프로젝트 최종 보고서

-소켓 통신을 사용한 채팅프로그램

개발자 명: 장은희

채팅프로그램

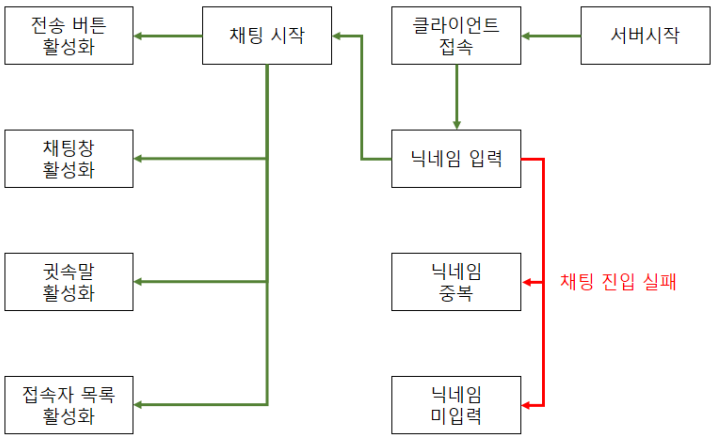
1. 프로젝트 주제 및 계획

개발 기간	2021.03.03 ~ 2021.03.15 (총 13일)
개발 운영체제	Window
개발 도구	Eclipse
개발 언어	Java, Java FX
개발 인원	1명
프로젝트 이름	채팅 프로그램
프로젝트 설명	TCP/IP를 이용한 자바 기반의 socket 프로그래밍인 채팅 프로그램이다. 하나의 서버 프로그램과 하나의 클라이언트 프로그램으로 구성되어 있으며, 각 클라이언트를 담당하는 Thread를 생성하여, 여러 명이 동시에 통신할 수 있도록 구현하였다.
주제 선정 의도	소켓 통신에 대한 공부와 이해를 위해, 일상생활에서 가장 많이 쓰이는 채팅 프로그램을 구현하였다.
기능	1) 입장/퇴장 알림 2) 닉네임 미입력 및 중복 체크 3) 접속 인원 4) 귓속말 5) 비속어 방지

1.1 개발 시간표

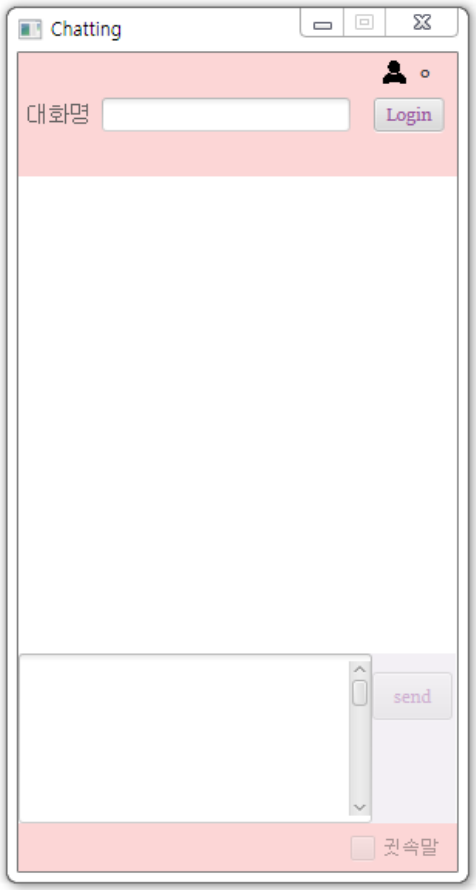
	3.03	3.04	3.05	3.06	3.07	3.08	3.09	3.10	3.11	3.12	3.13	3.14	3.15
분석													
소켓 통신 분석													
설계													
구현 기능 아이디어 기획													
GUI화면 구상													
구현													
클래스 및 기능 구현													
버그 수정 및 점검													

