

세상의 속도를
따라잡고 싶다면

Do
it!

특별 부록

PDF 전자책

자바 완전 정복

심화 편

부록 1 • 자바 네트워크

부록 2 • 자바 API의 함수형 인터페이스

김동형 지음



PDF 내려받기



부록 1 강의



부록 2 강의

QR 코드를
열어 보세요!

이지스퍼블리싱

부록 1. 자바 네트워크

자바 네트워크를 이해하려면 먼저 네트워크에 대해 알아야 한다. 따라서 먼저 IP와 포트 등 네트워크를 구성하는 기본 개념에 대해 알아 보고, 네트워크 정보를 저장 및 관리하는 자바 클래스인 `InetAddress`, `InetSocketAddress`를 사용하는 방법을 알아 본다. 마지막으로 이 장의 주요 내용인 TCP, UDP, 멀티캐스트 통신 프로토콜의 개념과 종류 등을 살펴보고, 이를 기반으로 각 통신 프로토콜 방식에 따른 원격지 호스트와의 데이터 입출력 방법을 예제를 통해 학습한다. 개념을 설명할 때 포장지, 택배 박스, 우체통 같은 비유적 표현들이 많이 나올 테니, 이를 통해 원리를 먼저 파악한 후 문법을 익힌다면 자바 네트워크를 이해하는데 훨씬 도움이 될 것이다.

- 네트워크의 기본 개념
- 자바 네트워크

네트워크의 기본 개념

자바의 네트워크를 이해하기 위해서는 먼저 기본적인 네트워크 이론에 대한 사전지식이 필요하다. 이번 절에서는 이러한 사전지식으로서 IP^{Internet Protocol}와 Port, TCP^{Transmission Control Protocol}와 UDP^{User Datagram Protocol}, 유니캐스팅^{Unicasting}, 브로드캐스팅^{Broadcasting}, 멀티캐스팅^{Multicasting}의 개념에 대해서 먼저 알아보자.

IP 주소

IP 주소^{Internet Protocol Address}란 인터넷상에서 장치간 통신을 위해 장치를 식별하는 주소이다. IP 주소체계는 IPv4와 IPv6로 나뉜다. 그럼 하나하나 알아보자.

IPv4

IPv4는 요즘 일반적으로 사용되는 IP 주소 체계로 32bit(4byte) 체계를 가지고 있다. 숫자로 말하면 약 43억 개의 장치를 식별할 수 있다는 말이다. 주소를 표현할 때는 1byte 크기의 숫자 4개를 점으로 분리하여 표현하는데, 이때 각 바이트값은 부호가 없는 정수^{unsigned integer}로 표현한다. 다시 말해 [0-255].[0-255].[0-255].[0-255]에 해당하는 숫자의 조합으로 표현되는 것이다(예, 192.168.0.2). 조합 가능한 모든 IP 주소가 장치를 식별하는데 사용되는 것은 아니며 특정 IP 주소들은 특별한 의미로 예약하여 사용된다. 예를 들어 127.0.0.1은 루프백^{loop back} 주소로 이 IP 주소로 데이터를 보내면 보낸 장치로 루프백 즉, 다시 돌아온다.

IPv6

시간에 지남에 따라 IPv4만으로는 인터넷상의 장치를 구분하기에 주소의 개수가 부족한 문제가 발생했다. 이를 해결하기 위한 방법으로 나온 것이 차세대 인터넷 프로토콜인 IPv6이다. IPv6는 주소 표현을 위해 128bit(16byte)를 2byte씩 콜론(:)으로 분리하여 표기하며, 각 자릿수는 16진수로 표현한다. 따라서 IPv6로 표현할 수 있는 모든 조합은 [0000-FFFF]:[0000-FFFF]:[0000-FFFF]:[0000-FFFF]:[0000-FFFF]:[0000-FFFF]:[0000-FFFF]:[0000-FFFF]와 같이 표현할 수 있다.

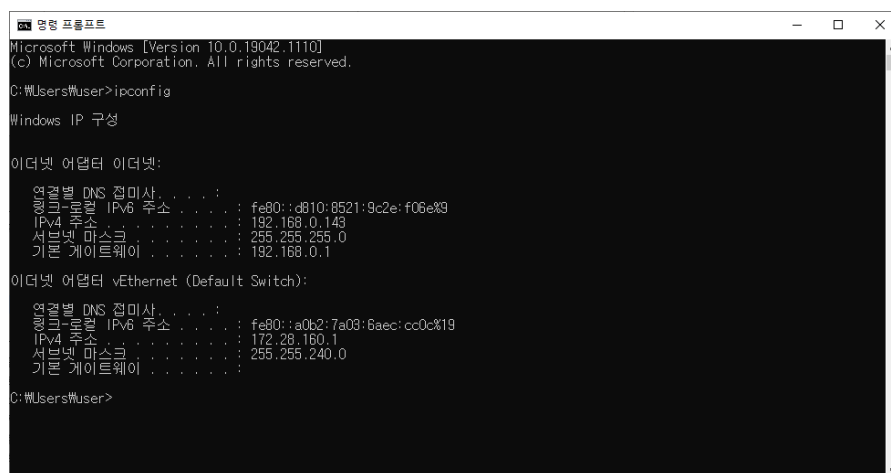
▶ IPv6 체계에서는 주소가 워낙 길다보니 분리 표기된 값이 모두 0인 경우 생략하는 등의 약식 표현이 존재한다.

그렇다면 IPv6를 사용하면 IP 주소가 부족한 문제를 해결할 수 있을까? IPv6가 사용하는 128bit로 표현할 수 있는 IP 주소의 개수를 하루 낱잡아 계산해보았다. 표현할 수 있는 모든 개수는 340,282,366,920,938,463,463,374,607,431,768,211,456이다. 밑줄 친 부분이 IPv4로 표현 가능한 개수의 자릿수이니 IPv6로 표현할 수 있는 주소의 범위가 얼마나 넓은지 어느 정도 가늠이 될 것이다.

내 시스템에서 IP 확인하기

IP 주소가 인터넷상에 모든 장치를 식별한다고 하였으니 내 컴퓨터도 인터넷만 된다면 IP가 할당되어 있을 것이다. 윈도우에서는 명령 프롬프트를 열고 **ipconfig** 명령을 입력하면 할당된 IP 주소를 확인할 수 있다.

▶ 인터넷이 된다는 의미는 인터넷을 가능하도록 하는 네트워크 어댑터가 설치되어 있고 인터넷 서비스를 받고 있다는 의미와 동일하다.



```
Microsoft Windows [Version 10.0.19042.1110]
(c) Microsoft Corporation. All rights reserved.

C:\Users\User>ipconfig

Windows IP 구성

이더넷 어댑터 이더넷:

    연결된 DNS 접미사 . . . . . : 
    링크-로컬 IPv6 주소 . . . . . : fe80::d810:8521:9c2e:f06e%9
    IPv4 주소 . . . . . : 192.168.0.145
    서브넷 마스크 . . . . . : 255.255.255.0
    기본 게이트웨이 . . . . . : 192.168.0.1

이더넷 어댑터 vEthernet (Default Switch):

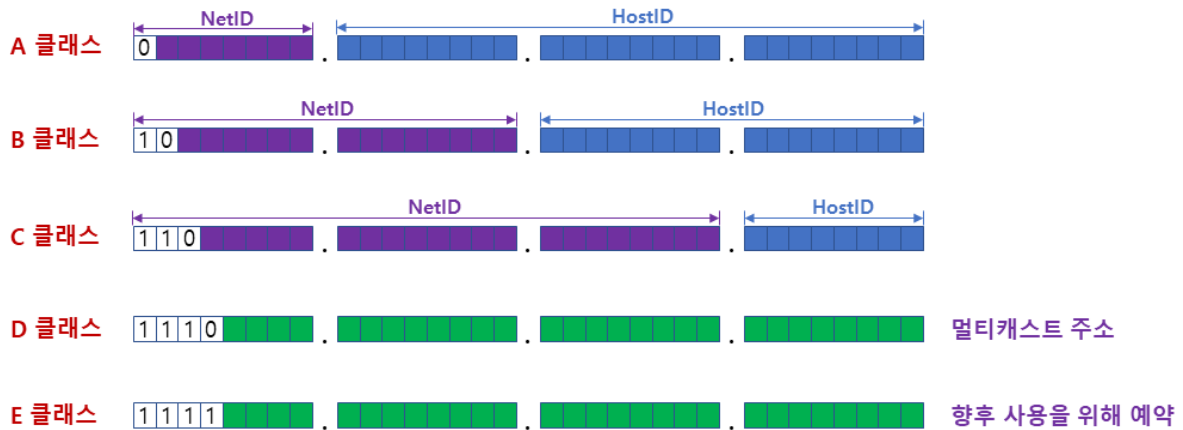
    연결된 DNS 접미사 . . . . . : 
    링크-로컬 IPv6 주소 . . . . . : fe80::a0b2:7a03:6aec:cc0c%19
    IPv4 주소 . . . . . : 172.28.160.1
    서브넷 마스크 . . . . . : 255.255.240.0
    기본 게이트웨이 . . . . . : 

C:\Users\User>
```

윈도우에서 할당된 IP 주소를 확인하는 방법

IP 주소의 분류

IP 주소는 할당할 수 있는 네트워크^{network} ID 수와 호스트^{host} ID 개수, 또는 사용 목적에 따라 5개의 클래스로 분류하여 사용한다.

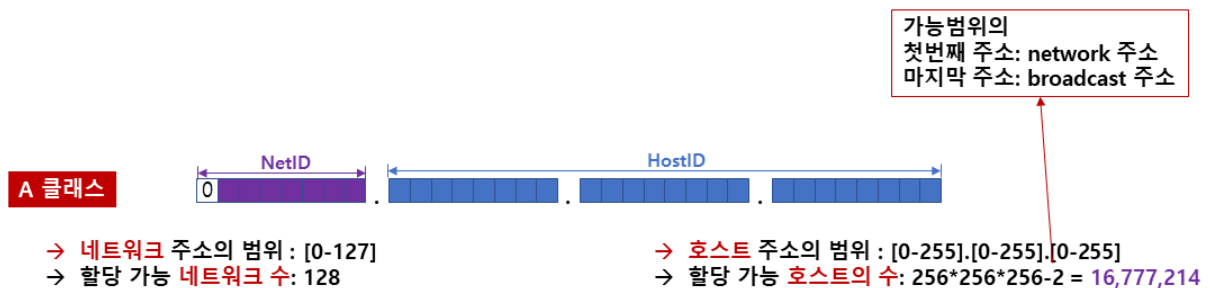


IP 주소 분류 5가지

먼저 A 클래스, B클래스, C 클래스는 할당할 수 있는 네트워크 ID의 수와 호스트 ID의 수에 따라 구분된다. 일반적으로 인터넷 통신을 위해 IP를 할당받는 경우 A 클래스~C 클래스에 속하는 IP를 할당받게 된다. D 클래스는 멀티캐스트 주소 할당을 위해 사용하고, E 클래스는 향후 사용을 위해 예약해 놓은 클래스이다. 그럼 이들 클래스 IP 주소에 대해서 좀 더 자세히 알아보자.

A 클래스

A 클래스는 전체 4byte의 IP 주소 중 1byte를 네트워크 ID로 사용하고 나머지 3byte를 호스트 ID로 사용한다.



A 클래스 IP에서의 네트워크 ID 및 호스트 ID 할당

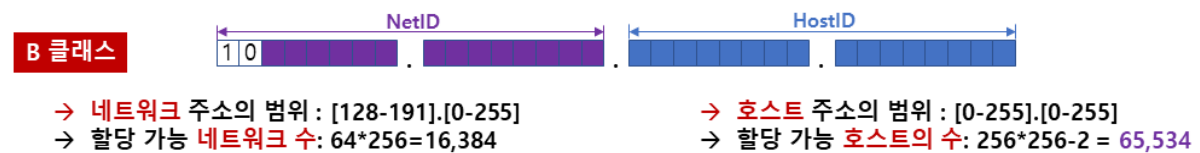
네트워크 ID의 첫 bit는 클래스 간의 구분을 위해 0의 값으로 고정되어 있다. 따라서 A 클래스로 할당 가능한 네트워크 수는 7bit, 즉 128개([0-127])의 네트워크를 할당할 수 있다. 각 네트워크는 3byte([0-255].[0-255].[0-255])로 호스트 ID를 표현하기 때문에 각 네트워크 당

$256 \times 256 \times 256 = 16,777,216$ 개의 호스트를 구분하여 표현할 수 있다. 다만 전체 호스트 ID 중 첫 번째 ID인 '-.0.0.0'의 경우 네트워크 ID 자체를 가리키고, 마지막 ID인 '-.255,255,255'의 경우 브로드캐스트 broadcast 주소로 사용된다. 따라서 하나의 A 클래스 네트워크 내에 실제 할당할 수 있는 정확한 호스트의 개수는 16,777,214개이다. 할당 호스트의 개수를 보면 유추할 수 있겠지만 A 클래스의 할당은 일반적으로 국가 단위로 이루어진다.

▶ 브로드캐스트 주소란 동일 네트워크에 연결된 모든 장치를 가리키는 주소를 말한다.

B 클래스

B 클래스는 네트워크 ID로 2byte를 사용하고, 클래스 간의 구분을 위해 처음 2bit는 '10'으로 고정되어 있다.

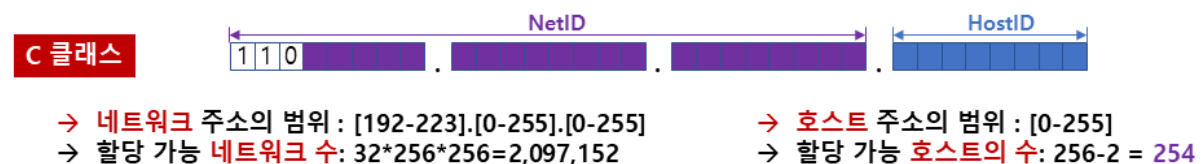


B 클래스 IP에서의 네트워크 ID 및 호스트 ID 할당

따라서 B 클래스의 네트워크는 [128-191].[0-255] 범위의 값으로 할당 가능하며, 전체 할당 가능한 네트워크 수는 $64 \times 256 = 16,384$ 이다. 호스트 ID로는 2byte([0-255].[0-255])가 사용되며 전체 할당 호스트 수는 네트워크 자체 주소(-.-.0.0)와 브로드캐스트 주소(-.-.255.255)를 제외한 $256 \times 256 - 2 = 65,534$ 개이다. 일반적으로 B 클래스는 할당 호스트 수의 규모상 기업 단위로 할당된다.

C 클래스

C 클래스는 처음 3bit가 '110'으로 고정되어 있는 3byte 네트워크 ID와 1byte의 호스트 ID로 구성된다.



C 클래스 IP에서의 네트워크 ID 및 호스트 ID 할당

따라서 가능한 조합의 네트워크 주소는 [192-223].[0-255].[0-255] 범위 내에 있어 전체 가능한 네트워크 수는 $32 \times 256 \times 256 = 2,097,152$ 개다. 반면 호스트는 1byte만 할당되기 때문에 [0-255] 범위 중 0(네트워크 자체 주소를 가리키는 경우)과 255(브로드캐스트 주소)를 제외한 254개의 호스트를 할당할 수 있다.

서브넷 마스크

앞서 할당하고자 하는 호스트 ID의 수에 따라 서로 다른 IP 클래스를 할당받아야 한다는 것은 이해했을 것이다. 하지만 호스트 ID의 개수에 따른 분류는 3가지 경우(A 클래스, B 클래스, C 클래스)만 제공하고 있으며, 이때 할당 가능한 호스트 수는 각각 16,777,214개, 65,534개, 그리고 254개다. 만일 A라는 회사에 1만대 정도의 호스트를 포함하는 부서가 4개 있고 이들 4개 부서의 네트워크를 분리하여 사용하고자 하는 경우를 생각해보자.



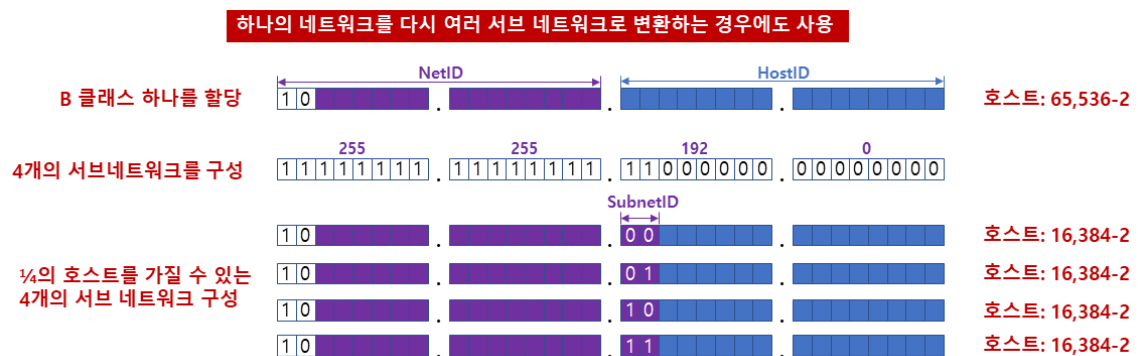
1만 대의 호스트 장치가 필요한 4개의 부서를 가진 A 회사에서의 IP 할당

C 클래스는 하나의 네트워크로는 고작 254개의 호스트만 IP 할당이 가능하니 불가능하다. 반면 B 클래스는 하나의 네트워크당 65,534개의 호스트에 IP 할당이 가능하다. 따라서 각각 1만대의 호스트를 할당하기 위해서는 B 클래스에서 4개의 네트워크를 할당받아야 한다. 각 네트워크당 무려 5만개 정도의 IP가 남아 돌지만 네트워크를 분리하여 할당하기 위해서는 어쩔 수 없는 일이다. 이러한 문제를 해결할 수 있는 방법이 바로 서브넷 마스크(subnet mask)이다.

서브넷 마스크는 네트워크 ID와 호스트 ID를 구분하는 마스크로 개념적으로는 호스트 ID의 일부를 네트워크 ID로 사용하는 것이다. 앞의 예시의 경우 일단 B 클래스의 네트워크를 1개

할당받고, 할당받은 65,536의 호스트를 다시 하위 네트워크로 분할하여 사용하는 것이다. 시스템은 IP 주소와 서브넷 마스크의 AND 연산(&) 결과를 네트워크 ID로 인식한다. 예를 들어 IP 주소가 '128.123.222.123'이고 서브넷 마스크가 '255.255.0.0'의 경우 네트워크 ID는 $128.123.222.123 \& 255.255.0.0 = 128.123.0.0$ 이 되는 것이다.

그러면 이제 하나의 네트워크를 서브넷 마스크를 이용하여 다시 여러 개의 서브 네트워크로 변환하는 예를 살펴보자. B 클래스는 2byte의 네트워크 ID를 사용하기 때문에 만일 서브넷 마스크로 255.255.0.0을 사용하는 경우 여전히 네트워크 ID로 2byte만을 사용한다는 의미가 된다. 만일 서브넷 마스크로 255.255.192.0을 사용한다면 이는 2byte+2bit를 네트워크 ID로 사용한다는 의미이다. 즉, 호스트 ID로 할당된 전체 byte중 처음 2bit를 네트워크 ID로 사용하기 때문에 하나의 네트워크를 4개(2bit)의 하위 네트워크로 분리하게 되는 것이다.



서브넷 마스크를 이용하여 B 클래스의 네트워크 1개를 4개의 하위 네트워크로 분할

4개로 분리된 각각의 하위 네트워크는 14bit를 이용하여 호스트 ID를 할당할 수 있기 때문에 네트워크 자체 주소와 브로드캐스트 주소를 제외하고 16,382개의 호스트에 IP를 할당할 수 있게 된다. 따라서 B 클래스 네트워크 1개를 할당받아 각각 호스트를 1만 대씩 포함하는 4개의 부서에 네트워크를 분리하여 IP를 할당할 수 있다. 서브넷 마스크 덕분에 비용을 4배나 줄인 것이다. 이렇게 서브넷 마스크를 사용하면 전체 호스트 개수를 수용할 수 있는 클래스의 IP를 할당받아 원하는 만큼의 호스트를 할당할 수 있는 여러 개의 하위 네트워크를 구성할 수 있다.

공인 IP와 사설 IP

IPv4로 사용할 수 있는 호스트의 개수는 앞서 알아본 바와 같이 43억개 정도이다. 전 세계의 컴퓨터뿐만 아니라 스마트폰 등 인터넷 호스트의 개수를 고려해보면 턱없이 부족한 숫자이다. 하지만 여전히 IPv4를 사용할 수 있는 이유는 바로 사설 IP의 사용 때문이다. IP는 호스트를 식별하는 범위에 따라 공인 IP와 사설 IP로 나뉜다.

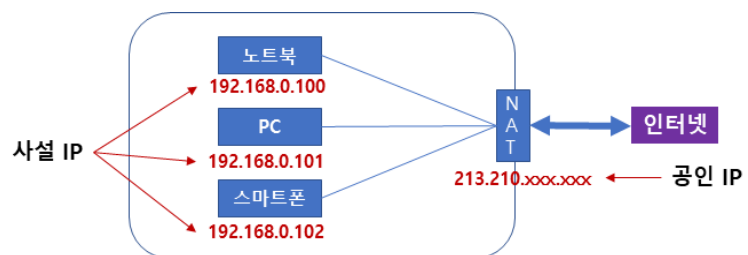
공인 IP는 인터넷상에서 각각의 호스트를 식별하는 IP로 나라별로 각각의 기관에서 관리를 하며 우리나라의 경우 인터넷진흥원에서 관리하고 있다. 보통 우리가 알고 있는 인터넷상에 유일무이한 주소인 IP가 바로 공인 IP인 것이다. 그렇다면 사설 IP는 무엇일까? 이는 내부망 내에서만 한정적으로 사용되는 IP이다. 각각의 내부망에서만 사용되기 때문에 네트워크마다 중복된 IP를 사용할 수 있다.

집에서 공유기를 설정해본 사람이라면 192.168.0.1에 접속하여 공유기에 접속해본 적이 있을 것이다. 이 공유기에 연결하여 사용하는 컴퓨터, 노트북, 스마트폰은 모두 192.168.0.[2-254] 사이의 IP가 할당된다. 만일 같은 회사의 공유기를 사용한다면 집집마다 같은 IP를 사용한다는 소리인데 이게 바로 사설 IP인 것이다. 어떻게 이게 가능할까? 사설 IP가 공유기를 거쳐 인터넷으로 나갈 때는 공인 IP로 바뀌어 전송된다. 즉, 공유기의 사용자는 실제로 하나의 공인 IP를 할당받아 여러 개의 사설 IP로 나누어 쓰는 셈인 것이다.

▶ 기본 설정된 공유기의 IP 주소는 제품마다 다를 수 있다.

사설 IP

- 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]



클래스별 사설 IP와 사용 예

각 클래스별로 특정 주소에 대해서 외부로는 나갈 수 없는 사설 IP를 지정해 놓고 있는데 A 클래스, B 클래스, C 클래스의 사설 IP 주소는 각각 10.[0-255].[0-255].[0-255], 172.[16-31].[0-255].[0-255], 192.168.[0-255].[0-255]의 범위를 갖는다.

포트

포트^{port}란 호스트 내에서 실행되고 있는 프로세스를 구분하기 위한 16bit(0~65,535)의 논리적 할당값을 의미한다. 쉽게 이야기해서 IP가 집주소라면 포트는 방번호에 해당하는 것이다. 즉, IP가 해당 컴퓨터를 찾기 위한 주소라면, 포트는 해당 프로세스에 대한 정보를 의미하는 것이다. 예를 들어 123.124.125.126 컴퓨터에 웹 페이지 정보를 포함하고 있는 데이터가 도착했다면, 도착한 데이터는 웹 브라우저 프로세스로 전달되어야 제대로 사용될 것이다. 즉, 웹 브라우저가 사용하는 포트 번호로 전달되는 것이다.

포트 번호로 사용할 수 있는 0~65,535번 중에서 0~1023번은 잘 알려진 포트^{well-known port}로 여러 인터넷 통신 프로세스가 미리 예약하여 사용한다. 다음은 그중 가장 대표적으로 사용되는 프로세스의 포트이다.

대표적인 잘 알려진 포트 및 서비스

포트	서비스		포트	서비스	
20	FTP-Data	FTP	80	HTTP	Web
21	FTP-Control	FTP	110	POP3	메일
23	Telnet	원격터미널	111	RPC	원격 프로시저 콜
25	SMTP-mail	메일	138	NetBIOS	윈도우 파일 공유
53	DNS	도메인이름→IP 주소	143	IMAP	메일
69	TFTP	간단한 FTP	161	SNMP	네트워크 관리

TCP와 UDP

인터넷을 통한 대표적인 통신 프로토콜(protocol)에는 TCP(Transmission Control Protocol)와 UDP(User Datagram Protocol)가 있다. 여기서 통신 프로토콜이란 두 호스트간 통신을 위해 정의된 통신 방식에 대한 규약 또는 약속이라고 생각하면 된다.

먼저 TCP는 신뢰성이 높은 통신 방식으로 패킷 전송시 오류가 발생하면 재전송을 수행한다. 연결형 프로토콜로 통신 과정 동안 연결 유지가 필요하다. 따라서 만일 여러 호스트들과 TCP 통신을 하는 경우 각각의 연결을 유지해 놓아야 하기 때문에 시스템 부하가 높아지는 단점이 있다. 전송 데이터의 크기가 제한이 없고 개념상 전화 통신과 비슷하며, 실시간성이 필요한 서비스보다는 파일 전송과 같이 전송 데이터의 신뢰성이 필요한 서비스에 주로 사용된다. TCP는 연결 지향형이기 때문에 모든 데이터가 같은 경로(이를 가상회선이라 함)로 이동하여 먼저 출발한 데이터가 항상 먼저 도착한다. 따라서 수신 순서는 전송 순서와 동일하다.

UDP는 전송 과정에서 전송 데이터에 오류가 발생하면 해당 데이터는 삭제되며 재전송하지 않기 때문에 신뢰성이 낮은 통신 프로토콜이다. 신뢰성이 낮다고 하니 쓰임이 별로 없어 보일 수도 있는데 전혀 그렇지 않다. 이는 데이터의 오류보다 데이터 전달의 실시간성이 중요한 서비스에 주로 사용된다. 예를 들어 인터넷으로 생방송 서비스하는 경우 간혹 데이터가 손실되었다고 재전송을 받는다면 더 이상 생방송이 되지 않을 것이다. 즉 이런 서비스의 경우 UDP를 사용하는 것이 적절하다. UDP는 전송데이터의 크기가 헤더를 포함하여 65,535bit로 한정되어 있어 그 이상의 데이터를 전송하고자 하는 경우 나누어 전송해야 한다. 전송하고자 하는 데이터는 데이터그램이라는 패킷으로 포장하여 전송한다. 비연결성 프로토콜이기 때문에 각각의 데이터그램은 다른 경로를 통해 수신지로 도착할 수 있으며, 이로 인해 수신 순서는 전송 순서와 다를 수 있다. 또한 수신 여부를 따로 확인하는 과정이 없기 때문에 상대적으로 빠른 속도로 동작하며 1:1뿐만 아니라 1:N 또는 N:N에도 적합하게 사용될 수 있다. UDP 통신은 개념적으로 우편(택배) 통신과 비슷하다.

