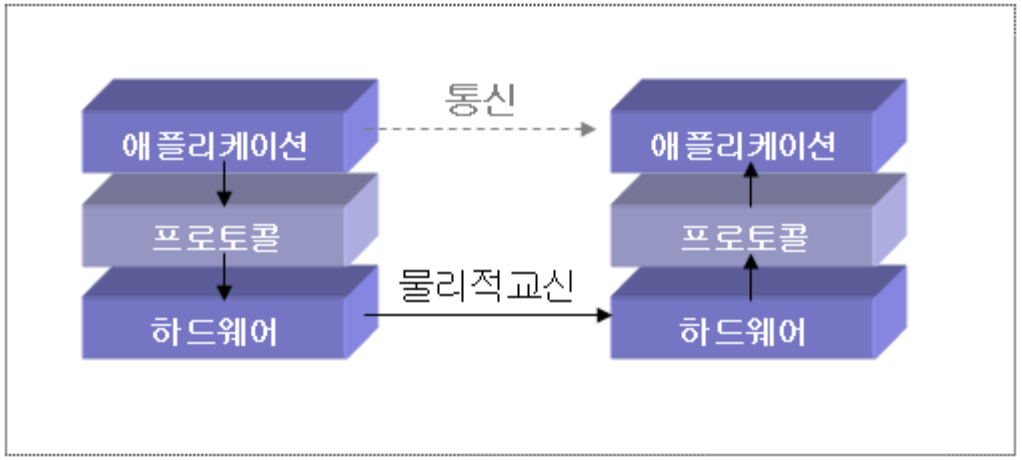
Java Network Programming

네트워크란?

- 같은 데이터 전송 프로토콜을 통해 교신하는 연결된 정보기기 집합
- 네트워킹
  - 하나의 장치로부터 다른 장치로 데이터를 옮기는 것
  - 같은 통신규약(프로토콜)을 사용



네트워크 통신구조

인터넷 주소(IP 주소)

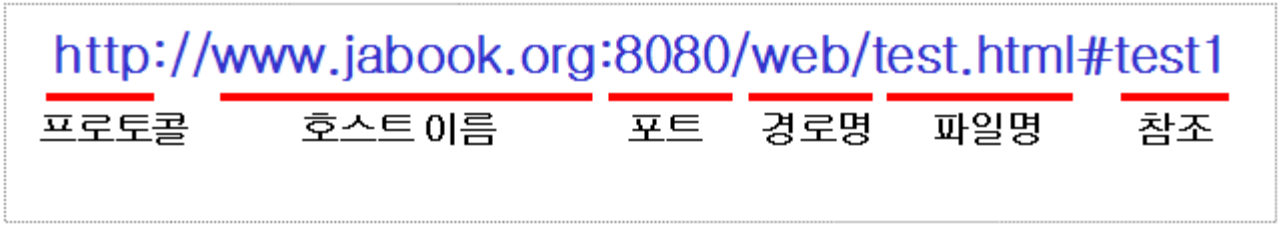
- 인터넷에 연결되어 있는 모든 컴퓨터를 하나의 호스트라 한다.
- 각각의 호스트는 인터넷 주소(Host or IP address)라 불리는 유일한 32비트 숫자로 구성된 주소체계를 이용하여 서로를 구분한다.
- IP 주소는 32비트 숫자를 그대로 다루기 힘들기 때문에, 8비트씩 끊어서 표현하고, 각 자리는 1바이트로 0에서 255까지의 범위를 갖게 된다.

  www.naver.com => 211.238.132.50

- IP 주소는 일반적으로 컴퓨터에 설치하는 랜카드에 저장되어 있다.

