TCP Chatroom Assignment

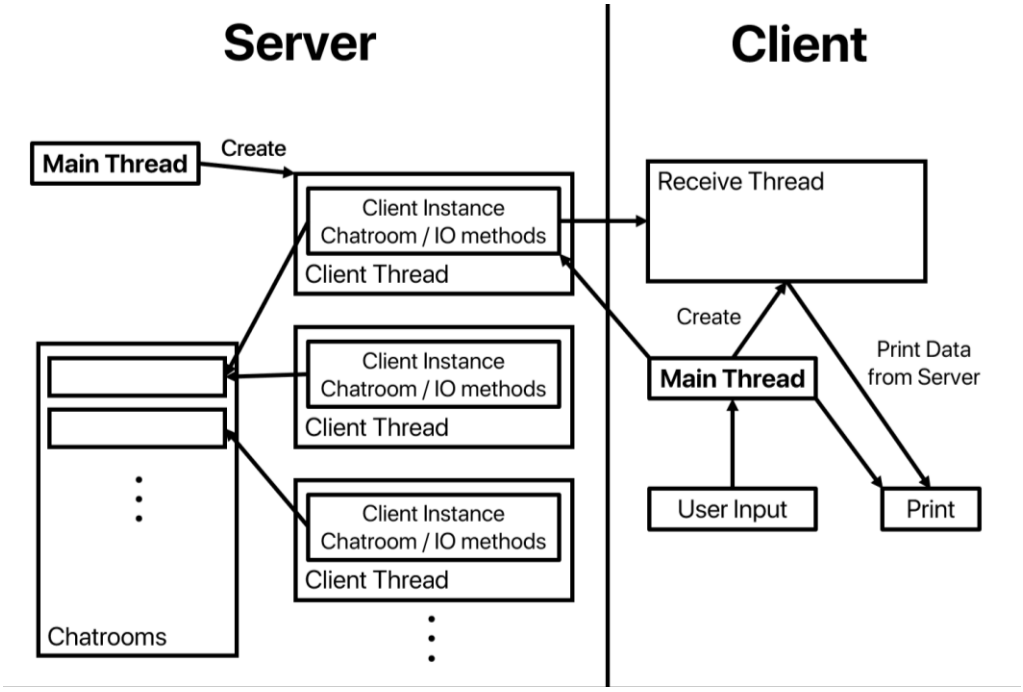
2021083681 장석규

Design _전반적인 Workflow

Server: 서버가 생성되면, 메인 스레드는 사용자의 입력을 기다린다. 사용자의 연결 요청이 들어오면 해당 사용자의 정보를 ClientInstance라는 인스턴스에 저장하고, 사용자가 처음으로 입력할 명령어(CREATE, JOIN, EXIT)을 처리한 후 해당 사용자의 입출력을 받을 스레드를 생성해 실행한다. 이후 작업은 모두 각 스레드가 담당하며, 메인 스레드는 새 새용자의 등록만 담당한다.

Client: 시작되면 스레드를 하나 생성해 이 스레드가 서버에서 데이터를 받는다. 메인 스레드는 사용자 입력을 받고 이에 따라 적절한 처리를 한다.

아래는 전반적인 프로그램의 구조와, 각 컴포넌트의 상호작용을 그림으로 핵심만 간략하게 나타낸 것이다.



Implementation _구체적인 구현

- **Server.java 파일의 클래스들**

Server class: 서버의 핵심 클래스

`main()`: 클라이언트가 채팅방을 만들 때까지 새로 온 클라이언트의 의사소통을 담당하며, 채팅방이 만들어지면 클라이언트와의 소통을 담당하는 객체를 스레드에 전달하고 다시 새 클라이언트의 입력을 받는다.

```
public class Server
{
    Run | Debug
    public static void main(String[] args) throws Exception
    {
        int port1 = Integer.parseInt(args[0]);
        int port2 = Integer.parseInt(args[1]);
        ServerSocket ss = new ServerSocket(port1);
        ServerSocket ss2 = new ServerSocket(port2);
        ArrayList<Chatroom> chatrooms = new ArrayList<>();

        while(true)
        {
            ClientInstance ci = new ClientInstance(ss);

            // Loop until gets either CREATE, JOIN or EXIT
            while(true)
            {…
        }
    }
}
```

Chatroom class: 챗방 이름과 클라이언트 목록 관리를 위한 클래스

```
class Chatroom
{
    String chatroomName;
    ArrayList<ClientInstance> clients = new ArrayList<>();

    public Chatroom(String chatroomName)
    {
        this.chatroomName = chatroomName;
    }
}
```

`ClientInstance class`: 클라이언트와의 소통을 담당하는 클래스.

각 클라이언트별로 소켓과 입출력을 담당할 I/O 도구들을 미리 준비해 놓고, receive/send 함수가 호출될 시 준비한 것들을 이용해 명령을 수행한다.