지금까지 설명한 TCP와 UDP의 특징을 정리하면 다음과 같다.

TCP와 UDP의 주요 특징 비교

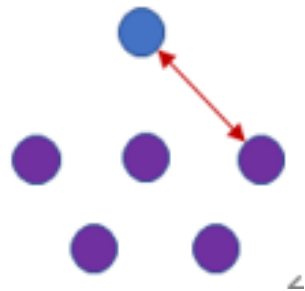
	TCP	UDP
신뢰성	높음	낮음
연결 방식	연결 지향	비연결 지향
패킷 교환 방식	가상 회선 방식	데이터그램 방식
전송 순서	전송 순서 보장	전송 순서 비보장
수신 여부 확인	수신 여부 확인	수신 여부 미확인
통신 방식	1:1 통신	1:1 또는 1:N 또는 N:N
속도	상대적으로 느림	상대적으로 빠름

유니캐스팅, 브로드캐스팅, 멀티캐스팅

마지막으로 네트워크 기본 이론 중 유니캐스팅, 브로드캐스팅 그리고 멀티캐스팅에 대해 알아보자. 이는 1:1 또는 1:N 통신에 관한 것으로 각각에 대해 살펴보면 다음과 같다.

유니캐스팅

유니캐스팅(unicasting)이란 두 장치 간의 1:1 통신을 일컫는 것으로, 특정 장치의 주소를 지정하여 통신하는 방식이다. 물리 주소(MAC 주소: 일반적으로 네트워크 카드의 고유값)를 기반으로 통신하는데, 받은 패킷에 자신의 물리 주소가 포함된 경우에만 데이터를 수신하여 사용하며 그렇지 않는 데이터들은 과감히 버려 버린다. 만일 동일한 내용을 여러 사용자에게 보내고자 하는 경우 유니캐스트를 사용하면 각각의 호스트별로 전송 데이터를 반복하여 보내야 하니 비효율적일 것이다.

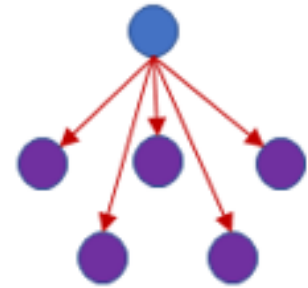


브로드캐스팅

브로드캐스팅(broadcasting)은 UDP 기반의 통신 방식으로 자신이 속해 있는 네트워크 내의 모든 호스트에게 데이터를 전달하는 1:N 통신 방식이다. 내부의 모든 호스트에게 동일한 데이터를

전송하는 경우 효율적으로 사용될 수 있다. 브로드캐스팅 시 수신지의 주소에는 브로드캐스트 주소를 사용한다.

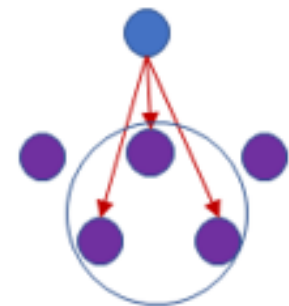
브로드캐스트 주소는 2가지 형태로 표현될 수 있는데, 첫 번째는 '네트워크 ID + 모든 호스트 ID 주소의 비트값 = 1'이다. 이 경우 해당 네트워크 ID 내의 모든 호스트에 데이터를 전달하게 된다. 예를 들어 128.234.0.0 네트워크에 연결된 모든 호스트로 전송하기 위한 브로드캐스트 주소는 128.134.255.255이다. 다만 이 경우 해당 네트워크의 라우터가 브로드캐스트를 지원해야 한다. 두 번째는 호스트 ID 뿐만 아니라 네트워크 ID의 모든 주소 비트값이 모두 1인 경우, 즉 255.255.255.255를 사용하는 방법이다. 이 경우 호스트가 속한 네트워크 내의 모든 호스트에 데이터가 전달된다.



멀티캐스팅

만일 네트워크 내에 6개의 단말이 있고 그중 3개의 단말에만 동일한 데이터를 전송하고자 하는 경우는 어떻게 해야 할까? 만일 유니캐스트로 보낸다면 동일한 데이터를 1:1로 3번 보내야 하고, 브로드캐스팅 방법을 사용하는 경우 원하지 않는 3개의 호스트가 해당 데이터를 받을 것이다. 이렇게 특정 그룹에게만 데이터를 전달하고 싶을 때 쓰는 기법이 바로 멀티캐스팅(multicasting)이다. 멀티캐스팅은 실제 호스트가 아닌 가상의 D 클래스 IP 주소에 데이터를 전달한다. 멀티캐스팅에 사용되는 D 클래스의 IP 주소는 [224-239].[0-255].[0-255].[0-255]이다.

이후 전달된 데이터는 해당 멀티캐스팅 IP 주소에 가입(join)된 호스트에게만 전달된다. 메신저 채팅으로 비유하면 1:1로 메시지를 보내거나(유니캐스팅) 광고와 같이 모든 사용자에게 데이터를 일괄적으로 보내는(브로드캐스팅) 대신 원하는 사람만 채팅방에 초대해서 메시지를 보내 그 사람들만 메시지를 읽을 수 있는 것(멀티캐스팅)과 같다.



자바 네트워크

네트워크에 대한 기본 개념을 알아보았으니 이제 자바의 네트워크를 본격적으로 알아보자. 어떤 프로토콜(TCP, UDP)을 사용하여 어떤 통신 방식(유니캐스팅, 멀티캐스팅, 브로드캐스팅)을 사용하든 IP 주소와 포트는 사용해야 한다. 자바에서는 주소를 저장하기 위해 대표적으로 InetAddress와 InetSocketAddress 클래스를 제공한다. 그럼 하나하나 살펴보자.

주소 저장 클래스

InetAddress 클래스의 객체 생성

InetAddress 클래스는 IP 주소와 호스트 이름을 저장 및 관리하는 클래스로 포트 번호는 저장하지 않는다. public 생성자를 제공하지 않기 때문에 다음과 같은 정적 메서드를 사용하여 객체를 생성한다.

InetAddress 객체 생성을 위한 정적 메서드

리턴 타입	메서드명	주요 내용
InetAddress	getByName(String host)	호스트 이름과 해당 IP 주소를 저장한 객체 리턴
	getByAddress(byte[] addr)	입력 IP 주소를 저장한 객체 리턴(값이 128 이상인 경우 (byte) 타입으로 캐스팅 필요)
	getByAddress(String host, byte[] addr)	호스트 이름과 입력 IP 주소를 저장한 객체 리턴(IP 주소값이 128 이상인 경우 (byte) 타입으로 캐스팅 필요)
	getLocalHost()	로컬 호스트 IP를 저장한 객체 리턴
	getLoopbackAddress()	루프백 IP(127.0.0.1)를 저장한 객체 리턴

InetAddress[]	getAllByName(String host)	여러 개의 IP를 사용하는 경우 모든 호스트 IP를 저장한 객체 리턴
---------------	---------------------------	--

getByName(String host) 메서드는 매개변수로 호스트 주소를 문자열로 입력받아 InetAddress 객체를 생성한다. 도메인 이름("www.google.com")으로 받을 수도 있고 IP 주소("127.217.26.132")를 직접 입력할 수도 있다. 두 번째 메서드인 getByAddress(byte[] addr)는 byte[] 배열값으로 IP 주소를 넘겨받아 InetAddress 객체를 생성한다. 이때 byte[]의 원소인 byte 자료형은 -128~127 사이의 값을 가지므로 IP 주소의 각 자릿수에 128보다 큰 값이 있는 경우 byte 타입으로 캐스팅이 필요하다. 이런 번거로움 때문에 getByName() 메서드도 IP 형태의 주솟값을 처리할 수 있으니 개인적으로는 getByAddress()보다는 캐스팅 걱정없는 getByName() 메서드를 사용하기를 권고한다.

▶ IP 주소는 0~255 사이의 값을 조합해 구성한다.

호스트 이름과 byte[]의 IP 주소를 모두 받는 getByAddress() 메서드도 오버로딩되어 있다. 여기서 주의해야 할 점은 InetAddress 객체 자체는 호스트 이름 또는 IP를 정보를 저장할 뿐이지 실제 해당 호스트 또는 IP가 있는지의 여부는 상관하지 않는다. 실제 해당 주소의 존재 여부는 InetAddress를 이용하여 접속을 시도할 때 비로소 확인할 수 있다. getLocalHost()는 현재 호스트의 IP를 가지고 있는 InetAddress 객체를 리턴하며 getLoopbackAddress()는 루프백(127.0.0.1) 주소를 저장한 객체를 리턴한다. 마지막으로 하나의 호스트가 여러 개의 IP를 사용하는 경우 getAllbyName() 메서드로 한 번에 배열 형태로 객체를 가져올 수도 있다.

InetAddress 클래스의 주요 메서드

InetAddress 클래스의 주요 메서드는 다음과 같다.

InetAddress 클래스의 주요 메서드

리턴 타입	메서드명	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 호스트 이름을 문자열로 리턴
boolean	isLoopbackAddress()	IP가 루프백 주소인지 확인
boolean	isMulticastAddress()	IP가 멀티캐스팅 영역의 주소인지 확인
boolean	isReachable(int timeout)	ping 명령으로 리턴 여부 확인

getAddress()는 객체에 저장하고 있는 IP 주소를 byte[] 형태로 리턴한다. IP 주소값이 128 이상이면 음숫값으로 변환한 후 출력한다.

▶ byte 자료형에 -128~127의 범위를 벗어난 값을 저장하는 경우 반대쪽 끝에서부터 값이 증가하거나 감소한다. 예를 들어 byte b = (byte) 130의 경우 -128(128), -127(129), -126(130)과 같이 변환되어 b = -126이 저장된다.

getHostAddress()와 getHostName()은 호스트의 IP 주소와 이름을 문자열로 리턴한다. isLoopbackAddress()와 isMulticastAddress()는 각각 저장하고 있는 IP 주소가 루프백 주소인지, 멀티캐스팅 주소인지를 불리언값을 리턴해 알려준다. 마지막으로 isReachable(int timeout)는 주어진 시간 동안 해당 주소로 ping 연결이 가능한지를 알려주는 메서드이다. 실제로 명령 프롬프트에서 'ping 호스트 이름' 또는 'ping IP 주소'를 입력하면 응답 시간을 확인할 수 있다.


```

Microsoft Windows [Version 10.0.19042.1110]
(c) Microsoft Corporation. All rights reserved.

C:\Users\User>ping www.google.com

Ping www.google.com [172.217.175.228] 32바이트 데이터 사용:
172.217.175.228의 응답: 바이트=32 시간=31ms TTL=115
172.217.175.228의 응답: 바이트=32 시간=30ms TTL=115
172.217.175.228의 응답: 바이트=32 시간=31ms TTL=115
172.217.175.228의 응답: 바이트=32 시간=30ms TTL=115

172.217.175.228에 대한 Ping 통계:
    패킷: 보낸 = 4, 받음 = 4, 손실 = 0 (0% 손실),
    왕복 시간(밀리초):
        최소 = 30ms, 최대 = 31ms, 평균 = 30ms
  
```

ping 요청을 이용한 www.google.com까지의 네트워크 진단

보안상의 이유로 ping 요청을 막아 놓은 서버가 많은데, 이 경우 실제 웹 브라우저로 접속은 가능하나 isReachable()는 false값을 리턴할 수 있다(예. www.naver.com).

▶ ping packet internet groper은 특정 호스트에게 패킷을 전송한 후 호스트가 이에 대한 응답 메시지를 보내면 이를 분석하여 해당 호스트까지의 네트워크 상태를 진단하는 명령이다.

추가적으로 InetAddress.toString() 메서드는 '호스트 이름/IP 주소'의 문자열을 리턴하도록 오버로딩되어 있어 System.out.println(InetAddress 객체)를 통해 쉽게 호스트 정보와 IP 정보를 확인할 수 있다.

[Do it 실습] InetAddress 클래스를 이용한 IP 주소의 저장 및 관리

InetAddressObjectAndMethod.java

```

01 package sec01_managingaddress.EX01_InetAddressObjectAndMethod;
02 import java.io.IOException;
03 import java.net.InetAddress;
04 import java.util.Arrays;
05
06 public class InetAddressObjectAndMethod {
07     public static void main(String[] args) throws IOException {
08         // 1. InetAddress 객체 생성
09         // 1-1. 원격지 IP 객체 생성
10         InetAddress ia1 = InetAddress.getByName("www.google.com");
  
```

```

11      InetAddress ia2 = InetAddress.getByAddress(new byte[]
12      {(byte)172,(byte)217,(byte)161,36});
13      InetAddress ia3 = InetAddress.getByAddress("www.google.com", new byte[]
14      {(byte)172,(byte)217,(byte)161,36});
15      System.out.println(ia1);
16      System.out.println(ia2);
17      System.out.println(ia3);
18      System.out.println();
19      // 1-2. 로컬/로프백 IP
20      InetAddress ia4 = InetAddress.getLocalHost();
21      InetAddress ia5 = InetAddress.getLoopbackAddress();
22      System.out.println(ia4);
23      System.out.println(ia5);
24      System.out.println();
25      // 1-3. 하나의 호스트가 여러 개의 IP를 가지고 있는 경우
26      InetAddress[] ia6 = InetAddress.getAllByName("www.naver.com");
27      System.out.println(Arrays.toString(ia6));
28      System.out.println();
29      // 2. InetAddress 메서드
30      byte[] address = ia1.getAddress();
31      System.out.println(Arrays.toString(address));
32      System.out.println(ia1.getHostAddress());
33      System.out.println(ia1.getHostName());
34      System.out.println();
35      System.out.println(ia1.isReachable(1000)); //true
36      System.out.println(ia1.isLoopbackAddress()); //false
37      System.out.println(ia1.isMulticastAddress()); //false 224-239.0-255.0-
38      255.0-255
39      System.out.println(InetAddress.getByAddress(new byte[]
40      {127,0,0,1}).isLoopbackAddress()); //true
41      System.out.println(InetAddress.getByAddress(new byte[]
42      {(byte)234,(byte)234,(byte)234,(byte)234}).isMulticastAddress()); //true
43      }

```

실행 결과

www.google.com/172.217.25.4

```
/172.217.161.36
www.google.com/172.217.161.36

DHKIM/192.168.123.101
localhost/127.0.0.1

[www.naver.com/125.209.222.141, www.naver.com/125.209.222.142]

[-84, -39, 25, 4]
172.217.25.4
www.google.com

true
false
false
true
true
```

10-17: 호스트 이름 또는 IP 주소를 이용하여 InetAddress 객체를 생성한다. 여기서 호스트 이름과 IP 주소를 함께 입력하는 경우 IP를 기준으로 저장한다. 즉, 호스트 이름에 오류가 있어도 이 객체를 통해 접속할 수 있다. InetAddress의 toString() 메서드는 '호스트 이름/IP 주소' 문자열을 출력한다.

20-23: 자신의 로컬 컴퓨터 이름과 주소를 저장하는 InetAddress 객체를 생성한다. 결괏값은 실행 환경마다 다르다. 루프백 주소(127.0.0.1)를 저장하는 InetAddress 객체도 생성하고 출력한다.

26-27: 다수의 서버를 사용하는 경우 getAllByName() 메서드를 이용하여 InetAddress[] 객체에 담아 Arrays.toString() 메서드로 한 번에 출력한다.

30-41: InetAddress 객체에 저장된 IP 주소를 byte[] 배열 또는 문자열로 받거나 호스트 이름을 가져와 출력한다. 저장하고 있는 IP까지의 네트워크 연결 가능 여부를 확인하고, 루프백 또는 멀티캐스트 IP인지도 확인하여 출력한다. 마지막으로 127.0.0.1이 루프백 주소인지, 234.234.234.234가 멀티캐스트 주소인지 확인한다.

참고로 www.google.com 또는 www.naver.com 등의 경우 동적으로 IP를 관리하기 때문에 예제를 실행할 때마다 다른 IP 값이 나올 수 있다. 하지만 결과로 나오는 모든 IP 주소는 해당 호스트의 주소이다.

SocketAddress 클래스의 객체 생성과 주요 메서드

SocketAddress는 IP 주소(호스트 이름)와 포트 번호를 함께 저장 관리하는 추상 클래스로 앞서 배운 **InetAddress에 포트 번호 관리 기능이 추가된 클래스**이다. 추상 클래스이기 때문에 객체를 생성하기 위해서는 SocketAddress 클래스를 상속받는 자식 클래스를 생성하고 추상 메서드를 재정의해야 한다. 늘 그렇듯이 자바는 이미 하위 클래스로 InetSocketAddress 클래스를 제공한다. 따라서 SocketAddress의 객체를 생성하기 위해 우리는 그저 다음과 같은 InetSocketAddress의 생성자를 호출하면 된다.



InetSocketAddress의 상속 구조

InetSocketAddress의 생성자

생성자	주요 내용
InetSocketAddress(int port)	IP 주소 없이 내부의 포트 번호만 지정
InetSocketAddress(String hostname, int port)	매개변수의 호스트 이름에 해당하는 IP와 포트 번호를 지정
InetSocketAddress(InetAddress addr, int port)	매개변수의 InetAddress(IP 정보)와 포트 번호를 지정

생성자를 살펴보면 포트만 저장할 수도 있고 IP 주소와 포트를 함께 저장할 수도 있다. IP 주소는 문자열 또는 앞서 살펴본 InetAddress 타입으로 입력받는다. 포트만 저장하는 경우는 뒤에서 살펴보겠지만 이미 수신한 패킷을 내부적으로 사용하는 경우에 주로 사용된다.

InetSocketAddress의 주요 메서드는 다음과 같이 3가지가 있다.

InetSocketAddress의 주요 메서드

리턴 타입	메서드명	주요 내용
InetAddress	getAddress()	저장 IP 주소를 InetAddress 타입으로 리턴
int	getPort()	포트 번호를 정수형으로 리턴
String	getHostName()	호스트 이름을 문자열로 리턴

IP 주소와 포트를 저장 및 관리하는 클래스이니 이들 주요 메서드는 각각의 정보를 추출하는 메서드이다. 먼저 getAddress() 메서드는 저장된 IP 주소를 InetAddress 객체에 담아 리턴하고, getPort()는 포트값을 정수형으로 리턴한다. InetSocketAddress 객체가 저장 및 관리하는 2개의 정보를 두 메서드로 알아볼 수 있는 것이다. 호스트 이름만을 알고 싶을 때에는 getHostName() 메서드를 사용하며, 이는 호스트 이름을 문자열로 리턴한다. InetSocketAddress 클래스 또한 '호스트 이름/IP 주소:포트 번호'의 문자열을 출력하도록 toString() 메서드가 오버로딩되어 있어 쉽게 저장 정보를 출력할 수 있다. InetAddress의 toString()과 비교하면 포트 번호만 추가된 형태이다.

[Do it 실습] SocketAddress 클래스를 이용한 IP 주소와 포트의 저장 및 관리

SocketAddressObjectAndMethod.java

```

01 package sec01_managingaddress.EX02_SocketAddressObjectAndMethod;
02 import java.net.InetAddress;
03 import java.net.InetSocketAddress;
04 import java.net.UnknownHostException;
05
06 public class SocketAddressObjectAndMethod {
07     public static void main(String[] args) throws UnknownHostException {
08         // 1. SocketAddress 객체 생성 (InetSocketAddress 생성자 사용)
09         InetAddress ia = InetAddress.getByName("www.google.com");
10         int port = 10000;

```

```

11      InetAddress isa1 = new InetAddress(port);
12      InetAddress isa2 = new InetAddress("www.google.com", port);
13      InetAddress isa3 = new InetAddress(ia, port);
14      System.out.println(isa1);
15      System.out.println(isa2);
16      System.out.println(isa3);
17      System.out.println();
18      // 2. SocketAddress의 메서드
19      System.out.println(isa2.getAddress());
20      System.out.println(isa2.getHostName());
21      System.out.println(isa2.getPort());
22  }
23  }

```

실행 결과

```

0.0.0.0/0.0.0.0:10000
www.google.com/216.58.220.196:10000
www.google.com/216.58.220.196:10000

www.google.com/216.58.220.196
www.google.com
10000

```

09-16: 생성자의 매개변수로 각각 포트 번호만 전달, String 타입의 호스트 이름과 포트 번호를 전달, 그리고 InetAddress 객체와 포트 번호를 전달하여 InetAddress 객체 생성 후 출력한다.

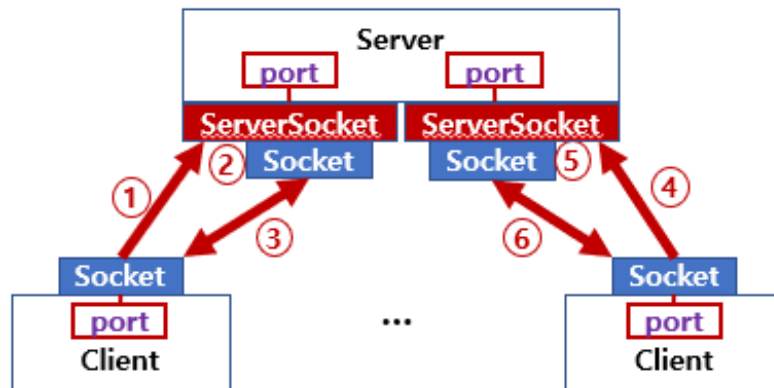
19-21: InetAddress 객체에서 IP 주소, 호스트 이름, 그리고 포트 정보를 추출하여 출력한다.

TCP 통신

네트워크의 개념도 알아봤고 자바의 주소 저장 및 관리 클래스도 알아보았으니 이제 본격적으로 자바를 네트워크 통신에 대해서 알아보자.

TCP 통신 메커니즘

TCP(Transmission Control Protocol) 통신은 앞서 살펴본 바와 같이 1:1 통신 방식이다. 좀 더 정확히 이야기하면 클라이언트와 서버 간의 1:1 통신이다. 실제 TCP 통신이 이루어지는 메커니즘을 살펴보면 다음과 같다.



TCP 통신이 이루어지는 메커니즘

①, ④: 클라이언트의 Socket으로 서버의 ServerSocket에 접속한다.

②, ⑤: 서버는 각 클라이언트와 통신할 수 있는 Socket 생성한다.

③, ⑥: 서버와 클라이언트의 Socket끼리 통신한다.

▶ 두 호스트끼리 통신할 때는 각각을 서버와 클라이언트로 구성한다 2인 이상의 클라이언트끼리 통신하려면 서버를 공유하여 클라이언트-서버-클라이언트 간에 통신해야 한다.

먼저 클라이언트는 Socket 객체를 통해 서버의 ServerSocket 객체에 연결을 요청한다. 연결 요청을 받은 서버의 ServerSocket 객체는 연결 요청을 한 클라이언트와 1:1 통신을 할 수 있는 Socket 객체를 생성한다. 이후 서버의 Socket 객체와 클라이언트의 Socket 객체는 통신을 종료할 때까지 연결을 유지하며 상호간에 데이터를 송수신한다. 만일 새로운 클라이언트가 서버의 ServerSocket 객체로 연결을 요청하면 앞의 과정으로 동일하게 반복하여 새로운 추가 클라이언트를 위한 새로운 소켓^{socket}이 추가된다.

TCP를 통해 통신을 수행하는 경우 기억해야 할 것이 하나 있다. 최초의 연결 요청은 항상 클라이언트(Socket)에서 서버(ServerSocket)로 이루어진다는 것이다. 따라서 클라이언트-

클라이언트 간 통신을 위해서는 두 클라이언트가 서버에 접속한 이후 서버가 전달자의 역할을 수행하는 클라이언트-서버-클라이언트 구조를 가져야 한다.

Socket 클래스의 객체 생성

TCP 통신에서 장치간 실제 데이터의 송수신 기능을 담당하는 Socket 클래스에 대해서 알아보자. Socket 클래스는 TCP 통신에서 두 호스트간의 입출력 스트림을 제공하는 클래스로 다음의 생성자를 통해 생성할 수 있다.

Socket 클래스의 생성자

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 port 정보를 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 port 정보를 포함하여 연결 요청)

첫 번째 생성자를 제외하고 모든 생성자는 매개변수로 호스트 주소(문자열 또는 InetAddress 타입)와 포트 번호를 입력받아 Socket 객체를 생성한다. 클라이언트에서 이들 생성자로 객체를 생성하면 **생성과 동시에 해당 주소로 연결을 요청**한다. 기본 생성자인 첫 번째 생성자는 연결 정보 없이 단순히 소켓만을 생성하는 예로 이후 연결 정보를 추가하여 연결을 요청할 수 있다.

Socket 객체가 생성되어 실제 연결이 이루어질 때, 연결하고자 하는 원격지 주소가 존재하지 않는 경우 UnknownHostException과 IOException이 발생할 수 있어 이에 대한 예외 처리를 해주어야 한다.

Socket 클래스의 주요 메서드

Socket클래스의 주요 메서드에는 다음과 같다.

Socket클래스의 주요 메서드

리턴 타입	메서드명	동작
void	connect(SocketAddress endpoint)	연결 정보가 없는 소켓에 원격지 주소 정보(SocketAddress)를 제공하여 timeout 시간 동안 연결 요청 수행(timeout=0인 경우 시간제약 없음, IOException 처리 필요)
void	connect(SocketAddress endpoint, int timeout)	
InetAddress	getInetAddress()	원격지 주소 정보: Socket에 연결된 원격지
int	getPort()	InetAddress와 port 리턴
InetAddress	getLocalAddress()	로컬 주소 정보: 로컬의 InetAddress와 port
int	getLocalPort()	정보 및 2가지 정보를 모두 포함한
SocketAddress	getLocalSocketAddress()	SocketAddress 리턴
InputStream	getInputStream()	입출력 스트림: 원격지와의 통신을 위한
OutputStream	getOutputStream()	입출력 스트림 리턴
int	getSendBufferSize()	송수신 버퍼사이즈: 송수신 버퍼사이즈의 설정
void	setSendBufferSize(int size)	및 읽기(default: 65536, SocketException 처리
int	getReceiveBufferSize()	필요)
void	setReceiveBufferSize(int size)	

먼저 connect() 메서드는 연결 정보가 없는 기본 생성자인 Socket()을 이용하여 객체를 생성한 경우 연결 요청을 수행할 원격지 주소 정보를 전달하는데 사용하는 메서드이다. 두 번째 매개변수로 정숫값(timeout)을 전달하면 전달된 시간 동안만 연결 요청을 수행하며, 해당 시간 안에 연결이 이루어지지 않는 경우 예외를 발생시킨다. 두 번째 매개변수를 생략하면 연결이 요청이 수락될 때까지 무한 대기한다. getInetAddress()와 getPort()는 현재 Socket에 연결된 **원격지의 IP 주소와 포트**를 각각 InetAddress와 int값으로 리턴한다. 이와는 반대로 getLocalAddress(), getLocalPort(), getLocalSocketAddress() 메서드는 로컬 주소 정보를 각각 InetAddress, int, SocketAddress 타입으로 리턴한다. TCP 통신에서 가장 중요한 메서드는 getInputStream()과 getOutputStream()으로 이들 메서드는 각각 byte 단위의 입출력 타입인 InputStream과 OutputStream 객체를 리턴한다. 이들 객체를 통해 데이터를 읽고 쓰는 방법은 자바 입출력에서 살펴본 byte 단위의 입출력 스트림(FileInputStream/FileOutputStream 등)에 쓰고 읽는 방식과 완벽하게 일치한다. 유일한 차이점은 이들 객체에 데이터를 읽고 쓰면 원격지의 데이터가 읽혀지거나 원격지 장치로 데이터가 써진다는 점이다. 마지막으로 데이터를 송신하거나 수신할 때 사용하는 버퍼사이즈를 읽거나 설정할 수도 있다. 참고로 송수신에 사용되는 디폴트 버퍼 사이즈는 65,536byte이다.