1.2 플로우차트를 이용한 코드 설계



2 사용자 인터페이스 및 기능 설명

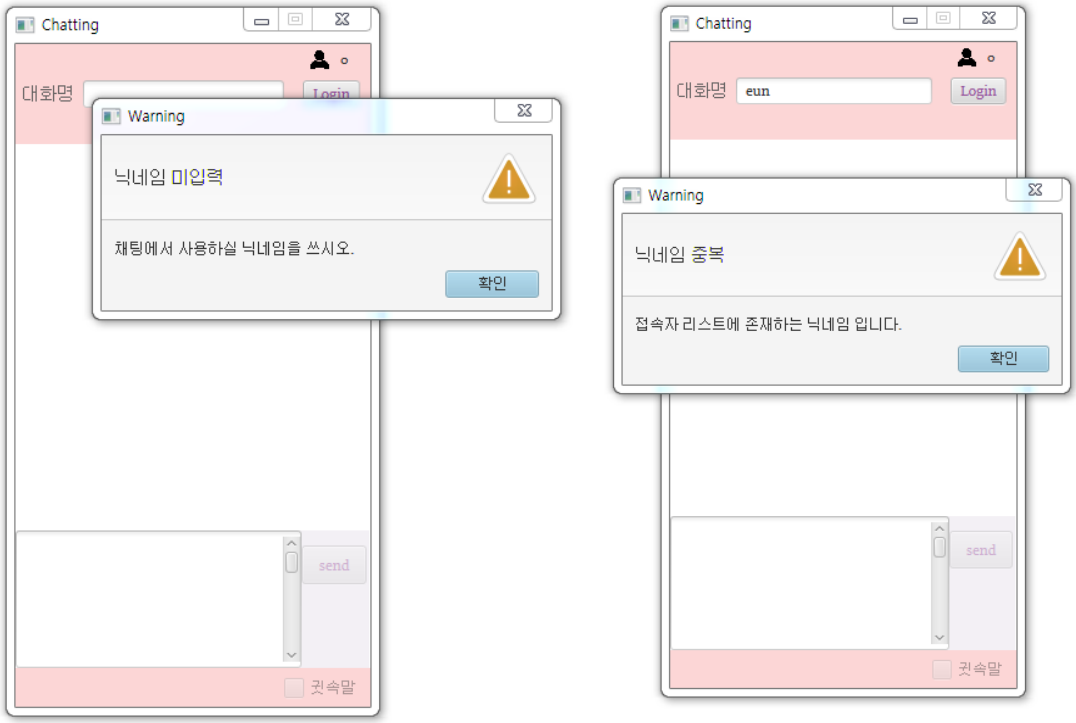
2.1 채팅 초기화면



사용자는 프로그램 실행 후, 대화명 입력 뒤 로그인 버튼을 누르면 채팅에 입장할 수 있다.

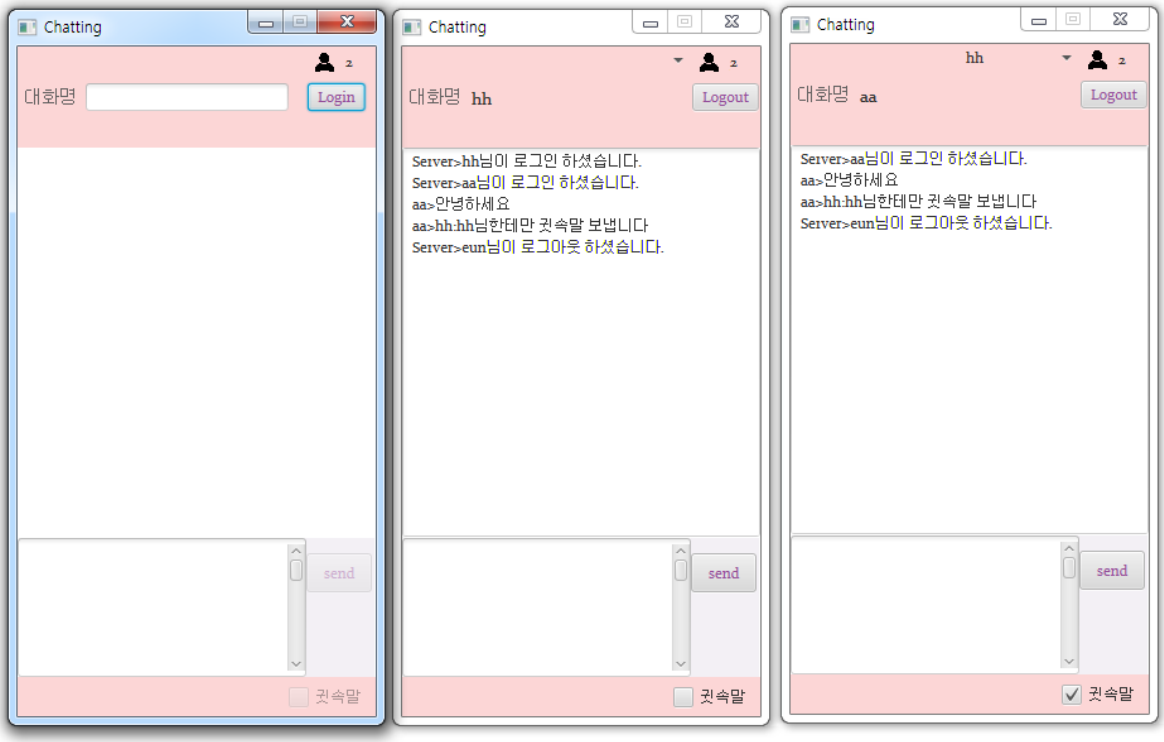
로그인 전에는 채팅창, 접속자 목록, 메시지 전송 버튼, 귓속말 체크박스 모두 비활성화 상태이다.

