REPORT

동국대학교
dongguk university

- 과목명 : 객체지향프로그래밍
- 담당교수 : 엄진영 교수님
- 학과 : 컴퓨터공학과
- 학번 : 2019112130
- 이름 : 조양진

객체지향프로그래밍 프로젝트 보고서

2019112130 조양진

목차

1. 문제분석

프로젝트: 채팅프로그램

VerySimpleChatServer.java
https://github.com/bethrobson/Head-First-Java/blob/master/chap15/VerySimpleChatServer.java

SimpleChatClient.java
https://github.com/bethrobson/Head-First-Java/blob/master/chap15/SimpleChatClient.java

- 위 2 개의 코드는 기본적인 채팅 프로그램의 Server 와 Client 코드이다
- 위 프로그램은 기본적인 서버 개설 및 채팅방 입장과 채팅기능만 구현되어있는 프로그램이다 . 아래 기능들을 추가해 보다 완성도 높은 채팅프로그램을 구현하시오

1) 채팅방 입장시 ID 생성하고 채팅 입력시 그사람 ID 가 옆에 뜨게하기 .(ID 생성시 GUI 를 통해 생성해야 합니다 . Console 창에 입력 X)
2) 채팅방 입장 , 퇴장시 채팅창에 뜨게하기
3) 채팅옆에 현재 시간 출력하기
4) 그 외 기능 구현시 난이도에 따라 +0~2 점을 부여

주어진 VerySimpleChatServer.java 파일과 SimpleChatClient.java 파일을 토대로 주어진 조건을 만족하는 기본적인 채팅 프로그램을 구현해야 합니다. 1 번 항목의 경우 채팅방 입장시 ID 를 생성하고, 해당 ID 로 채팅을 입력하면 채팅 기록 옆에 해당 ID 가 표시되도록 해야 합니다. 중요한 부분은 무조건 GUI 를 통해 ID 를 생성해야 하므로 Java 내의 Swing GUI 를 사용하여 ID 를 생성하게끔 수정해야 할 것입니다. 2 번 항목의 경우 client 가 채팅방에 입장, 퇴장한 것을 채팅창에 뜨도록 수정해야 합니다. 3 번

항목은 채팅 옆에 현재 시간을 출력하도록 수정해야 합니다. 이는 java 내의 시각을 다루는 클래스를 활용하면 될 것 같습니다. 마지막으로 그 외 기능을 구현하면 추가적인 부분점수를 받습니다.

2. 기존코드 리뷰 및 문제점과 개선 목표

A. VerySimpleChatServer.java

채팅 프로그램에 있어서 back bone에 해당하는 server의 소스코드입니다. 이 서버 위에서 다른 클라이언트들이 소켓 통신을 하여 데이터를 주고 받는 것인데, 해당 소스코드에는 어떠한 클래스와 메소드가 있는지 알아보겠습니다.

- *public static void main(String[] args)*

메인 함수에서는 new VerySimpleChatServer().go()를 통해 자신의 클래스 안의 go 메소드를 실행시킵니다.

- *public void go()*

go 메소드에서는 사용자가 서버를 가동했을 때 서버가 해야 할 일들이 있습니다. 우선 ArrayList 형 'clientOutputStream'을 동적 할당하여 나중에 client의 채팅을 입력받을 수 있게 합니다. try – catch 문으로 예외처리를 하여 본격적인 서버 실행에 있어 혹시 모를 오류를 배제합니다.
ServerSocket 클래스의 serverSock이라는 객체를 5000번 포트로 생성자 안에 인자를 넣어 생성합니다. 이후 while 무한 반복문을 사용하여 client 소켓과의 연결을 받습니다. PrintWriter 클래스의 writer 객체를 생성하여 이를 위에서 언급한 clientOutputStream에 add합니다.
이후 Thread 클래스를 사용하여 멀티스레딩을 사용합니다. 위에서 언급한 try-catch문으로 예외처리를 한 이유의 대부분의 지분은 이 멀티스레딩 때문입니다. 이렇게 여러 개의 스레드를 동시에 실행시켜야만 Client 프로그램을 여러 개 받을 수 있기 때문입니다.

- *public class ClientHandler implements Runnable*

다시 파일의 위로 올라오면 클라이언트 핸들러라는 클래스를 볼 수 있습니다. 이 클래스에서는 클라이언트의 입력값을 버퍼에서 읽어서 채팅창에 뿌려주는 역할을 합니다. cpublic ClientHandler(Socket clientSOcket)이 인풋값을 읽어오고 public void run 메소드가 tellEveryone 메소드를 호출하여 채팅창에 뿌려주는 역할을 합니다.

- *public void tellEveryone(String message)*

코드의 맨 마지막으로 가면 tellEveryone 메소드가 있습니다. 이 메소드는 클라이언트 핸들러에서 버퍼에서 읽어온 인풋을 채팅창으로 뿌리는 과정에서 필요한데, Iterator 클래스의 it을 생성하여 이를 clientOutputStream ArrayList와 연결합니다. 이후 while 문을 통해 안에 있는 내용을 flush하여 채팅창으로 뿌립니다.

B. SimpleChatClient.java

- *public static void main(String[] args)*

메인함수에서는 SimpleChatClient 자신의 클래스 안의 go 메소드를 호출합니다.

- *public void go()*

go 메소드에서는 Java Swing GUI 를 사용하여 채팅창을 구성하기 위한 객체들을 정의하고 setUpNetworking() 메소드를 호출합니다.

- *private void setUpNetworking()*

try-catch 예외처리문을 사용하여 127.0.0.1 (localhost)에서 5000번 포트를 통해 소켓을 열고 InputStreamReader, reader, writer와 같은 채팅 클라이언트에게 필요한 객체를 생성합니다.

- *public class SendButtonListener implements ActionListener*

SimpleChatClient 클래스 내부의 SendButtonListener 클래스는 위의 go 메소드안에서 JButton을 통해 메시지 보내기 버튼을 눌렀을 때 실행될 public void actionPerformed(ActionEvent ev)를 포함하고 있습니다. 따라서 클라이언트가 해당 버튼을 눌렀을 때 writer는 작성한 인풋값을 flush 하여 stream에 보냅니다.

현재 이 코드의 문제점은 클라이언트의 이름이 없어서 어떤 클라이언트가 어떤 클라이언트 프로그램인지 확인할 수 있는 방법이 없을 뿐더러, 어떤 클라이언트가 어떤 채팅을 보냈는지에 대한 정보도 주어지지 않아 이를 확인할 방법이 없습니다. 또한 채팅의 기록이 모호하여 이를 언제 보냈는지에 대한 데이터도 필요합니다. 마지막으로 현재 채팅방에 누가 들어와있고, 누가 나갔는지에 대한 정보도 필요합니다.

따라서 클라이언트 마다 프로그램 이름에 해당 클라이언트의 ID를 표시하여 한눈에 확인할 수 있도록 하며, 채팅 내용 앞에 해당 채팅을 보낸 사람의 ID를, 채팅 내용 뒤에 해당 채팅을 보낸 시점의 시각을 출력하겠습니다. 그리고 채팅방에 입장, 퇴장시에 해당 클라이언트의 ID와 시각을 함께 출력하여 어떤 사람이 채팅방에 있는지 확인할 수 있도록 하겠습니다.

3. 개선한 기능에 대한 설명

문제에 주어진 조건들을 만족하는 채팅 프로그램을 만들기 위해선 server보단 client 쪽을 수정하는 것이 간단할 것이라고 생각했습니다.

1) 채팅방 입장시 ID를 생성하고 채팅 입력시 그 사람 ID가 옆에 뜨게하기 (ID 생성시 GUI를 통해 생성해야 합니다.)

우선 해당 프로그램에는 ID를 저장하는 변수가 없기에,

```
static String userID;
```

을 SimpleChatClient.java의 public class SimpleChatClient 안에 선언 했습니다. 이 객체는 해당 클라이언트 프로그램의 유저 ID를 저장하고 있게 될 것입니다.