[Do it 실습] TCP 통신을 위한 Socket 클래스의 활용

SocketObject.java

```

01 package sec02_tcpcommunication.EX01_SocketObject;
02 import java.io.IOException;
03 import java.net.InetAddress;
04 import java.net.InetSocketAddress;
05 import java.net.Socket;
06 import java.net.UnknownHostException;
07
08 public class SocketObject {
09     public static void main(String[] args) throws UnknownHostException,
10         IOException {
11         // 1. Socket 객체 생성

```

```

11      Socket socket1 = new Socket();
12      Socket socket2 = new Socket("www.naver.com", 80);
13      Socket socket3 = new Socket("www.naver.com", 80,
14      InetAddress.getLocalHost(), 10000);
15      Socket socket4 = new Socket(InetAddress.getByName("www.naver.com"),80);
16      Socket socket5 = new Socket(InetAddress.getByName("www.naver.com"), 80,
17      InetAddress.getLocalHost(), 20000);
18      // 2. Socket 메서드
19      // connect 메서드 / 원격지 주소 정보 제공
20      System.out.println(socket1.getInetAddress() + ":" + socket1.getPort());
21      socket1.connect(new InetSocketAddress("www.naver.com", 80));
22      System.out.println(socket1.getInetAddress() + ":" + socket1.getPort());
23      System.out.println(socket2.getInetAddress() + ":" + socket2.getPort());
24      System.out.println();
25      // 로컬 주소 정보(지정한 경우 + 지정하지 않은 경우)
26      System.out.println(socket2.getLocalAddress()+":" +
27      socket2.getLocalPort());
28      System.out.println(socket2.getLocalSocketAddress());
29      System.out.println(socket3.getLocalAddress()+":" +
30      socket3.getLocalPort());
31      System.out.println(socket3.getLocalSocketAddress());
32      System.out.println();
33      // send/receive 버퍼 사이즈
34      System.out.println(socket2.getSendBufferSize()+",
35      "+socket2.getReceiveBufferSize());
36      }

```

실행 결과

```

null:0
www.naver.com/125.209.222.141:80
www.naver.com/125.209.222.141:80

/192.168.123.101:56130
/192.168.123.101:56130
/192.168.123.101:10000
/192.168.123.101:10000

```

65536, 65536

11-17: Socket 생성자로 기본 생성자를 사용하는 경우(socket1), 원격지 IP/포트만을 전달한 경우(socket2, socket4), 원격지 정보와 함께 로컬 호스트 IP/포트를 함께 전달한 경우(socket3, socket5)다. 여기서 IP 정보는 String 또는 InetAddress 객체로 전달한다(로컬 호스트의 정보를 제공하지 않는 경우 포트 번호는 비어 있는 포트로 임의 배정).

20-23: 원격지 정보를 포함하고 있지 않은 socket1으로부터 IP와 포트를 추출하면 각각 null과 0의 값을 가지며, 이 경우 connect() 메서드를 통해 원격지 연결 정보를 전달하여 연결을 수행한다. 이후 원격지 주소와 포트를 가져와 출력한다.

26-29: Socket 객체로부터 로컬 주소 정보와 포트 정보를 각각 가져와 출력할 수도 있고, 이 둘을 모두 포함하고 있는 InetAddress를 가져와 한번에 출력할 수도 있다.

32-33: Socket 객체의 송신 및 수신 버퍼 크기를 가져와 출력한다(디폴트 버퍼 크기는 65,536byte).

ServerSocket 클래스의 객체 생성

그럼 이번에는 서버에서 클라이언트 소켓으로부터의 연결 요청을 대기하고 연결 요청 시 통신을 위한 Socket 객체를 생성하는 역할을 수행하는 ServerSocket 클래스에 대해서 알아보자.

ServerSocket 객체 생성을 위해서는 다음 2개의 생성자를 사용한다.

ServerSocket 클래스의 생성자

생성자	동작
ServerSocket()	특정 포트 바인딩(binding) 없이 단순히 ServerSocket 객체만을 생성 (이후 바인딩 필요)

ServerSocket(int port)	매개변수로 입력된 port로 바인딩된 ServerSocket 객체 생성 (이후 ServerSocket으로의 연결 요청이 오면 바인딩된 port로 전달)
------------------------	---

첫 번째 생성자는 특정 포트로의 바인딩^{binding} 없이 단순히 ServerSocket 객체만을 생성하는 경우로 이후 특정 포트로 바인딩이 필요하다. 두 번째 생성자는 매개변수로 입력된 포트로 바인딩된 ServerSocket 객체를 생성한다. 여기서 **바인딩이란 원격지로부터의 수신 데이터를 특정 포트로 연결하는 것을 말한다.** 다시 말해 IP 정보를 이용하여 호스트까지 도착한 데이터를 어떤 프로세스로 보낼지를 결정하는 것이다. IP는 로컬 IP 주소로 고정되기 때문에 ServerSocket 객체 생성시에 IP 주소 정보는 포함되지 않는다. 즉, 최초 클라이언트로부터 연결이 수행되었을 때 자신의 어떤 프로세스와 연결할 지의 정보만 필요하기 때문에 생성자의 매개변수로 포트정보만 포함되는 것이다.

ServerSocket 클래스의 주요 메서드

ServerSocket 클래스의 주요메서드는 다음과 같다.

ServerSocket 클래스의 주요 메서드

리턴 타입	메서드명	동작
void	bind(SocketAddress endpoint)	바인딩 정보가 없는 서버 소켓에 바인딩 정보를 제공
boolean	isBound()	바인딩 여부 확인: ServerSocket이 바인딩 되어 있는지의 여부를 리턴
void int	setSoTimeout(int timeout) getSoTimeout()	연결 요청 리스닝 시간: Socket으로부터의 연결 요청을 리스닝(listening)하는 시간의 설정 및 가져오기(timeout=0 이면 무한 대기)
Socket	accept()	연결 요청 수락: 연결 요청이 수락된 후 통신을 위한 Socket 객체 리턴 (연결 수락까지 설정된 timeout 시간만큼 blocking)

먼저 bind() 메서드는 바인딩 정보를 포함하지 않는 객체 즉, ServerSocket 클래스의 기본 생성자로 생성한 객체에 바인딩 정보를 제공하는데 사용한다. 이 경우에는 로컬 IP가 아닌 다른 IP 지정도 가능하지만 일반적으로는 서버 자신의 IP가 사용된다. 여기서 한 가지 꼭 기억해야 할 것이 있다. 하나의 포트에는 하나의 프로세스만 바인딩이 가능하다. 만일 이미 사용중인 포트에 바인딩을 시도하면 IOException 예외가 발생한다. 따라서 다음과 같이 작성하면 자신의 컴퓨터에 사용중인 포트를 확인할 수 있다.

사용 중인 포트 확인 방법

```
for(int i=0; i<65536; i++) {  
    try {  
        ServerSocket serverSocket = new ServerSocket(i);  
    } catch(IOException e) {  
        System.out.println(i+"번째 포트 사용중 ...");  
    }  
}
```

isBound() 메서드는 현재 ServerSocket 객체가 바인딩되어 있는지의 여부를 boolean으로 리턴한다. ServerSocket 클래스는 클라이언트로부터의 연결 요청을 대기한다고 하였는데, setSoTimeout(int timeout) 메서드를 이용하여 객체 생성 이후 특정시간(timeout≠0)만 연결을 대기할 수도 있고 무한 대기(timeout=0)를 할 수도 있다.

▶ setSoTimeout(int timeout) 메서드로 별도의 대기 시간을 지정하지 않는 경우 timeout=0을 가져 연결 요청을 무한 대기한다.

설정된 대기 시간은 getTimeout() 메서드로 확인할 수 있다. 마지막으로 가장 중요한 accept() 메서드는 실제 연결 수락을 대기하는 메서드로 클라이언트로부터 연결 요청이 들어오면 해당 클라이언트와 통신할 수 있는 Socket 객체를 리턴한다. 이렇게 리턴된 Socket 객체가 바로 클라이언트 소켓과 통신하는 실제 주체가 되는 것이다. accept() 메서드는 연결 요청이 들어올 때까지는 설정된 시간(timeout)동안 블로킹(blocking) 되기 때문에 연결 수락은 별도의 스레드로 작성하는 것이 좋다. 참고로 ServerSocket 클래스의 bind()와 Socket의 connect()가 다소 혼동될

수 있는데 ServerSocket의 bind()는 서버에 연결 요청이 들어온 경우 해당 데이터를 전달할 연결 포트(프로세스)를 지정하는 것이고, Socket의 connect()는 원격지 특정 주소로의 연결을 수행하는 역할을 하는 메서드이다.

[Do it 실습] TCP 통신을 위한 ServerSocket 클래스의 활용

SeverSocketObject.java

```
01 package sec02_tcpcommunication.EX02_SeveSocketObject;
02 import java.io.IOException;
03 import java.net.InetAddress;
04 import java.net.InetSocketAddress;
05 import java.net.ServerSocket;
06 import java.net.Socket;
07
08 public class SeverSocketObject {
09     public static void main(String[] args) throws IOException {
10         // 1. ServerSocket 객체 생성
11         ServerSocket serverSocket1 = new ServerSocket();
12         ServerSocket serverSocket2 = new ServerSocket(20000);
13         // 2. ServerSocket 메서드
14         // bind
15         System.out.println(serverSocket1.isBound()); //false
16         System.out.println(serverSocket2.isBound()); //true
17         System.out.println();
18         // serverSocket1.bind(new InetSocketAddress("127.0.0.1", 10000));
19         serverSocket1.bind(new
20 InetSocketAddress(InetAddress.getLocalHost(),10000));
21         System.out.println(serverSocket1.isBound()); //true
22         System.out.println(serverSocket2.isBound()); //true
23         System.out.println();
24         // 사용 중인 TCP 포트 확인하기
25         for(int i=0; i<65536; i++) {
26             try {
27                 ServerSocket serverSocket = new ServerSocket(i);
28             } catch(IOException e) {
29                 System.out.println(i+"번째 포트 사용중 ...");
30             }
31         }
32     }
33 }
```

29	}
30	}
31	System.out.println();
32	// accept() 일반적으로는 별도의 쓰레드에서 실행
33	// setSoTimeout(): accept() 대기 시간
34	serverSocket1.setSoTimeout(2000);
35	try {
36	Socket socket = serverSocket1.accept(); //blocking
37	} catch(IOException e) {
38	System.out.println(serverSocket1.getSoTimeout()+"ms 시간이 지나 접
39	속대기를 종료합니다.");
40	}
41	}
42	}

실행 결과

false

true

true

true

135번째 포트 사용중 ...

139번째 포트 사용중 ...

445번째 포트 사용중 ...

...

60384번째 포트 사용중 ...

2000ms 시간이 지나 접속대기를 종료합니다.

11-12. 바인딩할 포트 번호 없이 기본 생성자를 사용하여 ServerSocket 객체(serverSocket1)를 생성하고, 바인딩 포트를 생성자 매개변수로 넘겨줘 ServerSocket 객체(serverSocket2)를 생성한다.

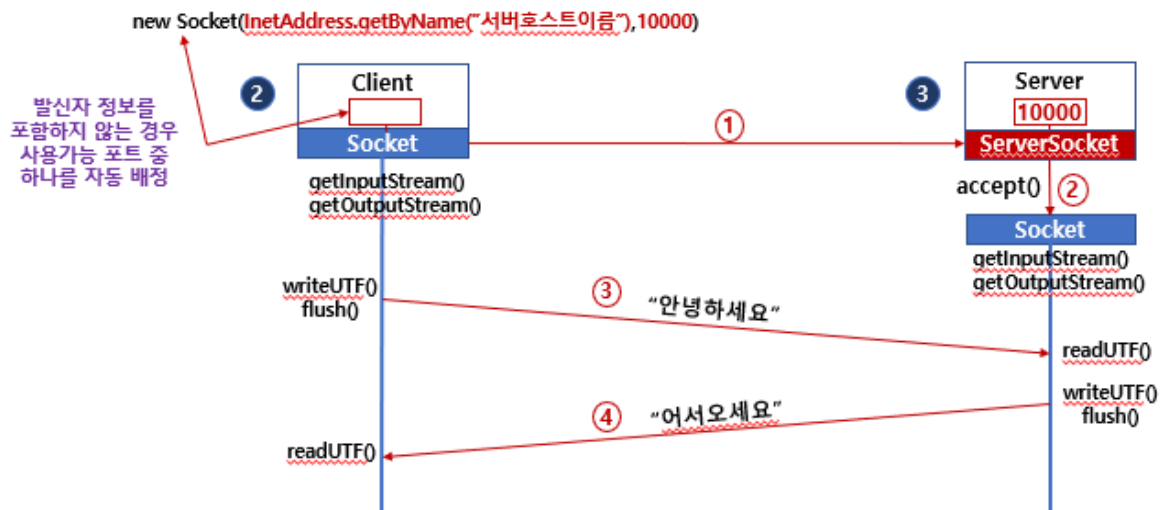
15-21. ServerSocket 객체의 바인딩 여부를 출력하며, 기본 생성자를 통해 생성한 바인딩 정보가 없는 객체의 경우 bind() 메서드를 이용하여 포트를 전달한다.

24-30. 포트의 가용 범위인 0~65,535까지 포트에 바인딩된 ServerSocket 객체를 생성한다. 바인딩하고자 하는 포트가 이미 사용 중일때는 IOException 예외가 발생한다(사용 중인 포트를 확인할 때도 사용 가능).

34-40. 클라이언트의 Socket으로부터의 연결 요청 대기 시간을 2초(2000ms)로 설정한 뒤 accept() 메서드로 응답을 대기한다. 대기시간 2초 내에 연결 요청이 없는 경우 IOException이 발생한다(대기시간을 설정하지 않거나 0으로 설정하면 무한대기).

TCP 통신 예제 1. 클라이언트와 서버 간의 텍스트 전송

TCP 통신에 관한 첫 번째 예로 클라이언트와 서버 간에 텍스트를 전송하는 예를 살펴보자. 우선 전체적인 동작 과정을 살펴보면 다음과 같다.



클라이언트와 서버간의 텍스트 전송 동작 과정

먼저 클라이언트는 Socket 객체 생성 후 서버로 연결 요청을 수행한다. 서버에서는 ServerSocket이 연결 요청을 처리하며 연결이 수락(accept)되면 해당 클라이언트와 통신할 수 있는 Socket 객체가 생성된다. 이 상태가 되면 클라이언트와 서버에 각각 서로 연결된 Socket 객체가 생성되어 있는 것이다. 이 시점에서 클라이언트와 서버는 서로 연결된 Socket 객체를 가지게 된다. 이제 Socket 객체의 getInputStream()과 getOutputStream() 메서드를 통해 InputStream, OutputStream 객체를 생성하면 이후 이들 객체를 통해 read(), write(), flush() 등의

메서드를 사용하여 데이터의 크기에 상관없이 텍스트를 송수신할 수 있다. 아래 예제에서는 다양한 타입의 데이터를 빠른 속도로 전송하기 위해서 Buffered(Input/Output)Stream과 Data(Input/Output)Stream을 함께 사용했다. Server가 연결 요청을 먼저 대기하고 있어야 클라이언트가 접속이 가능하기 때문에 다음 예제를 실행할 때 반드시 서버 쪽 프로그램을 먼저 실행시켜야 한다.

[Do it 실습] TCP 통신 예제 1-1. 클라이언트와 서버 간의 텍스트 전송 [ClientSide]

TCP_Text_ServerSide.java

```

01 package sec02_tcpcommunication.EX03_TCP_Text;
02 import java.io.BufferedInputStream;
03 import java.io.BufferedOutputStream;
04 import java.io.DataInputStream;
05 import java.io.DataOutputStream;
06 import java.io.IOException;
07 import java.net.InetAddress;
08 import java.net.Socket;
09 import java.net.UnknownHostException;
10
11 public class TCP_Text_ClientSide {
12     public static void main(String[] args) {
13         System.out.println("<<Client>>");
14         try {
15             Socket socket = new Socket(InetAddress.getByName("localhost"),
16 10000);
17             System.out.println("Server에 접속 완료");
18             System.out.println("접속 Server 주소 :
19                 "+socket.getInetAddress()+"."+socket.getPort());
20             DataInputStream dis = new DataInputStream(
21                 new BufferedInputStream(socket.getInputStream()));
22             DataOutputStream dos = new DataOutputStream(
23                 new BufferedOutputStream(socket.getOutputStream()));
24             dos.writeUTF("안녕하세요");
25             dos.flush();
26             String str = dis.readUTF();

```



27	System.out.println("server : "+str);
28	}
29	catch (UnknownHostException e) {}
30	catch (IOException e) {e.printStackTrace();}
31	}
	}

실행 결과

<<Client>>

Server에 접속 완료

접속 Server 주소 : localhost/127.0.0.1:10000

server : 어서오세요!

15-18. 원격지 IP의 10,000번 포트로 접속하기 위한 Socket 객체 생성 후 원격지 주소와 포트 번호를 출력한다(이 예제에서는 클라이언트와 서버가 같은 호스트에서 동작하여 원격지 IP에 로컬 주소를 입력).

19-26. Socket으로부터 입출력을 위한 InputStream과 OutputStream을 얻은 뒤 속도의 개선과 다양한 입출력을 위해 Buffered(Input/Output)Stream, Data(Input/Output)Stream 객체를 생성하여 입출력을 수행한다.

[Do it 실습] TCP 통신 예제 1-2. 클라이언트와 서버간의 텍스트 전송 [ServerSide]

TCP_Text_ServerSide.java

01	package sec02_tcpcommunication.EX03_TCP_Text;
02	import java.io.BufferedInputStream;
03	import java.io.BufferedOutputStream;
04	import java.io.DataInputStream;
05	import java.io.DataOutputStream;
06	import java.io.IOException;
07	import java.net.ServerSocket;
08	import java.net.Socket;
09	
10	public class TCP_Text_ServerSide {
11	public static void main(String[] args) {

```

12      System.out.print("<<Server>>");
13      ServerSocket serverSocket = null;
14      try {
15          serverSocket = new ServerSocket(10000);
16      } catch (IOException e) {
17          System.out.println("해당포트를 열 수 없습니다.");
18          System.exit(0);    // 프로그램 종료
19      }
20      System.out.println(" - Client 접속 대기...");
21      try {
22          Socket socket = serverSocket.accept();
23          System.out.println("Client 연결 수락");
24          System.out.println("접속 client 주소:" + socket.getInetAddress() +
25                                ":" + socket.getPort());
26          DataInputStream dis = new DataInputStream(
27              new BufferedInputStream(socket.getInputStream()));
28          DataOutputStream dos = new DataOutputStream(
29              new BufferedOutputStream(socket.getOutputStream()));
30          String str = dis.readUTF();
31          System.out.println("client: "+str);
32          dos.writeUTF("어서오세요!");
33          dos.flush();
34      } catch (IOException e) {}
35  }
36  }

```



실행 결과

```

<<Server>> - Client 접속 대기...
Client 연결 수락
접속 client 주소:/127.0.0.1:56480
client: 안녕하세요

```

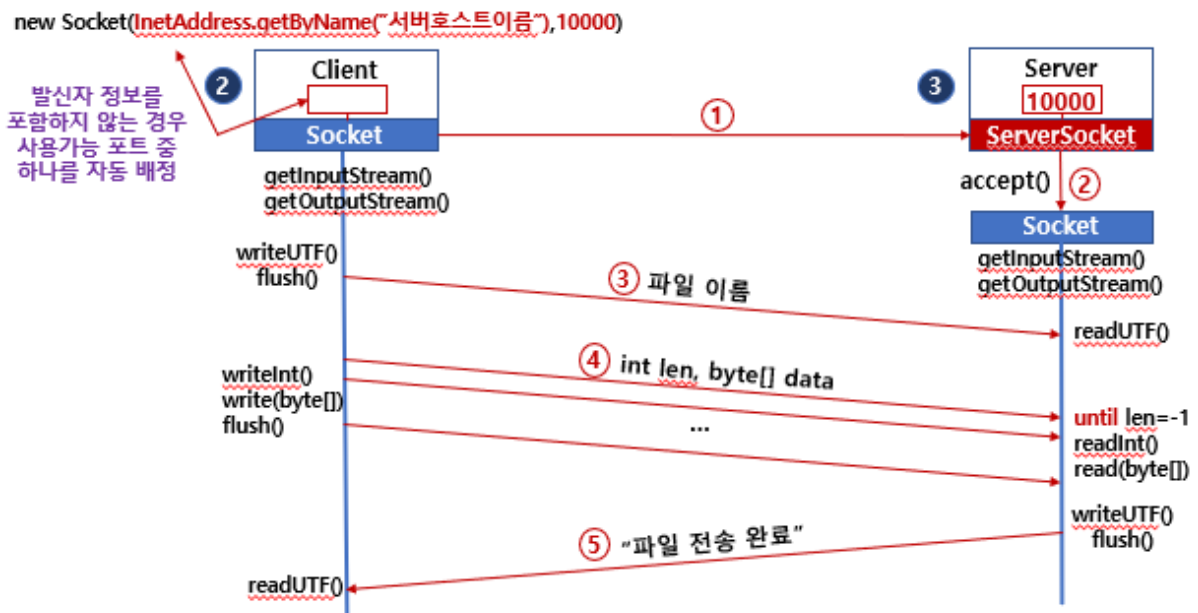
14-19. 10,000번 포트에 바인딩된 SerserSocket 객체를 생성하며 해당 포트가 이미 사용 중인 경우 프로그램을 종료한다.

22-25. `accept()` 메서드를 이용하여 연결 요청 시 요청한 클라이언트와 연결된 `Socket` 객체를 생성하고 원격지 주소(서버측 입장에서는 원격지는 클라이언트를 의미함)의 IP와 포트 번호 출력

26-33. `Socket`으로부터 입출력을 위한 `InputStream`과 `OutputStream`을 얻은 뒤 속도의 개선과 다양한 입출력을 위해 `Buffered(Input/Output)Stream`, `Data(Input/Output)Stream` 객체 생성하여 입출력 수행

TCP 통신 예제 2. 클라이언트와 서버 간의 파일 전송

두 번째 예제는 TCP 통신을 이용하여 클라이언트와 서버 간에 파일을 전송하는 예제이다. 전체적인 흐름은 다음과 같다.



클라이언트와 서버간의 파일 전송 동작 과정

클라이언트와 서버 간의 연결과 `InputStream`, `OutputStream` 객체를 얻는 과정은 앞의 예제와 동일하다. 입출력 스트림 객체 생성 이후 클라이언트는 전송할 파일 이름을 먼저 문자열로 전송한다. 이후 파일을 일정 크기로 나누어 전송하는데 이때 전송하는 파일 크기(`int` 타입)와 읽은 파일 데이터(`byte[]` 타입)으로 나누어 전송한다. 네트워크 전송 시 효율성을 고려하여 일정 크기(여기서는 2,048byte 단위)로 나누어 전송하는 것이 일반적이다. 여기서 단순히 읽은

데이터를 보내는 것이 아니라 읽은 데이터의 개수(파일의 마지막인 -1을 포함)를 함께 보내는 이유는 원격지에서 파일을 직접 읽는 것이 아니라 클라이언트가 파일을 읽은 후 데이터를 전송하기 때문이다. 읽은 데이터의 크기 또는 파일의 마지막 여부를 원격지에서는 알 길이 없기 때문에 이 정보를 함께 전송해야 하는 것이다. 마지막 데이터를 읽어 len=-1이 전송되면 원격지는 파일 전체를 읽었음을 알고, 자신은 “파일수신완료”를 출력하고, 클라이언트에게는 “파일전송완료” 문자열을 전송한다. 이후 클라이언트는 서버로부터 수신한 “파일전송완료” 문자열을 출력함으로써 모든 파일 전송 과정은 끝난다. 클라이언트와 서버 사이의 송수신 데이터 타입이 다양(String, byte[], int)하기 때문에, 이전 예제와 마찬가지로 이 예제에서도 다양한 데이터 타입의 입출력이 가능한 DataInputStream, DataOutputStream 클래스를 활용했다. 파일을 전송하기 위한 이러한 과정은 하나의 예일 뿐이며, 얼마든지 다양한 형태의 파일 전송 방법이 있을 수 있을 것이다. 앞의 예제와 마찬가지로 Server가 연결 요청을 먼저 대기하고 있어야 클라이언트가 접속이 가능하기 때문에 다음 예제를 실행할 때 반드시 서버 쪽 프로그램을 먼저 실행시켜야 한다.

[Do it 실습] TCP 통신 예제 2-1. 클라이언트와 서버 간의 파일 전송 [ClientSide]

TCP_File_ClientSide.java

```

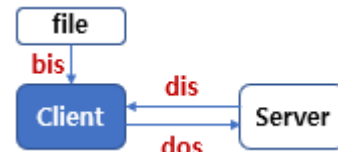
01 package sec02_tcpcommunication.EX04_TCP_File;
02 import java.io.BufferedInputStream;
03 import java.io.BufferedOutputStream;
04 import java.io.DataInputStream;
05 import java.io.DataOutputStream;
06 import java.io.File;
07 import java.io.FileInputStream;
08 import java.io.IOException;
09 import java.net.InetAddress;
10 import java.net.Socket;
11 import java.net.UnknownHostException;
12
13 public class TCP_File_ClientSide {
14     public static void main(String[] args) {

```

```

15      System.out.println("<<Client>>");
16      try {
17          Socket socket = new Socket(InetAddress.getByName("localhost"),
18      10000);
19
20          System.out.println("Server에 접속 완료");
21          System.out.println("접속 Server 주소 : "+ socket.getInetAddress() +
22                                  ":"+socket.getPort());
23
24          DataInputStream dis = new DataInputStream(
25              new BufferedInputStream(socket.getInputStream()));
26          DataOutputStream dos = new DataOutputStream(
27              new BufferedOutputStream(socket.getOutputStream()));
28          File file = new File("src/sec02_tcpcommunication/files_client/
29                                  ImageFileUsingTCP.jpg");
30
31          FileInputStream fis = new FileInputStream(file);
32          BufferedInputStream bis = new BufferedInputStream(fis);
33          System.out.println("파일전송:"+ file.getName());
34          // 1. 파일 이름 전송
35          dos.writeUTF(file.getName());
36          // 2. 파일 데이터 전송
37          byte[] data = new byte[2048];
38          int len;
39          while((len=bis.read(data))!=-1) {
40              dos.writeInt(len);      // 읽은 데이터의 길이
41              dos.write(data, 0, len); // 전송 데이터
42              dos.flush();
43          }
44          dos.writeInt(-1);
45          dos.flush();
46          String str = dis.readUTF();
47          System.out.println(str);
48      }
49      catch (UnknownHostException e) {}
50      catch (IOException e) {e.printStackTrace();}
51  }
52  }

```



실행 결과

<<Client>>

Server에 접속 완료

접속 Server 주소 : localhost/127.0.0.1:10000

파일전송:ImageFileUsingTCP.jpg

파일 전송 완료

17-20. 원격지 IP의 10,000번 포트로의 접속을 위한 Socket 객체 생성 후 원격지 주소와 포트 번호를 출력한다(이 예제에서는 클라이언트와 서버가 같은 호스트에서 동작하여 원격지 IP에 로컬 주소를 입력).

