**Term Project**

**JAVA**

-Swing을 이용한 Messenger 프로그램 작성-

이름 : 송승기

학번 : 201210578

교수님 : 정목동 교수님

제출 일자 : 18.06.25

|  |
| --- |
| 1-1) 채팅방 로그인 화면 |
| **class** Id **extends** JFrame **implements** ActionListener{  **static** JTextField *tf*=**new** JTextField(8);  JButton btn = **new** JButton("입력");    WriteThread wt;  ClientFrame cf;  **public** Id(){}  **public** Id(WriteThread wt, ClientFrame cf) {  **super**("로그인 화면");  **this**.wt = wt;  **this**.cf = cf;      setLayout(**new** FlowLayout());  add(**new** JLabel("이름"));  add(*tf*);  add(btn);    btn.addActionListener(**this**);    setBounds(300, 300, 450, 100);  setVisible(**true**);  }    **public** **void** actionPerformed(ActionEvent e) {  wt.sendMsg();  cf.isFirst = **false**;  cf.setVisible(**true**);  **this**.dispose();  } |
| swing 을 이용하여 로그인 화면을 구성하였다. textField로 아이디를 입력할 수 있도록 하였다. 그리고 layout은 FlowLayout으로 배치할 수 있도록 하였다. 그리고 버튼에 ActionListener 기능을 줘서 클릭하면 다음 페이지로 이동하도록 구성하였다. 그리고 dispose()로 전체 종료(exit)가 아니라 프레임만 종료되도록 하였다. |

1. 구성

|  |
| --- |
| 1-2) 채팅방 화면 구성(본화면)\_ClientFrame.java |
| **public** **class** ClientFrame **extends** JFrame **implements** ActionListener{  JTextArea txtA = **new** JTextArea();  JTextField txtF = **new** JTextField(15);  JButton btnTransfer = **new** JButton("전송");  JButton btnExit = **new** JButton("닫기");  **boolean** isFirst=**true**;  JPanel p1 = **new** JPanel();  Socket socket;  WriteThread wt;    **public** ClientFrame(Socket socket) {  **super**("SONGS'CHAT");  **this**.socket = socket;    Font f1 = **new** Font("맑은 고딕",Font.***BOLD***, 18);  txtA.setForeground(Color.***WHITE***);  wt = **new** WriteThread(**this**);  **new** Id(wt, **this**);  txtA.setBackground(Color.***green***);  add("Center", txtA);    p1.add(txtF);  p1.add(btnTransfer);  p1.add(btnExit);  add("South", p1);    btnTransfer.addActionListener(**this**);  btnExit.addActionListener(**this**);  setDefaultCloseOperation(***EXIT\_ON\_CLOSE***);  setBounds(300, 300, 350, 300);  setVisible(**false**);  }    **public** **void** actionPerformed(ActionEvent e){  String id = Id.*getId*();  **if**(e.getSource()==btnTransfer){    **if**(txtF.getText().equals("")){  **return**;  }  txtA.append("["+id+"] "+ txtF.getText()+"\n");  wt.sendMsg();  txtF.setText("");  }**else**{  **this**.dispose();  }  }  } |
| SONG’SCHAT 채팅프로그램 화면을 Swing으로 구성한다. TextArea로 대화창을 구성한다. 대화창의 배경 색깔을 Green으로 구성한다.( txtA.setBackground(Color.***green***;)그리고 전송과 닫기 버튼을 만들어 addActionListener을 추가해준다. 전송 버튼을 눌렸을 경우 입력이 되도록 하였고, 닫기 버튼을 누르면 dispose로 그 프레임을 종료되도록 설정해주었다. |

|  |
| --- |
| 1-3) 서버측 구성(클라이언트로부터 문자열 받아서 다른 클라이언트에게 문자열을 보내주는 스레드)  \_MultiChatServer |
| **class** EchoThread **extends** Thread{  Socket socket;  Vector<Socket> vec;  **public** EchoThread(Socket socket, Vector<Socket> vec){  **this**.socket = socket;  **this**.vec = vec;  }  **public** **void** run(){  BufferedReader br = **null**;  **try**{  br = **new** BufferedReader(  **new** InputStreamReader(socket.getInputStream()));  String str =**null**;  **while**(**true**){    str=br.readLine();    **if**(str==**null**){    vec.remove(socket);  **break**;  }    sendMsg(str);  }    }**catch**(IOException ie){  System.***out***.println(ie.getMessage());  }**finally**{  **try**{  **if**(br != **null**) br.close();  **if**(socket != **null**) socket.close();  }**catch**(IOException ie){  System.***out***.println(ie.getMessage());  }  }  } |
| 클라이언트로부터 문자열 받아서 다른 클라이언트에게 문자열을 보내주는 스레드이다. bufferedReader은 byte 단위의 문자를 입출력할 때 사용한다. 소켓을 생성하여 연결하고 클라이언트로 문자열을 받는다. 그리고 상대가 접속을 끊으면 멈춘다. 그리고 sendMsg를 통해 연결된 소켓들을 통해서 다른 클라이언트에게 문자열을 보내준다. |