ID를 GUI로 생성하기 위해서 저는 새로운 클래스를 만들어서 GUI를 구성했습니다. CreateNewAccount.java의 모든 소스코드는 클라이언트를 실행 시켰을 때 ID를 GUI를 통해 입력받아 생성하기 위해 작성한 코드입니다.

```
public class CreateNewAccount {
    static boolean isAccountCreated = false;
    public void go() {

        JFrame frame = new JFrame("New Account");
        Dimension dim = new Dimension(250,100);
        frame.setResizable(false);
        frame.setPreferredSize(dim);
        frame.setLocation(960, 540);

        JTextField textfield = new JTextField();
        textfield.setToolTipText("ID 를 입력해주세요.");

        JLabel label = new JLabel("LOGIN");
        label.setHorizontalAlignment(SwingConstants.CENTER);
        label.setVerticalAlignment(SwingConstants.CENTER);

        JButton button = new JButton("OK");
        button.addActionListener(new ActionListener(){
            public void actionPerformed(ActionEvent e){
                // textfield 에 있는 ID 를 SimpleChatClient 클래스 전달
                SimpleChatClient.userID = textfield.getText();
                // 현재 창 닫기
                frame.dispose();
                new SimpleChatClient().go();
            }
        });

        frame.add(label, BorderLayout.NORTH);
```

```
        frame.add(textfield, BorderLayout.CENTER);
        frame.add(button, BorderLayout.SOUTH);

        frame.pack();
        frame.setVisible(true);
        frame.setDefaultCloseOperation(frame.EXIT_ON_CLOSE);
        frame.setLocationRelativeTo(null);
    }

}
```

위 코드를 통해 GUI 창을 통해 사용자에게서부터 ID를 textfield 안에 입력받고, OK 버튼을 누름과 동시에 SimpleChatClient의 userID 안에 textfield 안의 값을 저장합니다.

이렇게 GUI로 ID를 입력받는 일은 클라이언트 프로그램을 실행시켰을 때 그 어떤 것보다도 우선시 되어야 하기 때문에, SimpleChatClient.java의 main 함수를 다음과 같이 CreateNewAccount().go()를 실행시키도록 하였습니다.

```
    public static void main(String[] args) {
        new CreateNewAccount().go();
    }
```

또한 반대로 CreateNewAccount에서는 ID를 입력받고 저장하는 일이 끝났을 때, SimpleChatClient().go()를 실행시켜 채팅창에 입장할 수 있도록 하였습니다.

이후 채팅을 입력했을 때 그 사람의 ID가 채팅 옆에 뜨도록 하였습니다.
이 부분은 SimpleChatClient.java의 public class SendButtonListener에서 처리하였는데, writer에게 입력한 인풋 값 이전에 해당 유저의 ID를 추가하여 채팅창에 해당 유저가 어떠한 채팅을 입력하였는지 알아볼 수 있도록 하였습니다.

```
public class SendButtonListener implements ActionListener {
        public void actionPerformed(ActionEvent ev) {
            try {
                // 1. 채팅 입력시 그사람 ID 가 옆에 뜨게하기
                // 3. 채팅옆에 현재 시간 출력하기
                writer.println(userID + ": " + outgoing.getText() + " (" + no
wTime2 + ")");
                writer.flush();

            }
            catch (Exception ex) {
                ex.printStackTrace();
            }
            outgoing.setText("");
            outgoing.requestFocus();
```

```
        }
    }
```

코드에 등장한 nowTime2는 3번 항목에서 설명드리겠습니다.

2) 채팅방 입장, 퇴장시 채팅창에 뜨게하기

1번 조건을 통해 이제 모든 클라이언트들은 각각의 userID를 갖게 되었습니다. 따라서 이를 사용하여 채팅방 입장, 퇴장시에 해당 ID와 함께 입장, 퇴장을 알리는 문구를 채팅방에 띄울 수 있습니다.

마찬가지로 server측이 아닌 client 측에서 수정하는 것이 편리하다고 생각하여 저는 SimpleChatClient.java의 public void go()부터 수정하였습니다. 해당 go() 메소드에서는 Java Swing GUI를 사용하여 채팅방의 GUI를 구성하는 것이 대부분이었습니다. 하지만 go 메소드는 채팅방이 생성됨과 동시에 가장 첫번째로 실행되는 메소드이기도 합니다. 따라서 저는 여기에 writer.append를 사용하여 입장시 문구를 출력하도록 수정하였습니다.

```
        frame.setSize(650, 500);
         frame.setVisible(true);
         // 2. 채팅방 입장시 채팅창에 뜨게하기
         writer.append(userID + " joined chat. (" + nowTime1 + ")"+ "\n");
         writer.flush();
```

마찬가지로 frame이 닫힐 때에도 go() 메소드 안의 frame의 windowListener에서 windowClosing이라는 메소드 안에 퇴장 문구를 writer에게 추가시켜 해당 유저가 챗을 떠남을 채팅창에 알릴 수 있도록 하였습니다.

```
// 2. 채팅방 퇴장시 채팅창에 뜨게하기
        frame.addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent e) {
                writer.append(userID + " left chat. (" + nowTime1 + ")"+ "\n"
);
                writer.flush();
            }
        });
```

코드에 등장한 nowTime1은 3번 항목에서 설명드리겠습니다.

3) 채팅 옆에 현재 시각 출력하기

3번 조건은 사실 이미 1번 항목을 구현하면서 반쯤 구현한 것이나 다름 없습니다. 왜냐하면 채팅을 입력할 때마다 ID를 같이 출력하는 것이 가능하다면 채팅을 입력할 때마다 현재시각을 같이 출력하는 것도 가능하기 때문입니다.

우선 java에서 시각과 포맷을 다루기 위해 java.text.SimpleDateFormat과 java.util.Date를 import 했습니다. 이후 SimpleChatClient 클래스의 객체 선언부에서 다음과 같은 코드를 통해 두가지 포맷의 시각을 저장하는 nowTime1과 nowTime2를 정의했습니다.

```
    SimpleDateFormat format1 = new SimpleDateFormat ( "yyyy-MM-dd HH:mm:ss");
    Date time1 = new Date();
    String nowTime1 = format1.format(time1);

    SimpleDateFormat format2 = new SimpleDateFormat ( "HH:mm");
    Date time2 = new Date();
    String nowTime2 = format2.format(time2);
```

nowTime1의 경우 년 – 월 – 일 시:분:초의 형태를, nowTime2는 시:분의 형태를 띄는 시각입니다. 매번 채팅을 칠 때마다 초까지 보기엔 불편하므로, 저는 유저 입장, 퇴장에는 nowTime1을, 일반 채팅 옆에는 nowTime2를 붙여서 출력하도록 하였습니다.

```
    public class SendButtonListener implements ActionListener {
        public void actionPerformed(ActionEvent ev) {
            try {
                // 1. 채팅 입력시 그사람 ID 가 옆에 뜨게하기
                // 3. 채팅옆에 현재 시간 출력하기
                writer.println(userID + ": " + outgoing.getText() + " (" + no
wTime2 + ")");
                writer.flush();

            }
            catch (Exception ex) {
                ex.printStackTrace();
            }
            outgoing.setText("");
            outgoing.requestFocus();
        }
    }
```

따라서 아까 채팅에 ID를 출력하는 것과 동일하게 SendButtonListener를 수정하였고, nowTime2를 붙여서 출력했습니다.

```
        // 2. 채팅방 입장시 채팅창에 뜨게하기
        writer.append(userID + " joined chat. (" + nowTime1 + ")"+ "\n");
        writer.flush();

        // 2. 채팅방 퇴장시 채팅창에 뜨게하기
        frame.addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent e) {
                writer.append(userID + " left chat. (" + nowTime1 + ")"+ "\n"
);
                writer.flush();
```

```
            }
        });
```

사용자 입장 퇴장시에는 nowTime1을 붙여서 출력하도록 하였습니다.

4) 그 외 기능 구현시 난이도에 따라 +0~2점을 부여

소소하지만 테스트를 하는 도중 어떤 창이 어떤 유저의 클라이언트인지 헷갈려서 SimpleChatClient.java의 go() 메소드에서 JFrame의 이름을 수정하여 확인하기 편하도록 수정했습니다.

```
JFrame frame = new JFrame("Simple Chat Client - " + userID);
```

또한 채팅창에 보내기 위한 입력값을 지워주는 Erase 버튼과, 채팅창을 깔끔하게 지울 수 있게 해주는 Clear 버튼을 만들었습니다. 마찬가지로 SimpleChatClient.java에서 다음과 같은 코드를 추가했습니다.

```
        JButton sendButton = new JButton("Send");
        // 4. 그 외 기능 구현시 난이도에 따라 +0~2 점을 부여
        JButton eraseButton = new JButton("Erase"); // outgoing JTextField 지우기
        JButton clearButton = new JButton("Clear"); // incoming JTextArea 비우기
        sendButton.addActionListener(new SendButtonListener());
        eraseButton.addActionListener(new EraseButtonListener());
        clearButton.addActionListener(new ClearButtonListener());
```