2.2 닉네임 미입력 및 중복체크



닉네임 칸이 공란 혹은 서버의 리스트 컬렉션에 이미 존재하는 닉네임일 경우, 경고 메시지가 뜨며 채팅 접속이 불가능해진다.

2.3 사용자 로그인 및 로그아웃 화면

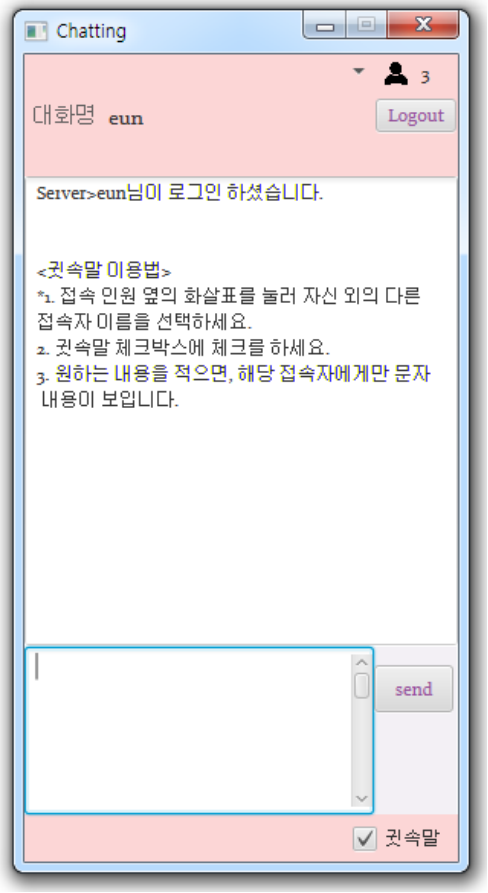


사용자가 입장 및 퇴장 시 채팅에 접속한 인원들에 한하여 알림 메시지를 전송함. 그와 동시에 입장한 인원의 스레드는 서버의 ArrayList에 추가하며, 접속 인원의 닉네임과 스레드를 HashMap에 추가한다. 퇴장 시, 서버의 리스트 컬렉션에서 사용자의 스레드와 닉네임을 삭제한다.

