

# Computer Networks PA #1 디자인 및 구현 문서

2020059152 오주영

## 1. 프로그램의 설계 및 구조의 기술

UDP의 Multicast 기능을 이용한 P2P 방식의 오픈 채팅 프로그램에 대한 구현 문서이다. 한 Peer가 다른 다수의 Peer들에게 메시지를 전송하기 위해 java.net 패키지의 MulticastSocket 클래스를 활용하였다. MulticastSocket의 joinGroup 메서드로 특정 multicast address에 참여할 수 있고, 이 주소로 보낸 메시지는, 참여하는 모든 호스트에게 전달이 된다. 처음엔 Peer 클래스와 별도로 Chatroom 클래스를 만들어, Peer가 메시지를 Chatroom에게 보내면 Chatroom이 접속된 Peer 목록 ArrayList에 대한 반복문을 돌며 하나 하나 메시지를 전송하는 방식을 고려했다. 하지만 이 방식은, Always-On Server 없이도 Peer들이 서로 직접적인 소통을 할 수 있다는 UDP의 특징에 반한다고 느껴 결국 채택하지 않았다. 해당 프로그램은 내가 구현한 Peer 클래스 하나로 작동한다. 다음은 이 Peer 클래스와 그 메서드들에 대한 설명이다.

### 멤버 변수

```
private MulticastSocket mSocket;  
private InetAddress ip;  
private String username;  
private int portNum;  
private boolean inGroup;
```

mSocket은 위 기술한 바와 같이, 특정 multicast address에 참여하고, 메시지를 송수신함에 사용한다. ip는 Peer가 참여할 multicast address이다. username은 Peer가 채팅방 내에서 가지는 이름이다. portNum은 포트번호이며, 프로그램 실행 인자로 받아온다. inGroup은 Peer가 채팅방에 참여한 상태인지 여부를 나타내는 boolean 변수이다.

## 생성자

```
public Peer(int portNum) {
    inGroup = false;
    try {
        mSocket = new MulticastSocket(portNum);
        this.portNum = portNum;
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

portNum, 즉 포트번호를 인자로 받아 mSocket을 이 포트에 binding한다.

## hashAddress

```
public String hashAddress(String roomName) {
    try {
        MessageDigest mssgDigest = MessageDigest.getInstance("SHA-256");
        byte[] digest = mssgDigest.digest(roomName.getBytes());

        int byteLen = digest.length;
        int x = digest[byteLen - 3] & 0xff;
        int y = digest[byteLen - 2] & 0xff;
        int z = digest[byteLen - 1] & 0xff;

        return "225." + x + "." + y + "." + z;
    } catch (Exception e) {
        e.printStackTrace();
    }

    return null;
}
```

문자열인 채팅방 이름을 인자로 받아 SHA-256 해시를 이용해 multicast address로 변환하는 메서드이다. MessageDigest의 digest 메서드의 반환값은 byte 배열이고, 이를 그대로 주소로 사용하면 각 byte는 signed로 해석되기 때문에 우리가 원하는 0~255 범위가 나오지 않는다. 이 점을 해결하기 위해 각 byte를 0xff 와 and 연산을 해 int promotion을 강제하면 된다. 이렇게 구한 multicast address를 문자열로 반환한다.

## join

