Programming Assingment #3

컴퓨터소프트웨어학부 2018062733 윤동빈

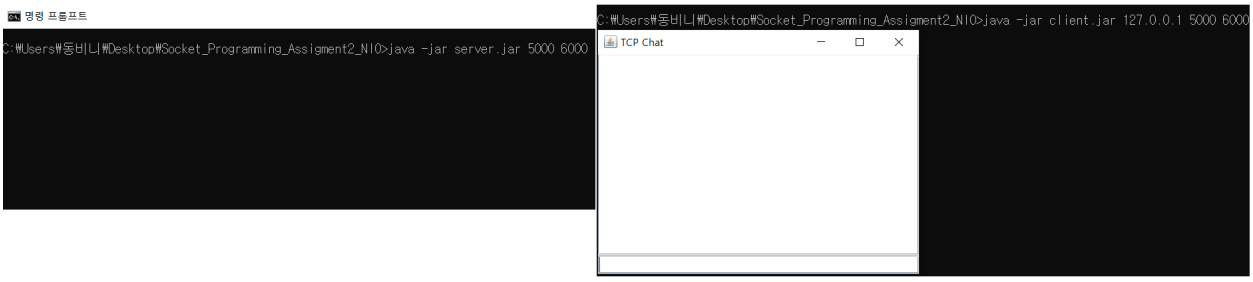
Notion에서 작성됐으며 PDF 파일로 추출하는 과정에서 코드 블럭 하이라이팅이 적용되지 않고 있습니다. 여기에서 하이라이팅이 적용된 내용을 확인하실 수 있습니다.

1. **실행 방법**

`server.jar` 또는 `client.jar` 파일이 있는 경로에서 각각 `java -jar server.jar {port1Number} {port2Number}`, `java -jar client.jar {ipAddress} {port1Number} {port2Number}` 를 입력한다.
(*server.jar*는 server 실행 파일을, *client.jar*는 client (채팅방) 실행 파일을 의미한다)

`client.jar` 실행 시에는 아래 그림과 같이 채팅방이 나타나며 `server.jar`는 별도의 알림이 뜨지 않는다.

여러 채팅방을 실행하려는 경우 명령 프롬프트를 원하는 사용자 수만큼 더 실행해서 진행한다.
아래는 *ipAddress*를 `127.0.0.1`로, *port1*, *port2 번호*를 각각 `5000`, `6000`으로 설정한 예시이다.



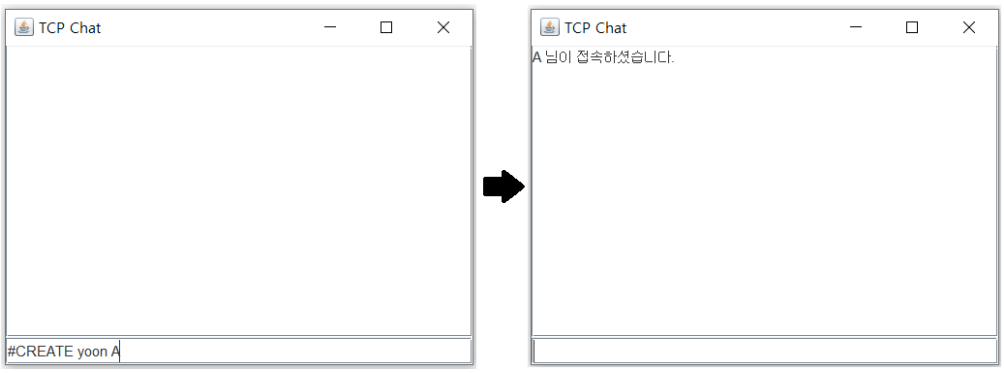
server.jar 실행 화면 *client.jar 실행 화면*

2. **명령어(CREATE, JOIN, PUT, GET, EXIT, STATUS) 및 채팅 예시**

채팅방 하단에 입력 칸을 통해서 명령어를 입력하거나 채팅을 할 수 있다.

2-1) **#CREATE**

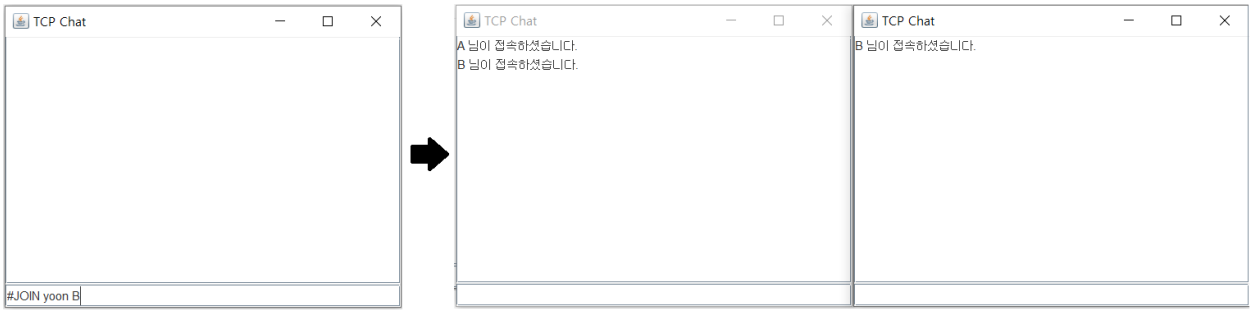
`#CREATE {생성할 채팅방 이름} {사용자 이름}` 을 입력해 채팅방을 생성할 수 있다.
접속 시 어떤 사용자가 접속했는지 알 수 있도록 채팅방에 메시지가 출력된다.



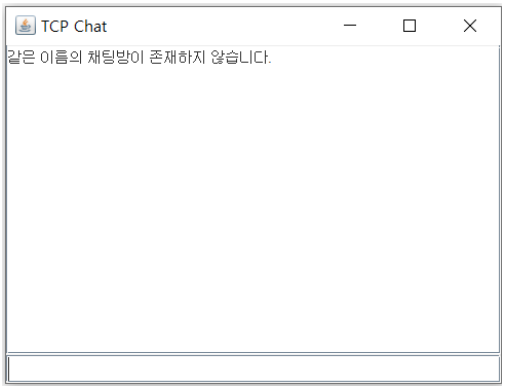
사용자 `A` 가 채팅방 `yoon` 을 생성

2-2) **#JOIN**

`#JOIN {참여할 채팅방 이름} {사용자 이름}` 을 입력해 채팅방에 접속할 수 있다.
접속 시 어떤 사용자가 접속했는지 알 수 있도록 채팅방에 메시지가 출력된다.



