▣ 서버 프로그램

◈ ChatServer 클래스
◈ ChatRunner 클래스
◈ ChatRoom 클래스

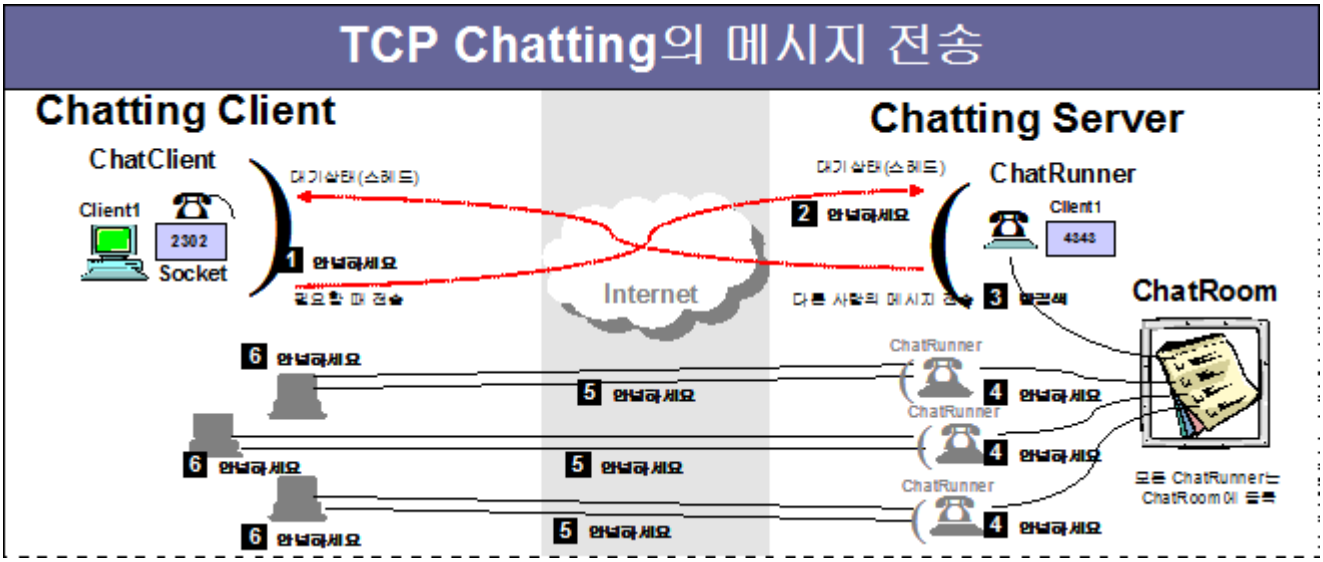
▣ 클라이언트 프로그램

◈ ChatClient 클래스
◈ ChatClientMain 클래스

▣ 서버

ChatServer 는 서버측에서 클라이언트의 접속을 받아 들이고, ChatRunner 라는 것을 생성한 후 ChatRoom 이라는 곳에 ChatRunner 의 참조값을 보관해 두는 역할을 합니다. 나머지는 ChatRunner 가 알아서 처리하게 됩니다. ChatRunner 는 스레드의 형식으로 연결설정을 유지하면서 클라이언트와 통신을 하게 됩니다.

클라이언트는 ChatClientMain 클래스에서 ChatClient 를 실행시키는 역할을 합니다. ChatClient 는 ChatServer 에 접속하게 되며, 접속이 되었다면 ChatClient 는 연결설정을 유지하면서 ChatRunner 와 통신하게 됩니다. 결국 ChatRunner 와 ChatClient 가 통신하는 것이 됩니다. ChatRunner 와 ChatClient 가 서로 통신하기 위해서는 내부에서 Socket 을 가지고 있어야 하며, Socket 에 연결된 입력과 출력 스트림이 개설된 상태이어야 합니다. 이러한 구조를 그림으로 나타내면 다음과 같습니다.