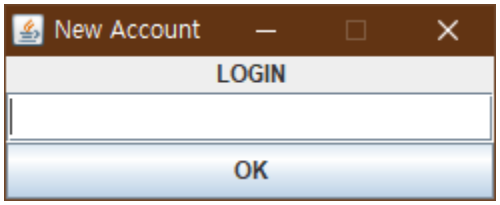
해당 ButtonListener는 다음과 같은 코드로 작동합니다.

```
    // 4. 그 외 기능 구현시 난이도에 따라 +0~2 점을 부여
    // outgoing JTextField 지우기
    public class EraseButtonListener implements ActionListener{
        public void actionPerformed(ActionEvent ev) {
            SimpleChatClient.this.outgoing.setText(null);
        }
    }
    // incoming JTextArea 비우기
    public class ClearButtonListener implements ActionListener{
        public void actionPerformed(ActionEvent ev) {
            SimpleChatClient.this.incoming.setText(null);
        }
    }
```

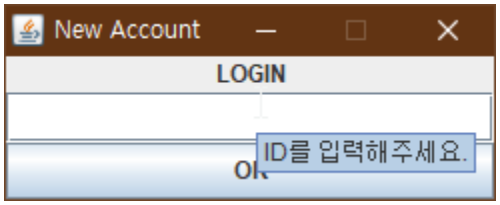
4. 개선한 기능에 대한 작동 화면

1) 채팅방 입장시 ID 생성하고 채팅 입력시 그사람 ID 가 옆에 뜨게하기 .(ID 생성시 GUI 를 통해 생성해야 합니다 . Console 창에 입력 X)

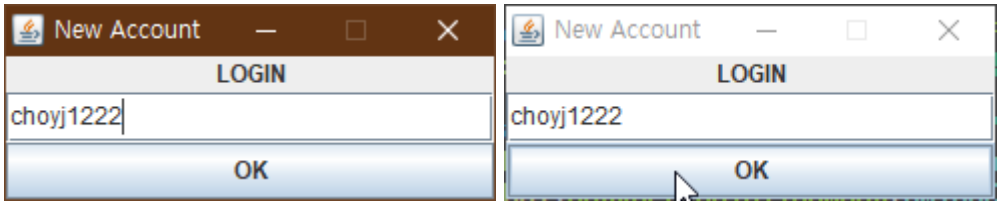
서버를 실행시키고 클라이언트를 실행시키면 다음과 같은 GUI 창이 뜹니다.



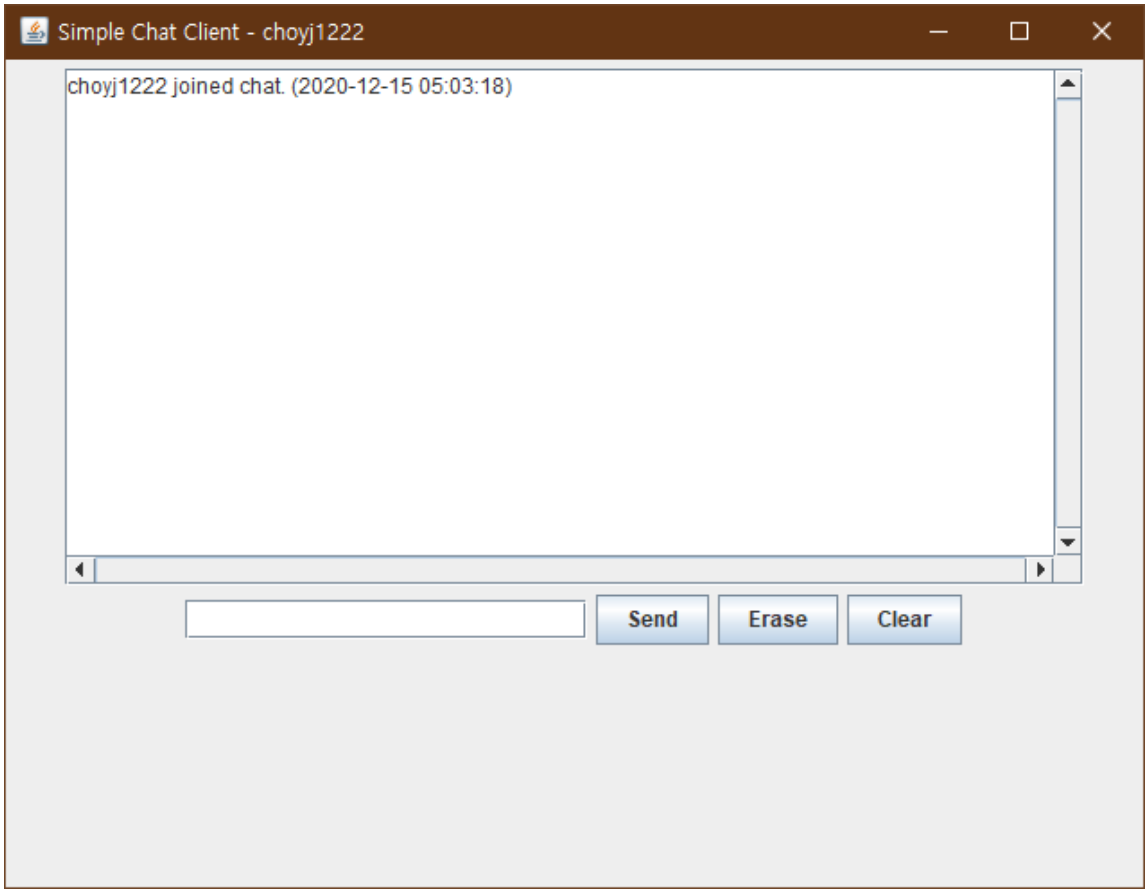
이 GUI 창이 ID 를 사용자에게 입력 받는 창입니다.

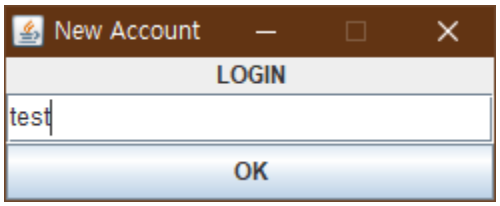


해당 JTextField 위에 마우스 커서를 두면 ID 를 입력해 달라는 문구가 나옵니다.

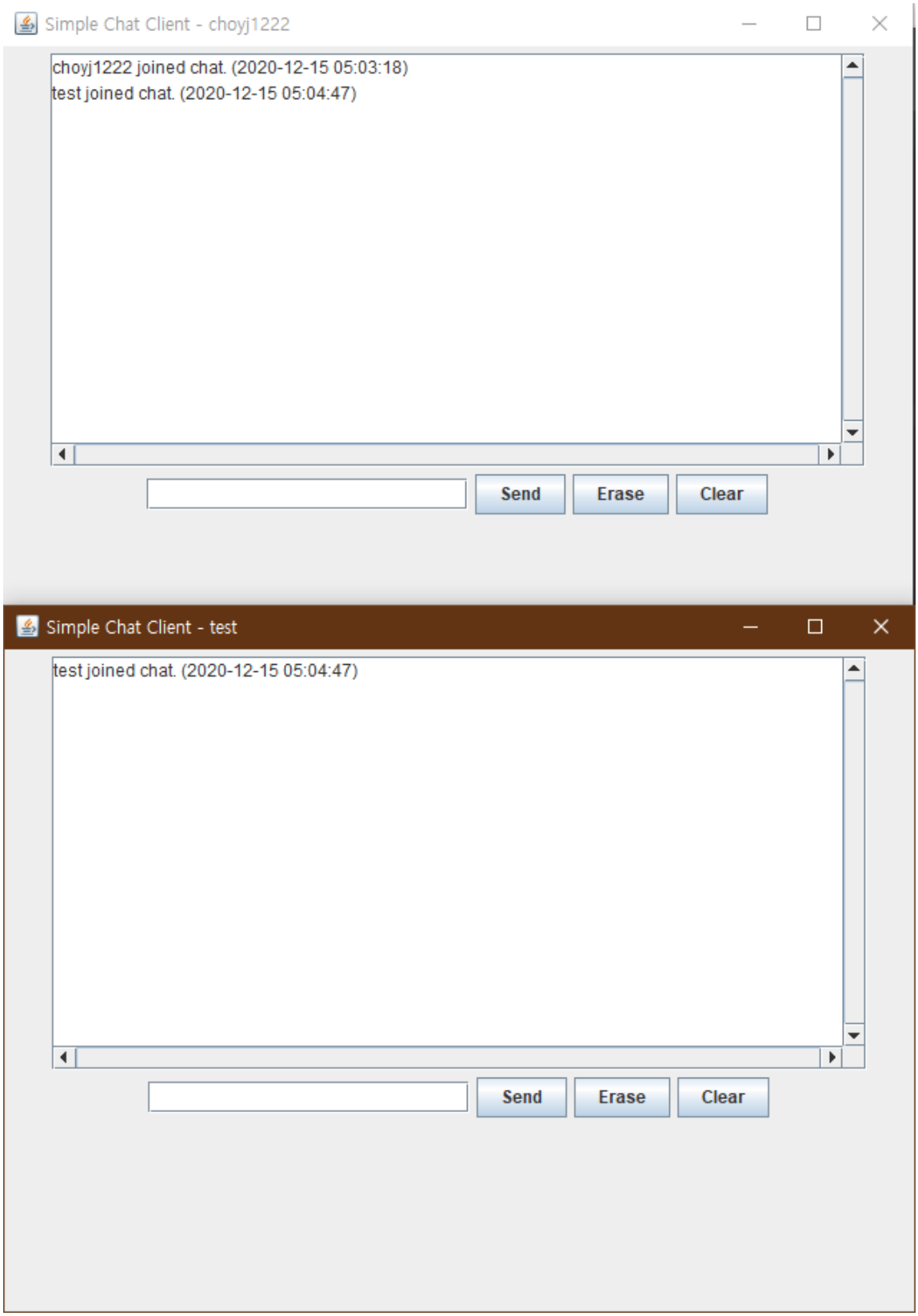


아이디를 입력하고 OK 버튼을 누르면 다음과 같이 채팅창이 뜹니다.