21-24. Socket으로부터 입출력을 위한 InputStream과 OutputStream을 얻은 뒤 속도의 개선과 다양한 입출력을 위해 Buffered(Input/Output)Stream, Data(Input/Output)Stream 객체를 생성한다.

25-43. 전송하고자 하는 파일과 연결된 BufferedInputStream 객체(bis)를 생성하고, 파일을 최대 2,048byte 크기로 나누어 읽은 뒤 DataOutputStream 객체(dos)를 통해 서버 측으로 전송한다. 이후 DataInputStream 객체(dis)를 통해 문자열을 수신한다(파일 전송 과정에서 읽은 파일의 길이를 함께 전송).

</코드설명>

[Do it 실습] TCP 통신 예제 2-2. 클라이언트와 서버 간의 파일 전송 [ServerSide]

TCP_File_ServerSide.java

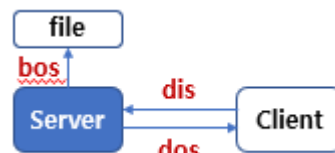
```
01 package sec02_tcpcommunication.EX04_TCP_File;
02 import java.io.BufferedInputStream;
03 import java.io.BufferedOutputStream;
04 import java.io.DataInputStream;
05 import java.io.DataOutputStream;
06 import java.io.File;
07 import java.io.FileOutputStream;
08 import java.io.IOException;
09 import java.net.ServerSocket;
10 import java.net.Socket;
11
```



```

12 public class TCP_File_ServerSide {
13     public static void main(String[] args) {
14         System.out.print("<<Server>>");
15         ServerSocket serverSocket = null;
16         try {
17             serverSocket = new ServerSocket(10000);
18         } catch (IOException e) {
19             System.out.println("해당포트를 열 수 없습니다.");
20             System.exit(0);    // 프로그램 종료
21         }
22         System.out.println(" - Client 접속 대기...");
23         try {
24             Socket socket = serverSocket.accept();
25             System.out.println("Client 연결 수락");
26             System.out.println("접속 client 주소:" + socket.getInetAddress() +
27                                 ":" + socket.getPort());
28             DataInputStream dis = new DataInputStream(
29                 new BufferedInputStream(socket.getInputStream()));
30             DataOutputStream dos = new DataOutputStream(
31                 new BufferedOutputStream(socket.getOutputStream()));
32             // 전송받은 파일 이름 출력
33             String receivedFileName = dis.readUTF();
34             System.out.println("파일수신:" + receivedFileName);
35             File file = new File("src/sec02_tcpcommunication/files_server/" +
36                                 receivedFileName);
37             FileOutputStream fos = new FileOutputStream(file);
38             BufferedOutputStream bos = new BufferedOutputStream(fos);
39             byte[] data = new byte[2048];
40             int len;
41             while((len=dis.readInt())!=-1) {
42                 dis.read(data,0,len);
43                 bos.write(data,0,len);
44                 bos.flush();
45             }
46             System.out.println("파일 수신 완료");
47             dos.writeUTF("파일 전송 완료");

```



48	<code>dos.flush();</code>
49	<code>} catch (IOException e) {}</code>
50	<code>}</code>
51	<code>}</code>

실행 결과

```
<<Server>> - Client 접속 대기...
Client 연결 수락
접속 client 주소:/127.0.0.1:56683
파일수신:ImageFileUsingTCP.jpg
파일 수신 완료
```

16-21. 10,000번 포트에 바인딩된 ServerSocket 객체를 생성하며 해당 포트가 이미 사용 중인 경우 프로그램을 종료한다.

24-27. accept() 메서드를 이용하여 연결 요청 시 요청한 클라이언트와 연결된 Socket 객체를 생성하고 원격지 주소(서버 측 입장에서 원격지는 클라이언트를 의미함)의 IP와 포트 번호를 출력한다.

28-31. Socket으로부터 입출력을 위한 InputStream과 OutputStream을 얻은 뒤 속도의 개선과 다양한 입출력을 위해 Buffered(Input/Output)Stream, Data(Input/Output)Stream 객체를 생성한다.

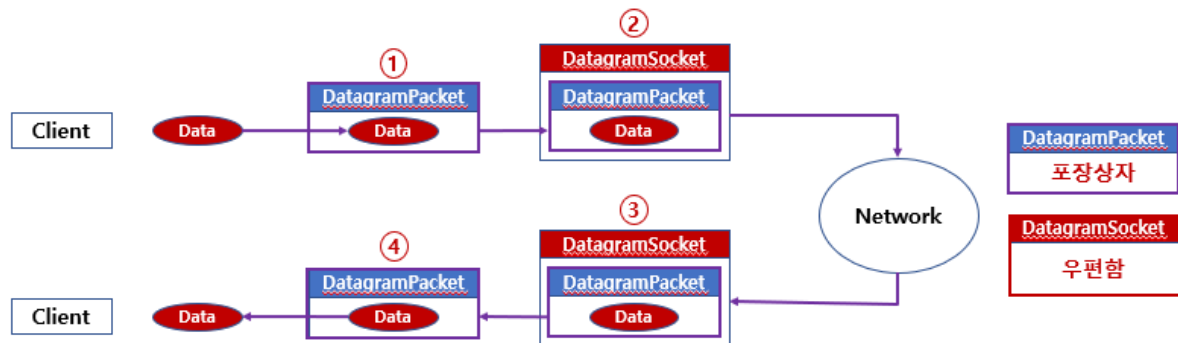
33-48. 쓰기 파일과 연결된 BufferedOutputStream 객체(bos)를 생성하고, DataInputStream 객체(dis)를 통해 읽은 읽은 길이와 읽은 데이터 정보를 파일에 쓴다. 데이터의 길이가 -1이 도착하면 반복문을 탈출하고 "파일 수신 완료"를 출력한 뒤 DataOutputStream 객체(dos)를 통해 "파일 전송 완료" 문자열을 출력한다.

UDP 통신

이번에는 비연결 지향 프로토콜인 UDP(User Datagram Protocol) 통신을 자바로 구현하는 방법에 대해서 알아보자.

UDP 통신 메커니즘

UDP 통신은 다음의 과정으로 진행된다.



UDP 통신이 이루어지는 메커니즘

- ①: 보낼 데이터를 `DatagramPacket`에 담는다(최대 65,508byte=65536-20(IP header)-8(UDP header)).
- ②: `DatagramPacket`을 원격지 `DatagramSocket`으로 전송한다(`DatagramSocket`의 `send()`).
- ③: `DatagramSocket`으로부터 `DatagramPacket`을 꺼낸다(`DatagramSocket`의 `receive()`). 수신 시에는 비어 있는 `DatagramPacket`을 하나 만들어 옮겨 담는다.
- ④: `DatagramPacket`으로부터 데이터를 꺼낸다.

▶ 보내고자 하는 데이터의 크기가 65508보다 큰 경우 여러 개의 `DatagramPacket`으로 나누어 전송한다.

먼저 보내고자 하는 데이터를 `DatagramPacket`으로 포장한다. `DatagramPacket`은 개념적으로 원격지로 데이터를 보내기 위해 포장하는 택배 상자같은 개념이다. 하나의 `DatagramPacket`은 헤더를 포함하여 최대 65,536byte로 한정되어 있다. 헤더에는 IP 헤더(20byte), UDP헤더(8byte)가 포함되기 때문에 실질적으로 하나에 `DatagramPacket`에 포함될 수 있는 최대 데이터 크기는 65,508byte가 된다. 따라서 만일 보내고자 하는 데이터가 65,508byte보다 크다면 여러 개의 `DatagramPacket`으로 나누어 전송해야 한다. `DatagramPacket`으로 데이터가 포장되면 자신의 `DatagramSocket` 객체의 `send()` 메서드를 이용하여 원격지의 `DatagramSocket`으로 `DatagramPacket`을 전송한다. 이후 수신 측의 `DatagramSocket`으로 `DatagramPacket`이 전달되면 해당 `DatagramPacket`에서 데이터를 꺼냄으로써 데이터가 전달된다. 이 과정은 택배 상자를 내 앞

우편함에 넣어 보내면 보내고자 하는 원격지의 우편함으로 배달되는 것과 같은 원리이다. DatagramSocket으로부터 DatagramPacket를 꺼낼 때는 DatagramSocket의 receive() 메서드가 사용된다. 이 과정은 우리가 알고 있는 택배를 보내고 받는 과정과 일치하지 않아 주의 깊게 볼 필요가 있다. 우리가 집앞 우편함에 택배 상자(DatagramPacket)가 도착하면 받은 택배 상자에서 물건(Data)을 꺼내지만, DatagramSocket에 DatagramPacket이 도착했을 때는 비어 있는 DatagramPacket을 하나 만들고 거기에 받은 DatagramPacket 내의 데이터를 옮겨담는다. 이후 옮겨담은 DatagramPacket에서 Data를 꺼내 사용하는 것이다. UDP 통신은 택배가 전송되는 과정과 개념적으로 잘 일치하는데, 유일하게 동일하게 적용되지 않는 부분이 바로 도착한 데이터를 꺼내오는 과정이다. 이 부분에 대해서는 이후 DatagramSocket의 메서드의 설명 과정에서 다시 한번 언급하겠다.

DatagramPacket 객체 생성

그럼 먼저 UDP 통신에서 데이터를 포장하는 클래스인 DatagramPacket에 대해서 알아보자. DatagramPacket은 하나의 패킷당 최대 65,508byte 크기를 가지는 byte[] 타입의 데이터를 포함할 수 있다. DatagramPacket의 객체는 아래와 같이 다양한 생성자를 이용하여 생성할 수 있다.

DatagramPacket 클래스의 생성자

생성자	동작
DatagramPacket(byte[] buf, int length)	주소 없이 단순히 데이터만 저장하는 패킷 (일반적으로 수신측에서 수신한 패킷을 저장하는데 사용)
DatagramPacket(byte[] buf, int offset, int length)	buf[]: 데이터 바이트 배열, offset: 시작 오프셋, length: 데이터의 길이
DatagramPacket(byte[] buf, int length, InetAddress address, int port)	데이터의 정보에 추가하여 수신측 주소 정보를 InetAddress와 port 번호로 추가한 패킷(택배 포장지에 주소를 써 놓는 개념)

DatagramPacket(byte[] buf, int offset, int length, InetAddress address, int port)	buf[]: 데이터 바이트 배열, offset: 시작 오프셋, length: 데이터의 길이
DatagramPacket(byte[] buf, int length, SocketAddress address)	데이터의 정보에 추가하여 수신측 주소 정보를 SocketAddress (InetAddress+Port)로 추가한 패킷(택배 포장지에 주소를 써 놓는 개념)
DatagramPacket(byte[] buf, int offset, int length, SocketAddress address)	buf[]: 데이터 바이트 배열, offset: 시작 오프셋, length: 데이터의 길이

이 생성자들은 공통적으로 포함 데이터를 의미하는 byte[] 타입의 배열을 입력매개변수로 가진다. 각 생성자들은 크게 3가지 형태로 나눌 수 있는데 도착지 주소가 없는 경우, 도착지 주소가 InetAddress와 포트로 구성된 경우, 그리고 SocketAddress로 지정된 경우이다. 두 번째와 세 번째는 결국 IP와 포트를 포함하고 있다는 의미이니 결국 포장지에 주소를 쓴 경우와 쓰지 않은 경우로 나눌 수 있을 것이다. length는 포함된 데이터의 길이이고 offset은 배열에서 데이터의 시작위치를 나타낸다.

DatagramPacket의 주요 메서드

다음은 DatagramPacket 클래스의 주요 메서드를 나타낸다.

DatagramPacket 클래스의 주요 메서드

리턴 타입	메서드명	동작
InetAddress void	getAddress() setAddress(InetAddress iaddr)	원격지 IP 주소 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 (패킷 당 전송 가능한 최대 raw 데이터=65,508 bytes)
int int void	getLength() getOffset() setLength(int length)	데이터 길이 읽기/설정 및 오프셋 읽기: 패킷에 포함되는 데이터의 길이 읽기 및 쓰기 오프셋 읽기

DatagramPacket은 택배 상자의 개념이라고 했다. 생성자에서도 본 것처럼 일반적으로 DatagramPacket에는 도착지(원격지)의 주소 정보를 포함하고 있다.

▶ 도착지(원격지) 주소 정보를 포함하고 있지 않은 Datagram의 경우 DatagramSocket의 connect() 메서드를 통해서만 전달 가능하며, 이에 대해서는 뒷부분에서 자세히 다룬다.

택배 상자에 도착지 주소를 쓰는 것과 같은 원리이다. getAddress()와 setAddress()는 DatagramPacket에 포함되어 있는 원격지의 IP 주소를 읽거나 설정하는 메서드이고, 비슷하게 getPort()와 setPort()는 원격지의 포트 정보를 읽거나 쓰는 메서드이다. IP 주소와 포트 정보를 한꺼번에 SocketAddress 타입으로 읽거나 쓰는 getSocketAddress(), setSocketAddress() 메서드도 제공한다. 패킷에서 데이터를 꺼낼 때는 getData() 메서드를 사용하며 패킷에 데이터를 넣을 때는 setData() 메서드를 사용한다. 이때 오프셋(offset)과 길이(length)를 함께 전달하여 전체 데이터의 일부분만을 가져올 수도 있고, 설정된 오프셋과 길이 정보를 가져오고 설정하는 getOffset(), getLength(), setLength() 메서드도 제공한다.

[Do it 실습] UDP 통신을 위한 DatagramPacket 클래스의 활용

DatagramPacketObject.java

```

01 package sec03_udpcommunication.EX01_DatagramPacketObject;
02 import java.net.DatagramPacket;
03 import java.net.InetAddress;
04 import java.net.InetSocketAddress;
05 import java.net.UnknownHostException;
06
07 public class DatagramPacketObject {
08     public static void main(String[] args) {
09         // 1. 데이터 생성
10         // byte[] : 65536 bytes(64Kbytes) -20byte(IP 헤더) -8 (UDP 헤더) =
11         65508byte
12         byte[] buf = "UDP-데이터그램패킷".getBytes();
13         // 2. DatagramPacket 객체 생성
14         // 2-1. 수신지 주소 미포함 패킷
15         DatagramPacket dp1 = new DatagramPacket(buf, buf.length);
16         DatagramPacket dp2 = new DatagramPacket(buf, 4, buf.length-4);
17         // 2-2. 수신지 정보(IP, Port) 포함 패킷
18         DatagramPacket dp3=null;
19         DatagramPacket dp4=null;
20         try {
21             dp3 = new DatagramPacket(buf, buf.length,

```

```

22         InetAddress.getByNames("localhost"), 10000);
23         dp4 = new DatagramPacket(buf, 4, buf.length-4,
24             InetAddress.getByNames("localhost"), 10000);
25     } catch (UnknownHostException e) { e.printStackTrace(); }
26     // 2-3. 수신지 정보(IP, Port) 포함 패킷
27     DatagramPacket dp5 = new DatagramPacket(buf, buf.length,
28         new InetSocketAddress("localhost", 10000));
29     DatagramPacket dp6 = new DatagramPacket(buf, 4, buf.length-4,
30         new InetSocketAddress("localhost", 10000));
31     // 3. DatagramPacket method
32     System.out.println("원격지 IP : " + dp1.getAddress()); //null
33     System.out.println("원격지 Port : " + dp1.getPort()); //-1
34     //System.out.println(dp1.getSocketAddress());
35     //IllegalArgumentException 예외
36     System.out.println("원격지 IP : " + dp3.getAddress());
37     //localhost/127.0.0.1
38     System.out.println("원격지 Port : " + dp3.getPort()); //10000
39     System.out.println("원격지 IP : " + dp3.getSocketAddress());
40     //localhost/127.0.0.1 : 10000
41     System.out.println();
42     System.out.println("포함데이터 : "+new String(dp1.getData()));
43     //UDP-데이터그램패킷
44     System.out.println("포함데이터 : "+new String(dp2.getData()));
45     //UDP-데이터그램패킷
46     System.out.println("포함데이터 : "+new String(dp2.getData(),
47         dp2.getOffset(), dp2.getLength()));
48     //데이터그램패킷
49     dp1.setData("안녕하세요".getBytes());
50     System.out.println("포함데이터 : "+new String(dp1.getData()));
51     //안녕하세요
52 }
53 }

```

실행 결과

원격지 IP : null

원격지 Port : -1


```
원격지 IP : localhost/127.0.0.1
원격지 Port : 10000
원격지 IP : localhost/127.0.0.1:10000

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

11-15. `getBytes()`를 이용하여 `String` → `byte[]`로 변환 뒤 `DatagramPacket`의 생성자로 전달하여 `DatagramPacket` 객체를 생성한다. 생성자의 매개변수에는 `byte[]` 타입의 데이터 이외에 데이터의 길이가 전달(`dp1`)되거나 오프셋(`offset`)과 데이터 길이가 함께 전달(`dp2`)된다(`dp1`과 `dp2` 객체는 원격지 정보를 포함하지 않음).

20-23. `dp1`과 `dp2`가 가지는 정보에 원격지 주소와 포트 번호, 즉 `DatagramPacket`이 전달될 정보를 포함하는 객체(`dp3`, `dp4`)를 생성한다(이때 원격지 주소와 포트 번호는 `InetAddress` 타입과 `int` 타입으로 각각 전달).

26-29. `dp3`와 `dp4`와 동일한 정보를 가지며 원격지 주소와 포트 번호를 `InetSocketAddress`로 한번에 전달하여 객체(`dp5`, `dp6`)를 생성한다.

31-37. `DatagramPacket`으로부터 원격지의 주소와 포트 번호를 각각 `InetAddress` 타입과 `int` 타입으로 가져올 수도 있고, 이 두 정보를 모두 포함하는 `InetSocketAddress` 타입으로 한번에 가져올 수도 있다. 원격지 정보를 포함하지 않는 `DatagramPacket`의 IP와 포트 번호를 출력하면 각각 `null`과 `-1`이 출력된다(이 경우 `getSocketAddress()` 메서드로 `InetSocketAddress`를 가져오려고 하면 `IllegalArgumentException` 예외 발생).

39-44. `DatagramPacket`으로부터 데이터를 추출하면 `DatagramPacket` 객체 생성 시 넘겨주었던 전체 `byte[]` 배열이 리턴되며, 오프셋 정보와 길이 정보를 가져와 해당 부분만을 발췌하여 사용한다.

45-46. `DatagramPacket`에 포함되어 있는 기존의 데이터를 `setData()` 메서드로 변경할 수 있다.

DatagramSocket 객체 생성

앞서 잠시 언급한 것처럼 DatagramSocket 클래스는 UDP 통신에서 DatagramPacket을 네트워크를 통해 전송하거나 수신하는 클래스로 우편함과 비슷한 개념이다. 쉽게 말해 송신측에서 데이터를 포장(DatagramPacket)하여 송신측 집앞의 우편함(송신측 DatagramSocket)에 넣으면 네트워크를 통해 수신측 집앞의 우편함(수신측 DatagramSocket)까지 배달되는 것이다. 수신측 DatagramSocket은 도착한 DatagramPacket을 바인딩된 포트로 전달한다. 개념적으로 우리 집앞까지 도착한 택배(DatagramPacket)가 내 방문 앞(바인딩된 포트)까지 배달된다는 것이다. 따라서 DatagramSocket에는 일반적으로 바인딩된 포트 정보가 포함된다.

DatagramSocket 클래스의 객체 생성에는 다음의 생성자들을 사용한다.

DatagramSocket 클래스의 생성자

생성자	동작
DatagramSocket()	기본 생성자로 객체를 생성하는 경우 객체 생성 호스트에 가용한 port에 바인딩된 DatagramSocket() 객체 생성
DatagramSocket(int port)	입력매개변수로 전달된 포트로 바인딩된 DatagramSocket() 객체 생성 (DatagramSocket에 DatagramPacket이 도착하면 바인딩된 Port로 전달)
DatagramSocket(int port, InetAddress laddr)	매개변수로 전달된 InetAddress의 port에 바인딩된 DatagramSocket 객체 생성
DatagramSocket(SocketAddress bindaddr)	매개변수로 전달된 SocketAddress에 바인딩된 DatagramSocket 객체 생성

첫 번째 생성자인 기본 생성자를 사용하여 DatagramSocket()을 생성하는 경우에는 객체 생성 호스트에서 가용한 포트가 임의로 지정되어 바인딩된다. 즉, 이 DatagramSocket에 도착한 DatagramPacket은 임의로 지정한 포트에 전달된다는 것이다. 나머지 생성자들은 바인딩하는 포트를 직접 지정하며, 추가적으로 IP 주소를 지정할 수도 있다. 당연히 자신에게 도착한 데이터이니 일반적으로는 IP 주소가 수신측 주소이겠지만 멀티캐스팅의 경우 특정 멀티캐스트 주소가 들어갈 수도 있다. 이들 생성자를 통해 DatagramSocket 객체를 생성하는 과정에서 SocketException 예외를 처리해주어야 한다. 이는 바인딩 과정이 실패한 경우 발생한다.

▶ 바인딩이 실패하는 가장 대표적인 경우는 다른 프로세스에서 사용 중인 포트에 바인딩을 하고자 하는 경우이다.

DatagramSocket 주요 메서드

DatagramSocket 클래스의 주요 메서드는 다음과 같다.

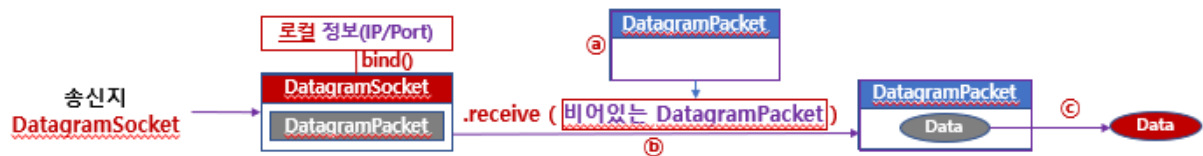
DatagramSocket 클래스의 주요 메서드

리턴 타입	메서드명	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: 바인딩된 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) (미설정 시 송수신 버퍼 크기는 65,536 byte)
int void	getSoTimeout() setSoTimeout(int timeout)	수신 리스닝 시간: DatagramPacket의 수신을 기다리는 시간 읽기 및 설정 (timeout=0이면 무한 대기) (시간 완료 시 SocketTimeoutException: 발생)

bind()는 DatagramPacket 도착 시 해당 패킷을 전달할 IP와 포트 정보를 지정하는 메서드로 말 그대로 바인딩을 수행한다. DatagramPacket을 원격지로 전송할 때는 send() 메서드를 사용하는데 이때 DatagramPacket 또는 DatagramSocket에 포함된 원격지 주소 정보를 사용한다. 도착한 DatagramPacket을 꺼낼 때는 receive() 메서드를 사용한다. 앞에서도 언급했던 바와 같이 데이터를 꺼낼 때 특이한 점은 도착한 DatagramPacket을 바로 꺼내는 것이 아니라 비워져있는 DatagramPacket 객체를 생성한 후 내부에 데이터를 옮겨 담는 방식을 사용한다. 즉 우편함에 택배 상자가 도착하면 새 택배 상자를 준비하여 도착한 택배 상자 속의 데이터만 새 상자로 옮겨

담는 식이다. 다음 그림과 표는 수신지에 도착한 DatagramPacket을 receive() 메서드를 통해 꺼내오는 과정을 나타낸다.



수신지 DatagramSocket에 도착한 DatagramPacket으로부터 Data를 꺼내오는 과정

- ㉠ 비어 있는 DatagramPacket을 생성한다(최대 65508 bytes=65536-20(IP header)-8(UDP header).
- ㉢ 수신한 DatagramPacket을 비어 있는 DatagramPacket으로 복사한다(DatagramSocket의 receive(비어 있는 DatagramPacket)).
- ㉣ 수신한 DatagramPacket으로부터 데이터를 추출한다(DatagramPacket의 **getData()**, **getOffset()**, **getLength()**).

자신의 DatagramSocket에 도착한 DatagramPacket을 꺼내기 위해서는 먼저 비어 있는 DatagramPacket 객체를 생성한 후 receive() 메서드의 매개변수로 넘겨준다. 이렇게 하면 넘겨준 DatagramPacket으로 데이터가 옮겨담아지며 이후 getData(), getOffset(), getLength() 등의 메서드를 이용하여 데이터를 꺼내는 것이다. 이상의 과정을 코드로 나타내면 다음과 같다.

수신지 DatagramSocket에 도착한 DatagramPacket으로부터 Data를 추출 예