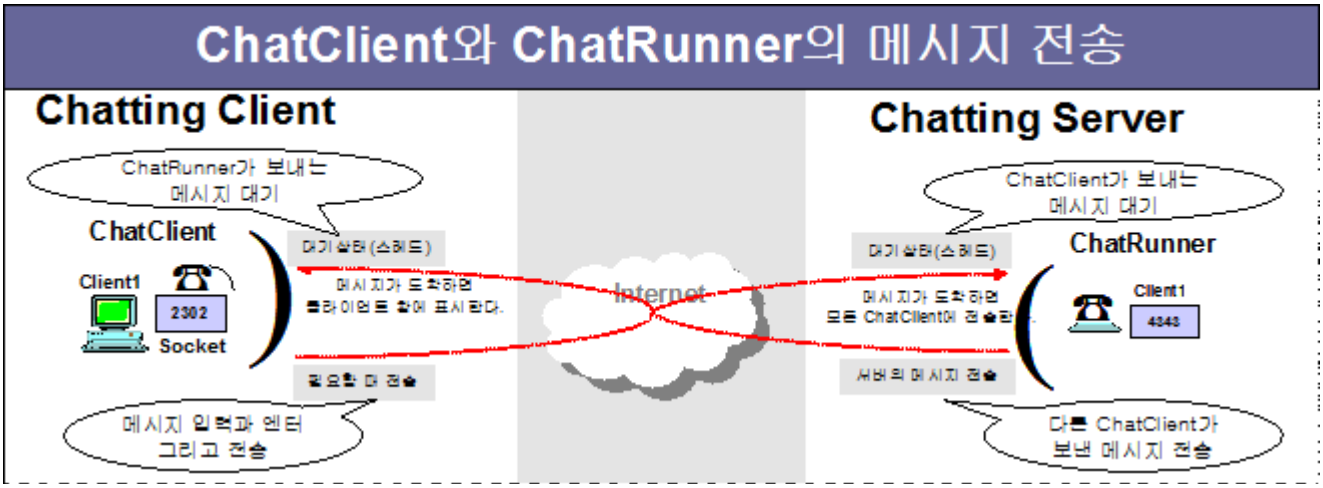


▣ 클라이언트

ChatClient 는 Socket 을 개설한 상태에서 입출력 스트림을 개설하게 되고, 입출력 스트림을 통해서 서버와 통신을 하게 됩니다. 이것은 ChatRunner 에게도 마찬가지입니다. ChatClient 가 출력 스트림을 통해서 메시지를

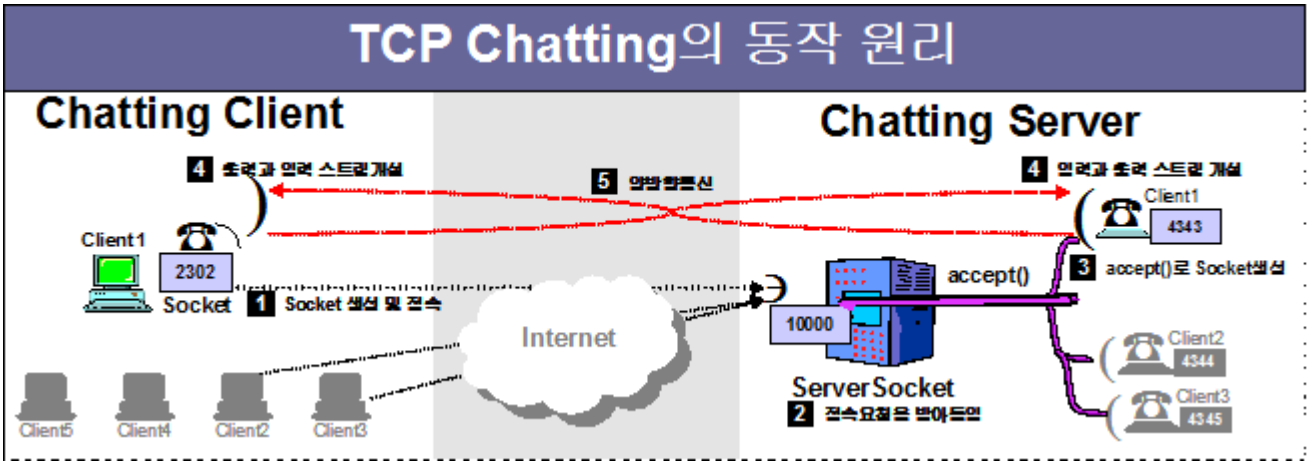
전송하면, ChatRunner 의 입력 스트림을 통해서 데이터가 들어오게 됩니다. 물론 ChatRunner 의 입력 스트림은 메시지가 들어오는 지를 감시하는 대기 상태에 있어야 합니다. ChatRunner 가 메시지를 받았다면 ChatRoom 에 들어있는 모든 ChatRunner 에게 메시지를 전달해서, 각각의 클라이언트에게 메시지를 전송하게 될 것입니다. 이러한 작업이 반복적으로 이루어지면서 채팅이 이루어지는 것입니다.

ChatClient 와 ChatRunner 의 입출력 스트림에서 하는 일을 좀 더 자세하게 알아보도록 하죠. 아래의 그림은 ChatClient 와 ChatRunner 의 입출력 스트림에서 하는 일을 도식으로 표현한 것입니다.



ChatClient 의 입력 스트림에서는 ChatRunner 가 보내는 메시지를 기다리게 됩니다. 지속적으로 데이터를 받아내기 위해서는 입력 스트림 자체는 스레드 안에서 작업이 이루어져야 합니다. 그리고 출력 스트림을 통해서 데이터를 ChatRunner 에게 전송할 것입니다. 텍스트 박스에 메시지를 입력하고 엔터를 누를 때 메시지가 전송되도록 이벤트 부분에서 작업을 하게 될 것입니다.