URL(Uniform Resource Locator)

- Uniform Resource Locator
- 인터넷상의 자원에 대한 고유한 주소
- 형식
  - protocol://hostname[:port]/path/filename#section



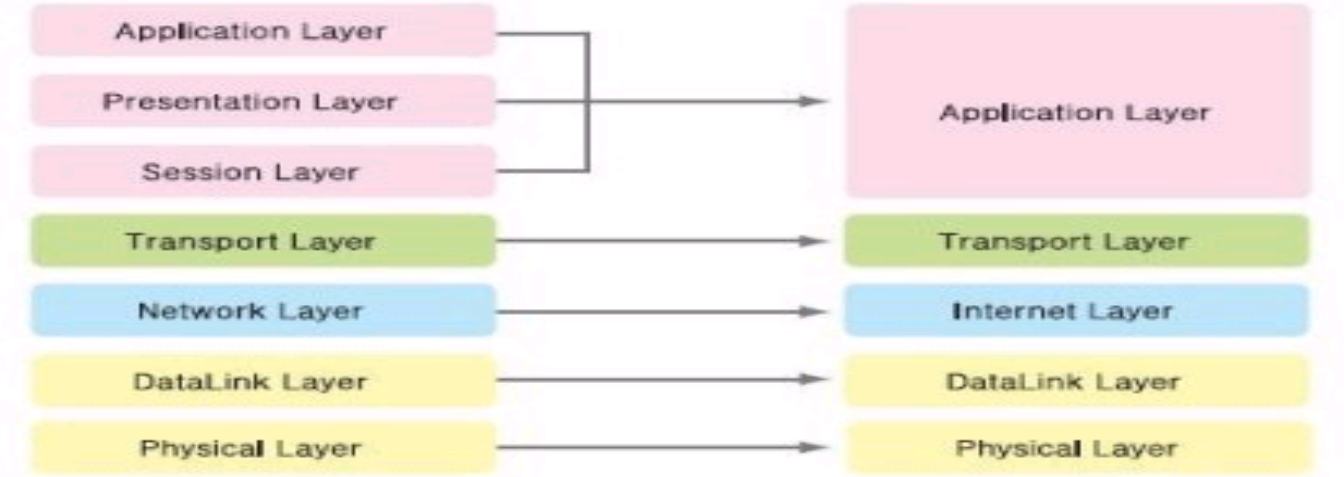
  - 프로토콜(protocol) : 자원에 접근하기 위해 사용되는 프로토콜
  - 호스트 이름(HostName) : 접근하고자 하는 자원이 있는 위치
  - 포트(Port) : TCP 연결에 대한 포트 번호. 생략시 기본포트번호 사용
    - (Echo:7 DayTime:13 FTP:21 Telnet:23 SMPT:25 HTTP:80)
  - 경로명(Path) : 접근하려는 파일이 있는 위치
  - 파일명(File) : 접근하려는 파일