```
// DatagramSocket 객체 생성
DatagramSocket ds = new DatagramSocket(new InetSocketAddress("localhost", 10002));
// 비어 있는 DatagramPacket 생성(최대 크기 지정)
byte[] receivedData = new byte[65508];
DatagramPacket dp = new DatagramPacket(receivedData, receivedData.length);
// DatagramSocket 객체의 receive() 메서드의 매개변수로 비어 있는 DatagramPacket 전달
ds.receive(dp);
// 데이터 꺼내기
byte[] data = dp.getData();
```

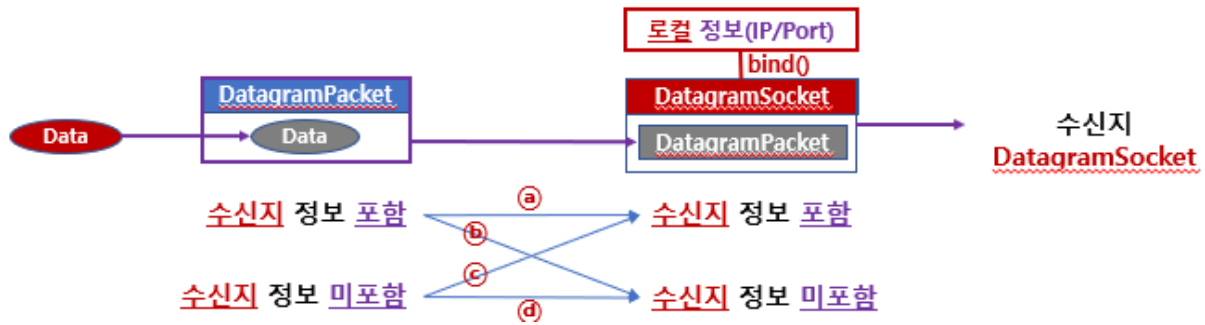
먼저 DatagramSocket 객체(ds)를 생성했다. 이후 이 객체에 DatagramPacket이 도착하면 도착한 데이터를 옮겨 담을 비어 있는 DatagramPacket 객체(dp)를 생성했다. 데이터의 크기는 DatagramPacket이 담을 수 있는 최대 크기를 지정했다. 이제 마지막으로 receive(dp)와 같이 실행하면 비어 있는 DatagramPacket 객체에 데이터가 옮겨지며, 이후 getData() 메서드를 통해 최종적으로 byte[] 데이터를 꺼내올 수가 있는 것이다.

다음 메서드인 connect()와 disconnect() 메서드는 눈여겨 볼 필요가 있다. UDP 통신 방식은 비연결 통신 방식이기 때문에 전송 데이터는 서로 다른 경로로 전송될 수 있다. 하지만 connect() 메서드를 이용하면 연결 지향성 통신처럼 송신측과 수신측의 DatagramSocket(우편함)을 직접 연결하는 것이다.



connect() 메서드를 이용한 송신지와 수신지 DatagramSocket 연결

이 경우 DatagramPacket에 송신지 주소를 넣지 않아도 송신측 우편함에 넣기만 하면 연결된 수신측 우편함으로 DatagramPacket이 전송된다. 마치 특정 수신지 우편함으로만 배달되는 전용 우체통이라고 생각하면 된다. 이렇게 connect() 메서드를 사용하여 송신측과 수신측을 직접 연결하는 경우에는 다른 주소로 전송되는 DatagramPacket은 당연히 전송할 수 없게 된다. 따라서 하나의 DatagramPacket이 원격지로 전달되기 위해 사용하는 주소의 모든 조합은 다음과 같이 4가지 경우가 있을 수 있다.



UDP 방식으로 송신 시 원격지 전송을 위해 사용하는 주소 정보의 4가지 경우

이들 각 경우에 대한 동작은 다음과 같다.

- ㉑ DatagramPacket 및 DatagramSocket에 연결된 수신지 주소로 전송한다. 이때 반드시 DatagramPacket의 주소와 DatagramSocket 연결 주소가 같아야 한다. 불일치 시 IllegalArgumentException가 발생한다.
- ㉒ DatagramPacket에 포함된 주소로 전송한다(비연결 지향성, DatagramSocket의 **send()**).
- ㉓ DatagramSocket에 연결(connect)되어 있는 수신지 주소로 전송한다(원격지 주소 연결: DatagramSocket의 **connect()** + DatagramSocket의 **send()**).
- ㉔ 전송 불가(NullPointerException)

▶ DatagramSocket에 원격지 주소정보가 있는 경우는 connect() 메서드를 이용하여 원격지 DatagramSocket이 연결된 경우이다.

먼저 ㉑의 경우는 DatagramPacket에도 원격지 주소 정보가 있고, DatagramSocket에도 원격지 주소 정보가 있는 경우이다. 이때에는 반드시 이들 2개의 원격지 주소가 반드시 일치해야 하며, 불일치시 IllegalArgumentException이 발생한다. ㉒의 경우는 가장 일반적인 경우로 DatagramPacket에는 원격지 주소가 있고, DatagramSocket에는 원격지 주소가 없는 경우이다. 이때에는 당연히 DatagramPacket에 포함된 원격지 주소로 전송된다. ㉓의 경우는 DatagramSocket에만 원격지 정보가 있는 경우로 이때는 connect() 메서드로 송신지와 수신지의 DatagramSocket이 연결되어 있는 상태이다. 따라서 DatagramPacket이 원격지 주소를 포함하고 있지 않아도 연결된 송신지로 전송된다. 마지막 ㉔의 경우는 DatagramPacket과 DatagramSocket

모두 원격지 주소 정보를 포함하고 있지 않으니 전송 자체가 불가능하게 된다. 두 DatagramSocket간의 연결은 disconnect() 메서드를 통해 해제할 수 있다. 그 밖에 원격지 주소 정보, 로컬 주소 정보, 송수신 버퍼 크기의 읽기 및 설정, 그리고 수신 대기 시간의 설정을 위한 메서드가 제공된다.

[Do it 실습] UDP 통신을 위한 DatagramSocket 클래스의 활용

DatagramSocketObject.java

```
01 package sec03_udpcommunication.EX02_DatagramSocketObject;
02 import java.io.IOException;
03 import java.net.DatagramPacket;
04 import java.net.DatagramSocket;
05 import java.net.InetAddress;
06 import java.net.InetSocketAddress;
07 import java.net.SocketException;
08 import java.net.UnknownHostException;
09
10 public class DatagramSocketObject {
11     public static void main(String[] args) {
12         // 1. DatagramSocket 객체생성
13         // (모든 DatagramSocket은 바인딩 되어 있어야 한다)
14         DatagramSocket ds1 = null;
15         DatagramSocket ds2 = null;
16         DatagramSocket ds3 = null;
17         DatagramSocket ds4 = null;
18         try {
19             ds1 = new DatagramSocket(); //비어 있는 포트로 자동 바인딩
20             ds2 = new DatagramSocket(10000);
21             ds3 = new DatagramSocket(10001,
22 InetAddress.getByNames("localhost"));
23             ds4 = new DatagramSocket(new InetSocketAddress("localhost",
24 10002));
25         } catch (SocketException | UnknownHostException e) {
26             e.printStackTrace();
27         }
```



```

28 // 2. DatagramSocket 메서드
29 // 2-1. 소켓의 바인딩 정보
30 System.out.println("ds1의 바인딩 정보 : " +
31                     ds1.getLocalAddress()+":"+ds1.getLocalPort());
32 System.out.println("ds2의 바인딩 정보 : " +
33                     ds2.getLocalAddress()+":"+ds2.getLocalPort());
34 System.out.println("ds3의 바인딩 정보 : " +
35                     ds3.getLocalAddress()+":"+ds3.getLocalPort());
36 System.out.println("ds4의 바인딩 정보 : " +
37                     ds4.getLocalAddress()+":"+ds4.getLocalPort());
38 // 2-2. 원격지 정보 저장 (connect() 메서드 사용)
39 System.out.println("원격지 주소 정보 : "+
40                     ds4.getInetAddress()+":"+ds4.getPort());
41 try {
42     ds4.connect(new InetAddress("localhost", 10003));
43 } catch (SocketException e) {
44     e.printStackTrace();
45 }
46 System.out.println("원격지 주소 정보 : "+
47                     ds4.getInetAddress()+":"+ds4.getPort());
48 ds4.disconnect();
49 System.out.println();
50 // 2-3. send()/connect(), receive()
51 // 전송 데이터그램 패킷 2개 생성
52 byte[] data1 = "수신지 주소가 없는 데이터그램 패킷".getBytes();
53 byte[] data2 = "수신지 주소가 있는 데이터그램 패킷".getBytes();
54 DatagramPacket dp1 = new DatagramPacket(data1, data1.length);
55 DatagramPacket dp2 = new DatagramPacket(data2, data2.length,
56     new InetAddress("localhost", 10002));
57 try {
58     /*불가능: 소켓-연결(connect) 정보 없음 + 패킷-수신지 정보 없음
59     ds1.send(dp1);
60     ds2.send(dp1);
61     ds3.send(dp1); */
62     ds1.connect(new InetAddress("localhost", 10002));
63     ds2.connect(new InetAddress("localhost", 10002));

```

```

64         ds3.connect(new InetSocketAddress("localhost", 10002));
65         // 가능: 소켓-연결(connect) 정보 있음 + 패킷-수신지 정보 없음
66         ds1.send(dp1);
67         ds2.send(dp1);
68         ds3.send(dp1);
69         ds1.disconnect();
70         ds2.disconnect();
71         ds3.disconnect();
72         // 가능: 소켓-연결(connect) 정보 없음 + 패킷-수신지 정보 있음
73         ds1.send(dp2);
74         ds2.send(dp2);
75         ds3.send(dp2);
76         // 소켓-연결(connect) 정보 있음 + 패킷-수신지 정보 있음
77         // 가능: 두 주소 일치: 소켓-연결(connect) 주소 = 패킷-수신지 주소
78         ds3.connect(new InetSocketAddress("localhost", 10002));
79         ds3.send(dp2);
80         ds3.disconnect();
81         /*불가능: 두 주소 불일치: 소켓-연결(connect) 주소 != 패킷-수신지 주
82 소
83         ds3.connect(new InetSocketAddress("localhost", 10003));
84         ds3.send(dp2); // IllegalArgumentException
85         ds3.disconnect(); */
86         // 데이터 수신
87         byte[] receivedData = new byte[65508];
88         DatagramPacket dp =
89             new DatagramPacket(receivedData, receivedData.length);
90         for(int i=0; i<7; i++) {
91             ds4.receive(dp);
92             System.out.print("송신자 정보 :
93 "+dp.getAddress()+":"+dp.getPort());
94             System.out.println(" : "+new String(dp.getData()).trim());
95         }
96         // 송수신 데이터 버퍼
97         System.out.println("송신버퍼 크기 : "+ ds1.getSendBufferSize());
98         System.out.println("수신버퍼 크기 : "+ ds1.getReceiveBufferSize());
99     } catch (IOException e) { e.printStackTrace(); }

```

	}
	}

실행 결과

ds1의 바인딩 정보 : 0.0.0.0/0.0.0.0:53397

ds2의 바인딩 정보 : 0.0.0.0/0.0.0.0:10000

ds3의 바인딩 정보 : /127.0.0.1:10001

ds4의 바인딩 정보 : /127.0.0.1:10002

원격지 주소 정보 : null:-1

원격지 주소 정보 : localhost/127.0.0.1:10003

송신자 정보 : /127.0.0.1:53397 : 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 : /127.0.0.1:53397 : 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

송신버퍼 크기 : 65536

수신버퍼 크기 : 65536

14-25. DatagramSocket의 기본 생성자로 객체를 생성하면 비어 있는 임의의 포트에 자동 바인딩된 객체가 생성되며, 생성자의 매개변수로 IP 주소를 포함하여 포트 정보를 넘겨주면 해당 포트에 바인딩된 DatagramSocket를 생성한다(원격지 주소 정보 없음).

28-35. 각 DatagramSocket이 바인딩된 IP 주소 및 포트 즉, 로컬 호스트에 대한 정보를 출력한다. 포트를 지정하지 않은 경우 비어 있는 포트 자동 지정되며 "localhost"는 루프백 주소 즉, 127.0.0.1을 가진다(원격지 주소 정보 없음).

37-46. 원격지 정보가 없는 DatagramSocket 객체의 원격지 주소 및 포트를 출력하면 각각 null과 -1이 출력되며, connect() 메서드를 이용하여 원격지 정보를 추가할 수 있다. 이 경우 disconnect()로 연결을 해제하기 전까지 두 DatagramSocket은 연결 상태를 유지한다.

50-69. 원격지 정보를 포함하지 않은 DatagramPacket 객체의 경우, 원격지 정보를 포함하지 않은 DatagramSocket의 send() 메서드로 전송 불가능하며, connect() 메서드를 통해 DatagramSocket에 원격지 정보 제공 후 send() 메서드로 전송할 수 있다. 전송 후에는 연결이 해제된다.

71-73. 원격지 정보를 포함하고 있는 DatagramPacket 객체는 원격지 정보를 포함하지 않은 DatagramSocket의 send() 메서드로 전송할 수 있다.

76-82. 원격지 정보를 포함하고 있는 DatagramPacket 객체를 connect() 메서드로 원격지 정보를 포함하고 있는 DatagramSocket의 send() 메서드로 전송시 두 원격지 정보는 반드시 일치해야 한다(불일치시 IllegalArgumentException 발생).

84-91. DatagramSocket으로부터 데이터 수신 시 비어 있는 DatagramPacket 객체(dp)을 생성하고 receive() 메서드의 매개변수로 넘겨주면, 수신된 데이터가 넘겨주었던 DatagramPacket 객체로 옮겨지며, 여기에서 송신자 정보, 실제 데이터 정보를 가져올 수 있다.

93-94. DatagramSocket 객체로부터 송신 및 수신 데이터 버퍼 크기를 가져와 출력한다(기본값은 65,536byte).

UDP 통신 예제 1. 클라이언트 간 텍스트 전송

첫 번째 UDP 통신 예제는 두 클라이언트 간 텍스트를 전송하는 예로 두 클라이언트는 DatagramSocket을 바인딩(각각 10000번과 20000번 포트에 바인딩)하고 연결(connect())없이 DatagramPacket에 수신측 주소를 넣어 전달하도록 했다.



UDP 통신을 이용한 클라이언트간 텍스트 전송

두 클라이언트를 직접 연결하지 않았으니 DatagramPacket에는 반드시 수신측 주소가 포함되어 있어야 한다. 예제의 동작 순서는 먼저 192.168.123.159 클라이언트가 DatagramPacket에 “안녕하세요”의 데이터와 함께 수신측 IP 주소(192.168.123.160)와 포트 번호(20000)를 넣어 DatagramSocket을 통해 전송하면 수신측은 받은 데이터를 바인딩된 20000 포트로 읽어들인다. 이후 같은 방식으로 이번에는 “반갑습니다”의 데이터를 DatagramPacket에 담아 192.168.123.159:10000 주소로 보내면 해당 IP의 DatagramSocket이 DatagramPacket을 수신하는 것이다. 각각의 데이터는 서로 다른 경로로 이동할 수 있으며 여러 개의 데이터를 연속적으로 보내는 경우 비연결성 UDP로 동작하기 때문에 수신 순서가 전송 순서와 다를 수 있다. 다만 일반적으로 전송 순서가 바뀌는 경우를 실험적으로 관찰하기는 쉽지 않다. 이 예제를 실행함에 있어 가장 먼저 클라이언트 A가 먼저 문자열을 전송할 때 클라이언트 B가 이를 받을 준비를 되어 있어야 한다. 따라서 **Client B 프로그램을 먼저 실행하고 Client A 프로그램을 실행해야** 한다.

[Do it 실습] UDP 통신 예제 1-1. 클라이언트 간 텍스트 전송 (비연결 통신) [Client A]

NonConnectedUDP_Text_ClientA.java

```

01 package sec03_udpcommunication.EX03_NonConnectedUDP_Text;
02 import java.io.IOException;
03 import java.net.DatagramPacket;
04 import java.net.DatagramSocket;
05 import java.net.InetSocketAddress;
06 import java.net.SocketException;
07
08 public class NonConnectedUDP_Text_ClientA {
09     public static void main(String[] args) {
10         System.out.println("<<ClientA>> - Text");
11         // 1.DatagramSocket 객체 생성(binding)
12         DatagramSocket ds = null;
13         try {
14             ds = new DatagramSocket(10000);
15         } catch (SocketException e) { e.printStackTrace();}
16         // 2. 전송데이터 생성 + DatagramPacket 생성
17         byte[] sendData = "안녕하세요".getBytes();

```

18	DatagramPacket sendPacket = new DatagramPacket(
19	sendData, sendData.length, new InetAddress("localhost", 20000));
20	// 3. 데이터그램 패킷 전송
21	System.out.println("송신 데이터 : " + new String(sendPacket.getData(),
22	0, sendPacket.getLength()));
23	try {
24	ds.send(sendPacket);
25	} catch (IOException e) {e.printStackTrace();}
26	// 4. 데이터그램패킷 수신
27	byte[] receivedData = new byte[65508];
28	DatagramPacket receivedPacket =
29	new DatagramPacket(receivedData, receivedData.length);
30	try {
31	ds.receive(receivedPacket);
32	} catch (IOException e) { e.printStackTrace();}
33	System.out.println("수신 데이터 : " +
34	new String(receivedPacket.getData(), 0, receivedPacket.getLength()));
35	}
36	}

실행 결과

<<ClientA>> - Text

송신 데이터 : 안녕하세요

수신 데이터 : 반갑습니다.

13-19. 10000번 포트에 바인딩된 DatagramSocket 객체를 생성하고, 127.0.0.1(localhost):20000 원격지 주소를 포함하는 DatagramPacket 객체를 생성한다.

21-25. DatagramSocket의 send() 메서드를 이용하여 원격지 주소를 포함하는 DatagramPacket을 전송한다.

27-34. 비어 있는 DatagramPacket 객체를 생성하고 DatagramSocket의 receive() 메서드의 매개변수로 넘겨주어 데이터를 수신한다. 이후 getData()와 getLength()를 이용하여 데이터 정보 추출하여 출력한다.

```

01 package sec03_udpcommunication.EX03_NonConnectedUDP_Text;
02 import java.io.IOException;
03 import java.net.DatagramPacket;
04 import java.net.DatagramSocket;
05 import java.net.SocketException;
06
07 public class NonConnectedUDP_Text_ClientB {
08     public static void main(String[] args) {
09         System.out.println("<<ClientB>> - Text");
10         // 1. DatagramSocket 객체 생성 (binding)
11         DatagramSocket ds = null;
12         try {
13             ds = new DatagramSocket(20000);
14         } catch (SocketException e) { e.printStackTrace();}
15         // 2. 데이터그램패킷 수신
16         byte[] receivedData = new byte[65508];
17         DatagramPacket receivedPacket =
18             new DatagramPacket(receivedData, receivedData.length);
19         try {
20             ds.receive(receivedPacket);
21         } catch (IOException e) { e.printStackTrace();}
22         System.out.println("수신 데이터 : " + new
23 String(receivedPacket.getData(), 0, receivedPacket.getLength()));
24         // 3. 전송데이터 생성 + DatagramPacket 생성
25         byte[] sendData = "반갑습니다.".getBytes();
26         DatagramPacket sendPacket = new DatagramPacket(sendData,
27             sendData.length, receivedPacket.getSocketAddress());
28         // 4. 데이터그램패킷 전송
29         System.out.println("송신 데이터 : " + new String(sendPacket.getData(),
30 0, sendPacket.getLength()));
31         try {
32             ds.send(sendPacket);
33         } catch (IOException e) {e.printStackTrace();}
34     }

```


우선 두 클라이언트의 IP 및 바인딩된 포트 정보는 앞의 예제와 동일하다. 우선 두 DatagramSocket을 connect() 메서드를 이용하여 연결했다. 이때 주의할 점은 connect() 메서드는 단방향 연결이기 때문에 양쪽 모두 송수신 과정을 연결하기 위해서는 각각 연결 과정을 거쳐야 한다. 연결된 이후 Client A는 파일 전송의 시작을 알리는 "/start" 문자열을 전송한 후 전송하고자 하는 파일의 데이터를 분할하여 DatagramPacket에 담아 전송한다. 전체 파일 전송이 끝나면 "/end" 문자열을 전송하여 모든 파일 내용이 전송되었음을 알린다. 이상의 모든 과정에서 파일 데이터를 포함하는 DatagramPacket에는 원격지 주소가 포함되지 않았다. 마지막으로 Client B는 파일 수신 완료 후 "파일 수신 완료"라는 문자열을 Client A에게 보냄으로서 모든 송수신은 종료된다. 이 예제 역시 Client A가 파일을 전송하기 전에 Client B는 이를 받을 준비가 완료되어야 한다. 따라서 **Client B → Client A 순서로 프로그램을 실행해야 한다.**

[Do it 실습] UDP 통신예제 2-1 : 클라이언트간 파일 전송 (연결 통신) [Client A]

ConnectedUDP_File_ClientA.java

```

01 package sec03_udpcommunication.EX04_ConnectedUDP_File;
02 import java.io.BufferedInputStream;
03 import java.io.File;
04 import java.io.FileInputStream;
05 import java.io.FileNotFoundException;
06 import java.io.IOException;
07 import java.net.DatagramPacket;
08 import java.net.DatagramSocket;
09 import java.net.InetSocketAddress;
10 import java.net.SocketException;
11
12 public class ConnectedUDP_File_ClientA {
13     public static void main(String[] args) {
14         System.out.println("<<ClientA>> - File");
15         // 1. DatagramSocket 생성 (binding 포함)
16         DatagramSocket ds = null;
17         try {
18             ds = new DatagramSocket(10000);

```

```

19         ds.connect(new InetSocketAddress("localhost", 20000));
20     } catch (SocketException e) { e.printStackTrace(); }
21     // 2. 파일 로딩
22     File file = new File("src\\sec03_udpcommunication\\files_clientA\\
23                                     ImageFileUsingUDP.jpg");
24     BufferedInputStream bis = null;
25     try {
26         bis = new BufferedInputStream(new FileInputStream(file));
27     } catch (FileNotFoundException e) { e.printStackTrace(); }
28     // 3. 데이터그램 패킷 전송
29     DatagramPacket sendPacket = null;
30     // 3-1. 파일이름 전송
31     String fileName = file.getName();
32     sendPacket = new DatagramPacket(fileName.getBytes(),
33                                     fileName.getBytes().length); //수신지 정보 없음
34     try {
35         ds.send(sendPacket);
36     } catch (IOException e) {e.printStackTrace(); }
37     // 3-2. 파일전송 시작을 알리는 사인 전송 (/start)
38     String startSign = "/start";
39     sendPacket = new DatagramPacket(startSign.getBytes(),
40                                     startSign.getBytes().length); //수신지 정보 없음
41     try {
42         ds.send(sendPacket);
43     } catch (IOException e) {e.printStackTrace(); }
44     // 3-3. 2048 사이즈로 나누어 실제 파일데이터 전송
45     int len;
46     byte[] filedata = new byte[2048]; //최대 65508이지만 2048로 나누어 전송
47     try {
48         while((len=bis.read(filedata)) !=-1) {
49             sendPacket = new DatagramPacket(filedata, len);
50             ds.send(sendPacket);
51         }
52     } catch (IOException e1) {
53         e1.printStackTrace();
54     }

```



```

55      // 3-4. 파일전송 끝을 알리는 사인 전송 (/end)
56      String endSign = "/end";
57      DatagramPacket sendPacket = new DatagramPacket(endSign.getBytes(),
58      endSign.getBytes().length); //수신지 정보 없음
59      try {
60          ds.send(sendPacket);
61      } catch (IOException e) {e.printStackTrace(); }
62      // 4. 데이터그램 텍스트 패킷 수신
63      byte[] receivedData = new byte[65508];
64      DatagramPacket receivedPacket = new DatagramPacket(receivedData,
65      receivedData.length);
66      try {
67          ds.receive(receivedPacket);
68      } catch (IOException e) {
69          e.printStackTrace();
70      }
71      System.out.println("수신데이터 : "+
72      new String(receivedPacket.getData()).trim());
73  }
74  }

```

실행 결과

<<ClientA>> - File

수신데이터 : 파일 수신 완료

16-20. 10000번 포트에 바인딩된 DatagramSocket 객체를 생성하고, connect() 메서드를 이용하여 127.0.0.1(localhost):20000 원격지 DatagramSocket과 연결을 수행한다.

22-36. 전송할 파일과 연결된 BufferedInputStream 객체를 생성하고, 파일 이름을 원격지 정보를 포함하지 않은 DatagramPacket 객체에 담아 send() 메서드로 전송한다(이 경우 원격지 정보는 DatagramSocket이 포함).

38-43. 파일 전송의 시작을 알리기 위해 "/start" 문자열을 전송한다.

45-54. BufferedInputStream 객체를 통해 파일로 부터 최대 2048byte를 반복적으로 읽어와 DatagramPacket에 읽은 데이터 길이와 함께 포함시켜 전송한다.

56-61. 파일 전송의 끝을 알리기 위해 "/end" 문자열을 전송한다.

63-72. 비어 있는 DatagramPacket을 생성하여 receive() 메서드로 데이터 수신 및 출력한다.

[Do it 실습] UDP 통신 예제 2-2. 클라이언트 간 파일 전송(연결 통신) [Client B]

ConnectedUDP_File_ClientB.java

```
01 package sec03_udpcommunication.EX04_ConnectedUDP_File;
02 import java.io.BufferedOutputStream;
03 import java.io.File;
04 import java.io.FileNotFoundException;
05 import java.io.FileOutputStream;
06 import java.io.IOException;
07 import java.net.DatagramPacket;
08 import java.net.DatagramSocket;
09 import java.net.InetSocketAddress;
10 import java.net.SocketException;
11
12 public class ConnectedUDP_File_ClientB {
13     public static void main(String[] args) {
14         System.out.println("<<ClientB>> - File");
15         // 1. DatagramSocket 생성 (binding 포함)
16         DatagramSocket ds = null;
17         try {
18             ds = new DatagramSocket(20000);
19             ds.connect(new InetSocketAddress("localhost", 10000));
20         } catch (SocketException e) { e.printStackTrace(); }
21         // 2. 데이터그램 패킷 수신
22         byte[] receivedData = null;
23         DatagramPacket receivedPacket = null;
24         // 2-1. 파일 이름 수신
25         receivedData = new byte[65508];
26         receivedPacket = new DatagramPacket(receivedData, receivedData.length);
27         try {
28             ds.receive(receivedPacket);
29         } catch (IOException e) { e.printStackTrace(); }
30         String fileName = new String(receivedPacket.getData(), 0,
```