사용자 `A` , `B` 가 채팅방 `yoon` 에 접속한 상태

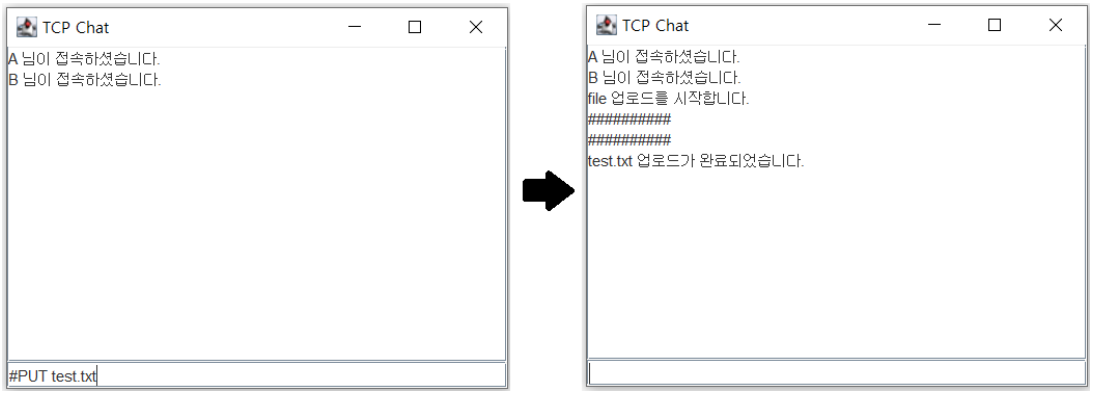


존재하지 않는 채팅방에 접속했을 때

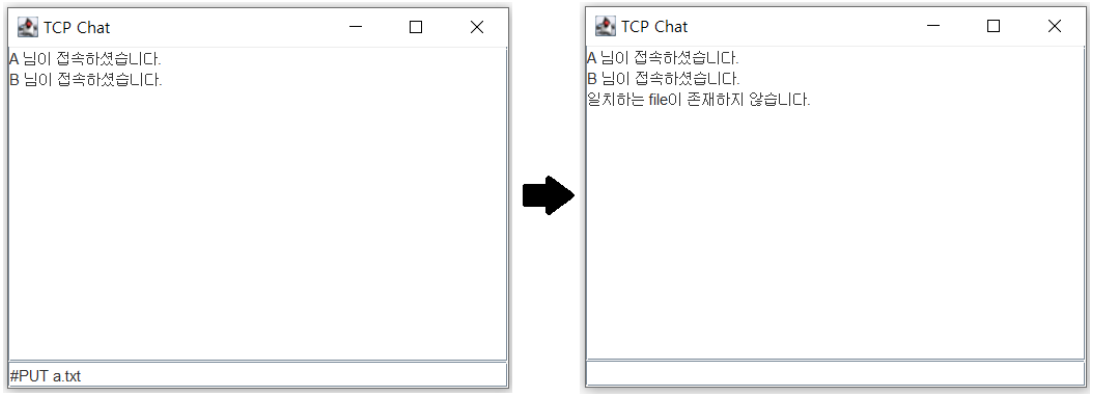
2-3) **#PUT**

`#PUT {fileName}` 을 입력하면 client는 server로 파일을 전송할 수 있다.
업로드 완료 시 메시지가 채팅방 화면에 출력된다.

업로드 된 파일은 `Socket_Programming_Assigment2/server/` 에서 확인할 수 있다.
(업로드 할 파일은 `Socket_Programming_Assigment2/client/upload` 에 있다고 가정)

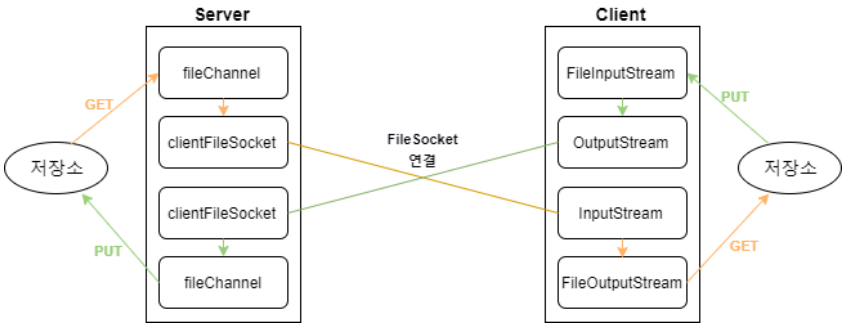


`test.txt` 파일을 client가 server로 업로드 (파일 크기: 640KB)



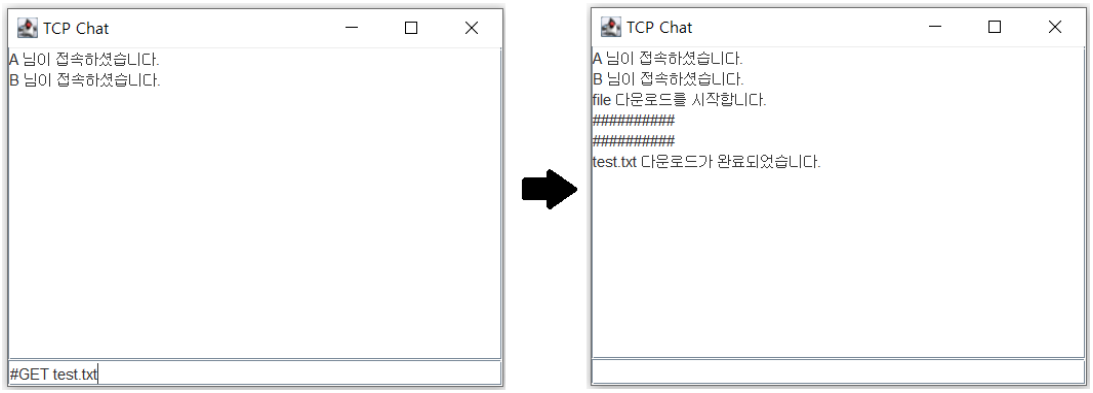
client가 입력한 file(`a.txt`)이 없는데 업로드를 시도한 경우

- **#PUT**, **#GET**에서 file 업로드/다운로드 매커니즘
  Server에서 clientFileSocket → fileChannel, fileChannel → clientFileSocket,
  Client에서 FileInputStream → OutputStream, InputStream → FileOutputStream
  으로 file 전송 시 64KB 단위로 처리한다.
  때문에 *#PUT*, *#GET* 명령어를 통해 전송하려는 file 크기가 640KB라면 `#`을 **총 20번** 출력한다.