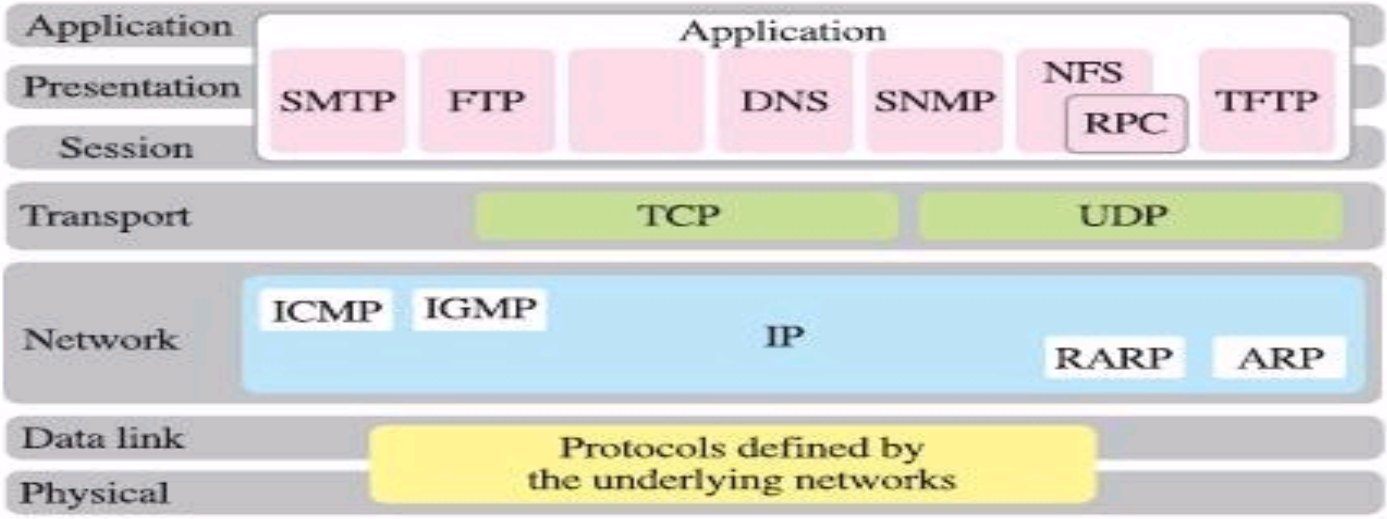
포트와 프로토콜

- 포트(Port)란 크게 2가지 의미를 갖는다.
- 첫번째는 컴퓨터의 주변장치를 접속하기 위한 물리적인 포트가 있다.
- 두 번째는 프로그램에서 사용되는 접속 장소인 논리적인 포트가 있다.
- 포트번호는 0부터 65536까지(16비트값) 이며, 0부터 511사이에 있는 포트는 FTP, TELNET, HTTP, SMTP등과 같은 표준 TCP/IP 어플리케이션용으로 예약되어 있고, 512에서 1024까지는 운영체제에 예약된 포트번호 이기 때문에 될 수 있는 한 사용을 하지 않는 것이 바람직하다.
- 예약된 포트번호의 대표적인 예로 80(HTTP), 21(FTP), 23(TELNET), 25(SMTP), POP3(110) 등이 있다.
- 프로토콜이란 클라이언트와 서버간의 통신규약이다.
- 대표적인 프로토콜의 예을 든다면 인터넷에서 다양한 이기종간의 통신을 지원하는 IP, TCP, UDP 등이 있다.