```

31                                     receivedPacket.getLength());
32     File file =
33         new File("src\\sec03_udpcommunication\\files_clientB\\"+fileName);
34     BufferedOutputStream bos =null;
35     try {
36         bos = new BufferedOutputStream (new FileOutputStream(file));
37     } catch (FileNotFoundException e) { e.printStackTrace(); }
38     System.out.println("수신파일이름 : "+fileName);
39     // 2-2. 시작태그와 끝태그를 기준으로 파일 수신
40     String startSign = "/start";
41     String endSign = "/end";
42     receivedData = new byte[65508];
43     receivedPacket = new DatagramPacket(receivedData, receivedData.length);
44     try {
45         ds.receive(receivedPacket);
46         if(new String(receivedPacket.getData(), 0,
47             receivedPacket.getLength()).equals(startSign)) {
48             while(true) {
49                 ds.receive(receivedPacket);
50                 if(new String(receivedPacket.getData(), 0,
51                     receivedPacket.getLength()).equals(endSign))
52                     break;
53                 bos.write(receivedPacket.getData(), 0,
54 receivedPacket.getLength());
55                 bos.flush();
56             }
57         }
58         bos.close();
59     } catch (IOException e) { e.printStackTrace(); }
60     // 4. 파일 전송 완료 메시지 응답
61     byte[] sendData = "파일 수신 완료".getBytes();
62     DatagramPacket sendPacket =
63         new DatagramPacket(sendData, sendData.length); //수신지 정보 없음
64     try {
65         ds.send(sendPacket);
66     } catch (IOException e) {e.printStackTrace(); }

```

67	}
	}

실행 결과

<<ClientB>> - File

수신파일이름 : ImageFileUsingUDP.jpg

14-20. 20000번 포트에 바인딩된 DatagramSocket 객체를 생성하고, connect() 메서드를 이용하여 127.0.0.1(localhost):10000 원격지 DatagramSocket과 연결을 수행한다.

25-38. 비어 있는 DatagramPacket을 생성하여 receive() 메서드로 처음으로 전송한 파일 이름을 수신하고, 쓰기 파일과 연결된 BufferedInputStream 객체를 생성한다.

40-58. 수신된 데이터가 "/start"인 경우 반복적으로 데이터를 수신하여 파일을 기록하고, 수신 데이터가 "/end"인 경우 반복문을 탈출한다.

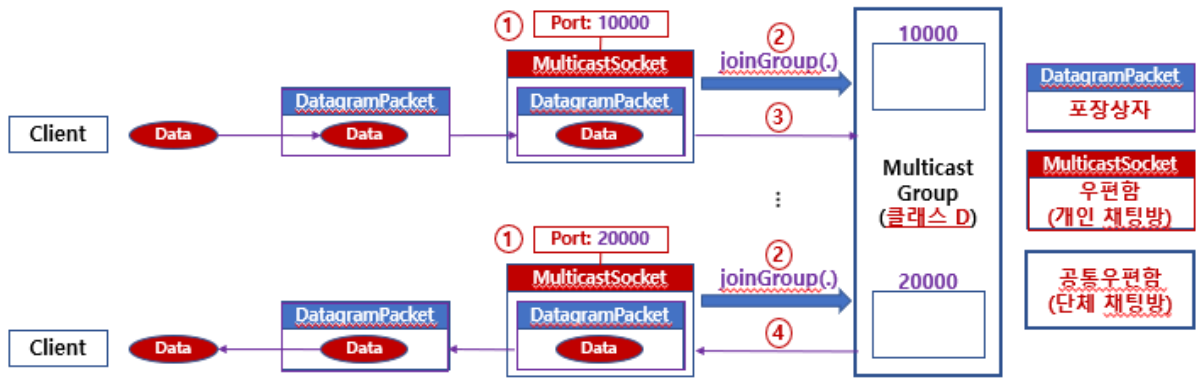
60-65. "파일수신완료" 문자열을 DatagramPacket 객체에 담아 DatagramSocket 메서드 send()로 전송한다(이 경우 원격지 정보는 DatagramSocket이 포함).

Multicast 통신

이번에는 멀티캐스팅 IP에 가입(join)한 호스트들간에 데이터를 송수신하는 멀티캐스팅에 대해서 알아보자.

Multicast 통신 메커니즘

멀티캐스트 통신은 UDP 통신에 기반하고 있어 데이터를 기본적으로 DatagramPacket으로 패키징하여 전송하는 점은 UDP 통신과 동일하다. 멀티캐스트 통신이 UDP 통신과 다른 점은 패키징된 DatagramPacket을 전송할 때 DatagramSocket 대신 MulticastSocket을 사용한다는 점이다. MulticastSocket 클래스에는 특정 멀티캐스트 IP 주소에 가입하는 기능(joinGroup())이 포함되어 있다. 전체적인 통신 메커니즘을 살펴보면 다음과 같다.



Multicast 통신 메커니즘

- ① 각 클라이언트는 Multicast를 지원하는 MulticastSocket을 생성한다. 이때 바인딩 정보를 필수로 포함한다. 바인딩 port 미지정 시 비어 있는 port를 자동으로 바인딩한다.
- ② 생성한 MulticastSocket을 멀티캐스팅 IP(클래스 D)를 가지는 Multicast Group에 조인한다(joinGroup(InetAddress)).
- ③ 송신: 보낼 데이터를 **DatagramPacket**에 담아 MulticastSocket을 통해 보낸다(send()).
- ④ 수신: Multicast Group에 패킷이 전달되면 패킷의 수신 포트에 조인된 모든 **Client**에게 패킷을 전송한다(receive()).

먼저 각각의 클라이언트들은 멀티캐스트를 지원하는 MulticastSocket 객체를 생성하며, 생성된 MulticastSocket은 필수적으로 바인딩 정보를 포함하고 있어야 한다. 이들 MulticastSocket은 클래스 D의 주소를 가지는 멀티캐스트 그룹에 joinGroup() 메서드를 이용하여 가입할 수 있다.

▶ 클래스 D의 IP 주소의 범위는 224.0.0.0 ~ 239.255.255.255이다.

이렇게 가입이 되면 이후에 멀티캐스트 그룹으로 전달되는 모든 DatagramPacket 중 자신이 바인딩된 포트에 전달되는 패킷은 수신이 가능해지는 것이다. 개념적으로 설명하면 멀티캐스트 그룹은 채팅 프로그램이고, 프로그램내 여러 바인딩 정보는 단체 채팅방(단톡방)의 개념이다. 즉 10000번 포트에 바인딩된 MulticastSocket이 멀티캐스트 그룹에 가입하면 10000번 단톡방에 가입한 것과 동일한 개념이다. 따라서 이후 10000번 단톡방에 쓰여지는 모든 글을 읽을 수 있게

되는 것이다. 각 클라이언트는 전송하고자 하는 데이터를 DatagramPacket에 포장하여 MulticastSocket을 통해 멀티캐스트 그룹으로 전송한다. 이때 전송되는 DatagramPacket에 포함된 포트 번호(바인딩 번호)에 따라 해당 DatagramPacket의 수신 클라이언트들이 결정되는 것이다. 만일 송신 클라이언트의 MulticastSocket이 바인딩된 포트 번호가 자신이 보낸 DatagramPacket의 원격지 주소의 포트와 동일하다면 자신도 자신이 보낸 DatagramPacket을 전달받게 될 것이다. 이후 데이터를 꺼내는 과정은 UDP와 동일하다.

MulticastSocket 객체 생성

MulticastSocket 객체는 다음 3가지 생성자 중 하나를 사용하여 생성할 수 있다.

MulticastSocket 클래스의 생성자

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

첫 번째 기본 생성자로 객체를 생성하는 경우 비워져 있는 포트로 자동 바인딩이 이루어진다. 이 경우 전송 측에서는 이 MulticastSocket 객체의 포트 번호를 알 길이 없어 이 객체로 먼저 데이터를 전송할 수는 없을 것이다. 다시 말해 기본 생성자를 사용한 MulticastSocket 객체 생성은 일반적으로 DatagramPacket을 송신하고자 하는 경우에 사용한다. 두 번째 생성자는

매개변수로 전달된 포트로 자신이 가입한 멀티캐스트 그룹에서 수신할 포트를 지정한다. 예를 들어 'new MulticastSocket(10000)'과 같이 생성했다면 해당 객체는 멀티캐스트 그룹으로 오는 여러 DatagramPacket 중 10000 포트로 전달되는 Datagram을 수신하겠다는 의미이다. 마지막은 포트 번호뿐만 아니라 IP 주소도 포함된 SocketAddress를 매개변수로 가지는 생성자이다.

MulticastSocket의 주요 메서드

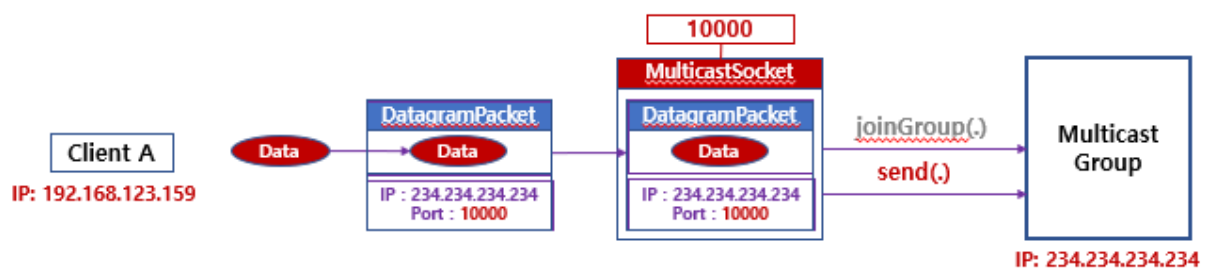
MulticastSocket은 기본적인 DatagramSocket의 기본적인 메서드와 함께 다음의 메서드를 포함한다.

MulticastSocket 클래스의 주요 메서드

리턴 타입	메서드명	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 때마다 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 바인딩된 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

첫 번째 getTimeToLive()와 setTimeToLive()는 패킷의 생존 기간을 읽거나 설정하는 메서드로 0~255 사이의 값을 가진다. 스위치나 라우터 등 단말을 하나 통과할 때마다 1씩 감소하기

때문에 만일 TimeToLive 값이 1인 경우에는 내부 네트워크에서만 사용할 수 있다는 의미이다. joinGroup()과 leaveGroup()은 가상의 멀티캐스트 그룹에 가입 또는 해제하는 메서드로 매개변수로 InetAddress가 사용된다. 당연히 이때 사용되는 IP 주소는 D 클래스에 해당하는 멀티캐스팅 IP 주소이다. 마지막 send()와 receive()는 상위 클래스인 DatagramSocket의 메서드로 UDP의 경우와 동일하게 DatagramPacket의 송신 및 수신을 수행하는 메서드이다. 다음 그림은 이들 메서드를 이용하여 멀티캐스트 그룹(IP: 234.234.234.234)으로 데이터를 전송하는 과정을 나타낸다.



멀티캐스트 그룹으로 데이터 전송 과정

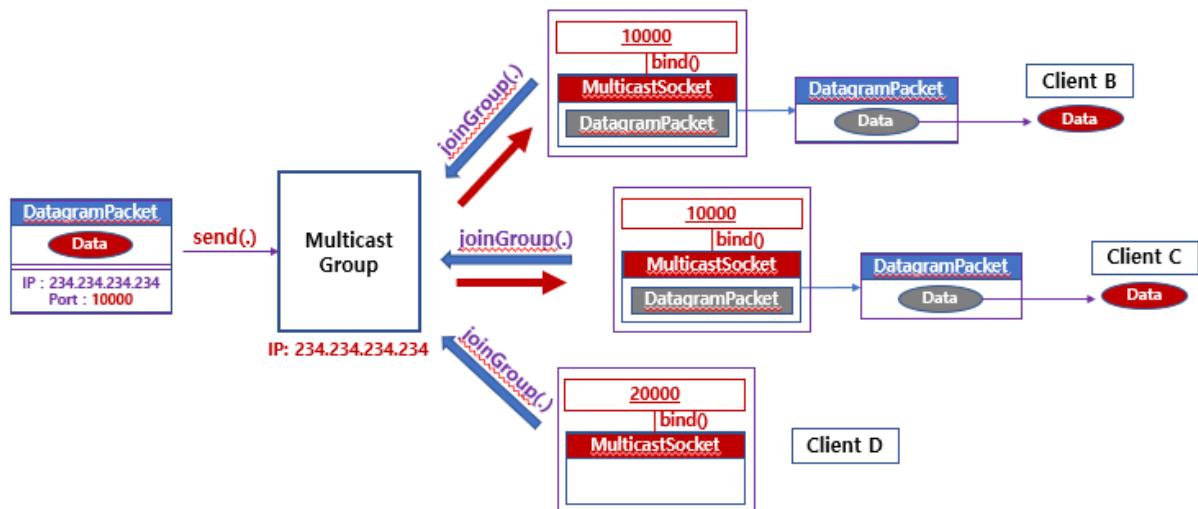
먼저 Client A는 10000번 포트에 바인딩된 MulticastSocket을 생성하고 이를 joinGroup() 메서드를 이용하여 IP=234.234.234.234를 가지는 멀티캐스트 그룹에 가입했다. 이는 이후 멀티캐스트 그룹으로 도착하는 DatagramPacket 중 10000 포트를 가지는 모든 패킷은 수신하겠다는 의미이다. 이후 전송데이터를 포함하는 DatagramPacket을 생성하고 원격지 주소 234.234.234.234:10000를 설정한 뒤 send() 메서드를 통해 멀티캐스트 그룹으로 DatagramPacket을 전송했다.

▶ 여기서 멀티캐스트 그룹에 가입여부와 상관없이 send() 메서드를 이용하여 멀티캐스트 그룹으로 DatagramPacket을 전송할 수 있다. 다만 가입이 안된 경우 데이터의 수신이 불가능하다.

멀티캐스트 그룹으로 전송된 DatagramPacket은 자신의 그룹에 가입되어 있는 여러 클라이언트들 10000번에 바인딩된 클라이언트에게 방금 수신한 DatagramPacket을 전송할 것이다. 따라서 10000번에 바인딩된 MulticastSocket을 가진 Client A는 자신이 보낸 데이터를 다시 받게 되는 것이다.

그럼 데이터를 수신하는 과정을 조금 더 자세하게 알아보자. 다음 그림은 하나의 멀티캐스트 그룹에 3개의 클라이언트(Client B, Client C, Client D)가 바인딩된 경우로, 가입된 클라이언트들은 각각 10000, 10000, 20000번에 바인딩 되어 있다.

▶ MulticastSocket은 TCP 통신의 Socket이나 UDP 통신의 DatagramSocket과는 다르게 하나의 포트에 여러 개의 MulticastSocket을 바인딩 할 수 있다. 애초에 1:N 통신을 위한 것이니 당연히 그래야 하는 기능이다.



하나의 멀티캐스트 그룹에 3개의 클라이언트가 바인딩된 경우의 데이터 수신

이 때 Client A가 234.234.234.234:10000의 원격지 정보를 가지고 있는 DatagramPacket을 이 멀티캐스트 그룹(234.234.234.234)에 전송하면 이 그룹에 가입이 되어 있으면서 10000번에 바인딩된 Client B와 Client C만 데이터를 수신하게 된다.

▶ 여기서는 따로 다루지 않지만 브로드캐스트 통신은 멀티캐스트 주소 대신에 브로드캐스팅 주소(255.255.255.255)로 데이터그램 패킷을 전송하는 것으로 가입(join)의 과정없이 내부 네트워크의 모든 단말에 전달된다.

[Do it 실습] Multicast 통신을 위한 MulticastSocket 클래스의 활용

MulticastSocketObject.java

```
01 package sec04_multicastcommunication.EX01_MulticastSocketObject;
02 import java.io.IOException;
03 import java.net.DatagramPacket;
04 import java.net.InetAddress;
05 import java.net.InetSocketAddress;
```

```

06  import java.net.MulticastSocket;
07  import java.net.UnknownHostException;
08
09  public class MulticastSocketObject {
10      public static void main(String[] args) {
11          // 멀티캐스트 : 클래스 D 224.0.0.0 ~ 239.255.255.255
12          // 1. MulticastSocket 객체 생성
13          MulticastSocket mcs1 = null;
14          MulticastSocket mcs2 = null;
15          MulticastSocket mcs3 = null;
16          try {
17              mcs1 = new MulticastSocket();
18                  //기본 생성자를 사용하는 경우 비어 있는 포트로 자동 바인딩
19              mcs2 = new MulticastSocket(10000);
20              mcs3 = new MulticastSocket(
21                  new InetSocketAddress(InetAddress.getByName("localhost"), 10000));
22          } catch (IOException e) {
23              e.printStackTrace();
24          }
25          System.out.println(mcs1.getLocalSocketAddress());
26          System.out.println(mcs2.getLocalSocketAddress());
27          System.out.println(mcs3.getLocalSocketAddress());
28          System.out.println();
29          // 2. MulticastSocket 주요 메서드 (이외 IllegalArgumentException 발생)
30          // getTimeToLive(), setTimeToLive(): 0~255
31          try {
32              System.out.println("TimeToLive: " + mcs1.getTimeToLive()); //1
33              mcs1.setTimeToLive(50);
34              System.out.println("TimeToLive: " + mcs1.getTimeToLive()); //50
35          } catch (IOException e) {
36              e.printStackTrace();
37          }
38          System.out.println();
39          // joinGroup(IP 정보:InetAddress), leaveGroup(IP 정보:InetAddress)
40          // send(데이터그램패킷), receive(데이터그램패킷)
41          try {

```

```

42         mcs1.joinGroup(InetAddress.getByName("234.234.234.234"));
43         mcs2.joinGroup(InetAddress.getByName("234.234.234.234"));
44         mcs3.joinGroup(InetAddress.getByName("234.234.234.234"));
45         byte[] sendData = "안녕하세요".getBytes();
46         DatagramPacket sendPacket = new DatagramPacket(sendData,
47             sendData.length, InetAddress.getByName("234.234.234.234"), 10000);
48         mcs1.send(sendPacket);
49         byte[] receivedData;
50         DatagramPacket receivedPacket;
51         receivedData = new byte[65508];
52         receivedPacket =
53             new DatagramPacket(receivedData, receivedData.length);
54         mcs2.receive(receivedPacket);
55         System.out.print("mcs2가 수신한 데이터 : " +
56             new String(receivedPacket.getData()).trim());
57         System.out.println(" 송신지 : " +
58             receivedPacket.getSocketAddress());
59         receivedData = new byte[65508];
60         receivedPacket =
61             new DatagramPacket(receivedData, receivedData.length);
62         mcs3.receive(receivedPacket);
63         System.out.print("mcs3가 수신한 데이터 : " +
64             new String(receivedPacket.getData()).trim());
65         System.out.println(" 송신지 : " +
66             receivedPacket.getSocketAddress());
67     } catch (UnknownHostException e) {
68         e.printStackTrace();
69     } catch (IOException e) {
70         e.printStackTrace();
71     }
}

```

실행 결과

0.0.0.0/0.0.0.0:49536

0.0.0.0/0.0.0.0:10000

```
/127.0.0.1:10000
```

```
TimeToLive: 1
```

```
TimeToLive: 50
```

```
mcs2가 수신한 데이터 : 안녕하세요 송신지 : /192.168.0.5:59466
```

```
mcs3가 수신한 데이터 : 안녕하세요 송신지 : /192.168.0.5:59466
```

13-27. MulticastSocket의 기본 생성자를 이용해 객체(mcs1)를 생성하고, 포트 번호만을 넘겨받아 10000번 포트에 바인딩된 객체(mcs2)를 생성한다. 그리고 IP와 포트(10000)를 모두 넘겨 받는 생성자를 사용해 객체를 생성하고 설정된 정보를 출력한다(포트를 지정하지 않은 경우 비어 있는 포트로 임의 배정).

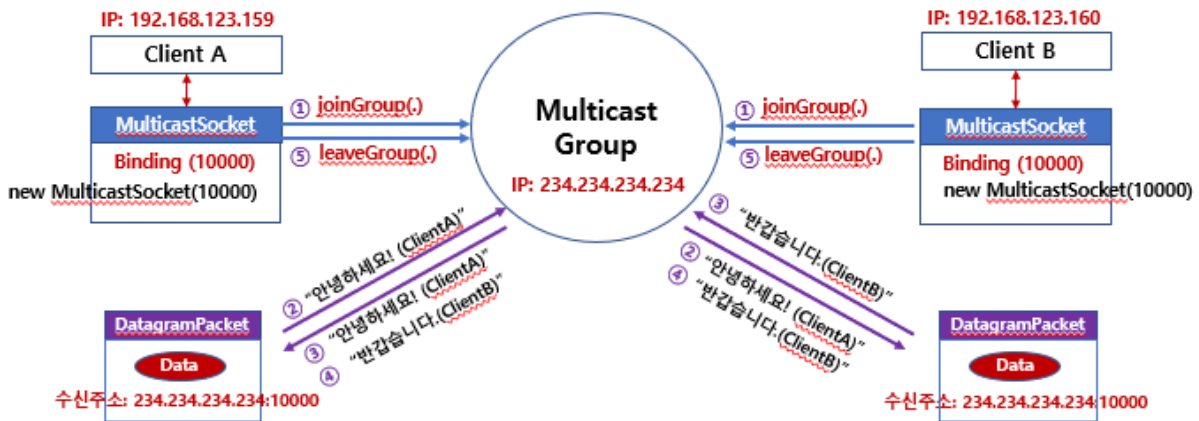
31-37. 디폴트 TimeToLive 값(=1) 출력 및 변경 이후 변경된 값을 출력한다.

41-48. 3개의 MulticastSocket 객체를 234.234.234.234 IP를 가지는 멀티캐스트그룹에 가입(join). 이후 "안녕하세요" 데이터를 포함하는 DatagramPacket을 멀티캐스트 그룹의 10000번 포트로 전송한다(이 경우 가입된 MulticastSocket 중 10000번 포트에 바인딩된 객체만 해당 데이터를 수신).

49-69. 10000번 포트에 바인딩되어 있는 두 MulticastSocket 객체(mcs1, mcs2)는 비어 있는 DatagramPacket 객체를 생성하여 receive() 메서드로 데이터를 수신한다(동일한 메시지를 함께 수신).

Multicast 통신 예제 1. 클라이언트간의 텍스트 전송

멀티캐스트 통신의 첫 번째 예제는 두 클라이언트간의 텍스트 전송 예로 2개의 클라이언트가 동일한 멀티캐스트 그룹에 조인하고 해당 그룹을 통해 데이터를 송신한다.



멀티캐스트 통신을 이용한 클라이언트간 텍스트 전송

멀티캐스트 그룹 IP 주소로는 D 클래스 IP 주소 중 하나인 234.234.234.234를 사용하였으며, 두 클라이언트의 MulticastSocket은 모두 10000번 포트에 바인딩되어 있으며, 이 멀티캐스트 그룹의 가입되어 있다. 이후 Client A가 234.234.234.234:10000의 수신 주소로 “안녕하세요 (ClientA)”를 전송하면 이 멀티캐스트 그룹의 10000번 포트에 바인딩된 모든 MulticastSocket 객체가 전달된 DatagramPacket을 수신한다. 두 클라이언트가 같은 포트로 바인딩되어 있기 때문에 계속 가입이 되어 있다면 데이터를 보낸 클라이언트도 자신의 보낸 데이터를 다시 받게 될 것이다. 이어서 Client B가 “반갑습니다.(Client B)”를 같은 방식으로 보내면 역시 두 클라이언트 모두 동일한 데이터를 수신하게 된다. 만일 가입을 해제(leaveGroup())한 후 데이터를 송신하거나, 두 클라이언트가 서로 다른 포트로 바인딩하면 자신이 보낸 데이터를 자신이 다시 받는 일은 없을 것이다. 다만 이 예제에서는 멀티캐스트 통신을 통해 동일 포트에 바인딩된 모든 호스트에 데이터가 전송됨을 보이기 위해 자신이 보낸 데이터도 자신이 받도록 작성했다. 이 예제 역시 **Client A가 보내는 첫 번째 문자열을 Client B가 수신할 수 있도록 Client B를 먼저 실행해야 한다.**

[Do it 실습] Multicast 통신 예제 1-1. 클라이언트간의 텍스트 전송 [Client A]

Multicast_Text_ClientA.java

```
01 package sec04_multicastcommunication.EX02_Multicast_Text;
02 import java.io.IOException;
03 import java.net.DatagramPacket;
04 import java.net.InetAddress;
```

```

05 import java.net.MulticastSocket;
06 import java.net.UnknownHostException;
07
08 public class Multicast_Text_ClientA {
09     public static void main(String[] args) {
10         System.out.println("<<ClientA>> - Text");
11         // 1. 멀티캐스팅 주소지 생성
12         InetAddress multicastAddress = null;
13         try {
14             multicastAddress = InetAddress.getByName("234.234.234.234");
15         } catch (UnknownHostException e) {
16             e.printStackTrace();
17         }
18         int multicastPort = 10000;
19         // 2. 멀티캐스트소켓 생성
20         MulticastSocket mcs = null;
21         try {
22             mcs = new MulticastSocket(multicastPort);
23         } catch (IOException e) {
24             e.printStackTrace();
25         }
26         // 3. 멀티캐스트 그룹에 조인
27         try {
28             mcs.joinGroup(multicastAddress);
29         } catch (IOException e) {
30             e.printStackTrace();
31         }
32         // 4. 전송 데이터그램 패킷 생성 + 전송
33         byte[] sendData = "안녕하세요!(ClientA)".getBytes();
34         DatagramPacket sendPacket = new DatagramPacket(sendData,
35             sendData.length, multicastAddress, multicastPort);
36         try {
37             mcs.send(sendPacket);
38         } catch (IOException e) {
39             e.printStackTrace();
40         }

```



```

41 // 5. 데이터그램 패킷 수신
42 receiveMessage(mcs); //자기가 보낸 데이터 수신
43 receiveMessage(mcs); //상대편이 보낸 데이터 수신
44 // 6. 멀티캐스트 그룹 나가기
45 try {
46     mcs.leaveGroup(multicastAddress);
47 } catch (IOException e) {
48     e.printStackTrace();
49 }
50 // 7. 소켓 닫기
51 mcs.close();
52 }
53 static void receiveMessage(MulticastSocket mcs) {
54     byte[] receivedData;
55     DatagramPacket receivedPacket;
56     receivedData = new byte[65508];
57     receivedPacket = new DatagramPacket(receivedData, receivedData.length);
58     try {
59         mcs.receive(receivedPacket);
60     } catch (IOException e) {
61         e.printStackTrace();
62     }
63     System.out.println("보내온 주소 : " +
64 receivedPacket.getSocketAddress());
65     System.out.println("보내온 내용 : " +
66                             new String(receivedPacket.getData()).trim());
67 }
}

```

실행 결과

<<ClientA>> - Text

보내온 주소 : /192.168.123.101:10000

보내온 내용 : 안녕하세요!(ClientA)

보내온 주소 : /192.168.123.101:10000

보내온 내용 : 반갑습니다!(ClientB)

12-31. 10000번 포트에 바인딩된 DatagramSocket 객체를 생성하고, 234.234.234.234 IP의 멀티캐스트 그룹에 가입한다.

33-40. "안녕하세요!(ClientA)" 데이터와 원격지 주소로 234.234.234.234:10000을 포함하는 DatagramPacket 객체를 생성하여 MulticastSocket의 send() 메서드로 전송한다.

42-43. 가입된 멀티캐스트 그룹에 도착하는 DatagramPacket 중 10000번 포트로 전달되는 데이터를 수신한다(receiveMessage() 메서드는 데이터를 수신하여 출력하도록 작성한 메서드).

45-52. 데이터 수신이 모두 끝난 후 leaveGroup() 메서드를 이용하여 멀티캐스트 그룹에 가입 해제한다.

53-66. 비어 있는 DatagramPacket을 생성한 뒤 receive() 메서드로 데이터를 수신하여 콘솔에 송신자 주소, 포함된 내용을 출력하도록 사용자 메서드를 작성한다.

[Do it 실습] Multicast 통신 예제 1-2. 클라이언트간의 텍스트 전송 [Client B]

Multicast_Text_ClientB.java

```
01 package sec04_multicastcommunication.EX02_Multicast_Text;
02 import java.io.IOException;
03 import java.net.DatagramPacket;
04 import java.net.InetAddress;
05 import java.net.MulticastSocket;
06 import java.net.UnknownHostException;
07
08 public class Multicast_Text_ClientB {
09     public static void main(String[] args) {
10         System.out.println("<<ClientB>> - Text");
11         // 1. 멀티캐스팅 주소지 생성
12         InetAddress multicastAddress = null;
13         try {
14             multicastAddress = InetAddress.getByName("234.234.234.234");
15         } catch (UnknownHostException e) {
16             e.printStackTrace();
17         }
18         int multicastPort = 10000;
```

```

19 // 2. 멀티캐스트소켓 생성
20 MulticastSocket mcs = null;
21 try {
22     mcs = new MulticastSocket(multicastPort);
23 } catch (IOException e) {
24     e.printStackTrace();
25 }
26 // 3. 멀티캐스트 그룹에 조인
27 try {
28     mcs.joinGroup(multicastAddress);
29 } catch (IOException e) {
30     e.printStackTrace();
31 }
32 // 4. 데이터그램 패킷 수신 대기
33 receiveMessage(mcs); //ClientA가 보낸 메시지 수신
34 // 5. 전송 데이터그램 패킷 생성 + 전송
35 byte[] sendData = "반갑습니다!(ClientB)".getBytes();
36 DatagramPacket sendPacket = new DatagramPacket(sendData,
37     sendData.length, multicastAddress, multicastPort);
38 try {
39     mcs.send(sendPacket);
40 } catch (IOException e) {
41     e.printStackTrace();
42 }
43 // 6. 데이터그램 패킷 수신 대기
44 receiveMessage(mcs); //자기가 보낸 메시지 수신
45 // 7. 멀티캐스트 그룹 나가기
46 try {
47     mcs.leaveGroup(multicastAddress);
48 } catch (IOException e) {
49     e.printStackTrace();
50 }
51 // 8. 소켓 닫기
52 mcs.close();
53 }
54 static void receiveMessage(MulticastSocket mcs) {

```

