
K G 아 이 티 뱅 크
JAVA
J A V A

Socket
&
ServerSocket

Socket & ServerSocket

- **네트워크 인터페이스를 통한 통신을 중계해주는 클래스**
  - TCP를 이용하기 때문에 신뢰성이 높은 통신방식
  - Socket : 클라이언트 롤에서 생성되는 객체
  - ServerSocket : 서버 롤에서 생성되는 객체
- **각각은 네트워크 인터페이스를 통한 연결/중계만 담당함**
  - 이 과정에서 InetAddress와의 연동이 필요하게 됨

```
import java.net.Socket;
import java.net.ServerSocket;

// 같은 Socket끼리는 연결되지 않으니 주의
ServerSocket server = new ServerSocket(8080);
Socket client= new Socket("localhost",8080);

// ServerSocket에서 받아들여야 함
Socket c1 = server.accept();
```

Socket & ServerSocket

- **Client는 연결을 시도하며, 이는 Socket 객체를 통해 진행됨**
  - 서버측에서는 자주적으로 연결을 시도하지 않음
- **연결이 수립된 이후부터는 Stream 객체를 생성하여 사용함**
  1. getOutputStream / getInputStream + Buffer처리가 필수
  2. 종료시에는 스트림 및 소켓에 대한 close가 모두 필요함

```
// Client의 경우 생성과 동시에 연결을 시도하게 됨
Socket client = new Socket("localhost", 8080);

// 생성되었으면 연결된 곳과 통신을 위한 스트림을 준비함
// 이 때 되도록 Buffer가 존재하는 스트림이 좋음
InputStream is = client.getInputStream();
OutputStream os = client.getOutputStream();

// 이후부터는 용도에 맞게 파일 입출력을 진행함
os.write(100);
System.out.println(os.read());
```

Socket & ServerSocket

❖ **서버는 ServerSocket을 통해서 연결을 받을 수 있게 준비함**

➢ 서버측에서는 자주적으로 연결을 시도하지 않음

❖ **이를 accept 메서드를 통하여 연결요청을 수락하고 이용함**

① 이 때도 Socket 객체를 생성하여 준비하여 사용하게 됨

② 이 때 Socket은 **상대방과의 연결**을 의미하게 됨

```
// 서버는 연결을 받을 수 있는 준비를 먼저해야 함
ServerSocket ss = new ServerSocket(8080);

// client가 요청하면 accept를 통해 Socekt을 생성
Socket s = ss.accept();

// 이후부터는 Socket을 통하여 Client와 통신을 진행
InputStream is = client.getInputStream();
OutputStream os = client.getOutputStream();
// 이 때 대화 목적이라면 Writer/Reader가 추천됨
.........
```

DatagramPacket
&
DatagramSocket

DatagramPacket & DatagramSocket

- **네트워크 인터페이스를 통한 통신을 중계해주는 클래스**
  - UDP를 이용하기 때문에 빠르지만 신뢰성이 낮은 방식
  - 간단한 데이터를 주고받는 통신은 UDP가 성능이 좋음
  - 신뢰성/연결지속성/데이터보장이 필요하면 절대 안됨
- **Socket과 달리 Stream이 아닌 byte로 모든 것을 처리함**
  - 필요에 따라 바이트 변환을 수행해야 하는 경우도 생김

```
import java.net.DatagramSocket;
import java.net.DatagramPacket;

// 통신을 위해선 양측에서 Socket이 필요함
DatagramSocket ds = new DatagramSocket(800);

// 이 Socket에 Packet를 "조립"하여 통신함
DatagramPacket dp = new DatagramPacket(....);
ds.send(dp);
ds.receive(dp);
```

DatagramPacket & DatagramSocket

- **DatagramSocket의 생성자는 크게 2가지가 있으며 주의해야 함**
  1. DatagramSocket() : 포트번호가 random으로 지정됨
  2. DatagramSocket(800) : 송수신포트의 기본값이 800이 됨
- **Server 쪽에서는 포트번호를 지정해주지 않으면 통신이 불가능**
  - 어느 출입구에서 수신하는지를 알 수 없어 전부 무시됨

```
// 포트번호가 없으면 임의로 설정됨
DatagramSocket ds = new DatagramSocket();
// 패킷을 주소와 포트번호를 모르면 못받음
ds.receive(packet)

// 서버측에서는 포트번호를 설정해야 함
ds = new DatagramSockect(800);
// 어차피 주소나 IP는 알고 있음
ds.receive(packet)
// 이후 적절한 변환과정을 거쳐 이용이 가능함
String result = new String(packet.getData(), ....);
```

DatagramPacket & DatagramSocket

- **DatagramPacket을 조립할 때는 용도에 따라 변수명을 설정함**
  1. 받는 용도(receive)와 보내는 용도(send)에 따라 달라짐
  2. 생성자에 얼마만큼의 정보를 넣어주는가에 따라 용도가 다름
- **생략할 경우, 미리 설정된 정보가 사용되니 주의해야 함**
  - 포트번호는 반드시 설정해주는 편이 좋음

```
// 받는 용도일 경우 배열과 그 크기만 전달함
byte[] buffer = new byte[1024];
DatagramPacket rp = new DatagramPacket(buffer, buffer.length);
// 만들어낸 패킷은 설정된 buffe에 따라 저장한도가 결정됨
ds.receive(rp);

// 보낼 때는 목적지와 포트번호를 넘겨줘야 함
InetAddress ip = InetAddress.getByName("localhost");
byte[] buffer = "ABCDEFG!".getBytes();
DatagramPacket sp =
            new DatagramPacket(buffer, buffer.length, ip, 800);
ds.send(sp); // 패킷에 설정된 정보를 토대로 전달됨
```