ChatRunner 의 입력 스트림 또한 ChatClient 가 보내는 메시지를 기다리게 됩니다. 지속적으로 데이터를 받아내기 위해서 입력 스트림 자체를 스레드의 형식으로 만드는 것은 ChatClient 와 동일합니다. 그리고 ChatRunner 의 출력 스트림에서는 ChatRoom 에 등록된 다른 ChatClient 가 전송하는 메시지가 있다면 그것을 전송하게 될 것입니다



1. Server

```
import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.IOException;
import java.io.OutputStream;
import java.net.ServerSocket;
import java.net.Socket;
import java.util.Collections;
import java.util.HashMap;
import java.util.Iterator;


public class ChatServer {
	private HashMap<String, OutputStream> clients;

	public static void main(String args[]) {
		// TODO Auto-generated method stub
		new ChatServer().start();
	}

	public ChatServer() {
		clients = new HashMap<String, OutputStream>();
		// Collections.synchronizedMap(clients)
	}

	public void start() {
		ServerSocket serverSocket = null;
		Socket socket = null;

		try {
			serverSocket = new ServerSocket(7777);
			System.out.println("서버가 시작되었습니다.");

			while(true) {
				socket = serverSocket.accept();
				System.out.println("[" + socket.getInetAddress() + " : " + socket.getPort() + "]" + "에서
접속하였습니다.");
				ServerReceiver thread = new ServerReceiver(socket);
				thread.start();
			}
```

```
        } catch(IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }

    public void sendToAll(String msg) {
        Iterator<String> it = clients.keySet().iterator();

        while(it.hasNext()) {
            try {
                DataOutputStream out = (DataOutputStream)clients.get(it.next());
                out.writeUTF(msg);
            } catch(IOException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
        }
    }

    class ServerReceiver extends Thread {
        private Socket socket;
        private DataInputStream in;
        private DataOutputStream out;

        public ServerReceiver(Socket socket) {
            this.socket = socket;
            try {
                in = new DataInputStream(socket.getInputStream());
                out = new DataOutputStream(socket.getOutputStream());
            } catch(IOException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
        }

        public void run() {
            String name = "";
            try {
                name = in.readUTF();
                sendToAll("#" + name + "님이 들어오셨습니다.");
```

```
                    clients.put(name, out);

                    System.out.println("현재 서버접속자 수는 " + clients.size() + "입니다.");

                    while(in != null) {
                        sendToAll(in.readUTF());
                    }
                } catch(IOException e) {
                    // TODO Auto-generated catch block
                    e.printStackTrace();
                } finally {
                    sendToAll( "#" + name + "님이 나가셨습니다.");
                    clients.remove( name);
                    System.out.println("[" + socket.getInetAddress() + " : " + socket.getPort() + "]" + " 에서
접속을 종료하였습니다.");
                    System.out.println("현재 서버접속자 수는 " + clients.size() + " 입니다.");
                }
            }
        }
}
```

2. Client

```
import java.io.BufferedReader;
import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.IOException;
import java.io.InputStreamReader;
import java.net.ConnectException;
import java.net.Socket;


public class ChatClient {

	public static void main(String args[]) {
		// TODO Auto-generated method stub
		if(args.length != 1) {
			System.out.println("USAGE: java ChatClient 대화명");
			System.exit(0);
		}

		try {
			Socket socket = new Socket("192.168.0.9", 7777);
			System.out.println("서버에 연결되었습니다.");

			Thread sender = new Thread(new ClientSender(socket, args[0]));
			Thread receiver = new Thread(new ClientReceiver(socket));

			sender.start();
			receiver.start();
		} catch(ConnectException e) {
			// TODO Auto-generated catch block
			e.printStackTrace();
		} catch(IOException e) {
			// TODO Auto-generated catch block
			e.printStackTrace();
		}
	}

	static class ClientSender extends Thread {
		private Socket socket;
		private DataOutputStream out;
```

```
        private String name;

        public ClientSender(Socket socket, String name) {
                this.socket = socket;
                try {
                        out = new DataOutputStream(socket.getOutputStream());
                        this.name = name;
                } catch(IOException e) {
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                }
        }

        public void run() {
                BufferedReader br = null;
                try {
                        br = new BufferedReader(new InputStreamReader(System.in));
                        if(out != null) {
                                out.writeUTF(name);
                        }

                        while(out != null) {
                                out.writeUTF( "[" + name + "]" + br.readLine());
                        }
                } catch(IOException e) {
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                } finally {
                        if (br != null) try { br.close(); } catch(IOException e) {}
                }
        }
}

static class ClientReceiver extends Thread {
        private Socket socket;
        private DataInputStream in;

        public ClientReceiver(Socket socket) {
                this.socket = socket ;
                try {
                        in = new DataInputStream(socket.getInputStream());
```

```
            } catch(IOException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
        }

        public void run() {
            while(in != null) {
                try {
                    System. out.println(in .readUTF());
                } catch(IOException e) {
                    // TODO Auto-generated catch block
                    e.printStackTrace();
                }
            }
        }
    }
}
```