```

55         byte[] receivedData;
56         DatagramPacket receivedPacket;
57         receivedData = new byte[65508];
58         receivedPacket = new DatagramPacket(receivedData, receivedData.length);
59         try {
60             mcs.receive(receivedPacket);
61         } catch (IOException e) {
62             e.printStackTrace();
63         }
64         System.out.println("보내온 주소 : " +
65         receivedPacket.getSocketAddress());
66         System.out.println("보내온 내용 : " +
67                             new String(receivedPacket.getData()).trim());
68     }
}

```

실행 결과

<<ClientB>> - Text

보내온 주소 : /192.168.123.101:10000

보내온 내용 : 안녕하세요!(ClientA)

보내온 주소 : /192.168.123.101:10000

보내온 내용 : 반갑습니다!(ClientB)

12-31. 10000번 포트에 바인딩된 DatagramSocket 객체를 생성하고, 234.234.234.234 IP의 멀티캐스트 그룹에 가입한다.

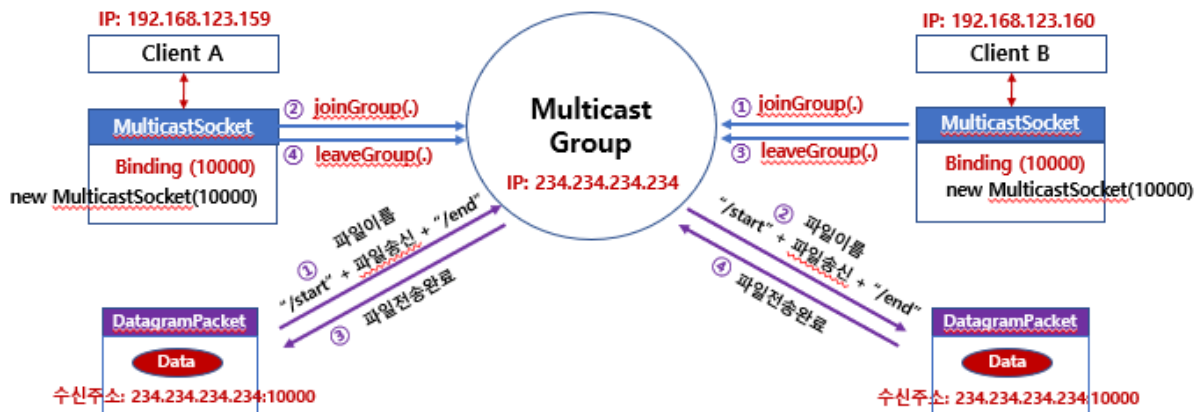
33-42. Client A가 멀티캐스트 그룹의 10000번 포트에 보낸 메시지를 수신한 뒤 "반갑습니다!(ClientB)" 데이터와 원격지 주소로 234.234.234.234:10000을 포함하는 DatagramPacket 객체를 생성하여 MulticastSocket의 send() 메서드로 전송한다.

43-50. Client B의 MulticastSocket은 10000번에 바인딩되어 있기 때문에, 자신이 보낸 데이터를 다시 자신이 수신하며, 이후 leaveGroup() 메서드를 이용하여 멀티캐스트 그룹에 가입 해제한다.

54-67. 비어 있는 DatagramPacket을 생성한 뒤 receive() 메서드로 데이터를 수신하여 콘솔에 송신자 주소, 포함된 내용을 출력하도록 사용자 메서드를 작성한다.

Multicast 통신 예제 2. 클라이언트 간의 파일 전송

이번에는 멀티캐스트 통신을 이용하여 두 클라이언트 간 파일을 전송하는 예제이다.



멀티캐스트 통신을 이용한 클라이언트간 파일 전송

앞의 예제와 동일하게 2개의 클라이언트에 10000번 포트에 바인딩된 MulticastSocket 객체를 생성했다. 다만 여기에서는 자신이 보낸 파일 데이터를 다시 자신이 받지 않도록 하기 위해 Client A는 파일 전송이 모두 끝난 후 멀티미디어 그룹에 가입했다. 파일 전송 과정에서는 네트워크상의 데이터 전송 효율성을 고려하여 파일은 2048byte씩 나누어 DatagramPacket으로 패키징하여 전송했다. 이때 실제 파일의 시작과 끝 지점을 알리기 위해 "/start"와 "/end"의 문자열 정보를 파일 전송의 시작과 끝에 추가하여 전송했다. 파일 전송이 모두 끝나면 Client A는 멀티미디어 그룹에 가입하여 이후에 멀티캐스트 그룹의 10000번 포트로 전달되는 DatagramPacket을 수신할 준비를 하며, Client B는 파일 수신 완료 이후 자신은 "파일수신완료"를 출력하고 "(ClientB) 파일 수신 완료" 문자열을 멀티미디어 그룹에 전송한다. 이 예제 역시 Client B가 파일 데이터를 수신할 준비를 먼저 해야 하기 때문에 Client B부터 실행한다.

[Do it 실습] Multicast 통신 예제 2-1. 클라이언트간의 파일 전송 [Client A]

Multicast_File_ClientA.java

```
0 package sec04_multicastcommunication.EX03_Multicast_File;
1 import java.io.BufferedInputStream;
0 import java.io.File;
2 import java.io.FileInputStream;
```

```

0  import java.io.FileNotFoundException;
3  import java.io.IOException;
0  import java.net.DatagramPacket;
4  import java.net.InetAddress;
0  import java.net.MulticastSocket;
5  import java.net.UnknownHostException;
0
6
0  public class Multicast_File_ClientA {
7      public static void main(String[] args) {
0          System.out.println("<<ClientA>> - File");
8          // 1. 멀티캐스팅 주소지 생성
0          InetAddress multicastAddress = null;
9          try {
1             multicastAddress = InetAddress.getByName("234.234.234.234");
0         } catch (UnknownHostException e) { e.printStackTrace(); }
1         int multicastPort = 10000;
1         // 2. 멀티캐스트소켓 생성
1         MulticastSocket mcs = null;
2         try {
1             mcs = new MulticastSocket(multicastPort);
3         } catch (IOException e) { e.printStackTrace(); }
1         // 3. 파일 로딩
4         File file = new
5         File("src\\sec04_multicastcommunication\\files_clientA\\ImageFileUsingMulticast.
1         jpg");
6         BufferedInputStream bis = null;
1         try {
7             bis = new BufferedInputStream(new FileInputStream(file));
1         } catch (FileNotFoundException e) { e.printStackTrace(); }
8         // 4. 파일데이터 전송
1         DatagramPacket sendPacket = null;
9         // 4-0. 파일이름 전송
2         String fileName = file.getName();
0         sendPacket = new DatagramPacket(fileName.getBytes(), fileName.length(),
2                                     multicastAddress, multicastPort);
1
2         try {

```

```

2         mcs.send(sendPacket);
2     } catch (IOException e) { e.printStackTrace(); }
3     System.out.println(fileName + " 파일 전송 시작");
2     // 4-1. 파일 시작 태그를 전송 (/start)
4     String startSign = "/start";
2     sendPacket = new DatagramPacket(startSign.getBytes(),
5 startSign.length(), multicastAddress, multicastPort);
2     try {
6         mcs.send(sendPacket);
2     } catch (IOException e) { e.printStackTrace(); }
7     // 4-2. 실제 파일 데이터 전송 (2048사이즈로 나누어서 파일 전송)
2     int len;
8     byte[] filedata = new byte[2048];
9     try {
3         while((len=bis.read(filedata))!=-1) {
0             sendPacket = new DatagramPacket(filedata, len,
3                                     multicastAddress, multicastPort);
1             mcs.send(sendPacket);
3         }
2     } catch (IOException e) { e.printStackTrace(); }
3     // 4-3. 파일 시작 태그를 전송 (/start)
3     String endSign = "/end";
3     sendPacket = new DatagramPacket(endSign.getBytes(), endSign.length(),
4                                     multicastAddress, multicastPort);
3
5     try {
3         mcs.send(sendPacket);
6     } catch (IOException e) { e.printStackTrace(); }
3     // 5. 멀티캐스트 그룹에 조인
7     try {
3         mcs.joinGroup(multicastAddress);
8     } catch (IOException e) { e.printStackTrace(); }
3     // 6. 데이터 수신 대기
9     receiveMessage(mcs);
4     // 7. 멀티캐스트 그룹 나가기
0
4     try {
1         mcs.leaveGroup(multicastAddress);

```

```

4      } catch (IOException e) { e.printStackTrace(); }
2      // 8. 소켓 닫기
4      mcs.close();
3  }
4  static void receiveMessage(MulticastSocket mcs) {
4      byte[] receivedData;
4      DatagramPacket receivedPacket;
5      receivedData = new byte[65508];
6      receivedPacket = new DatagramPacket(receivedData, receivedData.length);
4      try {
7          mcs.receive(receivedPacket);
4      } catch (IOException e) { e.printStackTrace(); }
8      System.out.println("보내온 주소 : " +
4      receivedPacket.getSocketAddress());
9      System.out.println("보내온 내용 : " +
5                          new String(receivedPacket.getData()).trim());
0      }
5  }
1  }

```


1	
6	
2	
6	
3	
6	
4	
6	
5	
6	
6	
6	
7	
6	
8	
6	
9	
7	
0	
7	
1	
7	
2	
7	
3	
7	
4	
7	
5	
7	
6	
7	
7	
7	
8	
7	
9	
8	
0	

8	
1	
8	
2	
8	
3	
8	
4	
8	
5	
8	
6	
8	
7	
8	
8	
8	
9	
9	
0	
9	
1	
9	
2	

실행 결과

<<ClientA>> - File

ImageFileUsingMulticast.jpg 파일 전송 시작

보내온 주소 : /192.168.123.101:10000

보내온 내용 : (ClientB) 파일 수신 완료

16-25. 10000번 포트에 바인딩된 DatagramSocket 객체를 생성한다(현재 시점에서 Client A는 멀티캐스트 그룹에 가입하지 않음).

27-42. 전송할 파일과 연결된 `BufferedInputStream` 객체를 생성하고, 파일이름과 원격지 주소로 `234.234.234.234:10000`을 포함하는 `DatagramPacket` 객체를 생성하여 `MulticastSocket`의 `send()` 메서드로 전송한다.

44-49. 파일 전송의 시작을 알리기 위해 `"/start"` 문자열을 전송한다.

51-59. `BufferedInputStream` 객체를 통해 파일에서 최대 `2048byte`를 반복적으로 읽어와 `DatagramPacket`에 읽은 데이터 길이와 함께 포함시켜 전송한다.

61-66. 파일 전송의 끝을 알리기 위해 `"/end"` 문자열을 전송한다.

68-78. `234.234.234.234` IP의 멀티캐스트 그룹에 가입한 뒤 자신이 바인딩된 `10000`번 포트의 데이터 수신 후 `leaveGroup()` 메서드를 이용하여 멀티캐스트 그룹에 가입을 해제한다.

80-92. 비어 있는 `DatagramPacket`를 생성한 뒤 `receive()` 메서드로 데이터를 수신하여 콘솔에 송신자 주소, 포함된 내용을 출력하도록 사용자 메서드를 작성한다.

[Do it 실습] Multicast 통신예제 2-2. 클라이언트간의 파일 전송 [Client B]

Multicast_File_ClientB.java

```
01 package sec04_multicastcommunication.EX03_Multicast_File;
02 import java.io.BufferedOutputStream;
03 import java.io.File;
04 import java.io.FileNotFoundException;
05 import java.io.FileOutputStream;
06 import java.io.IOException;
07 import java.net.DatagramPacket;
08 import java.net.InetAddress;
09 import java.net.MulticastSocket;
10 import java.net.UnknownHostException;
11
12 public class Multicast_File_ClientB {
13     public static void main(String[] args) {
14         System.out.println("<<ClientB>> - File");
15         // 1. 멀티캐스팅 주소지 생성
16         InetAddress multicastAddress = null;
17         try {
```

```

18         multicastAddress = InetAddress.getByName("234.234.234.234");
19     } catch (UnknownHostException e) { e.printStackTrace(); }
20     int multicastPort = 10000;
21     // 2. 멀티캐스트소켓 생성
22     MulticastSocket mcs = null;
23     try {
24         mcs = new MulticastSocket(multicastPort);
25     } catch (IOException e) { e.printStackTrace(); }
26     // 3. 멀티캐스트 그룹에 조인
27     try {
28         mcs.joinGroup(multicastAddress);
29     } catch (IOException e) { e.printStackTrace(); }
30     // 4. 파일 데이터 수신
31     byte[] receivedData;
32     DatagramPacket receivedPacket;
33     // 4-1. 파일 이름 수신
34     receivedData = new byte[65508];
35     receivedPacket = new DatagramPacket(receivedData, receivedData.length);
36     try {
37         mcs.receive(receivedPacket);
38     } catch (IOException e) { e.printStackTrace(); }
39     String fileName = new String(receivedPacket.getData()).trim();
40     System.out.println(fileName + " 파일 수신 시작");
41     // 4-2. 파일 저장을 위한 파일 출력 스트림 생성
42     File file = new
43     File("src\\sec04_multicastcommunication\\files_clientB\\" + fileName);
44     BufferedOutputStream bos=null;
45     try {
46         bos = new BufferedOutputStream(new FileOutputStream(file));
47     } catch (FileNotFoundException e) { e.printStackTrace(); }
48     // 4-3. 시작태그부터 끝태그까지 모든 데이터패킷의 내용을 파일에 기록
49     String startSign = "/start";
50     String endSign = "/end";
51     receivedData = new byte[65508];
52     receivedPacket = new DatagramPacket(receivedData, receivedData.length);
53     try {

```

```

54         mcs.receive(receivedPacket);
55         if(new String(receivedPacket.getData(), 0,
56             receivedPacket.getLength()).equals(startSign)) {
57             while(true) {
58                 mcs.receive(receivedPacket);
59                 if(new String(receivedPacket.getData(), 0,
60                     receivedPacket.getLength()).equals(endSign))
61                     break;
62                 bos.write(receivedPacket.getData(), 0,
63                     receivedPacket.getLength());
64                 bos.flush();
65             }
66         }
67     } catch (IOException e) { e.printStackTrace(); }
68     try {
69         bos.close();
70     } catch (IOException e) { e.printStackTrace(); }
71     System.out.println("파일 수신 완료");
72     // 5. 멀티캐스트 그룹 나가기
73     try {
74         mcs.leaveGroup(multicastAddress);
75     } catch (IOException e) { e.printStackTrace(); }
76     // 6. 전송 데이터그램 생성 + 전송
77     byte[] sendData = "(ClientB) 파일 수신 완료".getBytes();
78     DatagramPacket sendPacket = new DatagramPacket(sendData,
79         sendData.length, multicastAddress, multicastPort);
80     try {
81         mcs.send(sendPacket);
82     } catch (IOException e) { e.printStackTrace(); }
83     // 7. 소켓닫기
84     mcs.close();
85 }
86 }

```

실행 결과

<<ClientB>> - File

ImageFileUsingMulticast.jpg 파일 수신 시작 파일 수신 완료
--

16-29. 10000번 포트에 바인딩된 DatagramSocket 객체를 생성하고, 234.234.234.234 IP의 멀티캐스트 그룹에 가입한다.

31-47. 비어 있는 DatagramPacket을 생성하여 receive() 메서드로 처음으로 전송한 파일이름을 수신하고, 쓰기 파일과 연결된 BufferedInputStream 객체를 생성한다.

49-70. 수신된 데이터가 "/start"인 경우 반복적으로 데이터를 수신하여 파일을 기록하고, 수신 데이터가 "/end"인 경우 반복문을 탈출한다.

73-84. leaveGroup() 메서드를 이용하여 멀티캐스트 그룹에 가입 해제하고, "(ClientB) 파일 수신 완료" 데이터를 멀티캐스트 그룹(234.234.234.234)의 10000번 포트에 전송한다.

부록 2. 자바 API의 함수형 인터페이스

람다식은 객체지향형 문법체계 내에서 함수형 프로그램이 가능하도록 제공된 문법이라고 하였다. 다양한 클래스에서 공통으로 사용할 수 있는 기능을 함수로 설계하여 편하게 호출하여 사용할 수 있게 만들었다는 것이다. 목적 자체가 그렇다보니 자바는 대표적인 공통 기능 등을 미리 설계해 놓은 함수형 인터페이스를 제공한다. 자바 API가 제공하는 대표적인 함수형 인터페이스에 대해서 알아보자.

자바 API가 제공하는 함수형 인터페이스

자바 API가 제공하는 대표적인 함수형 인터페이스는 크게 `Consumer<T>`, `Supplier<T>`, `Predicate<T>`, `Function<T,R>`, `UnaryOperator<T>`/`BinaryOperator<T>`가 있다. 추가로 이들을 기준으로 특정 타입으로 확장한 다양한 함수형 인터페이스들이 존재한다.



자바 API가 제공하는 함수형 인터페이스

표준형 함수형 인터페이스와 확장형 함수형 인터페이스에 대해서 `Consumer<T>`를 기준으로 설명하면, 먼저 `Consumer<T>`는 `T`를 제네릭 변수로 가지는 함수형 인터페이스이다. 이후에 자세히 다루겠지만 내부에는 `void accept(T t)`의 추상 메서드가 정의되어 있다. 이의 확장형인 `IntConsumer`, `LongConsumer`, `DoubleConsumer`는 표준형에서의 제네릭 타입이 각각 `int`, `long`, `double`로 지정된 함수형 인터페이스를 의미한다. 또한 `BiConsumer<T,U>`는 표준형과 비교하여

제네릭 변수가 추가된 인터페이스를 나타낸다. 자바 API에서 제공하는 다른 함수형 인터페이스도 같은 방식으로 적용할 수 있다. 이들 인터페이스는 자바가 제공하는 API의 매개변수로 많이 사용된다. 즉 메서드 호출 시 매개변수로 이들 인터페이스의 객체를 생성하여 전달해야 하는데 이들 모두 함수형 인터페이스니 람다식으로 간단히 표현될 수 있는 것이다. 그럼 하나하나 자세히 알아보자.

Consumer<T>

표준형 Consumer<T>

Consumer<T> 함수형 인터페이스의 원형은 다음과 같다.

표준형 Consumer<T> 함수형 인터페이스의 원형

```
interface Consumer<T> {  
    public abstract void accept(T t);  
}
```

표준형 Consumer<T> 함수형 인터페이스는 입력(T)은 있지만 출력(void)은 없는 즉, 말 그대로 입력을 소비하는 소비자(consumer)의 역할을 수행하는 함수형 인터페이스로 accept() 추상 메서드를 포함한다.



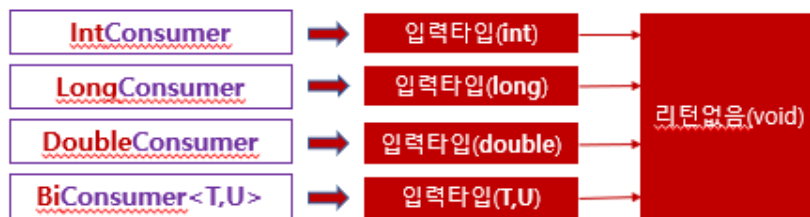
표준형 Consumer<T> 함수형 인터페이스의 입출력

다음은 각각 익명 이너클래스 방법과 람다식으로 Consumer<T> 인터페이스 타입의 객체를 생성한 후 구현한 메서드를 호출하는 예를 보이고 있다. 자바가 제공하는 함수형 인터페이스라는 점만 제외하면 앞서 알아본 방법과 완벽히 동일하다.

익명 이너클래스	<pre> Consumer<String> c = new Consumer<String>() { @Override public void accept(String t) { System.out.println(t); } }; c.accept("Consumer<T> 함수형 인터페이스"); </pre>
람다식	<pre> Consumer<String> c = (str)->System.out.println(str); c.accept("Consumer<T> 함수형 인터페이스"); </pre>

확장형 Consumer

확장형 Consumer 인터페이스란 위의 표준형 인터페이스에서 입력 자료형이 확정되거나 입력의 개수가 늘어난 경우를 말한다. IntConsumer, LongConsumer, DoubleConsumer는 제네릭 인터페이스인 Consumer<T>에서 T가 각각 int, long, double 타입으로 고정된 인터페이스를 말한다. BiConsumer<T,U>는 하나가 아닌 2개의 입력을 받아 소비(리턴하지 않음)하는 함수형 인터페이스이다. 따라서 이들 4개의 확장형 Consumer 인터페이스의 입출력을 정리하면 다음과 같다.



확장형 Consumer 함수형 인터페이스의 입출력

이 4개의 확장형 Consumer 인터페이스들은 모두 Consumer<T>와 동일하게 accept() 추상 메서드를 가진다. 참고로 자바에서 제공하는 함수형 인터페이스에서 리턴 타입이 기본 자료형이 아니거나 확장형 인터페이스의 리턴 타입이 모두 동일한 경우에는 모든 인터페이스의 추상 메서드 이름은 동일하다. 확장형 Consumer 인터페이스의 경우 리턴 타입이 void로 모두 동일하기 때문에 4개의 확장형 Consumer 인터페이스 모두 동일한 이름의 추상

메서드(accept())를 가진다. 이는 이후의 인터페이스들에도 동일하게 적용된다. 다음은 이들 4개의 확장형 Consumer 인터페이스의 객체를 람다식으로 정의 후 메서드를 호출한 예이다.

확장형 Consumer 인터페이스의 객체 생성 및 메서드를 호출의 예

```
IntConsumer c2 = (num)->System.out.println("int 값="+num);
LongConsumer c3 = (num)->System.out.println("long 값="+num);
DoubleConsumer c4 = (num)->System.out.println("double 값="+num);
BiConsumer<String, Integer> c5 = (name, age)->System.out.println("이름:"+name + " 나
이:"+age);
c2.accept(3);
c3.accept(5L);
c4.accept(7.8);
c5.accept("홍길동", 16);
```

c2~c4는 각각 입력타입을 int, long, double로 넘겨받아 그대로 화면에 값을 출력하였으며, c5의 경우 매개변수로 String 타입과 Integer 타입을 입력받아 이를 조합하여 화면에 출력하였다.

[Do it 실습] Consumer<T> 함수형 인터페이스(자바 API 제공)

Consumer_FuntionalInterface.java

```
1 package sec02_functioninterfaceinjavaAPI.EX01_Consumer_FuntionalInterface;
2 import java.util.function.BiConsumer;
3 import java.util.function.Consumer;
4 import java.util.function.DoubleConsumer;
5 import java.util.function.IntConsumer;
6 import java.util.function.LongConsumer;
7
8 public class Consumer_FuntionalInterface {
9     public static void main(String[] args) {
10         // 1. 익명이너클래스를 이용한 객체 생성
11         Consumer<String> c = new Consumer<String>() {
12             @Override
13             public void accept(String t) {
14                 System.out.println(t);
15             }
16         };
```

```

17 // 2. 람다식 표현
18 // 2-1. 기본 Consumer
19 Consumer<String> c1 = t->System.out.println(t);
20 c1.accept("Consumer<T> 함수형 인터페이스");
21 // 2-2. 확장형 Consumer
22 IntConsumer c2 = num->System.out.println("int 값="+num);
23 LongConsumer c3 = num->System.out.println("long 값="+num);
24 DoubleConsumer c4 = num->System.out.println("double 값="+num);
25 BiConsumer<String, Integer> c5 =
26     (name, age)->System.out.println("이름="+name + " 나이="+age);
27 c2.accept(5);
28 c3.accept(5L);
29 c4.accept(7.8);
30 c5.accept("홍길동", 16);
31 }
32 }

```

실행 결과

```

Consumer<T> 함수형 인터페이스
int 값=5
long 값=5
double 값=7.8
이름=홍길동 나이=16

```

11-20. 표준형 Consumer<T> 함수형 인터페이스의 객체를 익명 이너클래스 방식 및 람다식으로 생성 및 구현 메서드를 호출한다.

22-26. Consumer<T>의 제네릭 변수가 각각 int, long, double로 지정된 경우의 확장형 Consumer 인터페이스의 객체 생성과 2개의 입력변수를 가지는 BiConsumer<String, Integer> 객체를 람다식으로 생성한다.

27-30. 확장형 Consumer 객체의 구현 메서드를 호출한다. 모두 동일한 메서드를 가진다.

Supplier<T>

표준형 Supplier<T>

Supplier<T> 함수형 인터페이스의 원형은 다음과 같다.

표준형 Supplier<T> 함수형 인터페이스의 원형

```
interface Supplier<T> {  
    public abstract T get();  
}
```

Supplier<T>는 Consumer<T>의 반대 개념으로 입력(void)은 없지만 리턴 타입(T)이 존재하는 함수형 인터페이스이다. 즉, 입력 없이 결과를 제공하는 제공자(supplier) 역할을 수행하는 인터페이스인 것이다. 추상 메서드로는 get() 메서드를 가지고 있다.



표준형 Supplier<T> 함수형 인터페이스의 입출력

다음은 익명 이너클래스와 람다식을 이용해 Supplier<T> 객체를 생성한 예시이다.

익명 이너클래스	<pre>Supplier<String> s = new Supplier<String>() { @Override public String get() { return "Supplier<T> 함수형 인터페이스"; } }; System.out.println(s.get()); // Supplier<T> 함수형 인터페이스</pre>
람다식	<pre>Supplier<String> s1 = () -> "Supplier<T> 함수형 인터페이스"; System.out.println(s1.get()); // Supplier<T> 함수형 인터페이스</pre>

제네릭 타입으로는 String을 지정하여 객체를 생성하였으며, 구현된 메서드 내부에서는 별다른 동작없이 문자열만 리턴하고 있다.

▶ 앞서 살펴본 바와 같이 메서드 내부에서 return 문만 존재하는 경우, 이를 람다식으로 표현할 때는 중괄호와 함께

return 키워드의 생략이 가능하다.

확장형 Supplier

확정형 Supplier 역시 제네릭 인터페이스인 Supplier<T>에서 T의 자료형이 특정 자료형으로 고정된 인터페이스를 말한다. BooleanSupplier, IntSupplier, LongSupplier, DoubleSupplier는 제네릭 변수가 각각 boolean, int, long, double인 인터페이스를 말하는 것으로 더 이상 제네릭 인터페이스가 아니다.



확장형 Supplier 함수형 인터페이스의 입출력

여기서 주의해야 할 점은 이들 확장형 Supplier가 리턴 타입이 기본 자료형이면서 서로 다르기 때문에 다른 이름의 추상 메서드를 가진다. 다행스러운 것은 추상 메서드의 이름이 결정되는 일정한 규칙이 있는데 ‘기본 메서드명 + As + 리턴 타입명’의 형태를 가진다. 예를 들어 BooleanSupplier의 추상 메서드는 boolean getAsBoolean(){ }이 되는 것이다. 나머지도 동일한 방식으로 적용된다. 다음은 확장형 Supplier 인터페이스의 객체를 람다식으로 정의 후 메서드를 호출한 예이다.

확장형 Supplier 인터페이스의 객체 생성 및 메서드를 호출의 예

```
BooleanSupplier s2 = ()->false;
IntSupplier s3 = ()->2+3;
LongSupplier s4 = ()->10L;
DoubleSupplier s5 = ()->5.8;
System.out.println(s2.getAsBoolean()); //false
System.out.println(s3.getAsInt()); //5
System.out.println(s4.getAsLong()); //10
System.out.println(s5.getAsDouble()); //5.8
```

s2~s5는 Supplier<T>의 제네릭 변수 타입을 각각 boolean, int, long, double로 지정한 경우로 구현한 메서드 내부에서는 해당 타입의 값을 그대로 리턴하도록 작성하였다.

[Do it 실습] Supplier<T> 함수형 인터페이스(자바 API 제공)

Suppler_FuntionalInterface.java