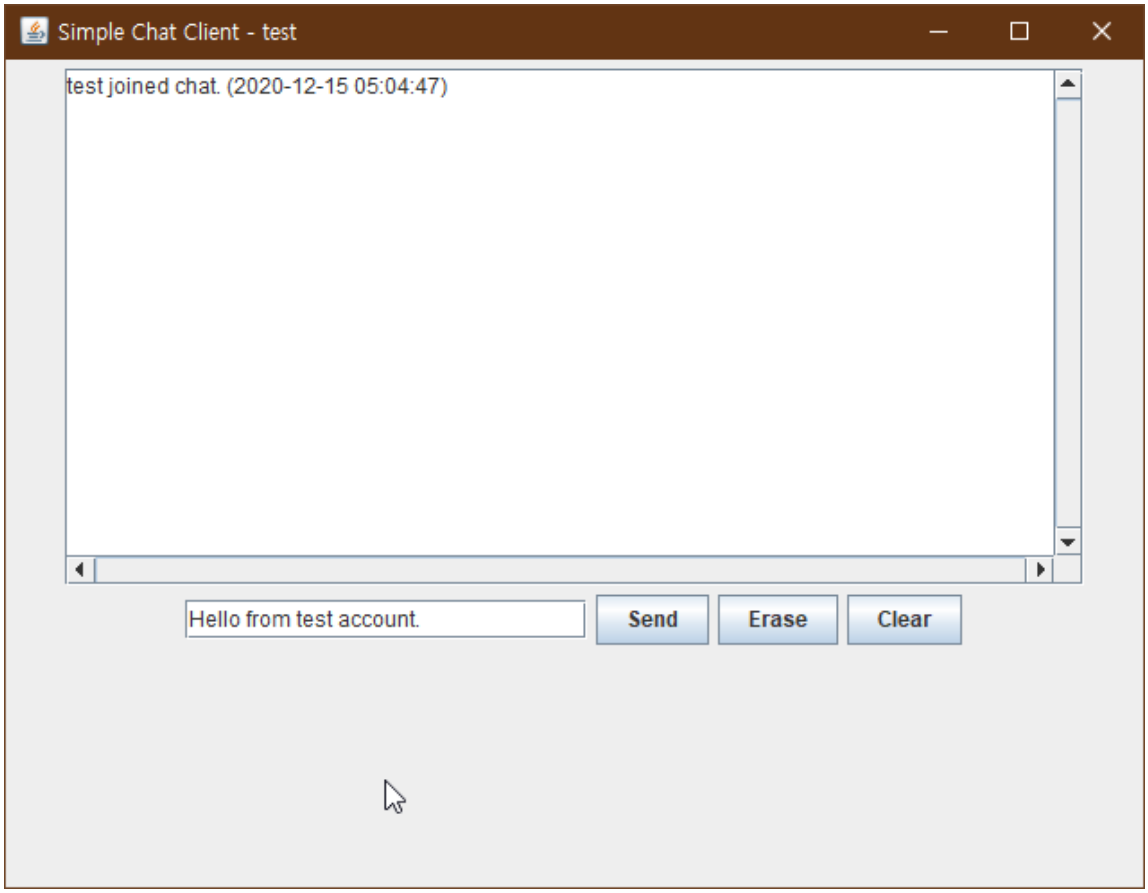


테스트를 위해 다른 클라이언트를 하나 더 실행시키고 test 라는 ID 를 입력시켜보겠습니다.

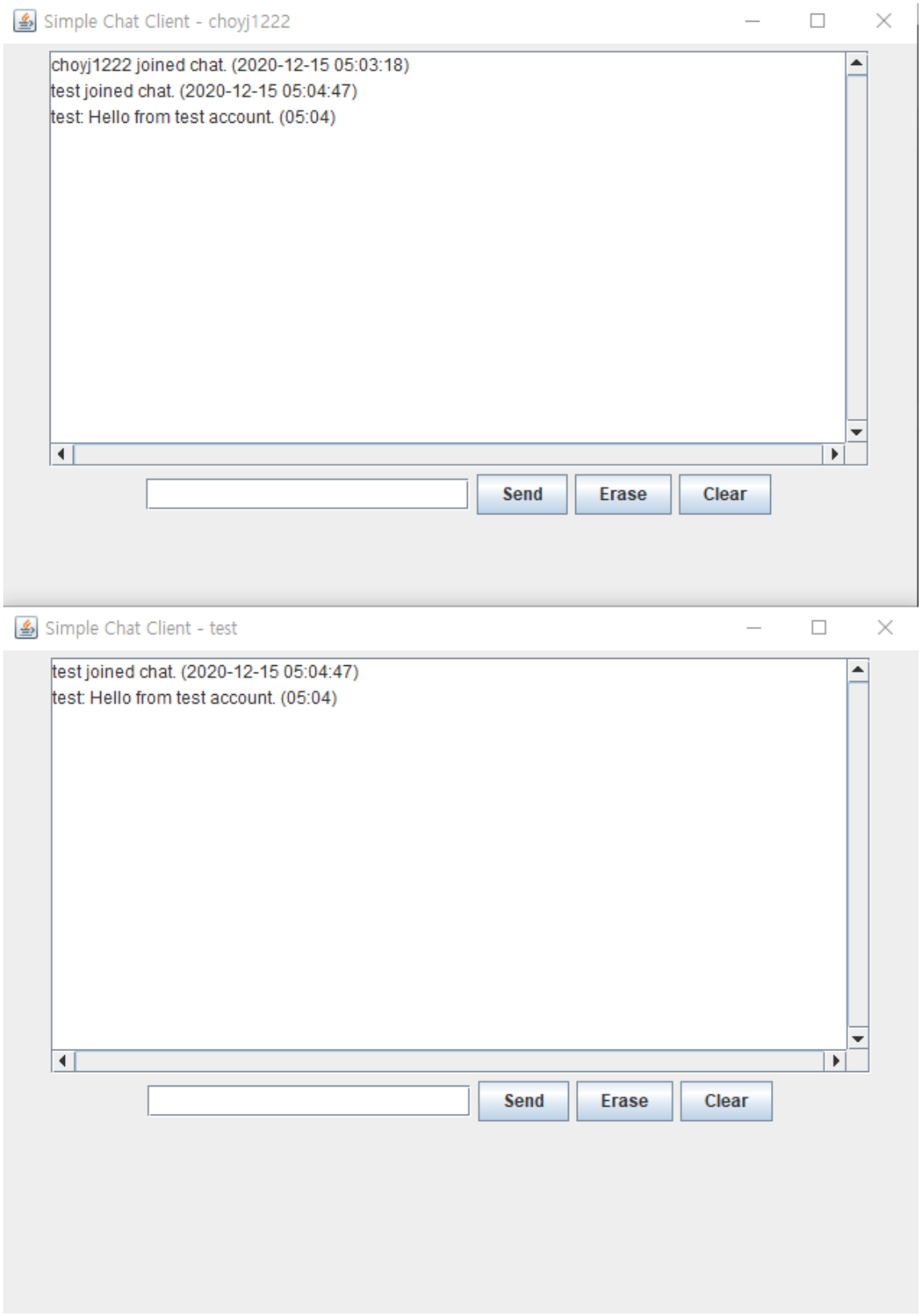


위와 같이 test 도 정상적으로 연결되고 채팅방에 들어갈 수 있었습니다.