|  |
| --- |
| 1-4) 서버측 구성(전송받은 문자열 다른 클라이언트들에게 보내주는 메서드)\_MultiChatServer |
| **public** **void** sendMsg(String str){  **try**{  **for**(Socket socket:vec){    **if**(socket != **this**.socket){  PrintWriter pw =  **new** PrintWriter(socket.getOutputStream(), **true**);  pw.println(str);  pw.flush();  }  }  }**catch**(IOException ie){  System.***out***.println(ie.getMessage());  }  }  } |
| for문을 돈다. 그러나 현재의 socket들이 데이터를 보낸 클라이언트인 경우를 제외하고 나머지 socket들에게 데이터를 보낸다. (단, 여기서 얻어온 소켓들은 내가 생성한 것이 아니기 때문에 닫지 않는다) |
| 1-5) 서버측 구성(메인 구성)\_MultiChatServer |
| **public** **class** MultiChatServer {  **public** **static** **void** main(String[] args) {  ServerSocket server = **null**;  Socket socket =**null**;    Vector<Socket> vec = **new** Vector<Socket>();  **try**{  server= **new** ServerSocket(1992);  **while**(**true**){  System.***out***.println("접속대기중..");  socket = server.accept();  vec.add(socket);    **new** EchoThread(socket, vec).start();  }  }**catch**(IOException ie){  System.***out***.println(ie.getMessage());  }  }  } |
| ServerSocket을 생성하여 ServerSocket 객체가 특정 포트(1992번)에서 클라이언트의 연결을 기다리고 있다는 것을 가정하고 socket = server.accpet()를 통해 서버는 클라이언트의 연결을 기다린다. 그리고 클라이언트와 연결된 소켓을 벡터에 담고(vec.add(socket)) 스레드를 구동한다. |

|  |
| --- |
| 1-6) 클라이언트측 구성(키보드로 문자열 입력받아 서버로 전송)\_MultiChatClient |
| **class** WriteThread{  Socket socket;  ClientFrame cf;  String str;  String id;  **public** WriteThread(ClientFrame cf) {  **this**.cf = cf;  **this**.socket= cf.socket;  }  **public** **void** sendMsg() {    BufferedReader br=  **new** BufferedReader(**new** InputStreamReader(System.***in***));  PrintWriter pw=**null**;  **try**{    pw=**new** PrintWriter(socket.getOutputStream(),**true**);    **if**(cf.isFirst==**true**){  InetAddress iaddr=socket.getLocalAddress();  String ip = iaddr.getHostAddress();  getId();  System.***out***.println("ip:"+ip+"id:"+id);  str = "["+id+"] 님 로그인 ("+ip+")";  }**else**{  str= "["+id+"] "+cf.txtF.getText();  }    pw.println(str);    }**catch**(IOException ie){  System.***out***.println(ie.getMessage());  }**finally**{  **try**{  **if**(br!=**null**) br.close();    }**catch**(IOException ie){  System.***out***.println(ie.getMessage());  }  }  }  **public** **void** getId(){  id = Id.*getId*();  }  } |
| 스레드에 키보드로부터 읽어오기(InputStreamReader) 위한 스트림객체를 생성한다(BufferedReader) 서버로 문자열을 전송하기 위한 스트림객체를 생성한다(PrinterWriter(socket.getOutputStream)). 첫번째 데이터는 id인데, 상대방에게 id와 함께 내 IP를 전송한다.(InetAddress를 이용하여 IP를 전송) |

|  |
| --- |
| 1-7) 클라이언트측 구성(서버로부터 전송된 문자열 읽어오기 위한 스레드)\_MultiChatClient |
| **class** ReadThread **extends** Thread{  Socket socket;  ClientFrame cf;  **public** ReadThread(Socket socket, ClientFrame cf) {  **this**.cf = cf;  **this**.socket=socket;  }  **public** **void** run() {  BufferedReader br=**null**;  **try**{  //서버로부터 전송된 문자열 읽어오기 위한 스트림객체 생성  br=**new** BufferedReader(  **new** InputStreamReader(socket.getInputStream()));  **while**(**true**){  //소켓으로부터 문자열 읽어옴  String str=br.readLine();  **if**(str==**null**){  System.***out***.println("접속이 끊겼음");  **break**;  }  //전송받은 문자열 화면에 출력  //System.out.println("[server] " + str);  cf.txtA.append(str+"\n");  }  }**catch**(IOException ie){  System.***out***.println(ie.getMessage());  }**finally**{  **try**{  **if**(br!=**null**) br.close();  **if**(socket!=**null**) socket.close();  }**catch**(IOException ie){}  }  }  } |
| 스레드에 서버로부터 전송된 문자열을 읽어오기 위한 스트림객체를 생성한다. 이때, socket에서 getInputstream을 통해 가져오 InputStreamReader로 읽는다. 소켓으로 문자열을 readLine을 통해 읽어준다. 입력이 없을때는 접속이 끊겼음을 보여주고 멈춘다. |