2.4 접속자 목록 확인

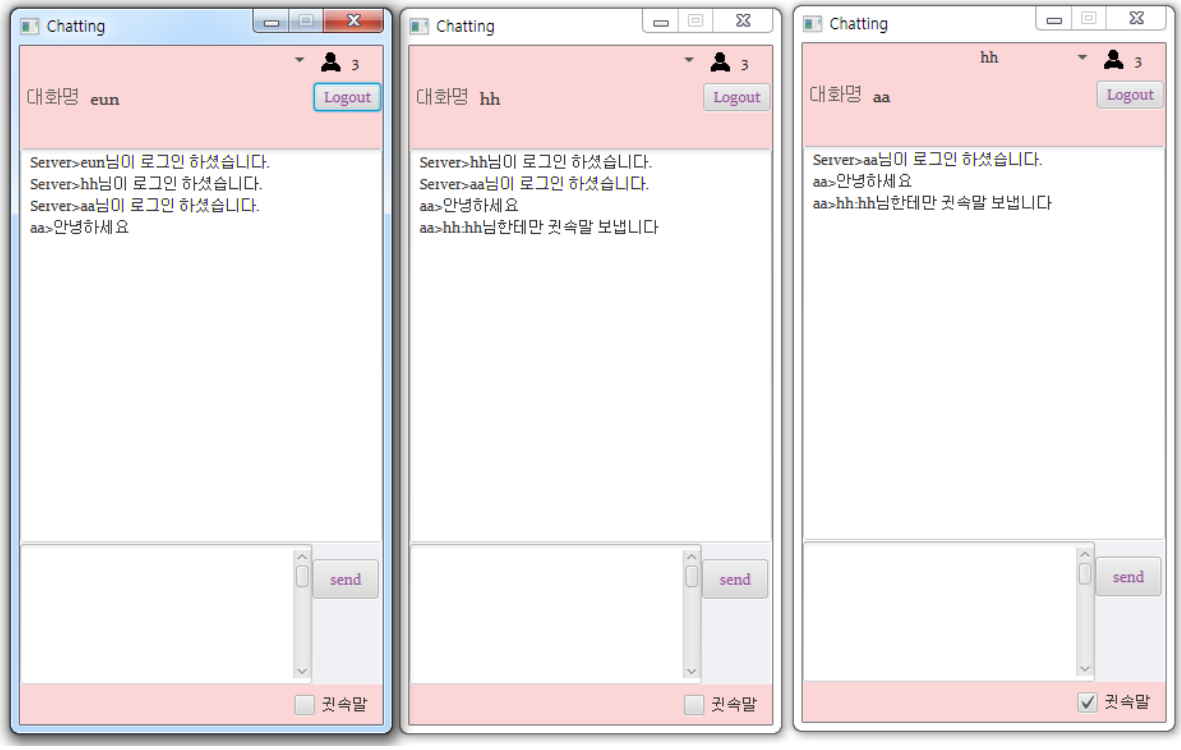
채팅창 오른쪽 상단에 접속 인원수를 표시하며, 접속 인원수 옆의 화살표 표시를 누를 시, 현재 접속한 인원들의 목록을 표시한다.

2.5 귓속말 기능 도움말



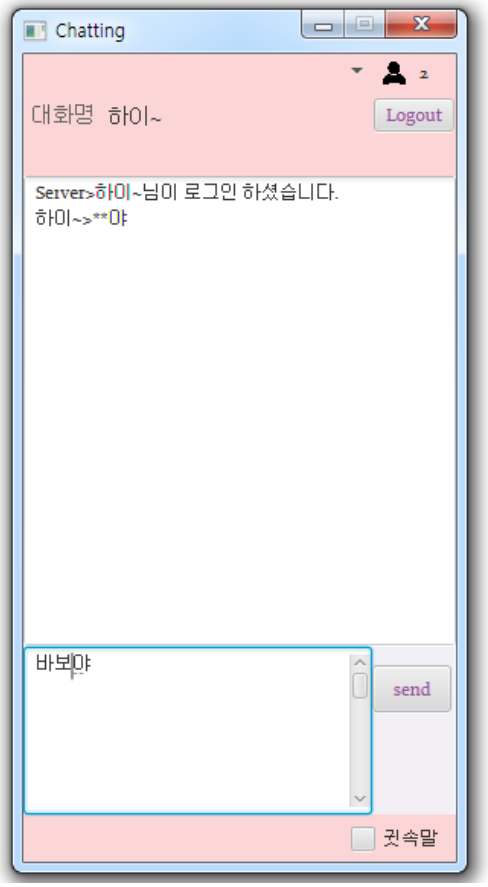
귓속말 체크박스를 체크한 후, 선택한 대상자가 없을 때, 혹은 자기 자신을 선택한 경우, 귓속말 사용법이 사용자의 채팅창에만 표시된다.

2.6 귓속말



귓속말 체크박스를 체크한 후, 접속자 목록에서 원하는 대상을 선택한 뒤 메시지를 보내면, 선택한 대상자와 보낸 사람만 채팅 확인이 가능하다.

