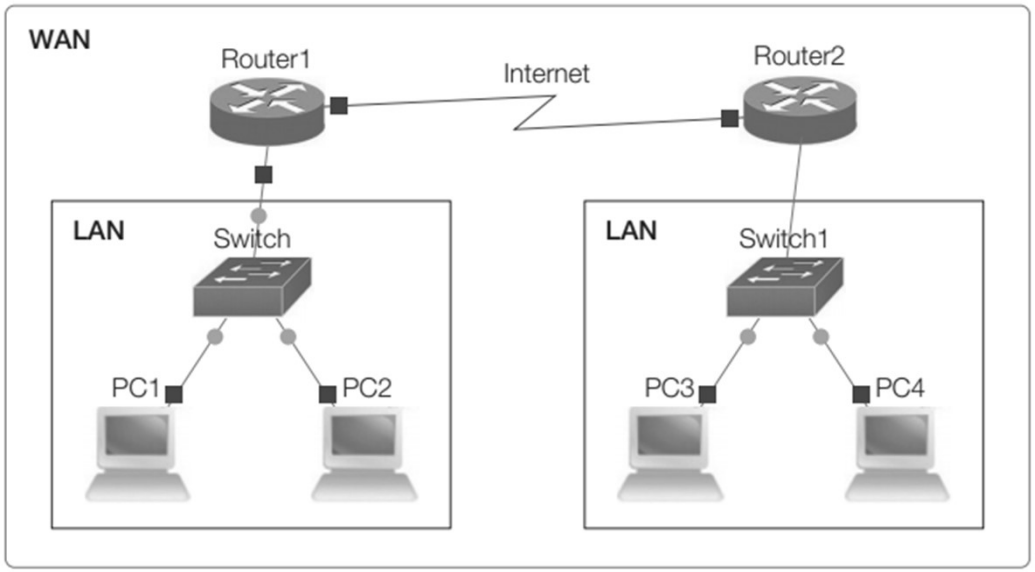
Network

목 차

네트워크 기초

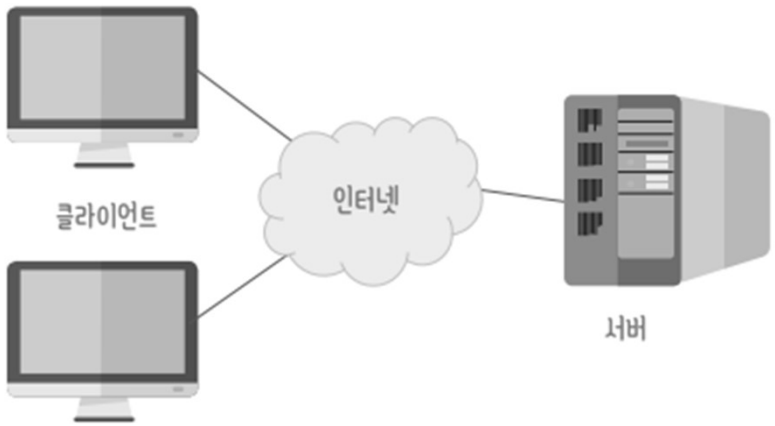
❖ 네트워크

- 네트워크: 여러 컴퓨터들을 통신 회선으로 연결한 것
- LAN: 가정, 회사, 건물, 특정 영역에 존재하는 컴퓨터를 연결한 것
- WAN: LAN을 연결한 것 = 인터넷



❖ 서버와 클라이언트

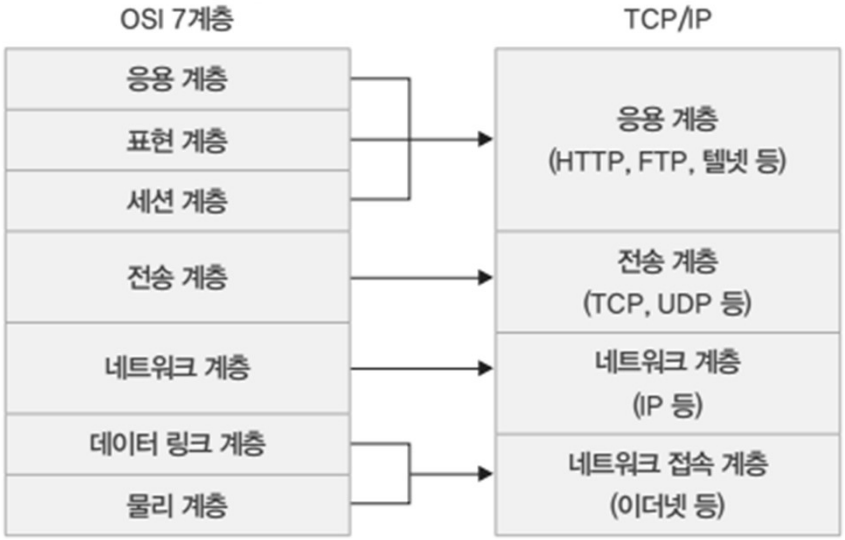
- 서버는 클라이언트보다 미리 실행되어 클라이언트의 요청을 대기
- 클라이언트는 서비스를 요청하기 전에 먼저 서버와 연결 시도
- 클라이언트가 연결을 요청하면 서버는 수용 혹은 거부
- 클라이언트와 서버가 서로 연결되면 클라이언트의 요청을 서버가 처리해서 클라이언트에 응답