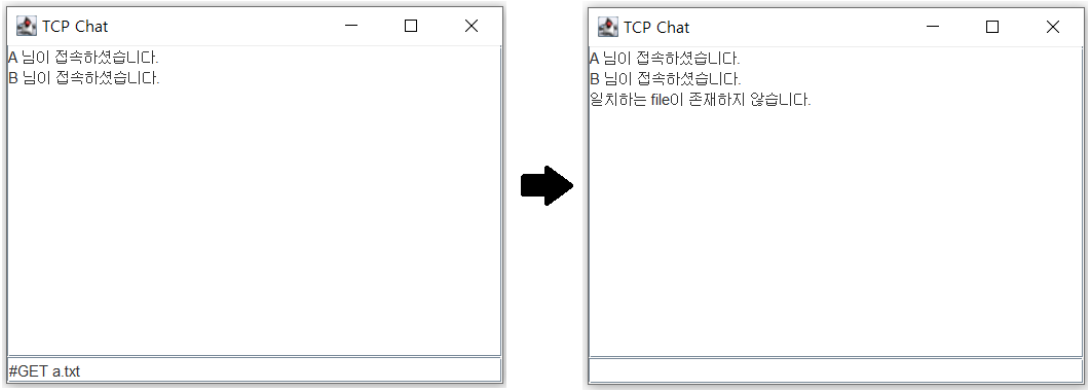


2-4) **#GET**

`#GET {fileName}` 을 입력하면 client는 server로 파일을 전송할 수 있다.
다운로드 완료 시 메시지가 채팅방 화면에 출력된다.
다운로드 된 파일은 `Socket_Programming_Assigment2/client/download/` 에서 확인할 수 있다.
(업로드 할 파일은 `Socket_Programming_Assigment2/client/upload`에 있다고 가정)



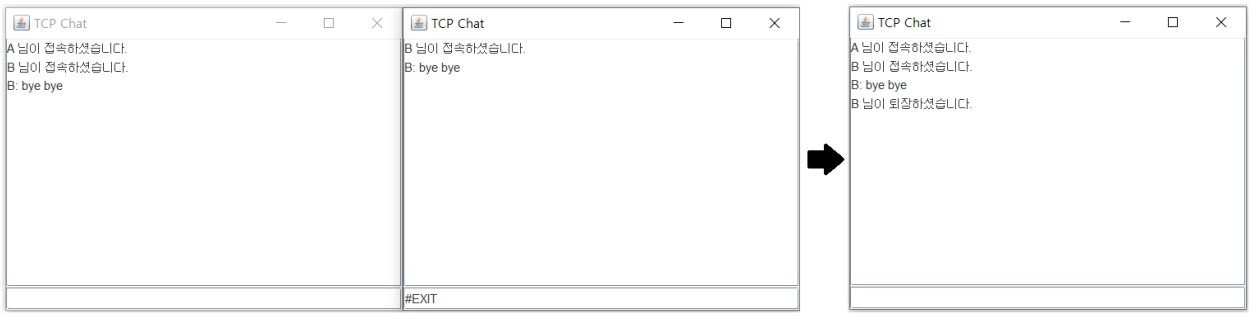
`test.txt` 파일을 client가 server에서 다운로드 (파일 크기: 640KB)



server에 입력한 file(`a.txt`)이 없는데 다운로드를 시도한 경우

2-5) **#EXIT**

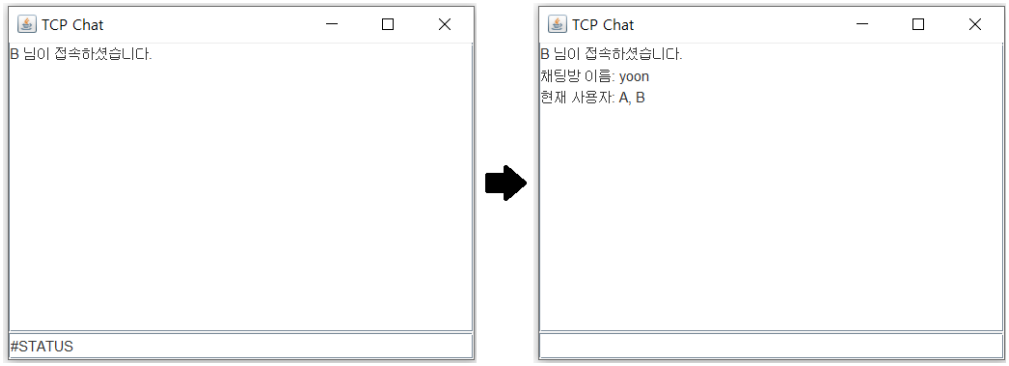
`#EXIT` 을 입력하면 채팅방이 종료되며, 어떤 사용자가 퇴장했는지 알 수 있도록 채팅방에 메시지가 출력된다.



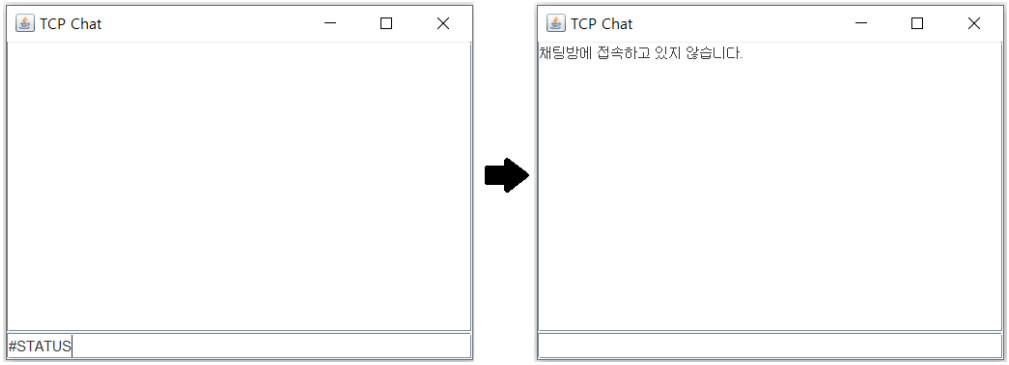
사용자 `B`가 명령어를 입력해 채팅방에서 퇴장한 경우

2-6) **#STATUS**

`#STATUS`를 입력하면 접속한 채팅방의 이름과 접속한 사용자의 이름을 확인할 수 있다.