```

@SuppressWarnings("deprecation")
public void join(String roomName, String username) {
    try {
        InetAddress newIp = InetAddress.getByName(hashAddress(roomName));
        if(newIp.equals(ip)) {
            System.out.println("### ALREADY IN " + roomName);
            return;
        }
        if(inGroup) {
            leave();
        }
        ip = newIp;
        mSocket.joinGroup(ip);
        inGroup = true;
        this.username = username;
        System.out.println("### Successfully joined "+roomName+"("+ip+)");
        sendMessage("has joined the room", false);
    } catch (Exception e) {
        e.printStackTrace();
    }
}

```

채팅방에 참여할 때 사용하는 메서드이다. 우선 인자로 받은 채팅방 이름을 위 hashAddress 메서드를 이용해 multicast address로 변환한다. 이미 참여하고 있는 address와 일치한다면, 이미 이 방에 참여하고 있다는 콘솔 메시지를 띄우고 메서드를 종료한다. 이런 예외처리 없이는 SocketException이 발생할 수 있다. 또 만약 이미 어떤 채팅방에 참여하고 있는 상태라면 후술할 leave 메서드를 호출해, 방을 옮기기 전에 먼저 다른 Peer들에게 퇴장함을 알리도록 구현했다. MulticastSocket의 joinGroup 메서드는 두개가 오버로딩되어 있는데, 하나는 InetAddress를 인자로 사용하고 다른 하나는 SocketAddress와 NetworkInterface를 인자로 사용한다. 내가 사용한 joinGroup은 전자에 속하는데, 이 메서드는 java 14 이후로 deprecated된 상태이다. 하지만 해당 과제에서는 InetAddress를 사용하는 것이 더 적합하다고 느끼고, NetworkInterface 클래스에 대한 이해도가 부족해 이 메서드를 사용했다. 위 @SuppressWarnings("deprecation")은 이 메서드를 사용함으로 인한, 콘솔에 나타나는 warning 메시지를 가려주기 위함이다. 성공적으로 joinGroup을 마치면 인자로 받아온 사용자 이름으로 갱신해주고, 성공적으로 이 채팅방에 참여했음을 콘솔 메시지로 알린다. 또 후술할 sendMessage 메서드를 통해 다른 모든 Peer들에게 참여 소식을 메시지로 전달한다.

## leave

```

@SuppressWarnings("deprecation")
public void leave() {
    if(!inGroup) {
        System.out.println("### NOT IN GROUP");
        return;
    }
}

```

```

sendMessage("has left the room", false);
try {
    mSocket.leaveGroup(ip);
    inGroup = false;
    ip = null;
} catch (Exception e) {
    e.printStackTrace();
}
System.out.println("### You have left the room");
}

```

채팅방에서 나갈 때 사용하는 메서드이다. 만약 채팅방에 참여하고 있지 않은 상태라면, 이에 대한 콘솔 메시지를 띄우고 메서드를 종료한다. 채팅방을 나가기 전에, 후술할 sendMessage 메서드를 통해 다른 모든 Peer들에게 퇴장함을 알리는 메시지를 보낸다. MulticastSocket의 leaveGroup 메서드를 통해 해당 multicast address 그룹에서 떠난다. 이 메서드 또한 deprecated된 상태이다. ip를 null로 해준은, 특정 채팅방을 나간뒤 바로 다시 참여하기 위해 join 메서드를 호출할 때 이미 참여하고 있는 채팅방으로 처리되지 않도록 하기 위함이다.

## sendMessage

```

public void sendMessage(String message, boolean isMessage) {

    if(!inGroup) {
        System.out.println("### YOU ARE NOT IN A ROOM");
        return;
    }

    if(message.length() > 512) {
        sendMessage(message.substring(0, 512), isMessage);
        sendMessage(message.substring(512), isMessage);
        return;
    }

    if(isMessage)
        message = username + ": " + message;
    else // notification
        message = "### " + username + " " + message;
    byte[] buffer = message.getBytes();

    try {
        DatagramPacket sendPacket =
            new DatagramPacket(buffer, buffer.length, ip, portNum);
        mSocket.send(sendPacket);
    } catch (Exception e) {
        e.printStackTrace();
    }
}

```

이 프로그램을 구현함에 있어서 가장 많은 변화를 겪은 메서드이다. 우선, 채팅 메시지를 512 byte의 chunk 단위로 나누어서 전송하기 위해 재귀 방식을 채택했다. 즉 메시지가 512 byte 보다 클 경우 첫 512 byte는 그냥 전송하고, 나머지 부분에 대해 재귀적으로 이 방식을 반복한다. boolean 인자 isMessage는 보낼 메시지가 진짜 메시지인지 알림(notification)인지를 구별하기 위함이다. 메시지는 일반적인 채팅 메시지를 뜻하며, 사용자 이름과 “:”를 앞에 붙여 다른 Peer들에게 전달한다. 알림은 Peer가 채팅방에 참여할 때 혹은 나갈때 전달할 내용으로, 앞에 “####”와 사용자 이름을 붙여 전달한다. 이렇게 기존 메시지 내용에 추가 정보를 덧붙인 후 DatagramPacket을 생성하고 mSocket을 통해 Packet을 전달한다.

## receiveMessage