`receive / send / receive_file / send_file` 네 가지 함수가 이름대로의 역할을 수행해 직접적으로 입출력을 수행한다.

```
class ClientInstance
{
    private Socket socket;
    private BufferedReader br;
    private PrintStream ps;
    private DataInputStream dis;
    private DataOutputStream dos;
    public String clientName;
    public int chatroom_code;

    public ClientInstance(ServerSocket ss)
    {
        try
        {
            socket = ss.accept();
            br = new BufferedReader(new InputStreamReader(socket.getInputStream()));
            ps = new PrintStream(socket.getOutputStream());
        } catch (Exception e)
        {
            e.printStackTrace();
        }

    }

    public String recieve() throws Exception
    {…
    }

    public String send(String msg)…

    public void recieve_file()…

    public void send_file(String fileName)…
}
```

`ClientThread class`: 각 client별로 입출력을 수행할 thread를 구성하는 클래스

`run()`: while문을 돌면서 클라이언트에게 오는 입력을 받고 명령어일 경우 적당한 처리를 하고, 명령어가 아닌 경우 서버에 데이터를 보낸 사용자의 챗방을 분석해 같은 챗방의 모든 인원들에게 send를 통해 메시지를 전송한다. 이때 전송되는 것이 파일인지 텍스트인지를 구분하기 위해 MESSAGE나 FILE 중 하나를 먼저 전송하고 데이터를 전송하며, 메시지의 경우 어떤 클라이언트가 전송한 것인지에 대한 정보를 FROM 뒤에 붙여 보낸다.

```
class ClientThread extends Thread
{
    ClientInstance ci;
    Chatroom chatroom;

    public ClientThread(ServerSocket ss, Chatroom chatroom, ClientInstance ci)
    {
        this.ci = ci;
        this.chatroom = chatroom;
        //System.out.println("New Client Joined");
    }

    //Gather data and commands from client, with loop
    @Override
    public void run()
    {
        while(true)
        {""
    }
}
```

- **Client.java 파일의 클래스들**

Client class: 클라이언트의 핵심 클래스

main(): 사용자로부터 입력을 받아 정보를 저장하고 서버에 연결을 요청한다. 연결이 완료되면 아래의 ReceiveThread를 실행시켜 서버로부터 오는 입력을 받을 수 있도록 하고, 메인 스레드는 while문을 끊임없이 돌며 사용자 입력을 받아 명령어이면 명령을 수행하고, 텍스트면 해당 입력을 서버로 전송해 챗방의 다른 멤버들이 문자를 받을 수 있도록 한다.

```
public class Client
{
    Run | Debug
    public static void main(String[] args)
    {
        String ip = args[0];
        int port1 = Integer.parseInt(args[1]);
        int port2 = Integer.parseInt(args[2]);
        Socket socket = null;
        try
        {
            socket = new Socket(ip,port1);
        }
        catch(Exception e)
        {
            e.printStackTrace();
        }

        Scanner sc = new Scanner(System.in);
        Stop stop = new Stop();
        RecieveThread rt = new RecieveThread(socket, stop);
        rt.start();

        // Loop to get user input
        while(true)
        {…

        sc.close();
    }
}
```

`ReceiveThread class`: 서버로부터 데이터를 받는 작업을 수행하는 클래스, 텍스트와 파일을 모두 받을 수 있다.

`run()`: while 문을 돌며 서버로부터 오는 입력을 받는 함수. 이때 서버는 반드시 MESSAGE나 FILE이라는 데이터 종류를 먼저 전송하므로, 해당 메시지를 먼저 수신해 종류를 파악하고 메시지의 종류에 맞는 입력 스트림을 사용한다. (메시지는 BufferedReader, 파일은 DataInputStream)

```
class RecieveThread extends Thread
{
    Stop stop;
    Socket socket;
    BufferedReader br;

    public RecieveThread(Socket socket, Stop stop)
    {
        try
        {
            br = new BufferedReader(new InputStreamReader(socket.getInputStream()));
        }
        catch(Exception e) {}
        this.socket = socket;
        this.stop = stop;
    }

    @Override
    public void run()
    {
        //Loop to get data that server sends
        while(!stop.stop)
        {…
    }


}
```

How to Run

```
> & (java.exe path) .\Server.java port1 port2
```

위 명령어를 통해 서버를 실행하고 나서,

```
> & (java.exe path) .\Client.java IP port1 port2
```

클라이언트를 실행하고 이후 다른 명령 입력 가능

Server log

```
Client Joined: in chatroom room, name name1
room  room
Client Joined: chatroom room, name name2
Client Joined: in chatroom room2, name name1
room(name1): Hi Other User
Sent Message to name1: Success
Sent Message to name2: Success
room(name2): Hello
Sent Message to name1: Success
Sent Message to name2: Success
Receiving File from Client name1 of chatroom room: a.txt
############################
File saved in server: a.txt
Sending File from Client name2 of chatroom room: a.txt
############################
File Sent: a.txt to client name2
room2(name1): no ody is in this room
Sent Message to name1: Success
```

Client named name1 in room

```
PS C:\Projects\Networking> & 'C:\Program Files\Java\jdk-19\bin\java.exe' .\Client.java localhost  2000 2001
#CREATE room name1
joined room
Hi Other User
FROM name1: Hi Other User
FROM name2: Hello
#PUT a.txt
############################
Upload Complete
#STATUS
Chatroom Name: room, Members:name1, name2,
```

Client named name2 in room

```
PS C:\Projects\Networking> & 'C:\Program Files\Java\jdk-19\bin\java.exe' .\Client.java localhost  2000 2001
#JOIN room name2
joined room
FROM name1: Hi Other User
Hello
FROM name2: Hello
#GET a.txt
Getting File
############################
File Saved to local: a.txt
```

Client named name1 in room2

```
PS C:\Projects\Networking> & 'C:\Program Files\Java\jdk-19\bin\java.exe' .\Client.java localhost  2000 2001
#CREATE room2 name1
joined room2
nobody is in this room
FROM name1: nobody is in this room
```