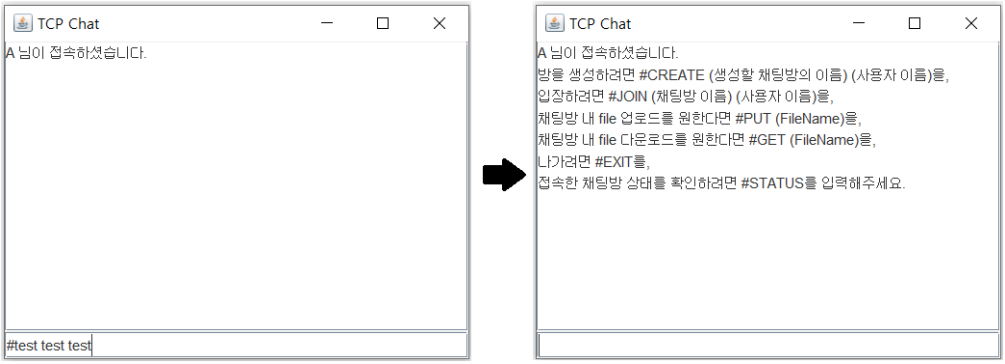
채팅방 이름이 `yoon`이고 접속한 사용자가 `A`, `B`가 있을 때



채팅방에 접속하지 않았는데 `#STATUS`를 입력했을 때

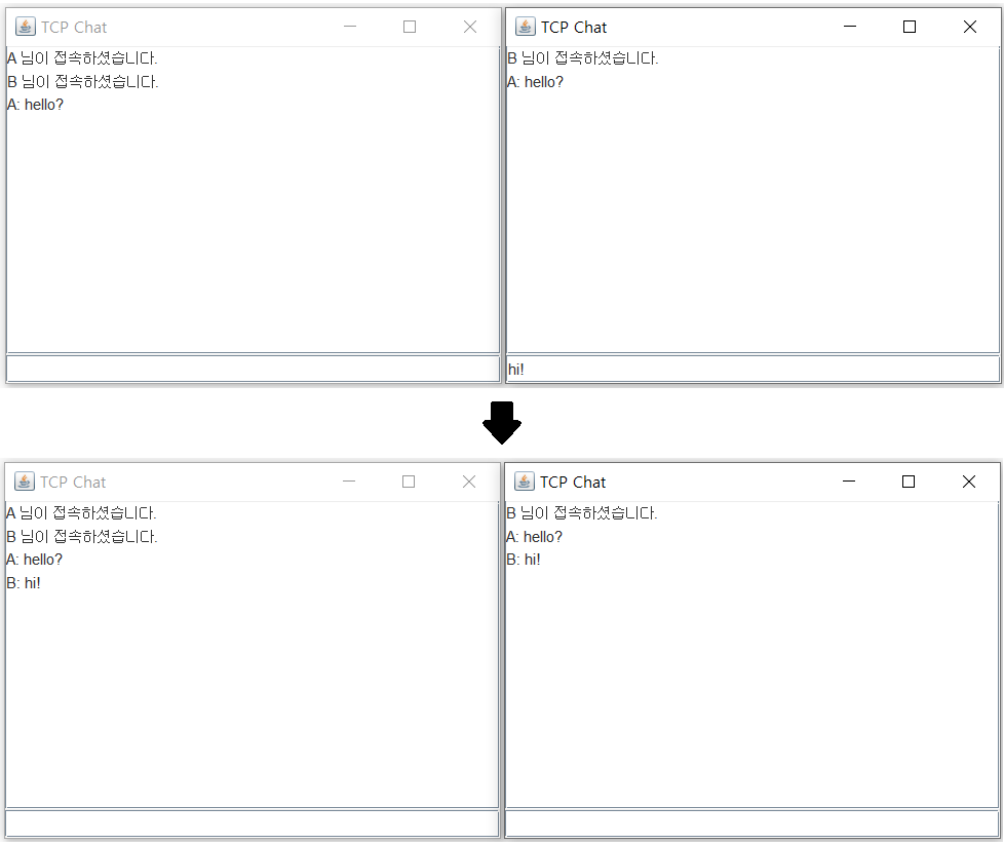
2-7) **잘못된 명령어 입력**

채팅 내 #으로 시작하는 문장은 명령어로만 사용되며 메시지를 전달하는 용도로 사용되지 않는다. 채팅방 접속 여부와 관계없이 #으로 시작하는 문장인데 잘못된 명령어를 입력하는 경우 무시하며 다음과 같은 메시지가 출력된다.

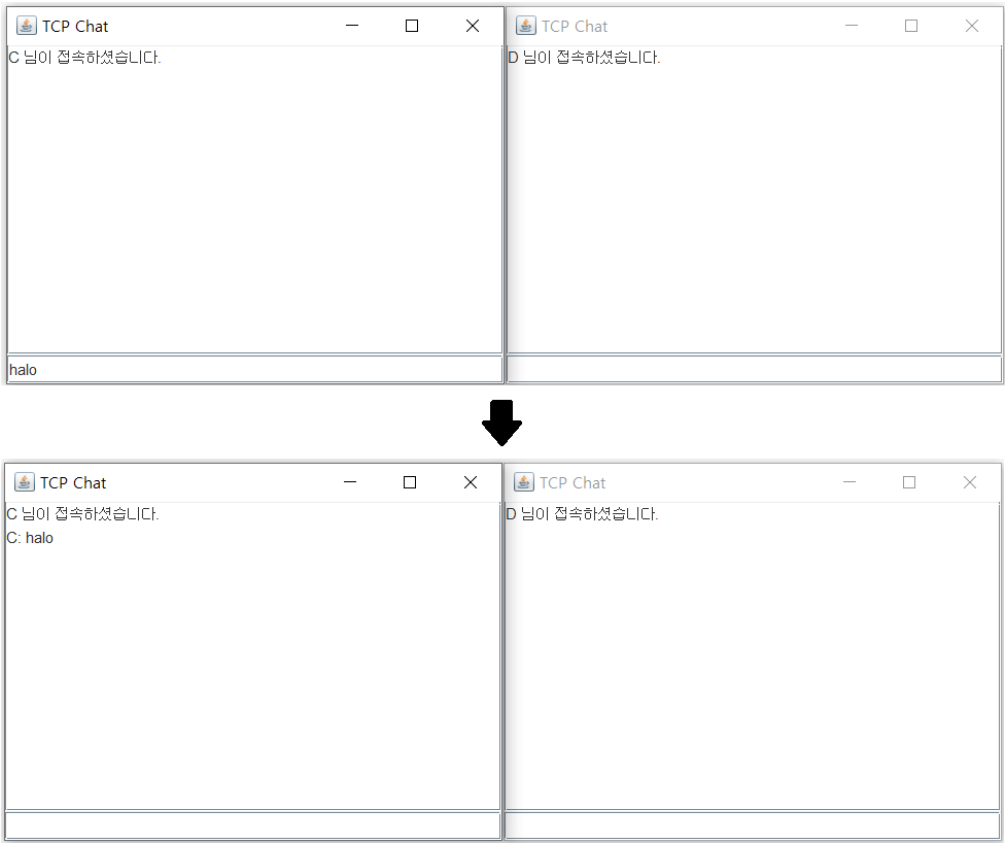


3. **채팅 예시**

사용자 A, B가 같은 채팅방에 있고 B가 입력했을 때 A와 B의 채팅방 화면에 동시에 메시지가 출력되는 모습이다.



사용자 C와 D는 서로 다른 채팅방에 입장해 서로의 접속 메시지를 확인하지 못하는 상황이며,
C가 입력한 메시지는 D가 있는 채팅방에서 확인할 수 없다.



