#Summary

우선 본 프로그램은 제대로 동작하지 않는다.

P2P 방식의 스워밍 프로토콜을 구현한다. command line argument로 5개의 IP주소와 port 번호 쌍이 주어진다. 이 중 첫번째를 seeder로 하고, 나머지 4개가 leecher이다. seeder는 server로부터 파일을 다운로드 받아 leecher들에게 전달한다. 그리고 leecher들은 자신의 bitMap을 갱신하며 다른 peer와 TCP 연결을 맺어 한번에 3개의 청크까지 다운받은 후, 연결을 끊고 다른 피어와 연결을 맺는 것을 반복한다. 받은 파일 chunk들은 자신만의 새로운 파일에 저장한다.

#FFILE

```
public static final int BUFFER_SIZE = 10240;
public RandomAccessFile raf;
public ArrayList<Integer> bitMap;
```

FFILE 클래스는 file로부터 읽어오기, file에 쓰기, BitMap 확인, 반환 등의 역할을 한다.

먼저 chunk가 10KB 단위로 송수신되므로 `BUFFER_SIZE`를 10240로 선언한다. 각각의 peer가 파일 chunk를 원본 파일의 순서대로 송신 또는 수신한다는 보장이 없기 때문에, 랜덤하게 파일에 접근하기 위해 RandomAccessFile `raf`를 선언한다. 그리고 각 peer의 chunk 보유 여부를 관리할 ArrayList `bitMap`을 선언한다.

가 있다.

```
public FFile(String filename){
        ...
}
```

각각의 peer마다 서로 다른 속성이 존재하므로 생성자를 새로 정의하였다. 인자로 String형 `filename`을 입력받는데, `filename`은 peer가 chunk를 수신하여 그 내용을 write할 file을 말한다. `raf`를 생성할 때 인자로 이 `filename`을 넘겨주게 되며, read와 write의 기능을 모두 할 수 있게 한다. 그리고 `bitMap`을 새로 생성한다.

```
public synchronized ArrayList<Integer> getBitMap(){
        ...
}
```

getBitMap( )은 해당 peer의 bitMap을 반환한다.

```
public synchronized byte[] readChunk(int chunk_num){
        ...
}
```

readChunk( )는 인자로 받은 chunk_num이 해당 peer의 bitMap에 존재한다면 `BUFFER_SIZE`만큼 읽어들이는 역할을 한다. 먼저 `BUFFER_SIZE` 크기의 byte배열 `buff`를 생성하고, `raf`에서 포인터를 0에 위치시킨 다음 `chunk_num`*`BUFFER_SIZE`만큼 떨어진 곳의 위치로 포인터를 이동시켜 `BUFFER_SIZE`만큼 read해서 `buff`에 저장하고, `buff`를 return한다.

```
public synchronized void writeChunk(int chunk_num, byte[] buff)
        ...
}
```

writeChunk( )는 새로 받은 `chunk_num`이 해당 peer의 bitMap에 존재하는 지 확인하고, 존재한다면 return한다. 존재하지 않는다면 bitMap에 `chunk_num`을 add하고, 편의상 bitMap을 sort한다. 그리고 파일의 포인터를 0에 위치시킨 다음 `chunk_num`*`BUFFER_SIZE`만큼 떨어진 곳의 위치로 포인터를 이동시켜 인자로 받은 `buff`의 내용을 write한다.

```
public synchronized void seederChunk(){
        ...
}
```

checkChunk( )는 처음에 seeder가 전체 파일의 chunk들을 받아오고, 그것을 bitMap에 add한다. chunk 번호는 1부터 시작하므로 `cnt`를 1로 초기화하고, 내용이 없을 때까지 반복문을 돌며 작업을 수행하고 `cnt`를 1씩 증가시킨다.

