 전송
byte[] sendData = "(ClientB) 파일 수신 완료".getBytes();
DatagramPacket sendPacket = new DatagramPacket(sendData, sendData.length, multicastAddress, multicastPort);
try {
  mcs.send(sendPacket);
} catch (IOException e) {  e.printStackTrace();  }

//#7. 소켓닫기
mcs.close();

}
```

1

2

3

```
<<ClientA>> - File ImageFileUsingMulticast.jpg 파일 전송 시작
보내온 주소 : /192.168.0.5:10000
보내온 내용 : (ClientB) 파일 수신 완료
```

```
<<ClientB>> - File ImageFileUsingMulticast.jpg 파일 수신 시작
파일 수신 완료
```

The End