네트워크 기초

❖ TCP와 UDP

- OSI 모델과 TCP/IP
- TCP
  - 전화와 유사한 연결 지향 프로토콜
  - 데이터 손실이 없고 데이터의 전달 순서가 보장
  - 연결 설정과 해제에 따른 시간적 부담이 발생
- UDP
  - 편지와 유사한 비연결 지향 프로토콜
  - 데이터 전달 속도가 빠름
  - 데이터 손실 발생 가능성이 있고 데이터의 전달 순서를 보장 않음



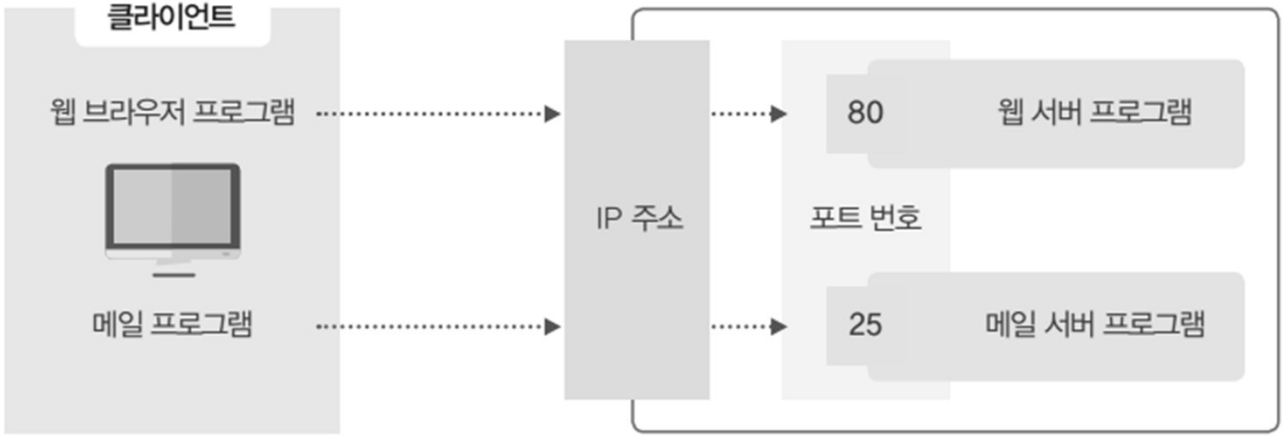
❖ IP 주소와 DNS 서버

- IP 주소
  - 컴퓨터에 부여된 고유 주소. 즉 인터넷 주소
  - 네트워크 어댑터마다 할당
  - 32비트로 구성되며, xxx.xxx.xxx.xxx 형식으로 표현
  - ipconfig(윈도우), ifconfig(맥OS) 명령어로 네트워크 어댑터에 어떤 IP 주소가 부여되어 있는지 확인

- DNS 서버
  - 114 전화국과 유사하며 URL을 IP 주소로 변환하는 서버
  - 숫자보다 문자열이 편하기 때문에 IP 주소보다는 www.naver.com처럼 URL을 사용

❖ 포트

- 논리적인 통신 연결 번호를 의미하며, 0~65,535사이의 번호를 사용
- 0~1,023은 인터넷주소관리기구가 특정 프로그램에 예약
- 서버 프로그램은 고정된 포트 번호를 사용. 예를 들어 파일 서버는 21번, 메일 서버는 25번, 웹 서버는 80번 포트를 사용
- 클라이언트 컴퓨터에서 웹 브라우저로 웹 서버에 접속하려면 해당 웹 서버의 IP주소 외에 80번 포트로 연결을 요청