```
public void receiveMessage() {
    try {
        byte[] receive = new byte[512];

        DatagramPacket receivePacket =
            new DatagramPacket(receive, receive.length);
        mSocket.receive(receivePacket);

        String message = new String(receivePacket.getData()).trim();
        String temp = message;
        // notification
        if(temp.charAt(0) == '#') {
            temp = temp.substring(4);
        }

        // Only if different username UNSURE is this acceptable?
        if(!temp.split(":", 2)[0].equals(username)) {
            System.out.println(message);
        }

    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

메시지나 알림을 받아오고 콘솔에 띄우는 메서드이다. 코드가 나름 복잡해진 이유는, 어떤 Peer가 보낸 메시지를 자신의 콘솔에선 띄우지 않도록 하기 위함이다. 우선 mSocket을 통해 Packet을 받고, 문자열로 변환한다. 그리고 이 문자열에 대한 사본 temp를 만들고, 이 temp를 통해 자신과 관련된 내용인지를 체크한다. temp의 첫 글자가 '#'이라면, #으로 시작하는 메시지는 전달할 수 없으니까 알림인 것이고, 알림을 전송한 Peer의 사용자 이름은 이 문자열의 4번째 인덱스부터에 나타날 것이다(“####”후에). 이 경우 substring 메서드를 통해 사용자 이름이 제일 첫부분에 오도록 만든다. 이외 그냥 메시지인 경우 문자열의 첫부분에 나타날 것이다. 이렇게 Peer의 사용자 이름이 나타날 위치를 통일시켰으니 “:”이나 “ ” 기준으로 split을해, 첫 문자열을 살펴 자신의 사용자 이름과 일치하지 않는 경우에만 콘솔에 메시지를 띄운다.

## main

```
public static void main(String[] args) {

    Peer peer = new Peer(Integer.parseInt(args[0]));

    // Input & send
    new Thread(new Runnable() {
        @Override
        public void run() {

            Scanner scan = new Scanner(System.in);
            while(true) {

                String input = scan.nextLine();

                if(input.isBlank()) continue;

                if(input.charAt(0) == '#') {

                    String[] userArgs = input.split(" ");

                    if(userArgs[0].equals("#JOIN"))
                        peer.join(userArgs[1], userArgs[2]);

                    else if(userArgs[0].equals("#EXIT"))
                        peer.leave();

                    else
                        System.out.println("### INVALID COMMAND: "+userArgs[0].substring(1));
                }
                else {
                    peer.sendMessage(input, true);
                }
            }
        }
    }).start();

    // Receive
    new Thread(new Runnable() {
        @Override
        public void run() {

            while(true)
                peer.receiveMessage();

        }
    }).start();

}
```

우선 실행 인자로 받은 포트번호로 Peer 객체를 생성한다. 다음 사용자의 입력을 받아 메시지를 전송할 쓰레드, 즉 client process 역할의 쓰레드 하나와 메시지를 받을 쓰레드, 즉 server process 역할의 쓰레드를 하나 만들어 병렬적으로 실행시킨다.