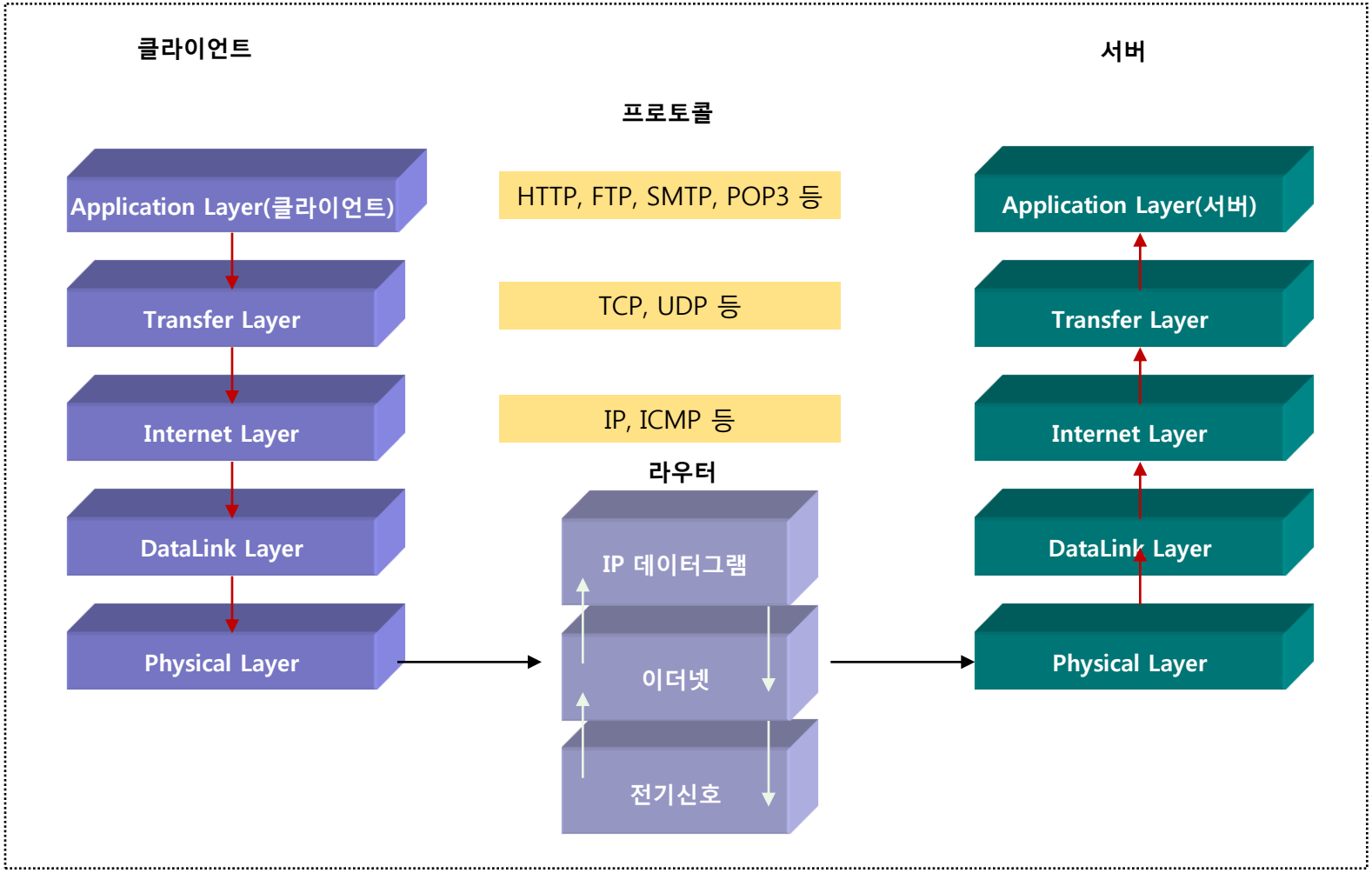
네트워크 통신 모델과 프로토콜

OSI-7Layer 모델과
TCP/IP 모델 비교





TCP와 IP 모델과 프로토콜

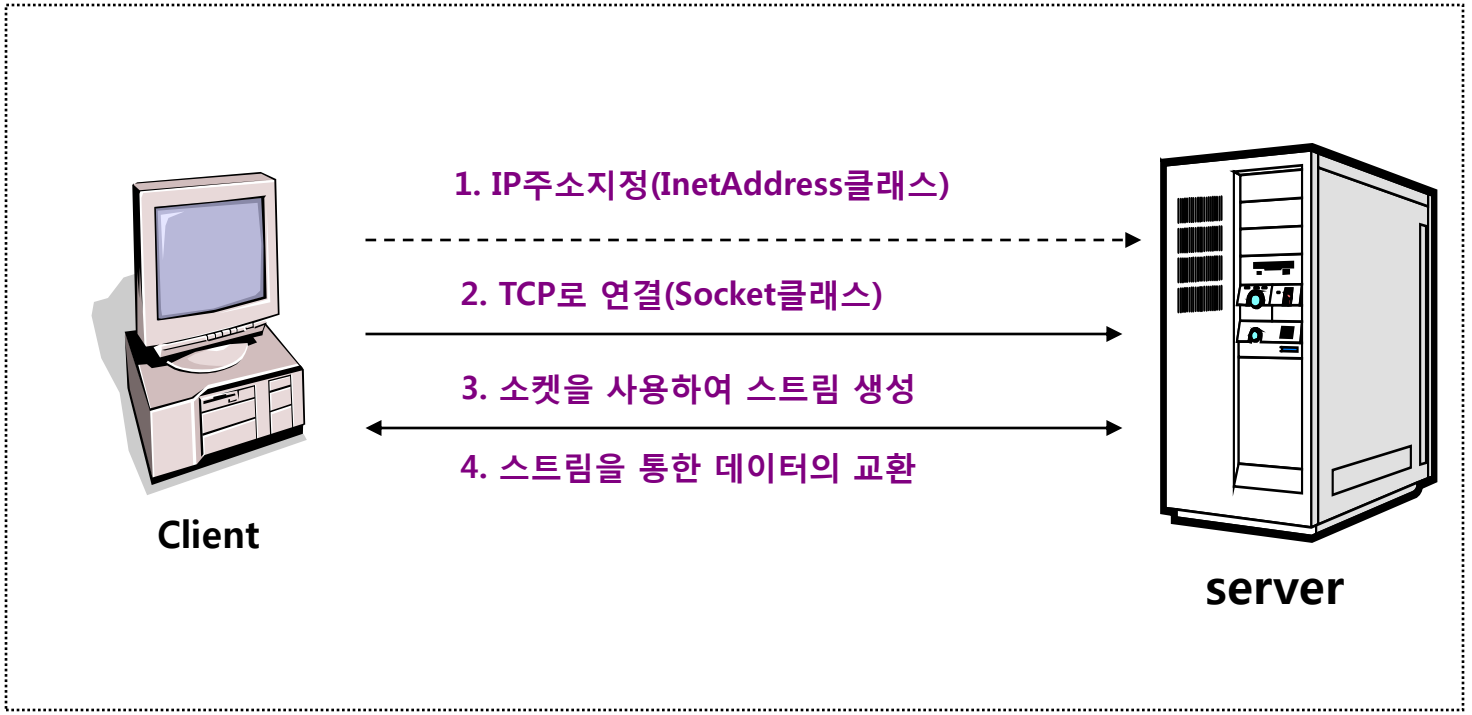


TCP와 UDP 프로토콜 소개

- 네트워크 통신에는 기본적으로 비연결지향(UDP)와 연결지향(TCP)으로 나눈다.
- TCP(Transfer Control Protocol)는 신뢰할 수 있는 end-to-end 스트림 통신 프로토콜이다.
- TCP는 전송자가 보낸 순서대로 수신자에게 바이트가 도착하는 것을 책임진다.