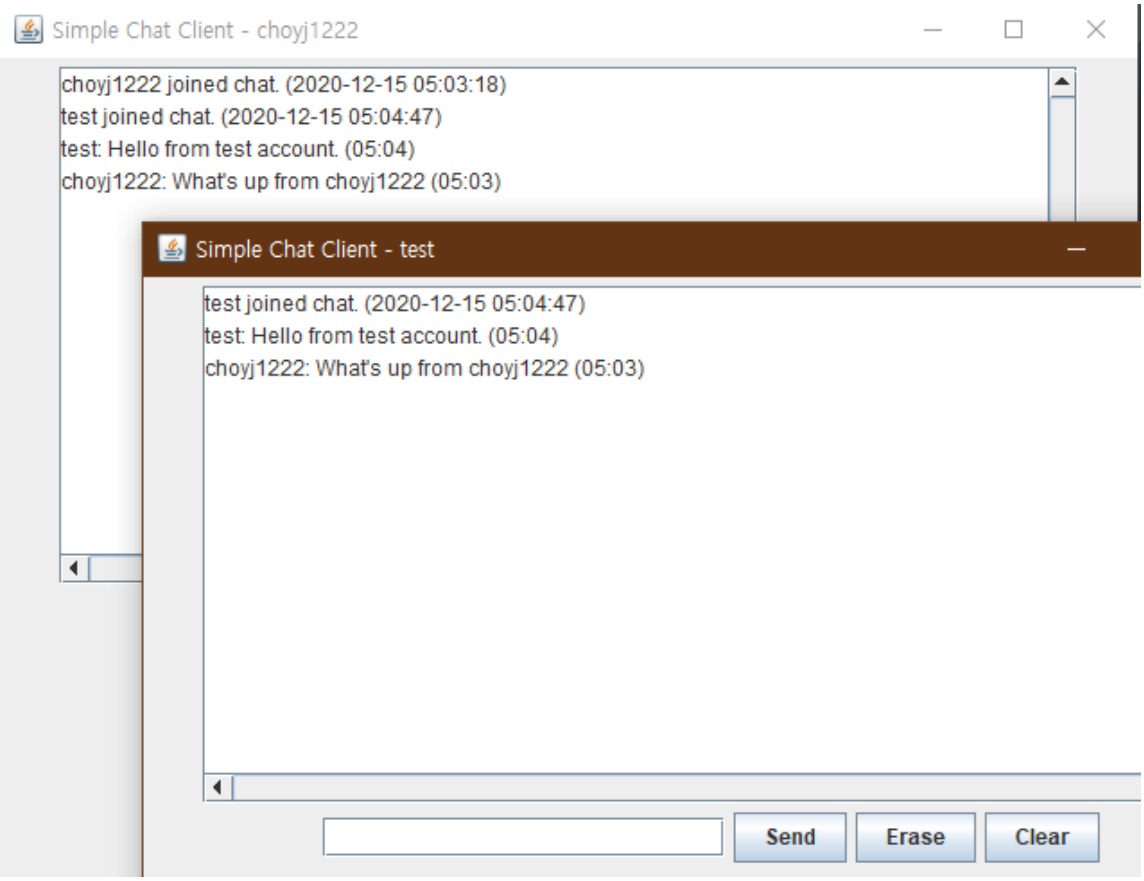
이번에는 채팅을 치면 해당 클라이언트의 ID 를 채팅창에 같이 출력하는 것을 보여드리겠습니다.



이 상태에서 Send 버튼을 누르면

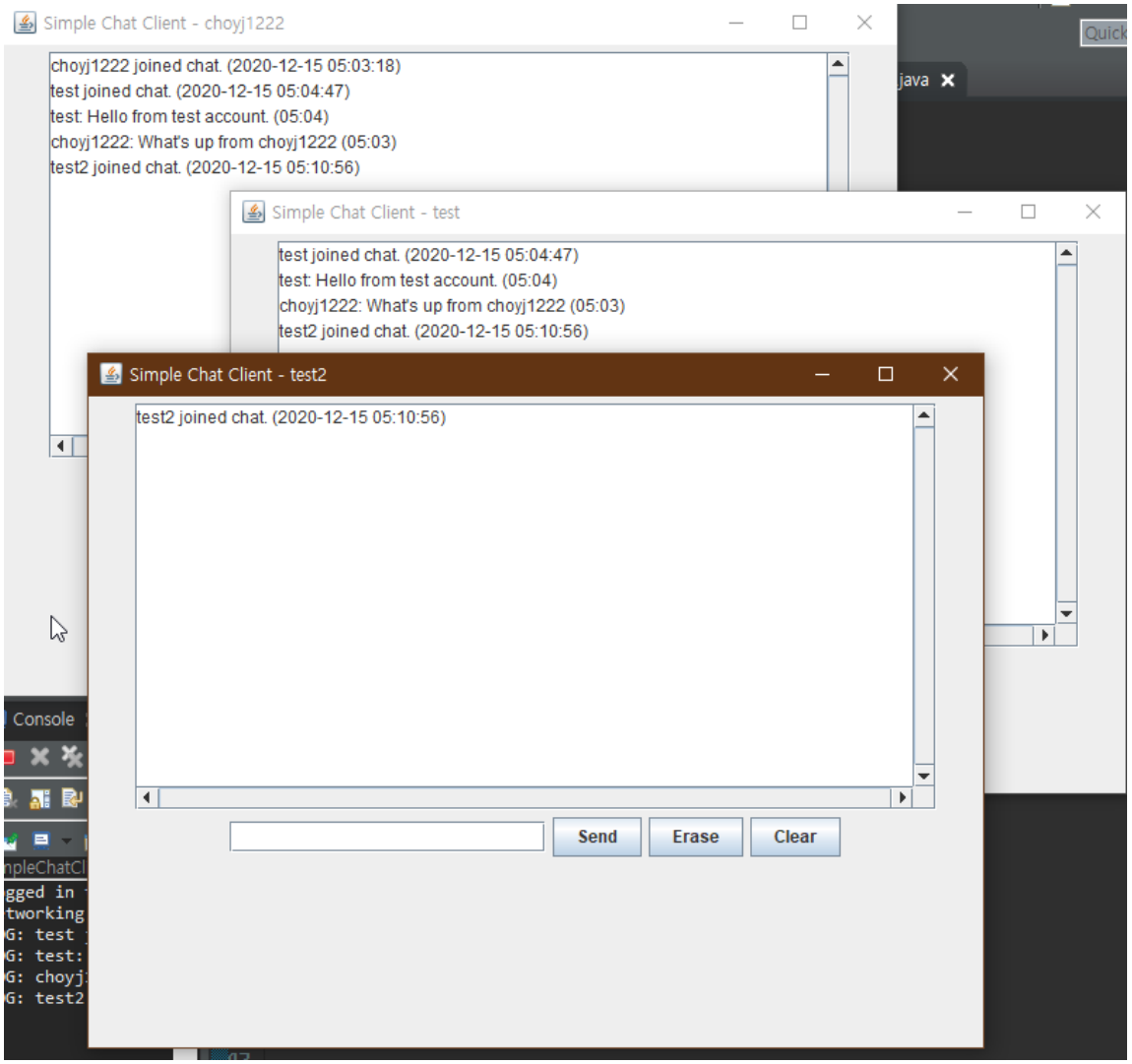


양쪽 클라이언트에서 정상적으로 출력됨을 확인할 수 있었습니다. 반대로 choyj1222 클라이언트에서 메시지를 보내보겠습니다.