구분명	범위	설명
Well Know Port Numbers	0~1023	국제인터넷주소관리기구(ICANN)가 특정 애플리케이션용으로 미리 예약한 Port
Registered Port Numbers	1024~49151	회사에서 등록해서 사용할 수 있는 Port
Dynamic Or Private Port Numbers	49152~65535	운영체제가 부여하는 동적 Port 또는 개인적인 목적으로 사용할 수 있는 Port

IP 주소 얻기

❖ InetAddress

- IP 주소를 나타내는 클래스
- java.net 패키지에 있으며 자바 프로그램에서 IP번호와 URL을 조사할 때 사용
- 생성자

```
static InetAddress[] getAllByName(String host)
static InetAddress getByAddress(byte[] addr)
static InetAddress getByAddress(String host, byte[] addr)
static InetAddress getByName(String host)
static InetAddress getLocalHost()
```

- 여기서 addr은 다음과 같이 사용

```
byte[] addr = new byte[4];
addr[0] = (byte)202;
addr[1] = (byte)179;
addr[2] = (byte)177;
addr[3] = (byte)21;
```

IP 주소 얻기

❖ InetAddress

- InetAddress 클래스가 제공하는 주요 메서드

메서드	설명
byte[ ] getAddress( )	IP 주소를 배열 타입으로 변환한다.
String getHostAddress( )	IP 주소를 String 타입으로 변환한다.
String getHostName( )	호스트 이름을 String 타입으로 변환한다.

- 예제 : sec01/InetAddressDemo

```
호스트 이름을 입력하시오 : www.hanbit.co.kr
www.hanbit.co.kr의 IP 주소 : 218.38.58.195
로컬 IP 주소 : 192.168.0.000
```

❖ 소켓

- Socket 클래스가 제공하는 주요 메서드

메서드	설명
void close( )	소켓을 닫는다.
void connect(SocketAddress endpoint)	소켓을 서버와 연결한다.
InetAddress getInetAddress( )	원격 컴퓨터의 InetAddress 객체를 가져온다.
InetAddress getLocalAddress( )	로컬 컴퓨터의 InetAddress 객체를 가져온다.
InputStream getInputStream( )	소켓에서 InputStream 객체를 가져온다.
OutputStream getOutputStream( )	소켓에서 OutputStream 객체를 가져온다.
int getLocalPort( )	로컬 컴퓨터의 포트 번호를 가져온다.
int getPort( )	원격 컴퓨터의 포트 번호를 가져온다.

❖ 소켓

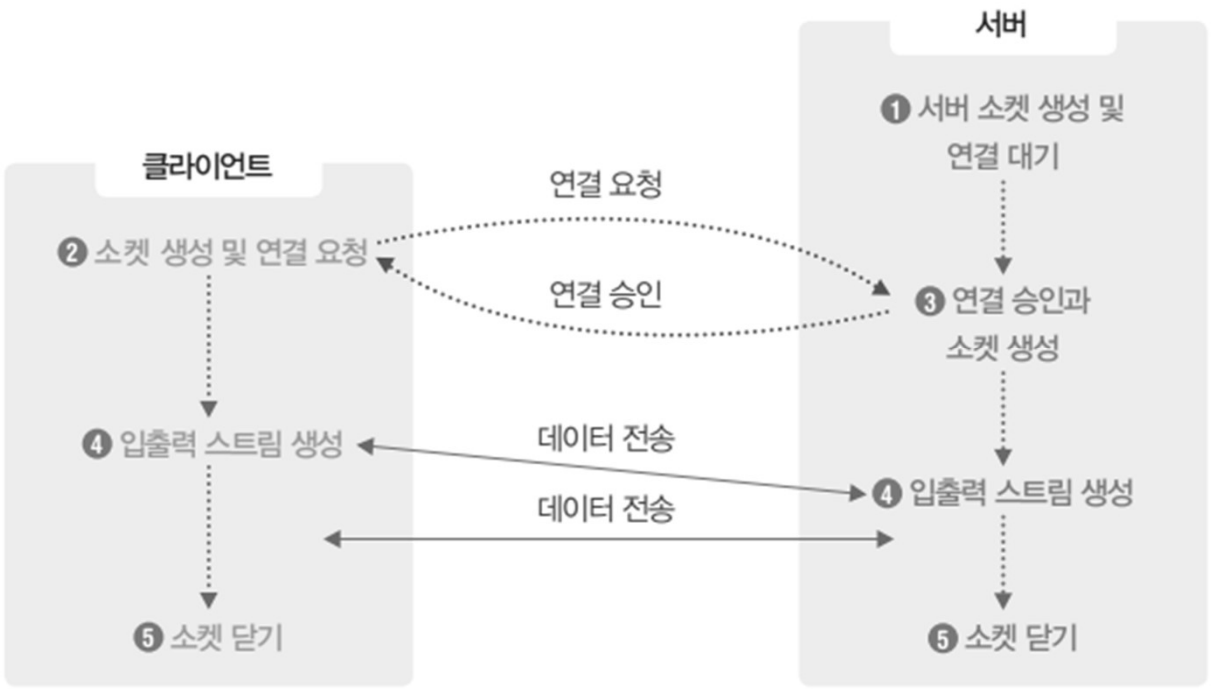
- 서버는 다수의 클라이언트와 상대하므로 클라이언트와 달리 연결 요청을 처리할 수 있는 ServerSocket 클래스를 사용
- ServerSocket은 클라이언트가 연결을 요청하면 대응하는 Socket 객체를 생성하는 역할
- 사용하는 ServerSocket클래스의 생성자

```
ServerSocket(int port)
```