2.7 비속어 방지



깨끗한 채팅창을 위한 비속어 방지 기능을 넣음. replace() 메서드로 비속어를 "**"로 대체한다.

3. 프로그램 구성 및 코드 설명

3.1 프로그램 구성

MultiChatServer 클래스	**[ChattingServer 프로젝트]** - start()메서드 : 클라이언트 소켓 연결 허용 및 클라이언트 스레드 생성. - msgSendAll(String msg)메서드 : 모든 클라이언트들에게 메시지를 전달하는 메서드. - ClientThread 클래스의 run()메서드 : 클라이언트가 서로 채팅을 주고 받을 시, 클라이언트로부터 데이터를 전송 받고 전송해 주는 메서드.
Main 클래스	**[Java_Chatting 프로젝트]** - init() 메서드 : 처음 클래스 시작 시, 실행하는 메서드, ip주소를 ChatClient클래스에 전달하고, 클라이언트 화면에 띄울 창 선택 메서드. - start(Stage primaryStage) 메서드 : 메인 창 설정 및 클라이언트 화면에 창을 띄어주는 메서드. - stop() 메서드 : 시스템 종료 메서드.
ChatClient 클래스	**[Java_Chatting 프로젝트]** - connectServer 메서드 : ip주소와 소켓을 통해 서버에 연결을 요청하는 메서드.
ChatController 클래스	**[Java_Chatting 프로젝트]** - initialize(URL arg0, ResourceBundle arg1) 메서드 : 채팅화면 초기 설정 클래스. - LoginHandle 메서드 : 로그인 버튼 클릭 시 실행되는 메서드. - change_Login() 메서드 : 로그인 후, 화면 설정 메서드. - change_Logout() 메서드 : 로그아웃 후, 화면 설정 메서드. - SendHandle() 메서드 : 메시지 전송버튼 클릭시 실행되는 메서드 - getMsg() 메서드 : 클라이언트가 서버로부터 데이터를 전송받는 메서드.

3.2 코드 설명

code	해설

[MultiChatServer – start()메서드]
`public void start() {`
`  try {`
`  ss = new ServerSocket(8888);`
`  System.out.println("server start");`
`  while (true) {`
`  s = ss.accept();`
`  ClientThread c = new ClientThread();`
`  clientThread.add(c);`
`  c.start();`
`  }`
`  } catch (Exception e) {`
`  System.out.println("[Multi Server]start() Exception !!");`
`  }`
`  }`
	[서버 시작]
	서버의 start() 메서드에서는 while문을을 통해 클라이언트의 연결 요청을 기다린다. 클라이언트에게서 연결 요청이 올 시, 소켓 연결을 허용하고, 서버의 ArrayList에 클라이언트의 스레드를 담고, 스레드를 실행시킨다.

[MultiChatServer – run()메서드]
`class ClientThread extends Thread{`
`//  DBConn dbConn = new DBConn();`
`  String msg;`
`  String[] rmsg;`
`  private BufferedReader inMsg = null;`
`  private PrintWriter outMsg = null;`
`  Client client;`
`  String ready = null;`
`  public void run() {`
`  boolean status = true;`
`  System.out.println("##Chatting start...");`
`  try {`
`  inMsg = new BufferedReader(new InputStreamReader(s.getInputStream()));`
`  outMsg = new PrintWriter(s.getOutputStream(), true);`
`  while (status) {`
`  msg = inMsg.readLine();`
`  rmsg = msg.split("/");`
`  String tmsg = "";`
`  if (msg != null) {`
`  if (rmsg[0].equals("Login")) {`
`  if(map.containsKey(rmsg[1])) {`
`  Thread getFromName = (Thread)map.get(rmsg[1]);`
`  for(ClientThread ct : clientThread) {`
`  if(ct.equals(getFromName)) {`
`  ct.outMsg.println("NoLogin/"+rmsg[1]);`
`  }`
`  }`
`  }else {`
`  map.put(rmsg[1], Thread.currentThread());`
`  System.out.println(Thread.currentThread());`
`  msgSendAll("Login/"+rmsg[1]);`
`  msgSendAll("Client_number/"+ map.size());`
`  }`
	[데이터 송수신]
	ClientThread클래스의 run()메서드에서는 클라이언트로부터 온 데이터를 수신하고 전송하는 역할을 담당한다.
	