- 데이터가 없어지거나 부분적으로 손상되는 상황을 막기 때문에 애플리케이션을 코딩하기가 쉬워진다.
- UDP(User Datagram Protocol)는 신뢰할 수 없는 end-to-end 데이터그램 서비스를 제공한다.
- UDP 프로토콜은 간단한 요청-응답 메커니즘에 기반하여, 패킷 손실 감지와 재전송은 상대편이 책임지도록 한다.
- TCP- 전화 , UDP- 편지 비유 한다.

TCP/IP 기반 자바 네트워크 프로그래밍 원리

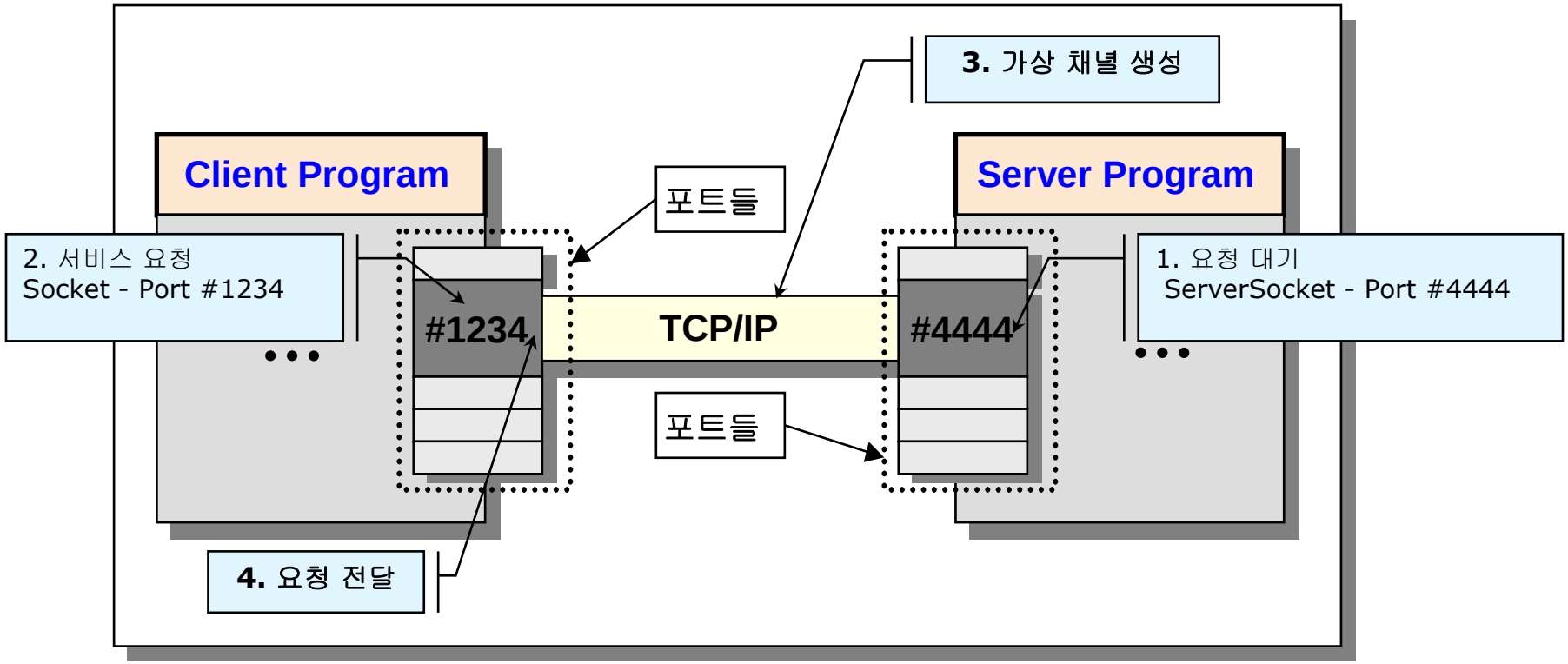
- 스트림(java.io) APIs + 네트워크(java.net) APIs 사용