- ServerSocket 클래스가 제공하는 주요 메서드

메서드	설명
Socket accept( )	클라이언트의 연결 요청을 받아 Socket 객체를 생성한다.
void close( )	서버 소켓을 닫는다.
public InetAddress getInetAddress( )	소켓에 연결된 인터넷 주소를 가져온다.

❖ 소켓을 이용한 클라이언트·서버 통신 과정



❖ 예제

- 문자열을 받는 간단한 서버 : sec02/SimpleServer
- 문자열을 보내는 간단한 클라이언트 : sec02/SimpleClient

- 명령 창에서 서버와 클라이언트의 통신 과정

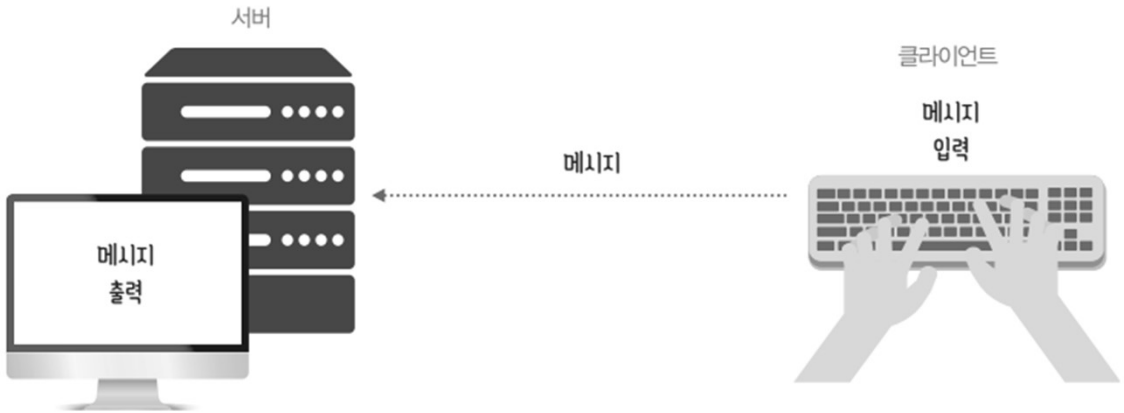
서버

```
명령 프롬프트
❶ D:\workspace\chap17\bin>java sec02. SimpleServer
❸ 받은 문자열 = 안녕, 단순 서버야.

D:\workspace\chap17\bin>
```

클라이언트

```
명령 프롬프트
❷ D:\workspace\chap17\bin>java sec02. SimpleServer
받은 문자열 = 안녕, 단순 서버야.

D:\workspace\chap17\bin>
```

❖ 에코 프로그램



❖ 예제 : 일대일 서버 및 클라이언트

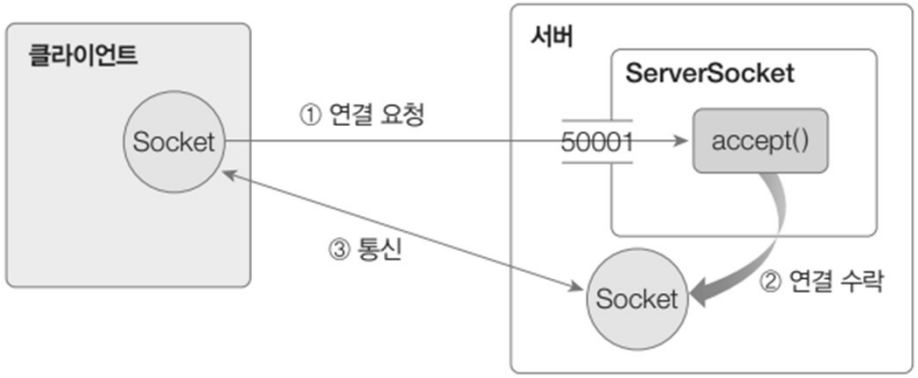
- 일대일 에코 서버 : sec03/Echo1Server
- 일대일 에코 클라이언트 : sec03/EchoClient