#UPLOAD

```
public ServerSocket server;
public FFile file;
```

Upload class는 Thread를 extend한다. upload는 즉 server의 역할을 하므로 ServerSocket인 `server` 객체와 FFile 객체인 `file`을 선언한다.

```
public Upload(ServerSocket ser, FFile ff) {
        ...
}
```

각각의 upload 객체마다 서로 다른 속성이 존재하므로 생성자를 새로 정의하였다. 인자로 ServerSocket 객체와 FFile 객체를 받아, 위에서 선언한 변수에 각각 할당해준다.

```
public void run() {
        ...
        socket = server.accept();
                System.out.println("Upload: Connected to "+socket.getInetAddress()+"
,"+socket.getPort());

        ArrayList<Integer> bitMap = file.getBitMap();
```

Thread를 상속받았으므로 run( )을 정의한다.

Socket 객체로 `socket`을 생성하여 `server`가 accept한 결과를 넘겨주고, 연결 완료 되었다는 메시지를 출력한다. 그리고 해당 peer의 bitMap을 생성하여 `file`의 getBitMap( )을 호출해서 return값을 할당해준다.

```
        ...
dos.writeUTF("BitMap from Uploader");
for (int i = 0; i < bitMap.size();i++) {
    int num = bitMap.get(i);
    dos.write(num);
}
dos.write(0);
```

Uploader가 보유한 chunk의 목록들을 Download에게 보내기 전에, BitMap을 전송하겠다는 메시지를 Downloader에게 전송한다. 그리고 `bitMap`의 크기만큼 반복문을 돌며 가진 `bitMap`의 number(index)를 전송한다. 전송을 끝마친 후 편의를 위해, 전송이 다 끝났다는 의미로 0을 전송한다.

```
        ...
int cnt = 0;
while(true){
    int num = dis.readInt();
    if(num == 0) break;
    dos.write(file.readChunk(num));
    cnt++;
    System.out.println("Upload: Send chunk "+num);
}
if(cnt==0){
    System.out.println("Upload: No chunk to Upload");
}
dos.write(0);
```

반복문을 돌며 Downloader측에서 보낸 메시지를 읽어들인다. 그리고 요청받은 번호에 해당하는 chunk에 해당하는 byte들을 자신의 `file`에서 읽어와 메시지를 전송한다. 총 몇개의 chunk를 전송했는 지 기록하는 cnt를 매 반복마다 1씩 증가시키고, chunk 전송 완료 시 메시지를 출력한다.

반복문이 다 끝난 후, 요청받은 파일을 모두 보냈다는 의미로 0을 전송한다. 그리고 cnt가 0이라면, 즉 보낸 chunk가 없다면 메시지를 출력한다.

#DOWNLOAD

```
public static final int BUFFER_SIZE = 10240;
public FFile file;
public String ip;
int port;
int cnt=0;
```

Download class는 Thread를 extend한다. `BUFEER_SIZE`를 10240으로 하고, FFile인 `file`과 `ip`, `port`를 선언하고 cnt를 0으로 초기화한다.

```
public Download(String IP, int Port, FFile ff) {
        ...
}
```

각각의 download 객체마다 서로 다른 속성이 존재하므로 생성자를 새로 정의하였다. 인자로 IP, Port, ff를 받아 위에서 선언한 변수에 할당해준다.

```
public void run() {
        ...
Socket server = new Socket(ip, port);
System.out.println("Download: connected with"+ip+", "+port);
}
```

Thread를 상속받았으므로 run( )을 정의한다.

인자로 ip와 port를 받아 Socket 객체로 `server`을 생성한 후, 연결 완료 메시지를 출력한다.

```
        ...
ArrayList<Integer> bitMap= file.getBitMap();
ArrayList<Integer> serverBitMap = new ArrayList<Integer>();
```

해당 peer의 chunk 목록을 저장할 ArrayList `bitMap`을 선언하여 file의 `getBitMap`을 호출해 현재의 chunkList를 업데이트한다. 그리고 server의 bitMap에 존재하는 chunk에 대해 요청해야 하므로, server의 bitMap을 저장할 ArrayList `serverBitMap`을 생성한다.

```
        ...
String st = dis.readUTF();
if(st.equals("BitMap from Uploader")){
    while (true) {
        int num = dis.readInt();
        if (num == 0) break;
        serverBitMap.add(num);
    }
}
```