4. **코드 설명**

파일은 `TCPServer.java`와 `TCPClient.java`로 나눠서 작성되었으며 각각 server, client로써 동작한다. `TCPServer.java`는 `TCPServer` class 내에 ClientService class를 가지며 이는 각 client에게 주어진 명령어에 적절한 서비스를 제공할 수 있도록 server에서 thread로 동작한다.

4-1) **TCPServer**

```
public class TCPServer {

    private ServerSocketChannel serverSocketChannel; //msg 송수신용 SocketChannel
    private ServerSocketChannel fileSocketChannel;   //file 송수신용 SocketChannel
    private Vector<ChatRoom> rooms;                  //server는 모든 채팅방을 관리
    private int port1;                               //채팅방 전용 port#
    private int port2;                               //file 송수신 전용 port#
    ...
```

`TCPServer` class 에서 사용되는 멤버 변수들이며 각 용도는 주석에 적힌 내용과 같다.

```
public TCPServer(int port1, int port2) {
    rooms = new Vector<>();
    this.port1 = port1;
    this.port2 = port2;

    try {

            //ServerSocketChannel 설정
            serverSocketChannel = ServerSocketChannel.open();
            serverSocketChannel.bind(new InetSocketAddress(port1));
            fileSocketChannel = ServerSocketChannel.open();
            fileSocketChannel.bind(new InetSocketAddress(port2));
            //non-blocking 모드로 설정
            serverSocketChannel.configureBlocking(false);
            fileSocketChannel.configureBlocking(false);

            //selector 생성 및 채널 등록
            Selector selector = Selector.open();
            serverSocketChannel.register(selector, SelectionKey.OP_ACCEPT);
            fileSocketChannel.register(selector, SelectionKey.OP_ACCEPT);
            ...
}
```

- `TCPServer`는 호출되면서 port 번호에 대한 인자 2개를 입력받고 멤버 변수를 초기화한다.

- msg, file 송수신에 쓰일 `serverSocketChannel`, `fileSocketChannel`를 각각 `port1`, `port2`로 바인딩 해준다. 이후 각 *ServerSocketChannel*로 들어오는 event를 받아서 독립적으로 수행하기 위해 non-blocking 모드로 설정해주고, `selector`에 등록해준다. 이 때 selector에서 해당 채널들을 accept하지 않았기 때문에 `SelectionKey.OP_ACCEPT`를 인자로 넣어서 이후에 accept event를 받을 수 있도록 설정해준다.

```
while(true) {

    //채널에서 이벤트 발생할 때까지 기다림
    selector.select();
    //해당되는 채널들 불러오기 (아래 while loop에서 연달아 처리)
    Iterator<SelectionKey> it = selector.selectedKeys().iterator();

    while(it.hasNext()) {
        //현재 처리할 이벤트 고르고 목록에서 삭제
        SelectionKey key = it.next();
        it.remove();

        if(key.isAcceptable()) { ... }
        else if(key.isReadable()) { ... }
    }
}
```

- `selector`는 앞에서 설정한 *ServerSocketChannel* 2개에서 발생하는 event를 기다린다. event 발생 시 해당되는 것들을 모아 `it`에 넣고 유형에 따라 순차적으로 처리한다.

- event 유형이 **accept**인지, **read**인지에 맞춰서 처리하도록 한다.
  (아래에서 설명 계속)

```
public ClientService(Socket clientSocket, Socket fileSocket) throws IOException {

    //필드 초기화
    socket = clientSocket;
    this.fileSocket = fileSocket;

    //serverSocket과 input/output 연결
    in = new BufferedReader(new InputStreamReader(socket.getInputStream()));
    out = new PrintWriter(socket.getOutputStream(), true);
}
```

- `ClientService`는 호출되면서 `TCPServer`로부터 소켓 2개를 입력받고 멤버 변수를 초기화한다.
  `serverSocket`과 정보를 주고받을 수 있도록 `in`, `out` 또한 초기화한다.

```
if(key.isAcceptable()) {
    //client 연결 요청

    //연결 요청한 client의 SocketChannel 생성
    ServerSocketChannel server = (ServerSocketChannel) key.channel();
    SocketChannel clientSocket = server.accept();

    SocketChannel serverSocket = null;
    SocketChannel fileSocket = null;

    if(server == serverSocketChannel)
        serverSocket = clientSocket;
    else if(server == fileSocketChannel)
        fileSocket = clientSocket;

    while(serverSocket == null || fileSocket == null) {
        selector.select();
        Iterator<SelectionKey> iter = selector.selectedKeys().iterator();
        if(!iter.hasNext()) continue;
        SelectionKey k = iter.next();
        if(!k.isAcceptable()) continue;
        iter.remove();

        ServerSocketChannel s = (ServerSocketChannel) k.channel();
        SocketChannel sc = s.accept();

        if(s == serverSocketChannel)
            serverSocket = sc;
        else if(s == fileSocketChannel)
            fileSocket = sc;
    }

    //non-blcoking 모드 설정
    serverSocket.configureBlocking(false);
    fileSocket.configureBlocking(false);

    //client class 초기화 및 selector에 읽기모드로 등록
```

```
    serverSocket.register(selector, SelectionKey.OP_READ, new Client(serverSocket, fileSocket));

}
```

- client로부터 accept event를 msg, file 송수신용 **총 2번**을 받는다. 그렇게 accept된 `serverSocket`, `fileSocket`을 non-blocking 모드로 설정해준 뒤 `Client` 생성자로 넣어줘 각 client에서 송수신에 활용할 수 있도록 해준다.

```
else if(key.isReadable()) {
    //읽기 요청한 client의 SocketChannel 및 정보
    SocketChannel clientSocket = (SocketChannel) key.channel();
    Client client = (Client) key.attachment();

    //client로부터 입력된 msg 확인
    ByteBuffer in = ByteBuffer.allocate(1024);
    try {
        clientSocket.read(in);
    } catch (Exception e) {
        //접속 종료 -> selector에서 삭제
        key.cancel();
        continue;
    }

    String clientMsg = new String(in.array()).trim();

    //의미없는 msg 무시
    if(clientMsg.equals("")) continue;

    if(clientMsg.charAt(0) == '#') {
        String[] msgs = clientMsg.split("\\s+"); //공백을 기준으로 파싱

        //예외 케이스 -> 전달 불가능
        if(msgs.length > 4) continue;

        ...
    } else {
        //채팅방에 입장하지 않은 경우
        if(client.getChatRoom() == null) {
            String msg = /* error 메시지 2-7) 참고*/;
            client.msgToMyChatRoom(msg);
            continue;
        }

        //채팅방에 속한 모든 clients에게 msg 전달
        String msg = client.getUsername() + ": " + clientMsg + "\n";
        client.msgToChatRoom(msg);
    }
    ...
}
```

- read event를 발생한 client 정보를 불러오고, 전송한 msg를 `clientMsg`로 받는다.
- 적절한 명령어 입력이 이루어지지 않은 경우의 예외처리