❖ 예제 : 일대다 서버 및 클라이언트

- 일대다 에코 서버 : sec03/Echo2Server
- 일대다 클라이언트 : 일대일 클라이언트와 동일

❖ TCP

- TCP는 연결형 프로토콜로, 상대방이 연결된 상태에서 데이터를 주고 받는 전송용 프로토콜
- 클라이언트가 연결 요청을 하고 서버가 연결을 수락하면 통신 회선이 고정되고, 데이터는 고정 회선을 통해 전달. TCP는 보낸 데이터가 순서대로 전달되며 손실이 발생하지 않음
- ServerSocket은 클라이언트의 연결을 수락하는 서버 쪽 클래스이고, Socket은 클라이언트에서 연결 요청할 때와 클라이언트와 서버 양쪽에서 데이터를 주고 받을 때 사용되는 클래스



❖TCP 서버

- TCP 서버 프로그램을 개발하려면 우선 ServerSocket 객체를 생성

```
ServerSocket serverSocket = new ServerSocket(50001);
```

- 기본 생성자로 객체를 생성하고 Port 바인딩을 위해 bind() 메소드를 호출해도 ServerSocket 생성

```
ServerSocket serverSocket = new ServerSocket();
serverSocket.bind(new InetSocketAddress(50001));
```

- 여러 개의 IP가 할당된 서버 컴퓨터에서 특정 IP에서만 서비스를 하려면 InetSocketAddress의 첫 번째 매개값으로 해당 IP를 줌

```
ServerSocket serverSocket = new ServerSocket();
serverSocket.bind( new InetSocketAddress("xxx.xxx.xxx.xxx", 50001) );
```

- Port가 이미 다른 프로그램에서 사용 중이라면 BindException이 발생. 다른 Port로 바인딩하거나 Port를 사용 중인 프로그램을 종료하고 다시 실행해야 함

- ServerSocket이 생성되면 연결 요청을 수락을 위해 accept( ) 메소드를 실행
- accept()는 클라이언트가 연결 요청하기 전까지 블로킹(실행 멈춘 상태) 클라이언트의 연결 요청이 들어오면 블로킹이 해제되고 통신용 Socket을 리턴

```
Socket socket = serverSocket.accept();
```

- 리턴된 Socket을 통해 연결된 클라이언트의 IP 주소와 Port 번호를 얻으려면 getRemoteSocketAddress ( ) 메소드를 호출해서 InetSocketAddress를 얻은 다음 getHostName()과 getPort() 메소드를 호출

```
InetSocketAddress isa = (InetSocketAddress) socket.getRemoteSocketAddress();
String clientIp = isa.getHostName();
String portNo = isa.getPort();
```

- ServerSocket의 close() 메소드를 호출해서 Port 번호를 언바이딩해야 서버 종료

```
serverSocket.close();
```

❖ TCP 클라이언트

- 클라이언트가 서버에 연결 요청을 하려면 Socket 객체를 생성할 때 생성자 매개값으로 서버 IP 주소와 Port 번호를 제공
- 로컬 컴퓨터에서 실행하는 서버로 연결 요청을 할 경우에는 IP 주소 대신 localhost 사용 가능

```
Socket socket = new Socket( "IP",  50001 );
```

- 도메인 이름을 사용하려면 DNS에서 IP 주소를 검색하는 생성자 매개값으로 InetSocketAddress 제공

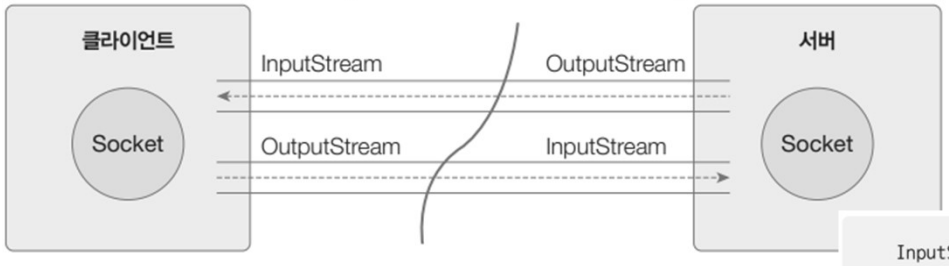
```
Socket socket = new Socket( new InetSocketAddress("domainName", 50001) );
```