Upload로부터 문자열을 읽어들이는데, 그것이 bitMap 보유 상태 전송 시작 메시지라면 반복문을 돌며 Uploader측에서 보낸 메시지를 읽어들인다. 이 때 메시지는 int형 `num`으로, Uploader가 가지고 있는 chunk의 번호들이 되겠다. 따라서 위에서 생성한 serverBitMap에 `num`을 add한다. `num`이 0이라면, 이것은 uploader측에서 보낸 전송 완료 메시지로 반복문을 종료한다.

```
cnt=0;
for (int i = 0; i < serverBitMap.size();i++) {
    if (cnt==3) break;
    int num = serverBitMap.get(i);
    if (bitMap.contains(num)==false) {
        dos.writeInt(num);
        byte[] buffer = new byte[BUFFER_SIZE];
```

```
            dis.read(buffer);
            file.writeChunk(num, buffer);
            System.out.println("Download: download chunk_num " + num);
            cnt++;
        }
    }
    dos.writeInt(0);

    if(cnt==0){
        System.out.println("Download: No chunk Downloaded");
    }
```

`cnt`를 0으로 초기화해주고, `serverBitMap`의 크기만큼 반복문을 돌며 다음과 같은 과정을 수행한다.

한번의 연결 당 3개의 chunk까지 전송받기 때문에, `cnt`가 3이 되면 반복문을 종료한다. `serverBitMap`에서 차례로 읽어 `num`에 저장한 다음, 해당 peer의 `bitMap`에 그 `num`이 존재하지 않는다면 `num`에 해당하는 chunk를 uploader에게 요청하고, `BUFFER_SIZE`만큼 읽어와 writeChunk( )를 통해 `file`에 쓴다. 그리고 해당 chunk를 다운받았다는 메시지를 출력하고, `cnt`를 1 증가시킨다.

모든 반복이 끝난 후 끝났다는 의미의 0을 전송한다.

마지막으로, 만약 `cnt`가 0이라면 chunk를 download 받지 못했다는 의미이므로 메시지를 출력한다.

#Seeder

```
public static final int BUFFER_SIZE = 10240;
private static FFile file;
private static Upload upload;
```

`BUFFER_SIZE`를 10240으로 선언하고 FFile 객체로 `file`, Upload 객체로 `upload`를 선언한다.

```
BufferedReader br = new BufferedReader(new FileReader("configuration.txt"));
String thisCon = null;
int thisPort = 0;
String[] ip = new String[5];
int[] port = new int[5];
```

configuration.txt 파일을 읽기 위해 BufferedReader로 `br`를 생성해주고, `br`의 내용을 한줄씩 읽어 저장할 `thisCon`을 선언한다. 그리고 현재 port 번호를 저장할 `thisPort`와 configuration file에서 읽어올 ip와 port를 저장할 `ip`, `port` 배열을 선언한다.

```
        ....
for (int i = 0; i < 5; i++) {
    thisCon = br.readLine();
    if (i==0) {
        thisPort = Integer.parseInt(thisCon.split(" ")[1]);
        break;
    }
    ip[i] = thisCon.split(" ")[0];
    port[i] = Integer.parseInt(thisCon.split(" ")[1]);
}
```

반복문을 돌며 `br`에서 줄 단위로 읽어와 `thisCon`에 저장한다. configuration.txt의 첫번째 ip와 port를 seeder의 ip와 port로 지정할 것이기 때문에 i가 0일때, 즉 첫번째 반복일때 `thisPort`에 첫번째 port를 저장하고 반복문을 중단한다. 나머지 경우에는 `ip`, `port` 배열에 각각 ip와 port들을 저장한다.

```
        ...
String thisfile = "origin-file";

file = new FFile(thisfile);
byte[] buff = new byte[BUFFER_SIZE];

file.seederChunk();

ServerSocket server = new ServerSocket(thisPort);
upload = new Upload(server, file);

upload.start();
```

Seeder은 원본 파일을 받아서 저장하고 있기 때문에 `thisfile`을 "origin-file"이라고 선언해준다. 그리고 `thisfile`을 인자로 넘겨 새로운 FFile인 `file`을 생성한다. `BUFFER_SIZE` 크기의 `buff` 배열을 선언하고, seederChunk( )를 호출하여 Seeder의 bitMap을 업데이트한다. ServerSocket을 생성하며 인자로 `thisPort`를 넘겨주고, 새로운 Upload 객체를 만들며 `server`와 `file`을 인자로 넘겨준 후 upload에 대해 start( )를 호출한다.