	클라이언트로부터 수신받은 메시지를 "/"마다 분리시켜 rmsg 배열에 저장한다.
	
	조건문을 통해, 조건문과 일치하는 대로 클라이언트에게 데이터를 송신한다. rmsg[0]이 "Login"과 일치할 시, 서버는 클라이언트의 로그인을 허용하는 데이터를 송신한다. 또한 클라이언트의 닉네임과 스레드를 Hashmap 컬렉션에 담고, 접속 인원을 알리기 위한 데이터를 함께 송신한다.

```
            String t="";
            for(Entry<String,Thread> entry : map.entrySet()) {
                t=t+entry.getKey()+"#";
            }

            msgSendAll("Client_List/"+t);
            System.out.println("이름:"+t);

        }else if(rmsg[0].equals("Logout")) {
            map.remove(rmsg[1]);
            msgSendAll("Logout/"+rmsg[1]);
            msgSendAll("Client_number/"+map.size());
        }else if(rmsg[0].equals("Whisper")) {
            Thread getName = (Thread)map.get(rmsg[2]);
            Thread getFromName = (Thread)map.get(rmsg[1]);

            for(ClientThread ct : clientThread) {

                if(ct.equals(getFromName) || ct.equals(getName)) {
                    ct.outMsg.println("Whisper/"+rmsg[1]+"/"+rmsg[2]+"/"+rmsg[3]);
                }
            }


        }else if(rmsg[1].contains("바보") || rmsg[1].contains("멍청이")) {
            msgSendAll("비속어 발지/"+rmsg[0]+"/"+rmsg[1]);
        }

        else {
            msgSendAll(msg);
        }
        msg=null;
    }
}

        map.remove(rmsg[1]);
        msgSendAll("Client_number/"+map.size());
        clientThread.remove(this);
        System.out.println("##" + this.getName() + "stop!!");
        this.interrupt();
        }catch(IOException e) {
            clientThread.remove(this);
            System.out.println("[ChatThread]run() IOException 발생!!");
    }
}
```

rmsg[0]이 "Logout"일 시, "Login"과 반대로 클라이언트의 스레드와 닉네임을 HashMap에서 제거하고, 접속자 목록을 업데이트해 주는 데이터를 클라이언트에 전송한다.

[Main – init()]

```
@Override
public void init() throws Exception {
   SocketConnect = new ChatClient("127.0.0.1");
   FXMLLoader loginLoader = new FXMLLoader(getClass().getResource("/fxml_source/Chatting.fxml"));
   login = loginLoader.load();
   chatController = loginLoader.getController();

   chatController.setSocket(SocketConnect);

   super.init();
}
```

[초기 화면 설정]

연결할 서버의 ip주소를 ChatClient클래스에 넘겨주고, 화면에 띄울 fxml파일을 설정한다.

[Main – start()]

```
@Override
public void start(Stage primaryStage) {
   try {
       primaryStage.initStyle(StageStyle.DECORATED);
         primaryStage.setMaximized(false);
         chatController.setPrimaryStage(primaryStage);
         login.setOnMousePressed(new EventHandler<MouseEvent>() {
            @Override
            public void handle(MouseEvent event) {
               xOffset = event.getSceneX();
               yOffset = event.getSceneY();
            }
         });

         login.setOnMouseDragged(new EventHandler<MouseEvent>() {
            @Override
            public void handle(MouseEvent event) {
               primaryStage.setX(event.getScreenX() - xOffset);
               primaryStage.setY(event.getScreenY() - yOffset);
            }
         });


      Scene scene = new Scene(login);
      primaryStage.setTitle("Chatting");
      primaryStage.setScene(scene);
      primaryStage.setResizable(false);
      primaryStage.show();

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

[클라이언트 창 초기 설정]

클라이언트가 채팅을 실행했을 시, 채팅창을 커서로 자유자재로 옮길 수 있도록 설정한다. 또한, 채팅창의 크기는 변경 불가능하도록 설정한다.

[ChatClient – connectServer()]