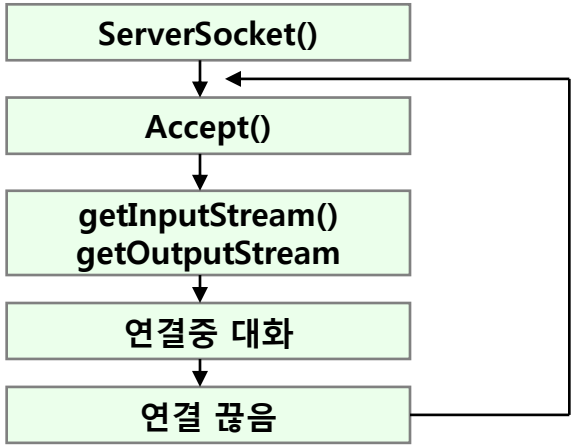
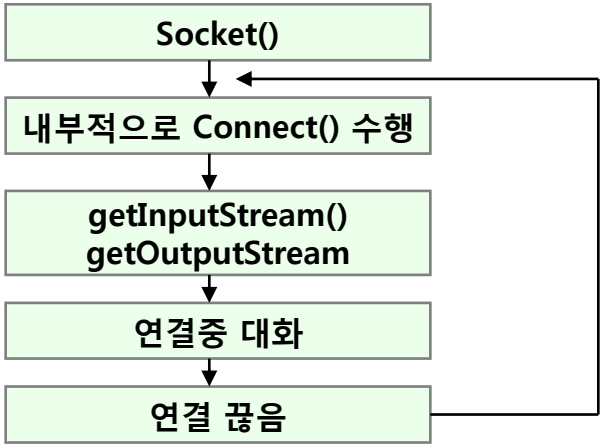
TCP 기반 소켓 프로그래밍

- 자바 프로그램은 OS에서 구현되는 소켓을 통해서 네트워크 통신한다.
- 소켓은 네트워크 통신의 끝부분을 나타낸다.
- 소켓은 실제 데이터가 어떻게 네트워크로 전송되는지 상관하지 않고 읽기/쓰기 인터페이스를 제공한다.
- 소켓은 네트워크 레이어와 트랜스포트 레이어가 캡슐화 되어 있기 때문에 네트워크의 레이어를 신경쓰지 않고 프로그램 할 수 있다.
- 자바는 TCP/IP 계층을 통한 프로그램을 지원하기 위해서 java.net.Socket 와 java.net.ServerSocket 소켓 클래스을 제공 하고 있다.
- 자바 클라이언트는 java.net.Socket 클래스의 객체를 생성하여 TCP 서버와 연결을 시도한다.
- 자바 서버는 java.net.ServerSocket 클래스의 객체를 생성하여 TCP 연결을 청취하여 클라이언트와 서버가 연결되게 된다.
- 클라이언트 편에 있는 소켓을 TCP 소켓 이라고 하고, 서버쪽에 있는 소켓을 TCP 서버 소켓이라고 부른다.