---

*#Leecher*1,2,3,4

```
private static Upload upload;
private static Download download;
```

Upload 객체로 `upload`, Download 객체로 `download`를 선언한다.

```
BufferedReader br = new BufferedReader(new FileReader("configuration.txt"));
String thisCon = null;
String thisFile = "copy-1";
String[] ip = new String[5];
int[] port = new int[5];
int thisPort = 0;
```

configuration.txt 파일을 읽기 위해 BufferedReader로 `br`를 생성해주고, `br`의 내용을 한줄씩 읽어 저장할 `thisCon`을 선언한다. 1번 Leecher에서 chunk들을 받아 저장할 파일인 `thisFile`을 "copy-1"로 선언하고, 현재 port 번호를 저장할 `thisPort`와 configuration file에서 읽어올 ip와 port를 저장할 `ip`, `port` 배열을 선언한다. 이때 2, 3, 4번 Leecher들의 `thisFile`은 "copy-2,3,4"이다.

```
        ...
for(int i = 0; i < 5; i++){
    thisCon = br.readLine();
    if (i==1) {
        thisPort = Integer.parseInt(thisCon.split(" ")[1]);
    }
    ip[i] = thisCon.split(" ")[0];
    port[i] = Integer.parseInt(thisCon.split(" ")[1]);
}
```

반복문을 돌며 `br`에서 줄 단위로 읽어와 `thisCon`에 저장한다. configuration.txt의 첫번째 ip와 port를 seeder의 ip와 port로 지정할 것이기 때문에 i가 1일때, 즉 두번째 반복일때 `thisPort`에 두번째 port를 저장하고 반복문을 중단한다. seeder의 ip와 port를 제외하면, 두번째 줄부터 순서대로 leecher 1, 2, 3, 4의 ip와 port이기 때문이다. 따라서 Leecher2, 3, 4에서는 각각 if 조건 안에 i==2, 3, 4가 들어간다. 나머지 경우에는 `ip`, `port` 배열에 각각 ip와 port들을 저장한다.

```
        ...
ServerSocket server = new ServerSocket(thisPort);

Random rand = new Random();
int random = rand.nextInt(4)+1;

FFile file = new FFile(thisFile);

upload = new Upload(server, file);
download = new Download(ip[random], port[random], file);

upload.start();
download.start();
```

먼저 `thisPort`를 인자로 하여 새로운 ServerSocket인 server을 생성한다. 랜덤으로 peer을 선택하기 위해 Random변수인 `random`을 선언하여 1~5사이의 랜덤값을 저장한다. 그리고 `thisfile`을 인자로 넘겨 새로운 FFile인 `file`을 생성한다. 인자로 server와 `file`을 넘겨 Upload를 생성하고, `ip`와 `port` 배열의 `random` 번째 ip와 port, `file`을 인자로 하여 Download 객체를 생성하여 각각 start( )를 호출한다.

#how to Execute

2018008331_박민경 _assignment2.zip 파일을 다운받아 압축을 해제한다. 압축 파일 안에 있는 모든 파일과 다운로드 할 파일은 하나의 폴더에 위치시키도록 한다.

1. window에서 5개의 cmd창을 띄운다.
2. 다섯개 창 모두 cd [.java 파일과 전송 파일이 존재하는 절대경로] 를 입력한다.

ex) cd C:\Users\박민경\Documents\Eclipse-workspace\BitTorrent-like Swarming Protocol\src

3. 하나의 창에서는 javac Seeder.java, 나머지 창에서는 javac Leecher1.java, javac Leecher2.java, javac Leecher3.java javac Leecher4.java로 각각을 컴파일한다.
4. Seecher을 먼저 실행시킨 후 각각의 창에서 Leecher1 / Leecher2 / Leecher3 / Leecher4를 실행한다

   ex) java Seeder / java Leecher1 / ...