```
public void connectServer() {
    try {
        socket = new Socket(ip, 8888);
        System.out.println("[Client]Server 연결 성공!!");
        inMsg = new BufferedReader(new InputStreamReader(socket.getInputStream()));
        outMsg = new PrintWriter(socket.getOutputStream(), true);
    } catch(Exception e) {
        System.out.println("[Client]connectServer() Exception 발생!!");
    }
}
```

[서버와의 연결]

ip주소와 포트번호로 원하는 서버에 연결한다.

[ChatController – initialize(URL arg0,ResourceBundle arg1)]

```
@Override
public void initialize(URL arg0, ResourceBundle arg1) {


    Platform.runLater(()->{
    checkBox.setDisable(true);
   choiceBox.setVisible(false);
   textField.setEditable(false);
   textArea.setVisible(false);
   send.setDisable(true);
   textArea.setEditable(false);
   //label.setVisible(false);
   username2.setVisible(true);
   });
}
```

[채팅 실행 시 초기 설정]

클라이언트가 로그인 하기 전까지 닉네임 칸 외의 모든 기능을 쓸 수 없도록 설정.

[ChatController – LoginHandle()]

```
public void LoginHandle() {


    name = username2.getText().toString();
    if(name.equals("")) {

       Alert alert = new Alert(AlertType.WARNING);
       alert.setTitle("Warning");
       alert.setHeaderText("닉네임 미입력");
       alert.setContentText("채팅에서 사용하실 닉네임을 쓰시오.");
       alert.showAndWait();


    }else {
       if(logout==0) {
           SocketConnect.getOutMsg().println("Login/"+name);

       }else {
          change_Logout();
          logout = 0;
          textArea.setVisible(false);
          SocketConnect.getOutMsg().println("Logout/"+name);

       }
    }
}
```

[로그인 버튼 기능]

클라이언트가 닉네임을 입력 후 로그인 버튼 클릭 시, 닉네임 미입력 혹은 중복이 아닐 경우, 소켓에 "Login" 문자와 클라이언트가 입력한 닉네임을 소켓에 담아 서버에 전송한다.

[ChatController – SendHandle()]

```
public void SendHandle() {
    String to_user = choiceBox.getValue().toString();
    if(checkBox.isSelected()==true) {
        if(to_user.equals("") || to_user.equals(name)) {
            Platform.runLater(()->{//wrapText="true" textarea 줄바꿈.
                textArea.appendText("\n\n<귓속말 이용법>\n*1. 접속 인원 중의 특정 한명을 골라 자신 외의 다른"
                        + " 접속자 이름을 선택하세요."
                           +"\n2. 귓속말 체크박스에 체크를 하세요."
                           +"\n3. 원하는 내용을 적으면, 해당 접속자에게만 문자 "
                           + "내용이 보입니다.\n");
            });


        }else {
            SocketConnect.getOutMsg().println("Whisper/"+name+"/"+to_user+"/"+textField.getText().toString());
        }
    }else {
        SocketConnect.getOutMsg().println(name+"/"+textField.getText().toString());
    }
    Platform.runLater(()->{
        textField.setText("");
        textField.requestFocus();
    });
}
```

[메시지 전송 버튼 기능]

귓속말 체크박스를 체크하지 않았을 시, 클라이언트는 클라이언트의 닉네임과 입력한 내용을 서버에 전송한다.

귓속말 체크박스를 체크했을 시, 접속자 목록에서 자신 혹은 빈칸을 선택했을 경우, 귓속말 이용법을 자신의 채팅창에 띄움.

접속자 목록에서 자신 외의 다른 접속자를 선택했을 시, "Whisper"이라는 문자와 자신의 닉네임, 그리고 선택한 접속자의 닉네임을 서버에 전송함.

[ChatController – getMsg()]

```
public void getMsg() {
    Thread thread = new Thread() {
        @Override
        public void run() {
            String[] list = null;
           String[] rmsg = null;
           String msg = null;
           while (true) {
              try {
                 msg =  SocketConnect.getInMsg().readLine(); // 서버에서 보낸 메시지를 읽음.
                 System.out.println("Log");
              } catch (IOException e1) {
                 System.out.println("소켓에러");
              }
              if (msg != null) {
                 System.out.println(msg+"client");
                 rmsg = msg.split("/");
                 if (rmsg[0].equals("Login")) {
                     String user=rmsg[1];
                     change_Login();
                     Platform.runLater(()->{
                         textArea.appendText("Server>"+ user +"님이 로그인 하셨습니다.\n");
                     });