- 입력되는 내용에 따라 적절한 서비스가 제공될 수 있도록 한다. #으로 시작되는 내용은 명령어 입력을, 그렇지 않은 내용은 채팅을 위해 입력된 내용으로 간주하도록 처리한다.
- 아래에 이어지는 내용은 #으로 시작하는 각 명령어에 대한 설명이다.

```
if(msgs[0].equals("#CREATE")) {
    //#CREATE (생성할 채팅방의 이름) (사용자 이름)

    //형식에 맞지 않은 케이스 처리
    if(msgs.length != 3) {
        String msg = "'#CREATE (생성할 채팅방의 이름) (사용자 이름)'으로 입력해야 합니다.";
        client.msgToMyChatRoom(msg);
        continue;
    }

    String roomname = msgs[1];
    String username = msgs[2];

    //이름이 입력되지 않은 경우
    if(username.equals("")) {
        String msg = "이름은 0자 이상이어야 합니다." + "\n";
        client.msgToMyChatRoom(msg);
        continue;
    }

    //같은 이름의 채팅방이 이미 존재하는 경우
    if(findChatRoom(roomname) != null) {
        String msg = "같은 이름의 채팅방이 존재합니다." + "\n";
        client.msgToMyChatRoom(msg);
        continue;
    }

    //채팅방 생성 및 설정
    ChatRoom chatRoom = new ChatRoom(roomname);
    client.setChatRoom(chatRoom);
    rooms.add(chatRoom);
    String msg = "#STATUS" + "\n";
    client.msgToMyChatRoom(msg);

    //채팅방에 client 정보 등록
    chatRoom.clients.add(client);

    //채팅방에 속한 clients에게 접속 알림
    msg = username + " 님이 접속하셨습니다." + "\n";
    client.msgToChatRoom(msg);
}
```

- `ChatRoom` class 인스턴스를 만들어 채팅방 정보를 입력하며 client 또한 해당 채팅방에 속할 수 있도록 `add()`를 호출한다.
- `msgToChatRoom()`을 호출해 해당 채팅방에 참여한 사용자에게 접속 알림 메시지를 보낸다.

```
else if(msgs[0].equals("#JOIN")) {
    //#JOIN (채팅방 이름) (사용자 이름)

    //형식에 맞지 않은 케이스 처리
    if(msgs.length != 3) {
        String msg = "'#JOIN (채팅방 이름) (사용자 이름)'으로 입력해야 합니다." + "\n";
        client.msgToMyChatRoom(msg);
        continue;
    }

    String roomname = msgs[1];
    String username = msgs[2];

    //이름이 입력되지 않은 경우
    if(username.equals("")) {
        String msg = "이름은 0자 이상이어야 합니다." + "\n";
        client.msgToMyChatRoom(msg);
        continue;
    }

    //이미 채팅방에 들어가 있는 경우
    if(chatRoom != null)
        chatRoom.clients.remove(this);

    //이미 채팅방에 들어가 있는 경우
    if(client.getChatRoom() != null) {
        client.msgToChatRoom(client.getUsername() + " 님이 퇴장하셨습니다." + "\n");
        ChatRoom chatRoom = client.getChatRoom();
        chatRoom.clients.remove(client);
    }

    //사용자 이름 등록
    client.setUsername(username);

    //채팅방 설정
    ChatRoom chatRoom = findChatRoom(roomname);

    if(chatRoom == null) {
        String msg = "같은 이름의 채팅방이 존재하지 않습니다." + "\n";
        client.msgToMyChatRoom(msg);
        continue;
    }

    //사용자가 접속한 채팅방 설정
    client.setChatRoom(chatRoom);

    //채팅방에 client 정보 등록
    String msg = "#JOIN" + "\n";
    client.msgToMyChatRoom(msg);
    chatRoom.clients.add(client);

    //채팅방에 속한 clients에게 접속 알림
    msg = username + " 님이 접속하셨습니다." + "\n";
    client.msgToChatRoom(msg);
}
```

- 적절한 명령어 입력이 이루어지지 않은 경우의 예외처리
  만약 어떤 채팅방에 들어가 있었고 다른 채팅방으로 갈 경우 채팅방에서 해당 사용자 정보 제거

- 입력한 이름과 같은 채팅방을 찾기 위해 `findChatRoom()`을 호출하고 client가 해당 채팅방에 속할 수 있도록 `add()`를 호출한다.
- `msgToChatRoom()`을 호출해 해당 채팅방에 참여한 사용자에게 접속 알림 메시지를 보낸다.

```
else if(msgs[0].equals("#EXIT")) {
    //#EXIT

    if(client.getChatRoom() != null) {
        client.msgToChatRoom(client.getUsername() + " 님이 퇴장하셨습니다." + "\n");

        //채팅방에 client 정보 삭제
        ChatRoom chatRoom = client.getChatRoom();
        chatRoom.clients.remove(client);
    }

    if(client.getUsername() != null)
        System.out.println(client.getUsername() + "님이 접속을 종료했습니다.");

    String msg = "#EXIT" + "\n";
    client.msgToMyChatRoom(msg);
}
```

- 적절한 명령어 입력이 이루어지지 않은 경우의 예외처리
- 채팅방이 종료 처리되도록 하기위해 client에 `#EXIT` 메시지 전달 (`msgToMyChatRoom()`)

```
else if(msgs[0].equals("#PUT")) {
    //#PUT (FileName)
    //file 전송: client -> server

    String fileName = msgs[1];

    //client에게 file 전송 요청
    out.println(clientMsg);

    //client와 연결된 SocketChannel
    SocketChannel clientFileSocket = client.getFileSocket();

    //server에 저장하도록 하는 FileChannel
    Path path = Paths.get("./server/" + fileName);
    FileChannel fileChannel = FileChannel.open(path, StandardOpenOption.CREATE, StandardOpenOption.WRITE);

    ByteBuffer inputBuffer = ByteBuffer.allocate(65536);
    int n;
    boolean errorFlag = false;

    //전송: SocktChannel -> FileChannel
    while ((n = clientFileSocket.read(inputBuffer)) > 0) {
        String check = new String(inputBuffer.array()).trim();
        if(check.equals("@ERROR@")) {
            errorFlag = true;
```

```
            break;
        }

        inputBuffer.flip();
        fileChannel.write(inputBuffer);
        clientSocket.write(ByteBuffer.wrap("#".getBytes()));

        inputBuffer.clear();
    }

    fileChannel.close();

    if(errorFlag) {
        File file = new File("./server/" + fileName);
        file.delete();
    } else {
        String msg = "\n" + fileName + " 업로드가 완료되었습니다." + "\n";
        client.msgToMyChatRoom(msg);
    }
}
```