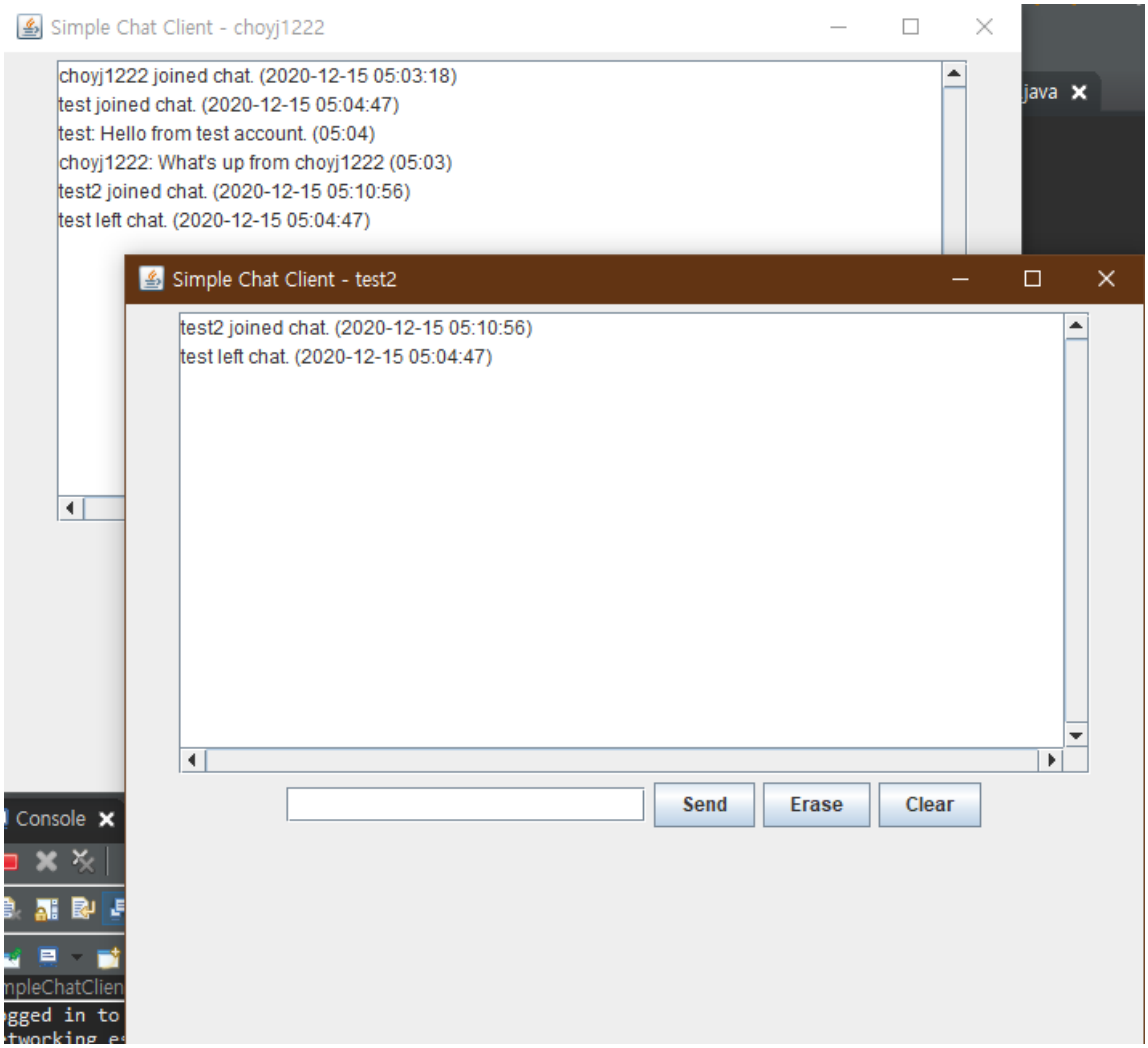


정상적으로 두 클라이언트 모두 채팅 내용과 ID 그리고 시각까지 확인할 수 있었습니다.

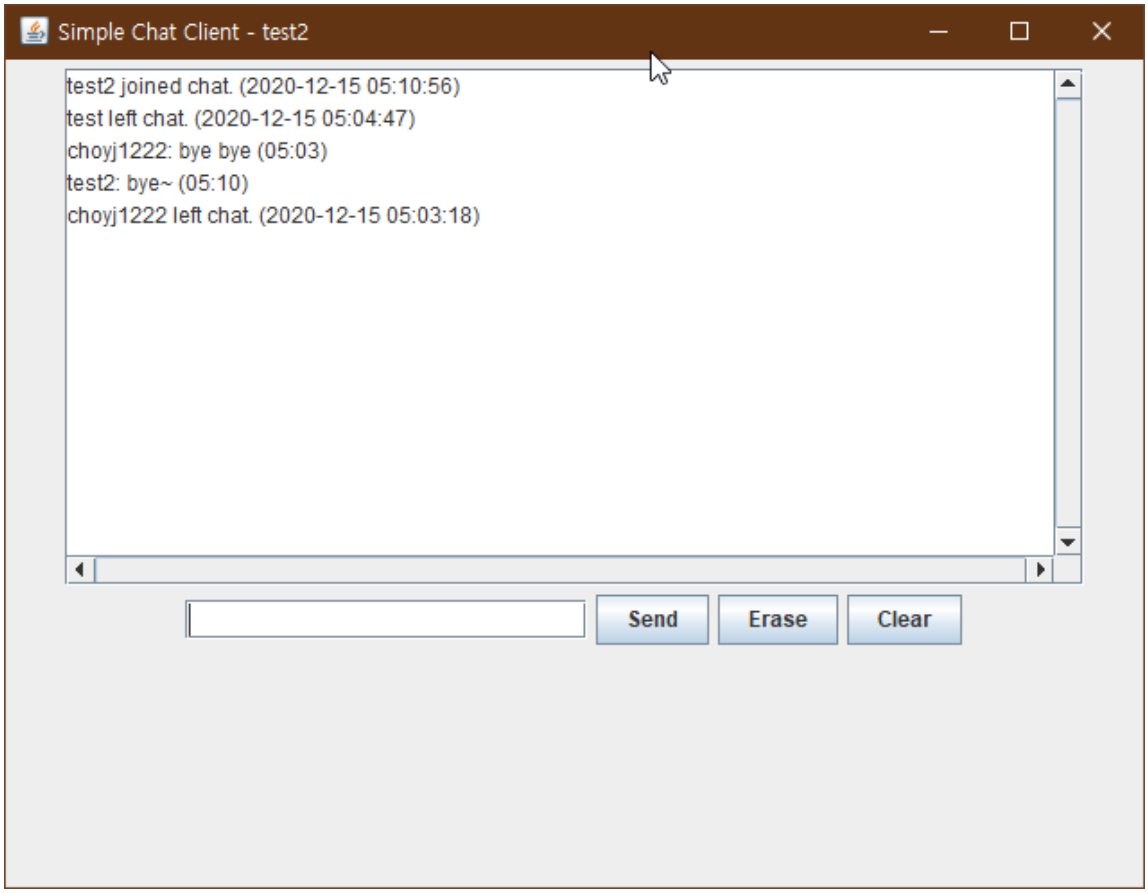
2) 채팅방 입장 , 퇴장시 채팅창에 뜨게하기



입장시 해당 userID 와 시각이 함께 출력되는 것을 확인할 수 있습니다.

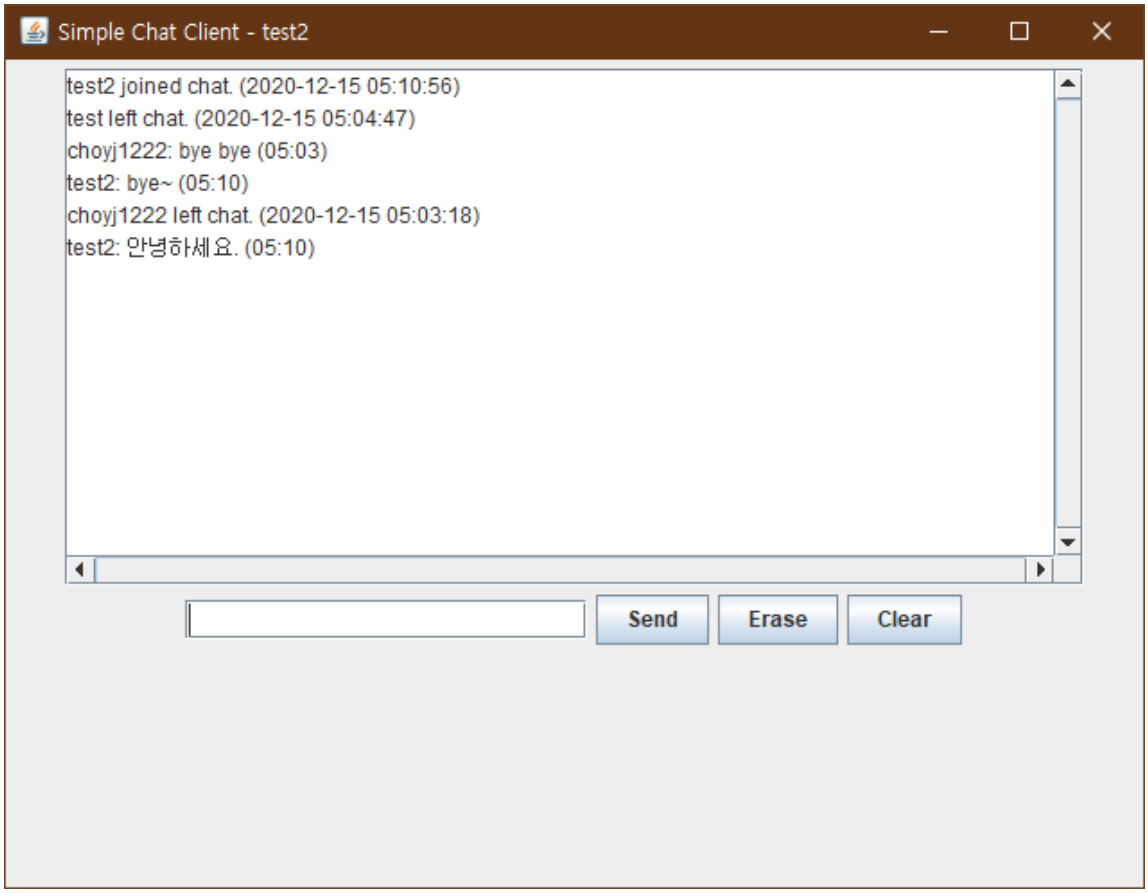


test 계정을 종료하였을 때의 퇴장 문구입니다.



choyj1222 계정이 퇴장했을 때의 퇴장 문구입니다. 정상적으로 입장, 퇴장 문구가 출력됨을 확인할 수 있습니다.