                 }else if(rmsg[0].equals("NoLogin")) {
                     System.out.println("2");
                     Platform.runLater(()->{
                         Alert alert = new Alert(AlertType.WARNING);
                         alert.setTitle("Warning");
                         alert.setHeaderText("닉네임 중복");
                         alert.setContentText("접속자 리스트에 존재하는 닉네임 입니다.");
                         alert.showAndWait();
                     });

                    }else if(rmsg[0].equals("Logout")) {
                     String user=rmsg[1];
                     Platform.runLater(()->{
                         textArea.appendText("Server>"+ user +"님이 로그아웃 하셨습니다.\n");
                     });

                 }
```

[ChatController – change_Logout]

```
public void change_Login() {

    Platform.runLater(()->{
        checkBox.setDisable(false);
        choiceBox.setVisible(true);
         username2.setVisible(false);
         username.setText(name);
         Login.setText("Logout");
         choiceBox.setValue("");
         logout = 1;
         textArea.setText("");
         textArea.setVisible(true);
         textField.setEditable(true); //채팅 치는 곳
         send.setDisable(false); //전송버튼


    });
}
```

[서버로부터의 데이터 수신]

서버로부터 받은 데이터를 "/" 마다 분리시켜 rmsg배열에 담는다.

rmsg[0]이 "Login"일 시, change_Login 메서드를 실행한다. 또한 모든 클라이언트의 채팅창에 클라이언트 입장을 알린다.

rmsg[0]이 "NoLogin"일 시, 클라이언트는 채팅에 접속할 수 없다.

[채팅 접속시 채팅창 기능 설정]

클라이언트가 채팅 접속에 성공했을 경우, 클라이언트가 채팅창의 모든 기능을 사용할 수 있도록 설정한다.

[ChatController – change_Logout()]

```
public void change_Logout() {

   Platform.runLater(()->{
       checkBox.setDisable(true);
         number.setText("0");
         choiceBox.setVisible(false);
         textField.setEditable(false);
         send.setDisable(true);
         username.setText("");
         username2.setVisible(true);
         username2.setText("");
         Login.setText("Login");
      });
}
```

[서버로부터의 데이터 수신]

클라이언트가 로그아웃 버튼을 클릭했을 경우, 닉네임 칸 외의 모든 기능을 쓸 수 없도록 설정한다. 또한 버튼에 적힌 "Login"버튼을 "Logout"으로 바꾼다.

4. 어려웠던 점 및 느낀 점

프로젝트를 진행하면서 어려웠던 점	**1. 배우지 않은 지식 활용** - TCP 소켓 통신 관련 부분의 지식이 없는 상태에서 해당 프로젝트를 진행하여, 많은 어려움을 겪었다. 하지만 프로젝트를 완성시키기 위해 관련 자료를 끊임없이 보았으며, 포기하지 않고 프로그래밍 개발을 계속하여 시도한 결과, 필요한 기능들을 구현할 수 있게 되었다. **2. 혼자 연구하고 개발하는 개인프로젝트** - 혼자 구현할 수 있는 기능을 생각하고, 모든 기능을 구현해야 되어 시간이 부족하였다. 하지만 혼자 연구하고, 노력한 끝에 자신의 역량을 더욱 발전 시킬 수 있었다. **3. 새로운 기능으로 인한 기존 기능 오류 발생** - 새로운 기능의 추가로 인해 기존 기능과 충돌하며 오류가 발생하는 상황이 많았음. 오류가 생길 시 바로 인터넷 서치를 통해 오류 원인을 찾고, 해결방법을 연구하였다.
느낀 점	- 새로 구현한 기능에 오류가 있을 수 있으므로, 복잡한 기능은 바로 원본 코드에 기능을 구현하는 것보다, 새로운 프로젝트에서 기능 구현을 실험 한 후 원본 코드에 삽입하는 방법으로 오류를 줄일 수 있었다. - 팀 프로젝트의 협업도 매우 중요하지만, 개인 프로젝트를 통한 자신의 역량 발전도 필요하다는 것을 느꼈다.