o 각 code block 의 구조 및 기능설명

< 파일 전송 프로그램 >
서버 띄움 –클라이언트 띄움 – 접속되면 서버가 클라이언트에게 파일전송 –
클라이언트에 전송된 파일이 콘솔로 찍히고 다운로드됨

(1) Simple_FTP_Server_김영민_20162820.java

<Code Block (1)>

```
import java.util.Scanner;
import java.io.DataInputStream;
import java.io.FileOutputStream;

public class Simple_FTP_Server {

	public static void main(String[] args) {
		// TODO Auto-generated method stub
		DataInputStream in = null;
		FileOutputStream out = null;
		ServerSocket listener = null;
		Socket socket = null;
```

DataInputStream in = null; // DataOutputStream 을 이용하면 기본 데이터 타입 단위로 전송 가능하다.

FileOutputStream out = null; // 디스크에 파일을 기록하는 Stream 이다.

ServerSocket listener = null; // 클라이언트에서 접속시 소켓을 생성 하는 Listener

<Code Block (2)>

```
try {
        listener = new ServerSocket(9999);
        System.out.println("연결을 기다리고 있습니다....");
        socket = listener.accept();
        System.out.println("연결되었습니다.");

        in = new DataInputStream(socket.getInputStream());
```

서버 소켓을 생성하고 클라이언트로부터 연결 요청을 대기 하는 try 문

<Code Block(3)>

```
while(true){
        int fileSize = in.readInt();
        String filename = in.readUTF();

        System.out.println("전송 받을 파일의 크기 "+ fileSize + ", 파일 " + filename );

        File file = new File(filename);
        out = new FileOutputStream(file);

        byte[] buffer = new byte[1024];
```

while(true){ // 클라이언트에서 데이터를 전송하는 순서대로 받는 다. 파일 사이즈를 정수로 받고, 파일 이름을 문자열로 받는다.

int fileSize = in.readInt(); // 파일의 용량을 클라이언트로 부터 전송 받는다.

String filename = in.readUTF(); // Path 가 제외된 파일 이름만을 클라이언트로 부터 전송 받는다.

File file = new File(filename); // 파일 생성 out = new FileOutputStream(file);

// 파일을 저장할 Stream 객체 생성 byte[] buffer = new byte[1024]; // 최대 1024 바이트씩 끊어서 파일 데이터

를 전송 받는 다.

<Code Block(4)>

```
for(int received = 0;received < fileSize;)
{
        int len = in.read(buffer);
        out.write(buffer,0,len);
        received += len;
}

System.out.println("전송 완료된 파일의 크기 "+ fileSize);
out.flush();
out.close();
```

클라이언트에서 보내어온 파일 사이즈만큼 전송 받을 때 까지 데이터를 읽 고 파일로 기록하는 것을 반복 하는 과정.

Int len = in.read(buffer); // DataInputStream 을 이용해 클라이언트로 부터 데이터 수신

out.write(buffer,0,len); // FileOutputStream 을 이용해 디스크에 파일로 기록

<Code Block(5)>

```
}catch(IOException e) {
        System.out.println(e.getMessage());
}finally {
        try{
                in.close();
                socket.close();
                listener.close();
        }catch(IOException e) {
                System.out.println("클라이언트와 통신 중 오류가 발생했습니다.");
        }
```

오류를 잡고 server 오류시 통신오류가 발생했습니다 print 문 출력!

in.close(); // DataInputStream 해제 후 소켓을 해제 해야 한다.

(2) Simple_FTP_Client_김영민_20162820

<Code Block 2-(1)>

```
public static void main(String[] args) {
        // TODO Auto-generated method stub
        FileInputStream in = null;
        DataOutputStream out = null;
        Socket socket = null;

        Scanner scanner = new Scanner(System.in);

        try {
                socket = new Socket("localhost", 9999);
                out = new DataOutputStream(socket.getOutputStream());
```

FileInputStream in = null; // 내 디스크에서 서버로 전송할 파일을 읽을 때 사용

DataOutputStream out = null; // 서버로 파일 데이터를 전송 할 때 사용. DataOutputStream 을 이용하면 기본 데이터 타입 단위로 전송 가능하다.