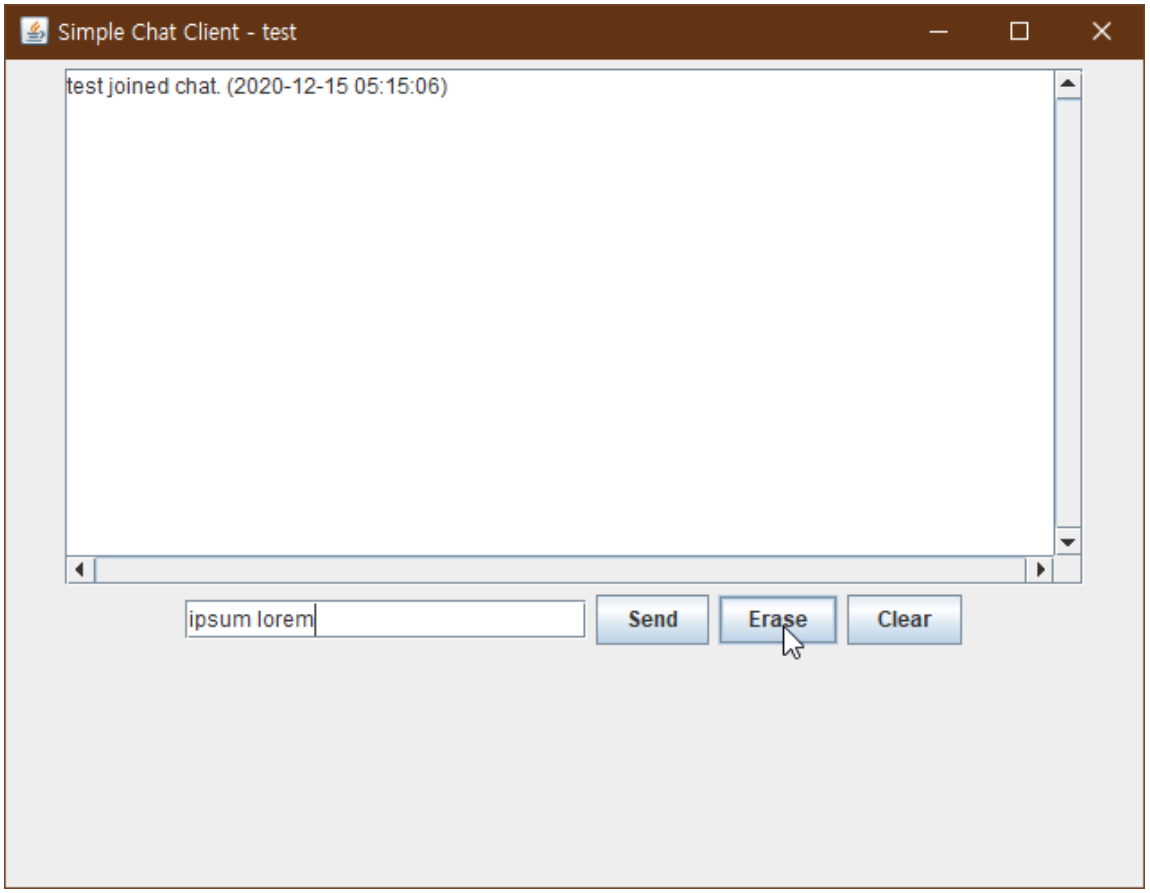
3) 채팅옆에 현재 시간 출력하기

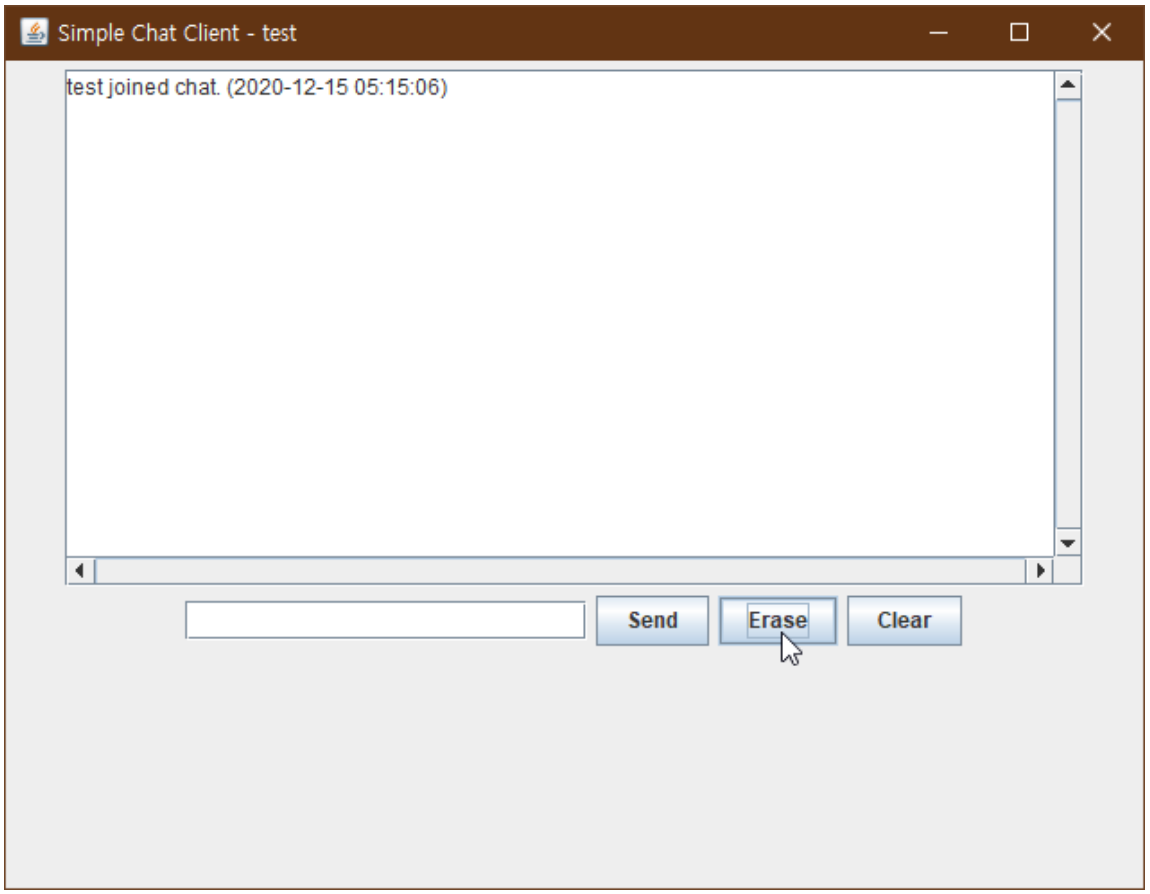


채팅창의 시각 출력은 위에서도 언급했듯이 입장, 퇴장의 경우에만 년-월-일 시:분:초의 형태, 일반적인 채팅의 경우 시:분의 형태로 채팅 옆에 출력되게 됩니다.