- 기본 생성자로 Socket을 생성한 후 connect() 메소드로 연결 요청 가능

```
socket = new Socket();
socket.connect( new InetSocketAddress("domainName", 50001) );
```

❖ 입출력 스트림으로 데이터 주고 받기

- 클라이언트가 연결 요청(connect())을 하고 서버가 연결 수락(accept())했다면, 양쪽의 Socket 객체로부터 각각 InputStream과 OutputStream을 얻을 수 있다.




```
InputStream is = socket.getInputStream();
OutputStream os = socket.getOutputStream();
```

- 상대방에게 데이터를 보낼 때에는 보낼 데이터를 byte[ ] 배열로 생성하고, 이것을 매개값으로 해서 OutputStream의 write() 메소드를 호출
- 문자열을 좀 더 간편하게 보내고 싶다면 보조 스트림인 DataOutputStream을 연결해서 사용

```
String data = "보낼 데이터";
byte[ ] bytes = data.getBytes("UTF-8");
OutputStream os = socket.getOutputStream();
os.write(bytes);
os.flush();
```

```
String data = "보낼 데이터";
DataOutputStream dos = new DataOutputStream(socket.getOutputStream());
dos.writeUTF(data);
dos.flush();
```

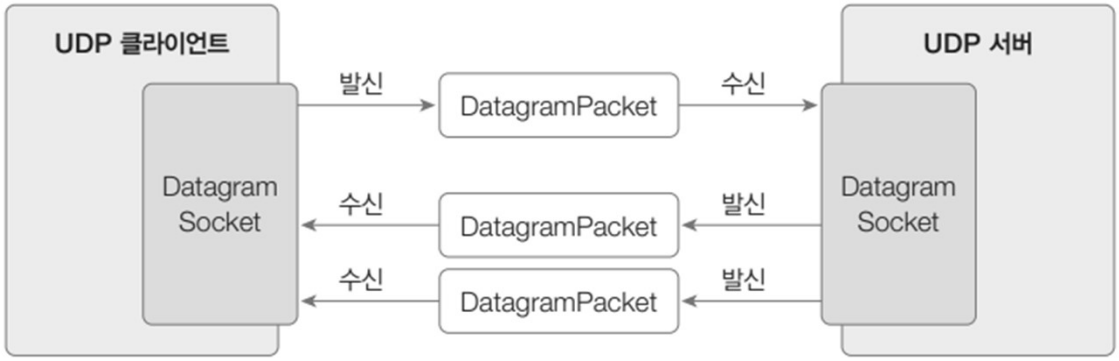
- 데이터를 받기 위해서는 받은 데이터를 저장할 byte[ ] 배열을 하나 생성하고, 이것을 매개값으로 해서 InputStream의 read() 메소드를 호출
- read() 메소드는 읽은 데이터를 byte[ ] 배열에 저장하고 읽은 바이트 수를 리턴
- 문자열을 좀 더 간편하게 받고 싶다면 보조 스트림인 DataInputStream을 연결해서 사용

```
byte[ ] bytes = new byte[1024];
InputStream is = socket.getInputStream();
int num = is.read(bytes);
String data = new String(bytes, 0, num, "UTF-8");
```

```
DataInputStream dis = new DataInputStream(socket.getInputStream());
String data = dis.readUTF();
```

❖ UDP

- 발신자가 일방적으로 수신자에게 데이터를 보내는 방식. TCP처럼 연결 요청 및 수락 과정이 없기 때문에 TCP보다 데이터 전송 속도가 상대적으로 빠름
- 데이터 전달의 신뢰성보다 속도가 중요하다면 UDP를 사용하고, 데이터 전달의 신뢰성이 중요하다면 TCP를 사용
- DatagramSocket은 발신점과 수신점에 해당하고 DatagramPacket은 주고받는 데이터에 해당



❖ UDP 서버

- DatagramSocket 객체를 생성할 때에는 다음과 같이 바인딩할 Port 번호를 생성자 매개값으로 제공

```
DatagramSocket datagramSocket = new DatagramSocket(50001);
```