Scanner scanner = new Scanner(System.in); // 콘솔 창에서 파일 이름 입력 받는 용도로 사용

out = new DataOutputStream(socket.getOutputStream()); // 기본 데이터 타입을 서버로 전송 할 수 있게 하는 스트림을 생성 try 문을통해 만들어준다.

<Code Block 2-(2)>

```
while (true) {
                String filename = scanner.next();
                File file = new File(filename);
                in = new FileInputStream(file);

                int fileSize = 0;
                byte [] buffer = new byte[1024];

                // 파일 사이즈를 측정
                while(true) {
                        int len = in.read(buffer);
                        if( len <= 0 )
                                break;

                        fileSize += len;
                }
                in.close();
```

반복해서 여러 파일을 하나씩 보낼 수 있게 반복문으로 동작 하고 또 안에 있는 while 문을 통해 파일사이즈를 계속 측정할 수 있게 만들어 주었다. 그 리고 마지막에 close 를 통해 파일 스트림을 닫고, 파일을 모두 읽어서 내부 포지션이 end 이므로, 아래에서 서버로 전송할 파일 스트림은 다시 생성
해서 쓴다.

String filename = scanner.next(); // 콘솔 창으로 파일 이름 입력 in = new FileInputStream(file); // 서버로 전송할 파일을 읽는 스트림 생성 byte [] buffer = new byte[1024]; // 서버로 1024 바이트씩 전송. 서버에서도

1024 바이트씩 전송 받도록 해둠

<Code Block 2-(3)>

```
String fileNameOnly = file.getName();
System.out.println("전송 할 파일 크기 " + fileSize + ", 파일명 " + fileNameOnly );

out.writeInt(fileSize);
out.writeUTF(fileNameOnly);

int sentSize = 0;
```

String fileNameOnly = file.getName(); // Path 를 제외한 파일 이름만 할당

파일 사이즈와 이름을 서버로 전송 하고, 서버에서는 파일 사이즈(정수)와 파일 이름(string)을 순서대로 읽으므로, 순서를 지켜서 보내게 만들어 주었다. out.writeInt(fileSize);

out.writeUTF(fileNameOnly); // Path 를 제외한 파일 이름만 서버로 전송 int sentSize = 0; // 전송 완료한 파일 사이즈 in = new FileInputStream(file); // 서버로 전송할 파일을 읽는 스트림을 다시 생성.

<Code Block 2-(4)>

```
in = new FileInputStream(file);
while(true) {
        int len = in.read(buffer);
        if( len <= 0 )
                break;

        out.write(buffer, 0, len);
        sentSize += len;

        if( sentSize >= fileSize )
                break;
}

out.flush();
in.close();
System.out.println("전송 완료 파일 크기 "+ fileSize);
```

읽어들인 데이터의 양(len) 만큼 buffer 로 할당 받은 파일 데이터를 서버로 전송하는 과정 uffer 의 배열은 1024 사이즈 이지만, 읽어들인 파일 데이터의 양(len)은 1024 이하 일 수 있고, 정확히 읽어들인 양(len) 만큼만 서버로 전송 한다.

int len = in.read(buffer); // 파일에서 데이터를 읽어 buffer 에 할당 하고, 읽은 byte 크기를 len 으로 반환 받는다. if(len <= 0) // 더 이상 파일로 부터 읽어들일 데이터가 없다.

If(sentSize >= fileSize) break; // 파일의 모든 데이터를 읽어서 서버로 전송 했다면, 반복문을 빠져 나간다

in.close(); // 파일 Stream 은 닫아준다.

<Code Block 2-(5)>

```
}catch(IOException e) {
        System.out.println(e.getMessage());

}finally {
        try {
                out.close();
                if(socket != null)
                        socket.close();

                scanner.close();
        }catch(IOException e) {
                System.out.println("서버와 통신 중 오류가 발생했습니다.");
        }
```

Server 와 마찬가지로 오류를 검출해내서 오류시 통신오류 발생 메시지 실행 아니면 out.close();을 통해 Socket 을 참조해서 생성했으므로 Socket 보 다 먼저 Close 한다.