- client로부터 정보를 64Kbyte 단위로 `clientFileSocket.read()`를 호출해 읽어온 후 `fileChannel.write()`를 호출해 `path`에 쓰는 작업을 반복한다. (`inputBuffer`는 64Kbyte 단위로 읽기 위한 용도로 활용된다)
- file 정보를 64Kbyte 단위로 읽을 때마다 채팅방에 `#`을 1개 출력한다.
- `errorFlag`는 찾는 file이 존재하지 않는 경우에 대한 예외 처리를 위한 변수로써 활용된다.

```
else if(msgs[0].equals("#GET")) {
    //#GET (FileName)
    //file 전송: server -> client

    String fileName = msgs[1];

    //client로 file 정보 보낼 SocketChannel
    SocketChannel clientFileSocket = client.getFileSocket();

    try {
        //server에서 file 불러오는 FileChannel
        Path path = Paths.get("./server/" + fileName);
        FileChannel fileChannel = FileChannel.open(path, StandardOpenOption.READ);

        String msg = "file 다운로드를 시작합니다." + "\n";
        client.msgToMyChatRoom(msg);

        ByteBuffer inputBuffer = ByteBuffer.allocate(65536);
        int n;

        while ((n = fileChannel.read(inputBuffer)) > 0) {
            inputBuffer.flip();
            clientFileSocket.write(inputBuffer);
            clientSocket.write(ByteBuffer.wrap("#".getBytes()));

            inputBuffer.clear();
        }
```

```
            fileChannel.close();

        } catch (NoSuchFileException e) {
            e.printStackTrace();
            String msg = "일치하는 file이 존재하지 않습니다." + "\n";
            client.msgToMyChatRoom(msg);
            clientFileSocket.write(ByteBuffer.wrap("@ERROR@".getBytes()));
        }

        //전송된 file 정보를 client에서 처리하도록 메시지 전송
        client.msgToMyChatRoom("\n");
        client.msgToMyChatRoom(clientMsg + "\n");
    }
```

- server에 해당 파일 정보를 64Kbyte 단위로 `path`에서 `fileChannel.read()`를 호출해 읽어온 후 `clientFileSocket.write()`를 호출해 저장하는 작업을 반복한다. (`inputBuffer`는 64Kbyte 단위로 읽기 위한 용도로 활용된다)
- file 정보를 64Kbyte 단위로 읽을 때마다 채팅방에 `#`을 1개 출력한다.
- `msgToMyChatRoom()`을 호출해 client에서 전송한 file 정보를 다운로드하도록 신호를 보낸다.
- `errorFlag`는 찾는 file이 존재하지 않는 경우에 대한 예외 처리를 위한 변수로써 활용된다.

```
else if(msgs[0].equals("#STATUS")) {
    //#STATUS

    //형식에 맞지 않은 케이스 처리
    if(msgs.length != 1) {
        String msg = "'#STATUS'로 입력해야 합니다." + "\n";
        client.msgToMyChatRoom(msg);
        continue;
    }

    //채팅방에 접속해 있지 않은 경우
    if(client.getChatRoom() == null) {
        String msg = "채팅방에 접속하고 있지 않습니다." + "\n";
        client.msgToMyChatRoom(msg);
        continue;
    }

    String msg = client.getChatRoom().getClients();
    client.msgToMyChatRoom(msg);
}
```

- 적절한 명령어 입력이 이루어지지 않은 경우의 예외처리
- `getClients()`를 호출해 해당 채팅방의 제목, 접속한 사용자 정보를 읽어온 뒤 `msgToMyChatRoom()`를 호출해 client에게 해당 정보를 전송한다.

```
else {
    //예외 케이스 -> 전달 불가능
    String msg = /* error 메시지 2-7) 참고*/;
    client.msgToMyChatRoom(msg);
}
```

- #으로 시작하는 내용을 입력한 경우 위에 해당되는 케이스가 없다면 입력 내용은 무시한다.
- 대신에 어떤 내용이 입력돼야 하는지에 도움을 줄 수 있는 내용을 출력하도록 client에게 해당 정보를 전송한다.

```
//같은 채팅방에 있는 모든 client에게 전송
public void msgToChatRoom(String msg) throws IOException {
    for(ClientService client : chatRoom.clients) {
        client.out.println(msg);
    }
}
```

- `ClientService`에 속하는 메소드로, 같은 채팅방에 속한 모든 사용자에게 `msg`를 전달한다.
  각 사용자 `client`에게 `out.println()`을 통해 전달하도록 한다.

```
public class Client {

    private ChatRoom chatRoom;          //현재 속해있는 채팅방
    private String username;            //사용자 이름
    private SocketChannel serverSocket; //msg 전달용
    private SocketChannel fileSocket;   //file 전달용

    public Client(SocketChannel serverSocket, SocketChannel fileSocket) throws IOException {
        this.serverSocket = serverSocket;
        this.fileSocket = fileSocket;
    }

    /* getter, setter method */
    ...
```

- 각 client의 정보를 저장하는 class로, 생성자에서는 `serverSocket`, `fileSocket`를 인자로 받아 멤버 변수를 초기화 한다.

```
//같은 채팅방에 있는 모든 client에게 전송
public void msgToChatRoom(String msg) throws IOException {
    for(Client c : chatRoom.clients)
        c.msgToMyChatRoom(msg);
}

//내 채팅방에만 표시
```

```
public void msgToMyChatRoom(String msg) throws IOException {
    serverSocket.write(ByteBuffer.wrap(msg.getBytes()));
}
```

- `msgToChatRoom()`은 해당 채팅방에 속한 모든 client에게 `msg`를 전달할 수 있도록 `msgToMyChatRoom()`를 호출한다.
- `msgToMyChatRoom()`은 호출한 client의 채팅방에만 `msg`가 표시될 수 있도록 `serverSocket.write()`을 호출한다.

```
public class ChatRoom {

    private String title;
    Vector<ClientService> clients;

    public ChatRoom(String title) {
        this.title = title;
        clients = new Vector<>();
    }

    public String getClients() {

        String ret = "채팅방 이름: " + title + "\n현재 사용자: ";

        for(int i=0; i<clients.size(); i++) {
            ClientService client = clients.get(i);
            ret += client.getUsername();

            if(i < clients.size()-1)
                ret += ", ";
        }
        ret += "\n";

        return ret;
    }
}
```

- 채팅방 정보를 갖는 `ChatRoom` class로, 채팅방 이름 `title`과 접속한 사용자 정보 `clients`를 멤버 변수로 갖는다.
- `getClients()`를 통해 채팅방 이름과 접속 사용자 정보를 반환한다.