- receive() 메소드는 데이터를 수신할 때까지 블로킹되고, 데이터가 수신되면 매개값으로 주어진 DatagramPacket에 저장

```
DatagramPacket receivePacket = new DatagramPacket(new byte[1024], 1024);
datagramSocket.receive(receivePacket);
```

- DatagramPacket 생성자의 첫 번째 매개값은 수신된 데이터를 저장할 배열이고 두 번째 매개값은 수신할 수 있는 최대 바이트 수

```
byte[] bytes = receivePacket.getData();
int num = receivePacket.getLength();
```

```
String data = new String(bytes, 0, num, "UTF-8");
```

- getSocketAddress () 메소드를 호출하면 정보가 담긴 SocketAddress 객체를 얻을 수 있음

```
SocketAddress socketAddress = receivePacket.getSocketAddress();
```

- SocketAddress 객체는 클라이언트로 보낼 DatagramPacket을 생성할 때 네 번째 매개값으로 사용

```
String data = "처리 내용";
byte[] bytes = data.getBytes("UTF-8");
DatagramPacket sendPacket = new DatagramPacket( bytes, 0, bytes.length,
socketAddress );
```

- DatagramPacket을 클라이언트로 보낼 때는 DatagramSocket의 send() 메소드를 이용

```
datagramSocket.send( sendPacket );
```

- UDP 서버를 종료하고 싶을 경우에는 DatagramSocket의 close() 메소드를 호출

```
datagramSocket.close();
```

❖ UDP 클라이언트

- 서버에 요청 내용을 보내고 그 결과를 받는 역할
- UDP 클라이언트를 위한 DatagramSocket 객체는 기본 생성자로 생성. Port 번호는 자동 부여

```
String data = "요청 내용";
byte[ ] bytes = data.getBytes("UTF-8");
DatagramPacket sendPacket = new DatagramPacket(
  bytes, bytes.length, new InetSocketAddress("localhost", 50001)
);
```

- 생성된 DatagramPacket을 매개값으로해서 DatagramSocket의 send() 메소드를 호출하면 UDP 서버로 DatagramPacket이 전송

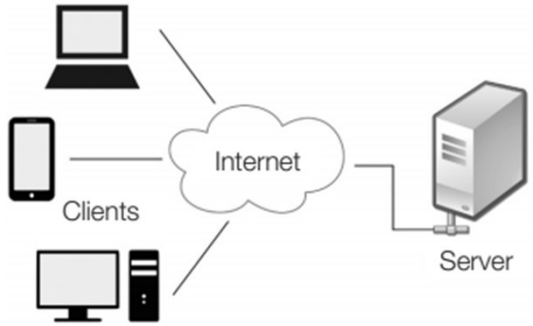
```
datagramSocket.send(sendPacket);
```

- DatagramSocket을 닫으려면 close() 메소드를 호출

```
atagramSocket.close();
```

❖ 서버의 동시 요청 처리

- 일반적으로 서버는 다수의 클라이언트와 통신. 서버는 클라이언트들로부터 동시에 요청을 받아서 처리하고, 처리 결과를 개별 클라이언트로 보내줌



- accept()와 receive()를 제외한 요청 처리 코드를 별도의 스레드에서 작업

TCP 서버

```
while(true) {
  Socket socket = serverSocket.
  accept();

  //데이터 받기
  …
  //데이터 보내기
  …
}
```

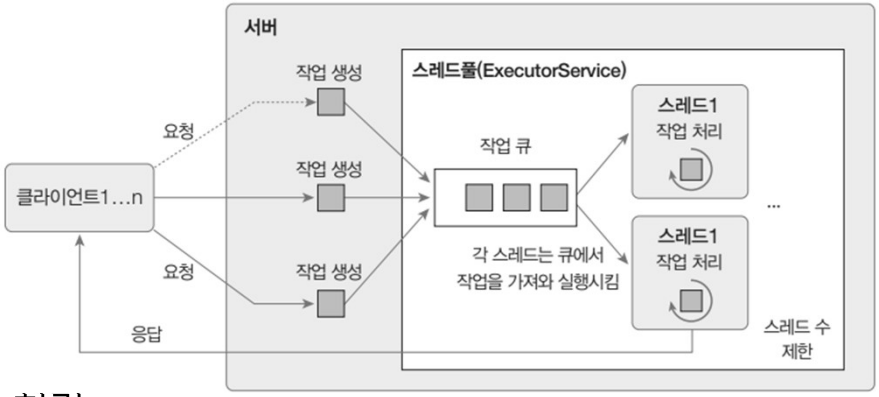
스레드로 처리

UDP 서버

```
while(true) {
  DatagramPacket receivePacket = …
  datagramSocket.
receive(receivePacket);
  …
  //10개의 뉴스를 클라이언트로 전송
  …
}
```