Socket 기반 Stream 통신



Socket 기반 Stream 통신

Socket Server Lifecycle	Socket Client Lifecycle
	

Server Algorithm	Client Algorithm
1. 소켓 생성 2. 소켓과 포트를 바인드(Bind)를 통한 이름 부여 3. 연결을 위한 listen queue 설정 4. 연결을 Accept 5. 연결을 통한 쓰기 작업 6. 연결로부터 읽기 작업 7. 연결 닫기	1. 소켓 생성 2. 서버 포트에 연결 3. 연결로부터 읽기 작업 4. 연결로부터 쓰기 작업 5. 연결 닫기

Socket 기반 Stream 통신

- 1. 클라이언트와 서버간의 통신은 먼저 서버의 실행으로부터 시작한다. 서버가 작동이 되면 TCP 서버소켓을 생성하게 된다.
- 2. TCP 서버 소켓으로 accept()을 통해서 클라이언트의 통신을 기다리고 있다.
- 3. 클라이언트가 클라이언트 TCP 소켓을 생성한다.
- 4. 클라이언트의 TCP 소켓이 생성되면 TCP 서버 소켓과 TCP 연결을 시도한다.
- 5. TCP 서버 소켓은 클라이언트의 클라이언트와 TCP 연결이 이루어 졌다면 2에서 언급한 accept()에 의해서 클라이언트와 통신할 수 있는 TCP 소켓을 생성하게 된다.
- 6. TCP 소켓을 서버에게 전달한다.
- 7. 클라이언트에 있는 TCP 소켓과 서버쪽에 있는 TCP 소켓은 TCP/IP 계층을 통해서 TCP 프로토콜로 통신을 할 수 있는 상태가 된다.
- 8. 클라이언트는 TCP 소켓으로 입출력 스트림을 생성한다.
- 9. 서버쪽도 TCP 소켓으로 입출력 스트림을 생성한다.
- 10 클라이언트는 출력 스트림으로 패킷을 전송한다.
- 11. 서버는 입력 스트림으로 패킷을 처리하고 다시 출력 스트림으로 패킷으로 전송하게 된다.

Socket 클래스

- 자바에서 TCP Socket은 java.net.Socket 클래스를 의미한다.
- 소켓 생성자는 서버와 접속하기 위해서는 서버의 ip address와 port를 알아야 한다.
- Socket 클래스의 생성자는 9가지로 이루어 졌으며 1개는 deprecated 되어 있다.
- Socket 클래스에서 가장 많이 사용하는 생성자는 아래와 같다.

```
public Socket(String host, int port)throws UnknownHostException IOException
```

- 소켓 객체를 생성할 때 2가지 예외 처리가 발생하는 첫번째 서버를 찾을 수 없을 경우와 port를 사용하지 않을 경우에 UnKownHostException 예외가 발생할 수 있고, 두번째는 네트워크의 실패, 또는 방화벽 때문에 서버에 접근할 수 없을 때 IOException 예외가 발생할 수 있다.

Socket을 이용한 입출력 스트림 생성

- TCP 소켓은 두 개의 네트워크 사이에서 바이트 스트림 통신을 제공한다.
- Socket 클래스에는 바이트를 읽기 위한 메서드와 쓰기 위한 메서드를 제공하고 있다.
- 이 두가지 메서드를 이용해서 클라이언트와 서버 간에 통신을 할 수 있다.
- 아래의 내용은 Socket 클래스에서 스트림을 생성할 수 있는 메서드이다.

```
java.io.InputStream getInputStream() throws IOException;
java.io.OutputStream getOutputStream() throws IOException;
```

- 소켓 클래스와 스트림 메서드를 이용한 간단한 코드를 살펴보자.

```
try{
Socket socket = new Socket("211.238.132.50",4000);
InputStream in = socket.getInputStream();
OutputStream out = socket.getOutputStream();
}catch(UnkwonHostException ukhe){
        ukhe.printStackTrace();
}catch(IOException ioe){
        ioe.printStackTrace();
}
```

Socket 정보와 종료

- Socket 클래스에서는 로컬의 IP 주소와 포트를 알 수 있는 메서드와 Socket으로 연결되어 있는 원격 호스트의 IP 주소와 포트를 알 수 있는 메서드를 제공하고 있다.
- Socket 클래스에 있는 중요 메서들 살펴보자.

```
public InetAddress getInetAddress() throws IOException;
public int getPort() throws IOException;
public InetAddress getLocalAddress() throws IOException;
public int getLocalPort() throws IOException;
```