|  |
| --- |
| 1-8) 클라이언트측 구성(메인)\_MultiChatClient |
| **public** **class** MultiChatClient {  **public** **static** **void** main(String[] args) {  Socket socket=**null**;  ClientFrame cf;  **try**{  socket=**new** Socket("127.0.0.1",1992);  System.***out***.println("연결성공!");  cf = **new** ClientFrame(socket);  **new** ReadThread(socket, cf).start();  }**catch**(IOException ie){  System.***out***.println(ie.getMessage());  }  }  } |
| 클라이언트 메인 부분이다. 소켓을 ipaddress가 127.0.0.1 이고 port 주소가 1992번에 연결시킨다. 환경상 주소를 127.0.0.1(자신) 으로 함으로써 단일 PC로 통신을 구현한다. |

1. 실행화면

|  |
| --- |
|  |
| 1. **MultiChatServer실행(서버 실행)** |
|  |
| 1. **MultiChatClient1 실행(클라이언트1 실행\_id값 입력전**) |
|  |
| 1. **MultiChatClient1 실행(클라이언트1 실행\_id값 입력후)** |

|  |
| --- |
|  |
| 1. **MultiChatClient2 실행(클라이언트2 실행\_id값 입력전**) |
|  |
| 1. **MultiChatClient2 실행(클라이언트2 실행\_id값 입력전**) |
|  |
| 1. **MultiChatClient1실행화면(클라이언트2가 접속했을때)** |

|  |
| --- |
|  |
| 1. **MultiChatClient1 실행화면(대화)** |
|  |
| 1. **MultiChatClient2 실행화면(대화)** |

1. 결론 및 보완점

**Port iddress:1992**

**Port iddress:1992**

127.0.0.1

127.0.0.1

Client 1(송승기)

Server

127.0.0.1

Client 2(김찬미)

**Port iddress:1992**

**Port iddress:1993**

**Port iddress:1991**

Java의 네트워킹 메소드를 이용하여 소켓 프로그래밍을 응용해 채팅 프로그램을 구현해 보았다. 채팅 프로그램의 GUI는 JAVA Swing을 이용하여 디자인을 해보았다. 자바 socket, ServerSocket객 체를 생성하여, 서버와 클라이언트의 IP주소와 Port주소를 이용해 연결해주고 그 소켓 통로를 이용하여 서로의 메시지를 주고 받는 형식으로 구현하였다. 위와 같은 그림으로 도식화 시킬 수 있을 것이다. Client와 Server은 1992번 포트로 처음에는 연결한다. 연결된 다음에는 다른 클라이언트와의 연결을 위해서 1992번 포트를 다른 포트로 돌려 연결되고 1992번 포트는 다시 열어준다. 위와 같이 Client 1과 연결된 후 1992번 포트를 비워준다. 이 때 사용되는 메소드는 server.accept() 이다. 이렇게 열린 포트로 Client 2가 연결을 하고, 연결한 후 또 1992번 포트를 열어준다. 이런식으로 서버-클라이언트 소켓 프로그래밍이 구현된다.

아직 많이 부족한 실력이라 미흡한 점이 많이 보인다. 화면을 구축하는데 에도 여러가지 어려움이 있었다. Swing의 setLayout을 이용하여 상황에 맞는 레이아웃을 집어넣었다. 이 프로젝트를 통해 알 수 있었던 부분은 소켓 프로그래밍에 대해 좀 더 숙달될 수 있었고, Swing 사용 부분에 대해서 응용력을 익힐 수 있었다. 그리고 exit와 dispose의 차이를 알 수 있었다. exit는 모든 프레임을 닫는 것이고 dispose는 해당 프레임을 닫는 것인데 상황에 맞게 유용하게 쓸 수 있을꺼 같다.

좀 더 보완하고자 하는 점은 로그인 화면의 부분이다. 아직 데이터베이스를 잘 다루지 못해 하지 못했지만, 로그인 부분을 DB를 이용하여 사용자 회원가입 기능을 추가할 수 있을것이다. id와 비밀번호등을 저장해놓고, 로그인시 id와 비밀번호를 쳐서 사용자가 있으면 채팅방으로 이동할 수 있게 설정을 해놓을 수도 있을 것이다. 그리고 친구 기능을 추가하여, 그 사용자에게 친구로 등록된 사용자(id)자 항목에 나열되어 있고, 그 사용자랑 1:1 및 다중 채팅을 할 수 있게 할 것이다. 그리고 위 그림에서 보면 ip주소가 모두 똑같이 127.0.0.1 로 똑같이 자신의 PC로 설정되어 있는데, 이 주소를 가상 머신을 이용하여, 서로 다른 IP주소를 가진 서버와 클라이언트 끼리도 통신을 할 수 있을 것이다.