스레드로 처리

- 스레드를 처리할 때 클라이언트의 폭증으로 인한 서버의 과도한 스레드 생성을 방지하기 위해 스레드풀을 사용하는 것이 바람직



❖ TCP EchoServer 동시 요청 처리

- 스레드풀을 이용해서 클라이언트의 요청을 동시에 처리

```
12
13  public class EchoServer {
14    private static ServerSocket serverSocket = null;
15    private static ExecutorService executorService =
          Executors.newFixedThreadPool(10);
16
17    public static void main(String[] args) {
18      System.out.println("-------------------------------------------------");
19      System.out.println("서버를 종료하려면 q를 입력하고 Enter 키를 입력하세요.");
20      System.out.println("-------------------------------------------------");
```

10개의 스레드로 요청을 처리하는 스레드풀 생성

❖ UDP NewsServer 동시 요청 처리

- 스레드풀을 이용해서 클라이언트의 요청을 동시에 처리

```
10  public class NewsServer extends Thread {
11    private static DatagramSocket datagramSocket = null;
12    private static ExecutorService executorService =
          Executors.newFixedThreadPool(10);
13
14    public static void main(String[] args) throws Exception {
15      System.out.println("-----------------------------------------------------");
16      System.out.println("서버를 종료하려면 q를 입력하고 Enter 키를 입력하세요.");
17      System.out.println("-----------------------------------------------------");
```

10개의 스레드로 요청을 처리하는 스레드풀 생성

❖ JSON

- 네트워크로 전달하는 데이터 형식
- 두 개 이상의 속성이 있으면 객체 { }로 표기. 두 개 이상의 값이 있으면 배열 [ ]로 표기

객체 표기	{
	"속성명": 속성값,
	"속성명": 속성값,
	…
	}
		속성명: 반드시 "로 감싸야 함
		속성값으로 가능한 것
		– "문자열", 숫자, true/false
		– 객체 { … }
		– 배열 […]

배열 표기	[항목, 항목, …]	항목으로 가능한 것 – "문자열", 숫자, true/false – 객체 { … } – 배열 […]

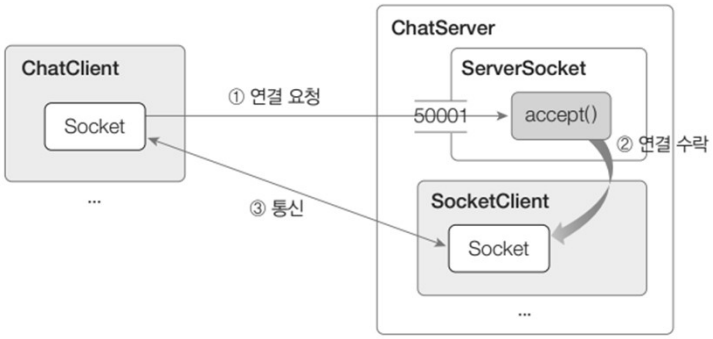
```
{
  "id": "winter",
  "name": "한겨울",
  "age": 25,
  "student": true,
  "tel": { "home": 02-123-1234", "mobile": "010-123-1234" },
  "skill": [ "java", "c", "c++" ]
}
```

- JSON 라이브러리 다운로드: https://github.com/stleary/JSON-java

클래스	용도
JSONObject	JSON 객체 표기를 생성하거나 파싱할 때 사용
JSONArray	JSON 배열 표기를 생성하거나 파싱할 때 사용

❖ 채팅 서버와 클라이언트 구현

- TCP 네트워킹을 이용해서 채팅 서버와 클라이언트를 구현
- 채팅 서버: ChatServer는 채팅 서버 실행 클래스로 클라이언트의 연결 요청을 수락하고 통신용 SocketClient를 생성하는 역할
- 채팅 클라이언트: 단일 클래스 ChatClient는 채팅 서버로 연결을 요청하고, 연결된 후에는 제일 먼저 대화명을 보내며 다음 서버와 메시지를 주고 받음



클래스	용도
ChatServer	– 채팅 서버 실행 클래스 – ServerSocket을 생성하고 50001에 바인딩 – ChatClient 연결 수락 후 SocketClient 생성
SocketClient	– ChatClient와 1:1로 통신
ChatClient	– 채팅 클라이언트 실행 클래스 – ChatServer에 연결 요청 – SocketClient와 1:1로 통신

Thank You!