- 소켓은 시스템에 의해 자동으로 종료 되는 경우가 있다. 예를 들어 프로그램이 종료되거나, 가비지 컬렉터에 의해 처리되는 경우에 소켓이 자동으로 종료 된다.
- 소켓이 시스템에 의해 자동으로 닫히는 것은 바람직 하지 않다.
- 소켓의 사용이 끝나면 연결을 끊기 위해서는 소켓의 close() 를 호출 해야 한다.
- 이런 소켓을 닫는 코딩은 일반적으로 finally 블록에서 처리 한다.
- 소켓이 닫히더라도 getInetAddress(), getPort() 등의 메서드는 사용할 수 있으나, getInputStream(), getOutputStream()을 사용하게 되면 IOException을 발생하게 된다.

ServerSocket 클래스

- 클라이언트와의 TCP연결을 받기 위해서는 java.net.ServerSocket 클래스의 객체를 생성해야 한다.
- 서버 소켓은 네트워크 통신을 수행하기 위해 서버 소켓을 바로 사용하는 것이 아니라 클라이언트의 TCP 연결 요청에 대한 java.net.Socket 객체를 생성하는 역할을 한다.
- 서버 소켓 객체를 생성하여 accept() 메서드가 하는 역할은 클라이언트의 TCP 요청이 있을 때 까지 블러킹 되는 메서드이다.
- 클라이언트의 TCP 요청이 들어 왔다면 accept() 메서드는 Socket 객체를 리턴하게 된다.
- 그런 후에 다른 클라이언트의 TCP 요청을 기다리게 되므로 일반적으로 accept()메서드는 무한 루프 블록에 안에 존재 하게 되고, 리턴 되는 java.net.Socket 클랫의 객체를 이용하여 스레드를 만들게 된다.

ServerSocket 생성

- 서버 소켓 생성자는 바인딩에 사용할 TCP 포트 번호를 인자로 받는다.
- 만약 기존의 TCP 포트 번호가 바인딩되어 있다면 java.io.IOException를 발생하게 될 것이다.
- 일반적인 서버 소켓 생성자를 구체적으로 살펴 보면 다음과 같다.

```
public ServerSocket(int port) throws IOException, SecurityException;
```

- 만약 어떤 프로세스가 이미 특정한 포트로 바운드된 소켓을 생성한다면 IOException를 발생하게 되며, port번호는 잘 알려진 포트 번호는 제외를 하는 것이 바람직하다.

ServerSocket의 Socket 연결 받기

- 서버소켓의 주요 작업은 들어오는 연결 요청들을 수신하고 각 요청을 java.net.Socket 객체를 생성하는 것이다.
- 이런 역할을 수행하는 것이 ServerSocket 클래스의 accept() 메서드이다.
- 클라이언트에 들어오는 요청이 없다면 요청이 올 때까지 accept() 메서드는 블록화 되거나 타임아웃이 되면 종료한다.
- accept() 메서드의 형식은 아래와 같다.

```
Socket accept() throws IOException, SecurityException;
```

- accept() 메서드는 일반적으로 무한루프로 처리한다.
- 클라이언트의 TCP 요청이 오면 accept() 메서드를 통해 java.net.Socket 클래스의 객체를 생성한 후에 다른 클라이언트의 TCP 요청이 기다리게 되기 때문에 accept() 메서드를 무한루프로 처리 해야 한다.
- accept()메서드에 의해 리턴된 java.net.Socket클래스의 객체는 Thread로 처리하여 계속 유지 시킬수 있게 한다.

ServerSocket의 정보와 종료

- 서버 소켓 클래스에서는 클라이언트의 TCP Connection으로 생성된 Socket를 이용하여 클라이언트의 IP와 port를 알아 낼 수 있다.

```
Socket tcpSocket = serverS.accept();
System.out.println("클라이언트의 IP 주소 : "+
                    tcpSocket.getInetAddress().getHostName());
```

- 서버의 IP 주소를 알수 있는 메서드는 아래와 같다.

```
public InetAddress getInetAddress();
```

- 서버의 port를 알수 있는 메서드는 아래와 같다.

```
public int getLocalPort();
```

- java.net.Socket 클래스의 종료 처럼, 서버 소켓의 종료는 close()메서드로 간단하게 종료할 수 있다.