4-2) **TCPClient**

```
public class TCPClient extends JFrame implements ActionListener {

    private static final long serialVersionUID = 2018062733L;

    private JTextField textField;  //text 입력
    private JTextArea textArea;    //text 출력
    private BufferedReader in;     //ClientService로부터 정보 받는 입력스트림
    private PrintWriter out;       //ClientService로 정보 보내는 출력스트림
    private Socket serverSocket;
```

```
    private Socket fileSocket;
    private String ip;
    private int port1;
    private int port2;
    ...
```

`TCPClient` class 에서 사용되는 멤버 변수들이며 각 용도는 주석에 적힌 내용과 같다.

```
public TCPClient(String title, String ip, int port1, int port2) throws IOException {
    super(title);
    this.ip = ip;
    this.port1 = port1;
    this.port2 = port2;

    textField = new JTextField();
    textArea = new JTextArea();
    JScrollPane pane = new JScrollPane(textArea); // 스크롤바

    getContentPane().add(textField, BorderLayout.SOUTH); //텍스트 입력창 -> 하단
    getContentPane().add(pane, BorderLayout.CENTER);     //텍스트 출력창 -> 중앙

    textField.addActionListener(this);     //입력되는 텍스트 인식
    textArea.setFocusable(false);          //입력창에만 쓸 수 있도록 함

    setVisible(true);
    setBounds(300, 50, 400, 300);
    setDefaultCloseOperation(EXIT_ON_CLOSE);

    serverSocket = new Socket(ip, port1);
    fileSocket = new Socket(ip, port2);

    in = new BufferedReader(new InputStreamReader(serverSocket.getInputStream()));
    out = new PrintWriter(serverSocket.getOutputStream(), true);
    ...
```

- `TCPClient` class는 채팅방 프로그램 이름 `title`, 서버 IP주소 `ip`, port# 2개 `port1`, `port2`를 인자로 받으며 멤버 변수를 초기화한다.
- `serverSocket`과 `fileSocket`은 각각 `port1`, `port2`로 연결되도록 하고, `in`, `out`은 `serverSocket`과 연결된 입출력 스트림이다.
- 이외 내용은 채팅방 프로그램 설정과 관련된 내용이다.

```
while(true) {
    String msg = in.readLine();
    String[] msgs = msg.split("\\s+"); //공백을 기준으로 파싱

    ...
    else if(msg.equals("#EXIT")) {
        //채팅방 종료
        System.exit(0);
```

```
    } else if(msg.equals("#JOIN") || msg.equals("#STATUS")) {
        //채팅방 입장 시 내용 초기화
        textArea.setText("");
    } else {
        //채팅방에 해당 msg 표시
        textArea.append(msg + "\n");

        //채팅량이 많을 때 최신 내용을 먼저 볼 수 있도록 함
        textArea.setCaretPosition(textArea.getText().length());
    }
}
```

- client로 들어오는 msg는 `msg`로 받고, 입력된 내용은 유형에 맞게 처리하도록 한다.
- *#EXIT*가 입력된 경우 채팅방을 종료하고,
  #JOIN 또는 *#STATUS*가 입력된 경우 채팅방 화면을 초기화한다.
  (*#PUT*, *#GET*에 대한 내용은 아래에서 계속)
  이외에는 채팅방에 `msg`를 출력한다.

```
else if(msgs[0].equals("#GET")) {
    //#GET: client -> server file 다운로드

    String fileName = msgs[1];

    InputStream is = fileSocket.getInputStream();
    FileOutputStream fos = new FileOutputStream("./client/download/" + fileName);

    byte[] fileBuf = new byte[65536];
    int n;
    boolean errorFlag = false;

    while ((n = is.read(fileBuf)) != -1) {
        String check = new String(fileBuf).trim();
        if(check.equals("@ERROR@")) {
            errorFlag = true;
            break;
        }

        fos.write(fileBuf, 0, n);

        textArea.append("#");
        textArea.setCaretPosition(textArea.getText().length());

        int remainSize = is.available();
        if(remainSize == 0) break;
    }

    fos.flush();
    fos.close();

    if(errorFlag) {
        File file = new File("./client/download/" + fileName);
        file.delete();
    } else {
        //채팅방에 해당 msg 표시
        textArea.append("\n" + fileName + " 다운로드가 완료되었습니다." + "\n");
        //채팅량이 많을 때 최신 내용을 먼저 볼 수 있도록 함
```

```
        textArea.setCaretPosition(textArea.getText().length());
    }
}
```

- server로부터 파일 정보를 64Kbyte 단위로 얻기 위해 `is.read()`를 호출해 읽어온 후 `fos.write()`를 호출해 쓰는 작업을 반복한다. (64Kbyte 단위로 읽기위해 `fileBuf` 활용)
- file 정보를 64Kbyte 단위로 읽을 때마다 채팅방에 `#`을 1개 출력한다.
- `errorFlag`는 찾는 file이 존재하지 않는 경우에 대한 예외 처리를 위한 변수로써 활용된다.

```
public void actionPerformed(ActionEvent e) {
    String command = textField.getText();
    textField.setText("");

    //공백 무시
    if(command.equals("")) return;
    String[] commands = command.split("\\s+");

    if(commands[0].equals("#PUT")) {
        //#PUT: client -> server file 업로드

        String fileName = msgs[1];

        try {

            FileInputStream fis = new FileInputStream("./client/upload/" + fileName);
            OutputStream os = fileSocket.getOutputStream();

            textArea.append("file 업로드를 시작합니다." + "\n");
            textArea.setCaretPosition(textArea.getText().length());

            byte[] fileBuf = new byte[65536];
            int n;

            while ((n = fis.read(fileBuf)) > 0) {
                os.write(fileBuf, 0, n);

                textArea.append("#");
                textArea.setCaretPosition(textArea.getText().length());
            }

            textArea.append("\n");
            textArea.setCaretPosition(textArea.getText().length());

        } catch (FileNotFoundException fne) {
            fne.printStackTrace();

            textArea.append("일치하는 file이 존재하지 않습니다." + "\n");
            textArea.setCaretPosition(textArea.getText().length());

            OutputStream os;
            try {
                os = fileSocket.getOutputStream();
                os.write("@ERROR@".getBytes());
            } catch (IOException ioe) {
```

```
                ioe.printStackTrace();
            }

        } catch (IOException ioe) {
            ioe.printStackTrace();
        }
    }

    System.out.println("command: " + command);
    out.println(command);
}
```

- 채팅방에서 내용을 입력할 때마다 호출되는 함수이다.
- *[#PUT 관련 내용이 입력된 경우]*
  server로 파일 정보를 64Kbyte 단위로 전송하기 위해 `fis.read()`를 호출해 읽어온 후 `os.write()`를 호출해 쓰는 작업을 반복한다. (64Kbyte 단위로 읽기위해 `fileBuf` 활용)

  file 정보를 64Kbyte 단위로 읽을 때마다 채팅방에 `#`을 1개 출력한다.
  `errorFlag`는 찾는 file이 존재하지 않는 경우에 대한 예외 처리를 위한 변수로써 활용된다.
- server로 입력된 내용을 전송한다. (`out.println()`)