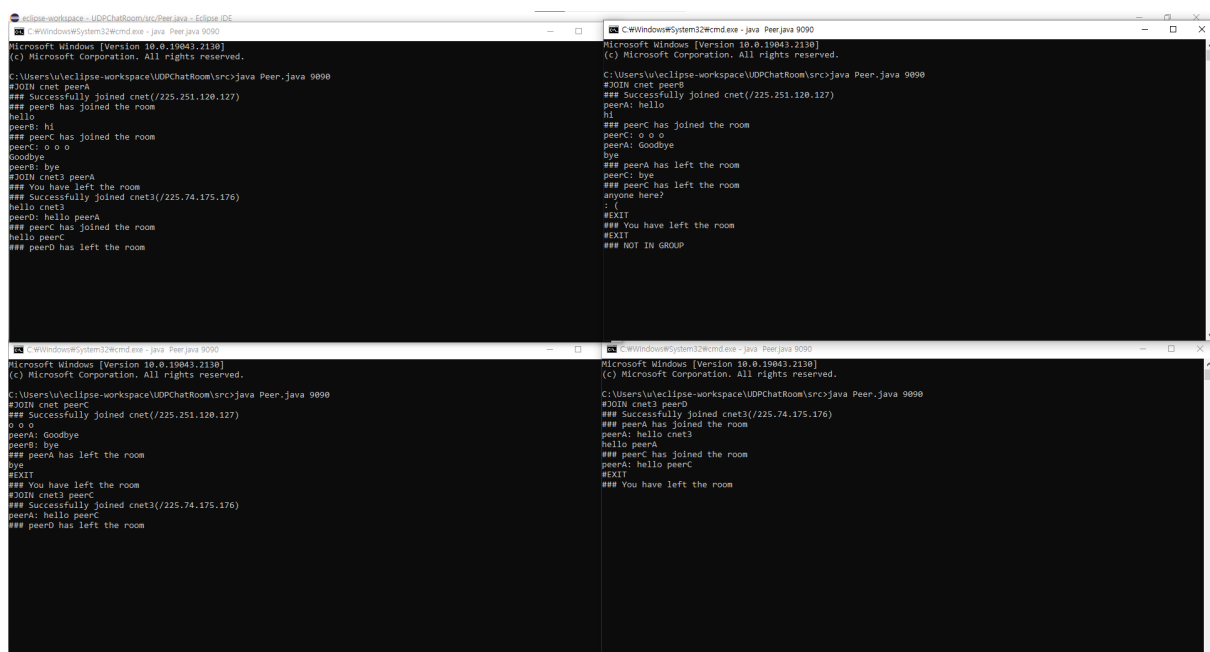
## 컴파일 및 실행방법

Peer.java가 있는 경로로 들어가 콘솔창을 실행합니다. 저는 **cmd**를 사용했습니다.

jdk 버전이 11 이상이면, 별도의 컴파일 없이 **java Peer.java 포트번호** 를 입력해 실행할 수 있습니다.

만약 11 미만의 jdk 버전이라면, 1. **javac Peer.java** 를 입력해 컴파일을 합니다. 2. **java Peer 포트번호** 를 입력해 실행합니다.

## 예시 실행 화면



```
Microsoft Windows [Version 10.0.19043.2138]
(c) Microsoft Corporation. All rights reserved.

C:\Users\uleclipse-workspace\UDPChatRoom\src>java Peer.java 9090

#JOIN cnet peerA
### Successfully joined cnet((225.251.120.127))
peerA: hello
hi
peerB: hi
### peerC has joined the room
peerC: o o o
Goodbye
peerD: bye
#JOIN cnet3 peerA
### You have left the room
### Successfully joined cnet3((225.74.175.176))
hello cnet3
peerD: hello peerA
### peerC has joined the room
hello peerC
### peerD has left the room

Microsoft Windows [Version 10.0.19043.2138]
(c) Microsoft Corporation. All rights reserved.

C:\Users\uleclipse-workspace\UDPChatRoom\src>java Peer.java 9090

#JOIN cnet peerB
### Successfully joined cnet((225.251.120.127))
peerA: hello
hi
peerC: o o o
peerA: goodbye
bye
### peerA has left the room
peerC: bye
### peerC has left the room
anyone here?
: {
#EXIT
### You have left the room
#EXIT
### NOT IN GROUP

Microsoft Windows [Version 10.0.19043.2138]
(c) Microsoft Corporation. All rights reserved.

C:\Users\uleclipse-workspace\UDPChatRoom\src>java Peer.java 9090

#JOIN cnet peerC
### Successfully joined cnet((225.251.120.127))
o o o
peerB: Goodbye
peerB: bye
### peerA has left the room
bye
#EXIT
### You have left the room
#JOIN cnet3 peerC
### Successfully joined cnet3((225.74.175.176))
peerA: hello peerC
### peerD has left the room
```