```

1  package sec02_functioninterfaceinjavaAPI.EX02_Suppler_FuntionalInterface;
2  import java.util.function.BooleanSupplier;
3  import java.util.function.DoubleSupplier;
4  import java.util.function.IntSupplier;
5  import java.util.function.LongSupplier;
6  import java.util.function.Supplier;
7
8  public class Suppler_FuntionalInterface {
9      public static void main(String[] args) {
10         // 1. 익명 이너클래스 방법 객체 생성
11         Supplier<String> s = new Supplier<String>() {
12             @Override
13             public String get() {
14                 return "Supplier<T> 함수형 인터페이스";
15             }
16         };
17         System.out.println(s.get()); //Supplier<T> 함수형 인터페이스
18         // 2. Supplier<T> 함수형 인터페이스
19         // 2-1. 표준형
20         Supplier<String> s1 = ()-> "Supplier<T> 함수형 인터페이스";
21         System.out.println(s1.get()); //Supplier<T> 함수형 인터페이스
22         // 2-2. 확장형
23         BooleanSupplier s2 = ()->false;
24         IntSupplier s3 = ()->2+3;
25         LongSupplier s4 = ()->10L;
26         DoubleSupplier s5 = ()->5.8;
27         System.out.println(s2.getAsBoolean()); //false
28         System.out.println(s3.getAsInt()); //5
29         System.out.println(s4.getAsLong()); //10
30         System.out.println(s5.getAsDouble()); //5.8

```


31	}
32	}

실행 결과

```
Supplier<T> 함수형 인터페이스
Supplier<T> 함수형 인터페이스
false
5
10
5.8
```

11-21. 표준형 `Supplier<T>` 함수형 인터페이스의 객체를 익명 이너클래스 방식 및 람다식으로 생성 및 구현 메서드를 호출한다.

23-26. `Supplier<T>`의 제네릭 변수가 각각 `boolean`, `int`, `long`, `double`로 지정된 경우의 확장형 `Supplier` 인터페이스 객체를 람다식으로 생성한다.

27-30. 확장형 `Supplier` 객체의 구현 메서드를 호출한다. 리턴 타입이 기본 자료형이면서 각각 서로 다른 타입을 리턴하기 때문에 내부 포함 메서드의 이름이 서로 다르다.

Predicate<T>

표준형 Predicate<T>

먼저 `Predicate<T>` 함수형 인터페이스의 원형은 다음과 같다.

표준형 Predicate<T> 함수형 인터페이스의 원형

```
interface Predicate<T> {
    public abstract boolean test(T t);
}
```

`Predicate<T>` 함수형 인터페이스는 임의 자료형의 입력(`T`)을 받아 `boolean` 타입을 출력하는 함수형 인터페이스이다. 즉, 어떤 입력에 대해 `true` 또는 `false`의 결과를 예측하는

예측기(predicate)의 역할을 수행하는 인터페이스인 것이다. 추상 메서드의 이름은 test()이며 리턴타입은 항상 boolean으로 고정되어 있다.



표준형 Predicate<T> 함수형 인터페이스의 입출력

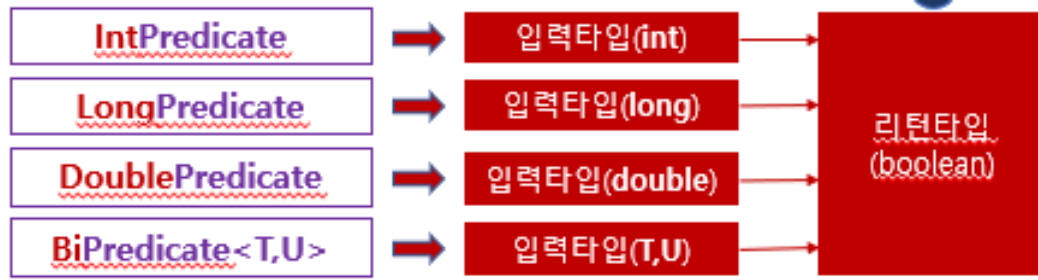
다음은 익명 이너클래스와 람다식으로 Predicate<T> 타입의 객체를 생성한 예시이다.

익명 이너클래스	<pre>Predicate<String> p = new Predicate<String>() { @Override public boolean test(String t) { return (t.length() > 0) ? true : false; } }; System.out.println(p.test("안녕")); // true</pre>
람다식	<pre>Predicate<String> p1 = (str)->(str.length())>0? true:false; System.out.println(p1.test("안녕")); //true</pre>

구현한 메서드 내부에서는 String 객체를 매개변수로 입력받으며, 전달받은 문자열이 공백의 경우에는 false를 한 글자 이상인 경우 true를 리턴한다.

확장형 Predicate<T>

앞에서 다룬 다른 인터페이스들과 마찬가지로 확장형 Predicate 인터페이스 역시 표준형인 Predicate<T>에서 제네릭 변수 T를 특정 타입으로 고정시키거나 2개의 입력을 가지는 인터페이스이다. IntPredicate, LongPredicate, DoublePredicate는 각각 제네릭 변수가 int, long, double로 고정된 경우의 인터페이스로 더 이상 제네릭 인터페이스가 아니다. BiPredicate<T, U>는 표준형 Predicate<T>의 입력을 2개로 늘린 확장형 인터페이스이다. 앞서 여러 번 언급한 것처럼 리턴 타입이 기본 자료형이지만 모두 동일한 타입을 리턴하기 때문에 모든 확장형 Predicate는 동일한 이름(test())의 추상 메서드를 가진다.



확장형 Predicate 함수형 인터페이스의 입출력

다음은 이들 4개의 확장형 Predicate 인터페이스의 객체를 람다식으로 정의 후 메서드를 호출한 예이다.

확장형 Predicate 인터페이스의 객체 생성 및 메서드를 호출의 예

```
IntPredicate p2 = (num)->(num%2==0)? true:false;
LongPredicate p3 = (num)->(num>100)? true:false;
DoublePredicate p4 = (num)->(num>0)? true:false;
BiPredicate<String, String> p5 = (str1, str2)->str1.equals(str2);
System.out.println(p2.test(2));           //true
System.out.println(p3.test(85L));         //false
System.out.println(p4.test(-5.8));        //false
System.out.println(p5.test("안녕", "안녕")); //true
```

먼저 p2~p4는 Predicate<T>에서 제네릭 변수 T를 각각 int, long, double로 지정한 경우이다. p2는 매개변수로 넘어온 정수값이 짝수의 경우 true, 홀수인 경우 false를 리턴하도록 작성하였다. p3와 p4는 각각 매개변수로 넘어온 값이 100, 그리고 0보다 큰 경우 true, 그렇지 않은 경우 false를 리턴한다. 마지막 p5는 2개의 문자열을 매개변수로 입력받아 두 문자열이 같은 경우 true, 다른 경우 false를 리턴한다.

```

1  package sec02_functioninterfaceinjavaAPI.EX03_Predicate_FuntionalInterface;
2
3  import java.util.function.BiPredicate;
4  import java.util.function.DoublePredicate;
5  import java.util.function.IntPredicate;
6  import java.util.function.LongPredicate;
7  import java.util.function.Predicate;
8
9  public class Predicate_FuntionalInterface {
10     public static void main(String[] args) {
11         // 1. 익명 이너클래스 방법으로 객체 생성
12         Predicate<String> p = new Predicate<String>() {
13             @Override
14             public boolean test(String t) {
15                 return (t.length()>0)? true:false;
16             }
17         };
18         System.out.println(p.test("안녕")); //true
19         // 2. Predicate<T>의 람다식 표현
20         // 2-1. 표준형 Predicate<T>
21         Predicate<String> p1 = (str)->(str.length()>0?true:false);
22         System.out.println(p1.test("안녕")); //true
23         System.out.println();
24         // 2-2. 확장형 Predicate
25         IntPredicate p2 = (num)->(num%2)==0?true:false;
26         LongPredicate p3 = (num)->(num>100)?true:false;
27         DoublePredicate p4 = (num)->(num>0)?true:false;
28         BiPredicate<String, String> p5 = (str1, str2)->str1.equals(str2);
29         System.out.println(p2.test(2)); //true
30         System.out.println(p3.test(85L)); //false
31         System.out.println(p4.test(-5.8)); //false
32         System.out.println(p5.test("안녕", "안녕")); //true
33     }
34 }

```

실행 결과

```
true
true

true
false
false
true
```

11-21. 표준형 Predicate<T> 함수형 인터페이스의 객체를 익명 이너클래스 방식 및 람다식으로 생성한다.

24-27. Predicate<T>의 제네릭 변수가 각각 int, long, double로 지정된 경우의 확장형 Predicate 인터페이스의 객체 생성과 2개의 입력변수를 가지는 BiPredicate<String, String> 객체를 람다식으로 생성한다.

28-31. 확장형 Predicate 객체의 구현 메서드 호출한다. 모두 동일한 메서드를 가진다.

Function<T, R>

표준형 Function<T, R>

Function<T, R> 함수형 인터페이스의 원형은 다음과 같다.

표준형 Function <T, R> 함수형 인터페이스의 원형

```
interface Function<T, R> {
    public abstract R apply(T t);
}
```

Function<T, R>은 T 타입의 입력을 받아 이를 R 타입으로 변환하는 함수형 인터페이스이다. 즉, 입력으로 받은 T 타입을 R 타입으로 변환하는 변환기(function)의 역할을 수행하는 인터페이스인 것이다. Function<T, R> apply()라는 이름의 추상 메서드를 가진다.



그림 0-1 표준형 Function<T, R> 함수형 인터페이스의 입출력

다음은 익명 이너클래스와 람다식으로 String의 입력을 Integer로 변환하는 Function<T, U>의 객체를 생성하는 예시이다. 구현 메서드 내부에서는 입력 받은 문자열의 길이를 정수값으로 리턴하였다.

익명 이너클래스	<pre> Function<String, Integer> f = new Function<String, Integer>() { @Override public Integer apply(String t) { return t.length(); } }; System.out.println(f.apply("안녕")); //2 </pre>
람다식	<pre> Function<String, Integer> f1 = str->str.length(); System.out.println(f1.apply("안녕")); //2 </pre>

확장형 Function

확장형 Function은 먼저 표준형인 Function<T,R>에서 입력 타입을 특정 타입으로 고정하거나 입력 타입의 수를 늘린 인터페이스와 출력 타입을 고정시킨 인터페이스로 구성된다.

입력 타입 고정 또는 2개의 입력을 가지는 확장형 Function

IntFunction<R>, LongFunction<R>, DoubleFunction<R>은 표준형 Function<T, R>에서 입력으로 사용된 제네릭 변수 T를 각각 int, long, double로 고정한 인터페이스이다. BiFunction<T, U, R>은 T와 U 타입의 입력을 받아 R 타입으로 변환하는 함수형 인터페이스이다. 이들 4개의 인터페이스 모두 리턴 타입은 R 타입으로 동일하기 때문에 모든 확장형 Function 인터페이스의 추상 메서드는 표준형과 동일한 apply()이다.



입력 타입 고정 또는 2개의 입력을 가지는 확장형 Function 함수형 인터페이스의 입출력

다음은 입력 타입 고정 또는 2개의 입력을 가지는 확장형 Function 함수형 인터페이스의 객체를 람다식으로 정의 후 메서드를 호출한 예이다.

입력 타입이 확장된 확장형 Function 함수형 인터페이스의 객체 생성 및 메서드 호출의 예

```
IntFunction<Double> f2 = (num)->(double)num;           //int->double
LongFunction<String> f3 = (num)->String.valueOf(num);  //long->문자열
DoubleFunction<Integer> f4 = (num)->(int)num;          //double->int
BiFunction<String, Integer, String> f5 = (name, age)->"이름: "+name+", 나이 : "+age;
System.out.println(f2.apply(10));                      //10.0
System.out.println(f3.apply(20L));                     //20
System.out.println(f4.apply(30.5));                    //30
System.out.println(f5.apply("홍길동", 16));            //이름: 홍길동, 나이 : 16
```

구현된 기능을 하나씩 살펴보면 f2는 int 값을 입력받아 Double 타입으로 바꾸어 리턴하며, f3는 long 값을 입력받아 이를 문자열로 변환한다. f4는 double 값을 입력받아 이를 Integer 타입으로 다운캐스팅하여 리턴하며, f5는 String과 Integer 2개의 값을 입력받아 이들이 각각 조합된 문자열을 리턴한다.

출력 타입이 고정된 확장형 Function

그럼 이번에는 입력 타입 대신 출력 타입이 고정된 경우를 살펴보자. ToIntFunction<T>, ToLongFunction<T>, ToDoubleFunction<T> 인터페이스는 표준형 Function<T, R>에서 출력 타입을 각각 int, long, double로 고정시킨 확장형 Function 인터페이스이다. 인터페이스의 이름도 출력이 고정되었음을 직관적으로 알 수 있도록 ToXXXXFunction의 형태를 가진다. 출력 타입이 기본

자료형이고 모두 다르기 때문에 '기본 메서드명 + As + 리턴 타입명'의 규칙성에 따라 이들 인터페이스는 각각 `applyAsInt()`, `applyAsLong()`, `applyAsDouble()` 메서드를 추상 메서드로 가진다.



출력 타입이 고정된 확장형 Function 함수형 인터페이스의 입출력

다음은 출력 타입이 고정된 확장형 Function 함수형 인터페이스의 객체를 람다식으로 정의 후 메서드를 호출한 예이다.

출력 타입이 확장된 확장형 Function 함수형 인터페이스의 객체 생성 및 메서드 호출의 예

```

ToIntFunction<String> f6 = (str)->str.length();           //str->int
ToLongFunction<Double> f7 = (num)->num.longValue();       //double->long
ToDoubleFunction<Integer> f8 = (num)->num/100.0;          //int->double
System.out.println(f6.applyAsInt("반갑습니다."));         //6
System.out.println(f7.applyAsLong(58.254));               //58
System.out.println(f8.applyAsDouble(250));                //2.5
  
```

f6은 문자열을 매개변수로 입력받아 문장열의 길이를 int 값으로 리턴하고, f7은 Double 값을 입력 받아 이를 long 값으로 변환하여 리턴한다. 마지막으로 f8은 Integer 값을 입력받아 이를 100.0으로 나눈 double 값을 리턴한다.

[Do it 실습] Function<T, R> 함수형 인터페이스(자바 API 제공)

Function_FunctionalInterface.java

```

1 package sec02_functioninterfaceinjavaAPI.EX04_Function_FunctionalInterface;
2 import java.util.function.BiFunction;
3 import java.util.function.DoubleFunction;
4 import java.util.function.Function;
5 import java.util.function.IntFunction;
6 import java.util.function.LongFunction;
7 import java.util.function.ToDoubleFunction;
8 import java.util.function.ToIntFunction;
9 import java.util.function.ToLongFunction;
  
```



```

10
11 public class Function_FunctionalInterface {
12     public static void main(String[] args) {
13         // 익명 이너클래스 방법으로 객체 생성
14         Function<String, Integer> f = new Function<String, Integer>() {
15             @Override
16             public Integer apply(String t) {
17                 return t.length();
18             }
19         };
20         System.out.println(f.apply("안녕"));    //2
21         // Function<T, R> 랴다식 표현
22         // 표준형
23         Function<String, Integer> f1 = str-> str.length();
24         System.out.println(f1.apply("안녕"));    //2
25         // 확장형 (입력(T)이 고정)
26         IntFunction<Double> f2 = (num)->(double)num;;    //int->double
27         LongFunction<String> f3 = (num)->String.valueOf(num);    //long->String
28         DoubleFunction<Integer> f4 = (num)->(int)num;;    //double->int
29         BiFunction<String, Integer, String> f5 = (name, age)->"이름은 = "
30             + name + " 나이는 = " + age; //String, Integer->String
31         System.out.println(f2.apply(10));    //10.0
32         System.out.println(f3.apply(20L));    //20
33         System.out.println(f4.apply(30.5));    //30
34         System.out.println(f5.apply("홍길동", 16)); //이름은 = 홍길동 나이는 =
35
36         System.out.println();
37         // 확장형(출력(R)이 고정)
38         ToIntFunction<String> f6 = (str)->str.length();    //String->int
39         ToLongFunction<Double> f7 = (num)->num.longValue(); //double->long
40         ToDoubleFunction<Integer> f8 = (num)->num/100.0;    //Integer->double
41         System.out.println(f6.applyAsInt("반갑습니다"));    //5
42         System.out.println(f7.applyAsLong(58.254));    //58
43         System.out.println(f8.applyAsDouble(250));    //2.5
44     }
}

```

실행 결과

```
2
2
10.0
20
30
이름은 = 홍길동 나이는 = 16

5
58
2.5
```

14-24. 표준형 `Function<T, R>` 함수형 인터페이스의 객체를 익명 inner클래스 방식 및 람다식으로 생성한다.

26-30. `Function<T, R>`의 제네릭 변수 중 `T`가 각각 `int`, `long`, `double`로 지정된 경우의 확장형 `Function<R>` 인터페이스의 객체 생성과 2개의 입력변수(`String`, `Integer`)를 받아 `String`을 리턴하는 `BiFunction<String, Integer, String>` 객체를 람다식으로 생성한다.

31-34. 제네릭 변수 `T`가 고정되거나 확장된 `Function<R>` 및 `BiFunction<T, U, R>` 객체의 구현 메서드 호출. 모두 동일한 메서드를 가진다.

37-39. `Function<T, R>`의 제네릭 변수 중 `R`이 각각 `int`, `long`, `double`로 지정된 경우의 확장형 `Function<T>` 인터페이스의 객체를 람다식으로 생성한다.

40-42. 제네릭 변수 `R`이 고정된 확장형 `Function<T>` 객체의 구현 메서드를 호출한다. 리턴 타입이 기본 자료형이면서 각각 서로 다른 타입을 리턴하기 때문에 내부 포함 메서드의 이름이 서로 다르다.

UnaryOperator<T>, BinaryOperator<T>

표준형 UnaryOperator<T>, BinaryOperator<T>

마지막으로 알아볼 자바 API에서 제공하는 함수형 인터페이스는 UnaryOperator<T>와 BinaryOperator<T>로 각각의 원형은 다음과 같다.

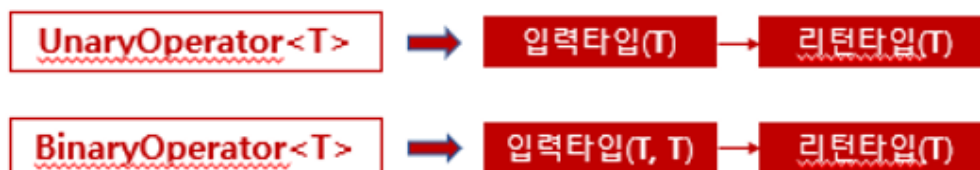
표준형 UnaryOperator<T> 함수형 인터페이스의 원형

```
interface UnaryOperator<T> {  
    public abstract T apply(T t);  
}
```

표준형 BinaryOperator<T> 함수형 인터페이스의 원형

```
interface BinaryOperator<T> {  
    public abstract T apply(T t1, T t2);  
}
```

UnaryOperator<T>와 BinaryOperator<T>는 T 타입의 입력을 받아 T 타입의 값을 리턴하는 함수형 인터페이스이다. 즉, T 타입의 입력값을 일정 연산을 통해 결과를 계산하고, 이 결과를 연산 결과로 리턴하는 연산기(operator) 기능을 수행하는 인터페이스이다. 접두어로 사용된 Unary와 Binary는 입력의 개수를 나타내는 것으로 각각 입력이 1개와 2개인 경우이다. 이들 인터페이스들은 Function<T,R> 인터페이스에서 출력이 입력과 동일한 T 타입인 특수한 경우로 볼 수 있으며, 실제로 추상 메서드 이름 또한 Function<T, R>과 동일한 apply()이다.



표준형 UnaryOperator<T>와 BinaryOperator<T>의 함수형 인터페이스의 입출력

다음은 익명 이너클래스와 람다식으로 각각 Integer 타입과 String 타입 입력의 연산을 수행하는 UnaryOperator<T>와 BinaryOperator<T> 객체의 생성 예시이다.

익명 이너클래스	<pre> UnaryOperator<Integer> uo = new UnaryOperator<Integer>() { @Override public Integer apply(Integer t) { return t * 2; } }; System.out.println(uo.apply(5)); // 10 BinaryOperator<String> bo = new BinaryOperator<String>() { @Override public String apply(String t, String u) { return t + u; } }; System.out.println(bo.apply("안녕", "하세요")); // 안녕하세요 </pre>
람다식	<pre> UnaryOperator<Integer> uo1 = value -> value * 2; System.out.println(uo1.apply(5)); // 10 BinaryOperator<String> bo1 = (value1, value2) -> value1 + value2; System.out.println(bo1.apply("안녕", "하세요")); // 안녕하세요 </pre>

UnaryOperator<Integer> 객체인 uo와 uo1내의 구현 메서드에서는 Integer 객체를 입력받아 이를 2배한 후 다시 Integer로 리턴한다. BinaryOperator<String> 객체인 bo와 bo1내의 구현메서드에서는 2개의 문자열을 입력받아 이를 연결한 하나의 문자열을 리턴한다.

확장형 UnaryOperator, BinaryOperator

확장형 UnaryOperator와 BinaryOperator는 각각의 표준형 인터페이스에서 제네릭 타입을 특정타입으로 고정한 인터페이스이다. 고정된 타입은 각각 int, long, double이며 확장된 인터페이스의 이름은 UnaryOperator와 BinaryOperator 앞에 Int, Long, Double이 붙은 형태를 가진다. 리턴 타입이 기본 자료형이면서 모두 다르기 때문에 ‘기본 메서드명 + As + 리턴

타입명'의 규칙에 따라 각각 포함하고 있는 추상 메서드의 이름은 `applyAsInt()`, `applyAsLong()`, `applyAsDouble()`이다.



확장형 `UnaryOperator`와 확장형 `BinaryOperator` 함수형 인터페이스의 입출력

다음은 이들 확장형 `UnaryOperator`와 확장형 `BinaryOperator` 함수형 인터페이스의 객체를 람다식으로 정의 후 메서드를 호출한 예이다.

UnaryOperator와 확장형 BinaryOperator 인터페이스의 객체 생성 및 메서드를 호출의 예

```
IntUnaryOperator uo2 = (num)->num*10;           //int->int
LongUnaryOperator uo3 = (num)->num+20L;          //long->long
DoubleUnaryOperator uo4 = (num)->num*10.0;        //double->double
System.out.println(uo2.applyAsInt(10));           //100
System.out.println(uo3.applyAsLong(20L));          //40
System.out.println(uo4.applyAsDouble(30.5));       //305.0

IntBinaryOperator bo2 = (num1,num2)->num1+num2;   //int->int
LongBinaryOperator bo3 = (num1,num2)->num1*num2;  //long->long
DoubleBinaryOperator bo4 = (num1,num2)->num1/num2; //double->double
System.out.println(bo2.applyAsInt(10,20));         //30
System.out.println(bo3.applyAsLong(20L, 10L));     //200
System.out.println(bo4.applyAsDouble(42.0, 12.0)); //3.5
```

확장형 `UnaryOperator` 객체인 `uo2`, `uo3`, `uo4`의 구현 메서드에서는 매개변수로 입력된 값을 각각 곱하기 10, 더하기 20, 곱하기 10.0 한 결과를 리턴한다. 마지막으로 확장형 `BinaryOperator`

객체인 bo2, bo3, bo4는 각각 입력된 2개의 값의 덧셈, 곱셈, 나눗셈 결과를 리턴하도록 내부 메서드를 구현하였다.

[Do it 실습] UnaryOperator<T>와 BinaryOperator<T> 함수형 인터페이스(자바 API 제공)

UnaryBinaryOperator_FunctionalInterface.java

```

1  package sec02_functioninterfaceinjavaAPI.EX05_UnaryBinaryOperator_FunctionalInterface;
2  import java.util.function.BinaryOperator;
3  import java.util.function.DoubleBinaryOperator;
4  import java.util.function.DoubleUnaryOperator;
5  import java.util.function.IntBinaryOperator;
6  import java.util.function.IntUnaryOperator;
7  import java.util.function.LongBinaryOperator;
8  import java.util.function.LongUnaryOperator;
9  import java.util.function.UnaryOperator;
10
11 public class UnaryBinaryOperator_FunctionalInterface {
12     public static void main(String[] args) {
13         // 1. 익명 이너클래스를 이용한 객체 생성
14         // Function<T, T> = UnaryOperator<T>
15         UnaryOperator<Integer> uo = new UnaryOperator<Integer>() {
16             @Override
17             public Integer apply(Integer t) {
18                 return t*2;
19             }
20         };
21         BinaryOperator<String> bo = new BinaryOperator<String>() {
22             @Override
23             public String apply(String t, String u) {
24                 return t+u;
25             }
26         };
27         System.out.println(uo.apply(5)); //10
28         System.out.println(bo.apply("안녕", "하세요")); //안녕하세요
29         System.out.println();
30         // 2. 람다식

```

```

31 // 표준형
32 UnaryOperator<Integer> uo1 = value -> value*2;
33 System.out.println(uo1.apply(5)); //10
34 BinaryOperator<String> bo1 = (value1, value2)->value1+value2;
35 System.out.println(bo1.apply("안녕", "하세요")); //안녕하세요
36 System.out.println();
37 // 확장형
38 IntUnaryOperator uo2 = (num)->num*10; //int->int
39 LongUnaryOperator uo3 = (num)->num+20L; //long->long
40 DoubleUnaryOperator uo4 = (num)->num*10.0; //double->double
41 System.out.println(uo2.applyAsInt(10)); //100
42 System.out.println(uo3.applyAsLong(20L)); //40
43 System.out.println(uo4.applyAsDouble(30.5)); //305.0
44 System.out.println();
45 IntBinaryOperator bo2 = (num1, num2)->num1+num2; //int->int
46 LongBinaryOperator bo3 = (num1, num2)->num1*num2; //long->long
47 DoubleBinaryOperator bo4 = (num1, num2)->num1/num2; //double->double
48 System.out.println(bo2.applyAsInt(10, 20)); //30
49 System.out.println(bo3.applyAsLong(20L, 10L)); //200
50 System.out.println(bo4.applyAsDouble(40.2, 12.0)); //3.35
51 }
52 }
53

```

실행 결과

```

10
안녕하세요

10
안녕하세요

100
40
305.0

```

30

200

3.35

16-29. 표준형 `UnaryOperator<T>`와 `BinaryOperator<T>` 함수형 인터페이스의 객체를 익명 이너클래스 방식으로 생성 및 구현 메서드를 호출한다.

33-36. 표준형 `UnaryOperator<T>`와 `BinaryOperator<T>` 함수형 인터페이스의 객체를 람다식으로 생성 및 구현 메서드를 호출한다.

39-44. `UnaryOperator<T>`의 제네릭 변수가 각각 `int`, `long`, `double`로 지정된 경우의 확장형 `UnaryOperator` 인터페이스의 객체 생성과 구현 메서드를 호출한다. 리턴 타입이 기본 자료형이면서 각각 서로 다른 타입을 리턴하기 때문에 내부 포함 메서드의 이름이 서로 다르다.

46-51. `BinaryOperator<T>`의 제네릭 변수가 각각 `int`, `long`, `double`로 지정된 경우의 확장형 `BinaryOperator` 인터페이스의 객체 생성과 구현 메서드를 호출한다. 리턴 타입이 기본 자료형이면서 각각 서로 다른 타입을 리턴하기 때문에 내부 포함 메서드의 이름이 서로 다르다