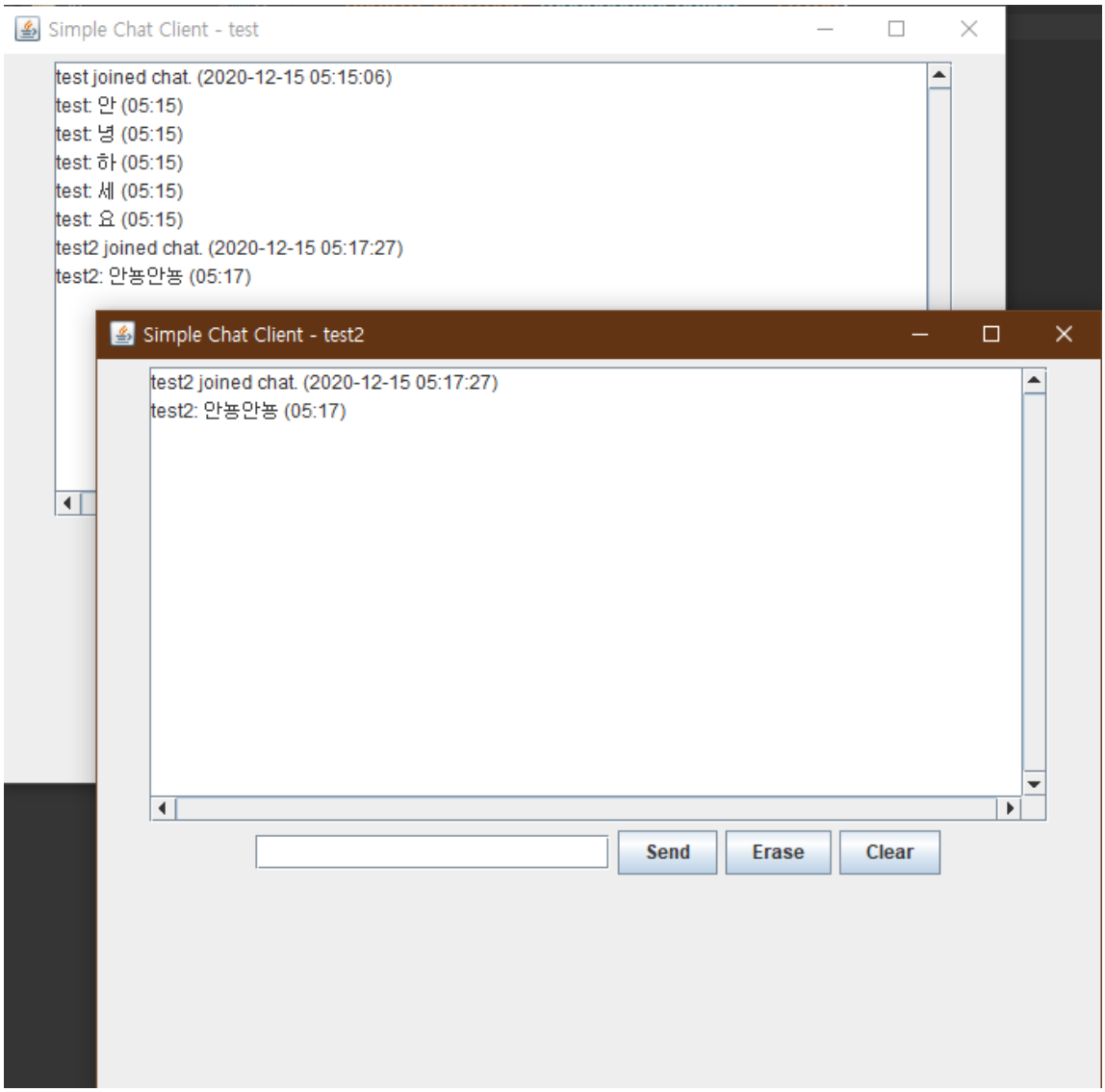
4) 그 외 기능 구현시 난이도에 따라 +0~2 점을 부여

제가 추가적으로 구현한 기능에 대해 설명드리겠습니다.

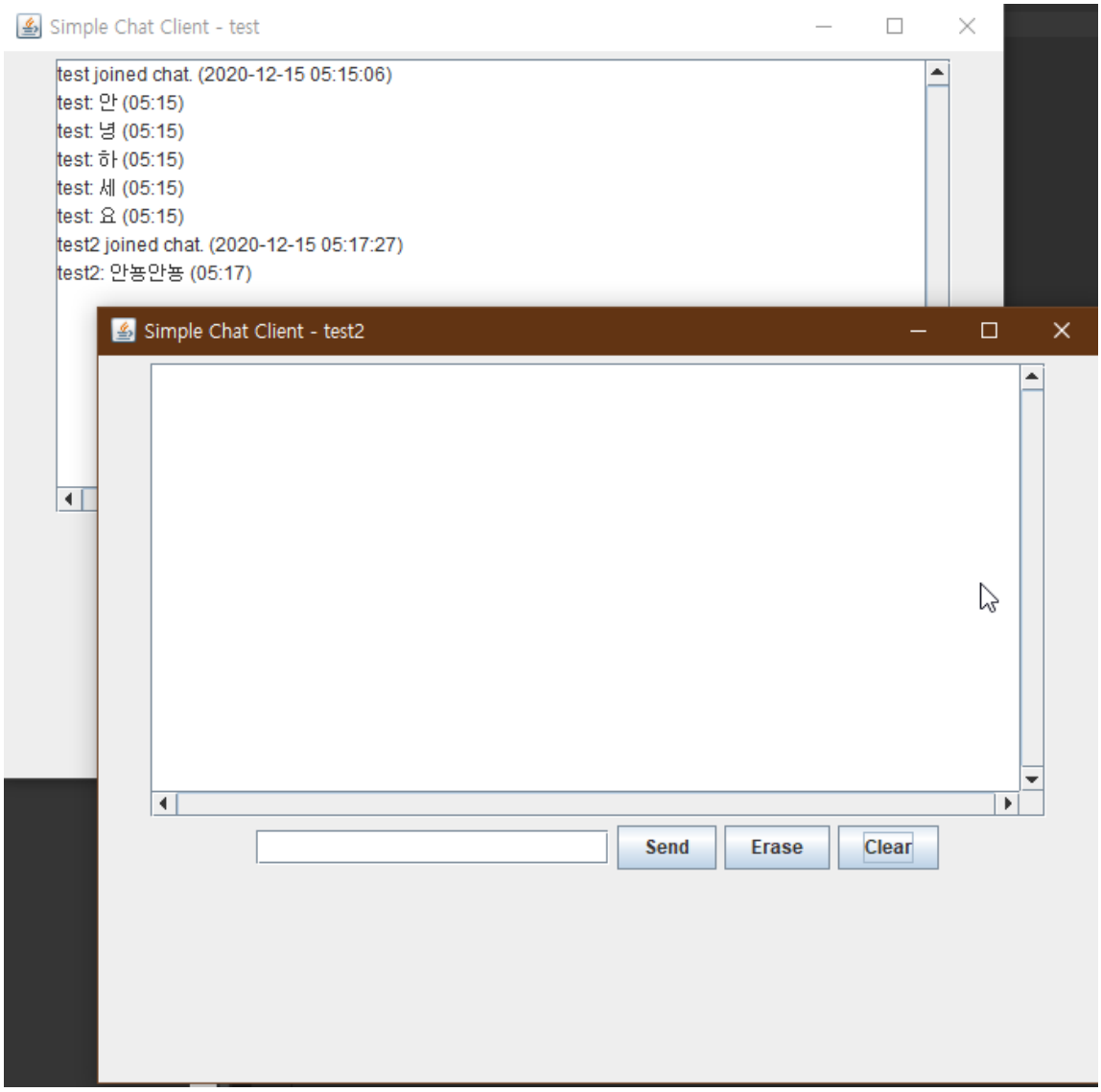

Simple Chat Client - test
test joined chat. (2020-12-15 05:15:06)
ipsum lorem
Send
Erase
Clear


Simple Chat Client - test
test joined chat. (2020-12-15 05:15:06)
Send
Erase
Clear

Erase 버튼을 누르면 해당 JTextField 에 있는 입력값이 초기화 됩니다. 제가 Erase 버튼을 누르자 작성해두었던 ipsum lorem 이라는 문구가 사라졌음을 확인할 수 있습니다.



clear 버튼은 해당 채팅창의 목록을 지울 수 있게 해줍니다. 물론 서버에 있는 데이터를 지우는 것이 아니고 local client 각각의 채팅창에만 적용됩니다. 위와 같은 상태에서 test2 클라이언트의 Clear을 누르면



test의 클라이언트에는 기록이 남아있지만 test2 클라이언트의 채팅기록은 사라졌음을 확인할 수 있습니다.

5) Future Work

이번 객체지향 프로그래밍 프로젝트: 채팅 프로그램을 진행하면서 Java 언어를 통해 로컬 웹 서버를 여는 방법과 프로그램간의 소켓 통신, 그리고 멀티쓰레딩 기법을 사용한 다중 프로세스가 작동하는 방법에 대해 공부해볼 수 있는 좋은 기회였습니다. 제가 다음번에 이 프로젝트를 다시 수정한다면 추가해보고 싶은 것이 몇 개 있습니다.

- 서버 내장 채팅봇

디스코드라는 채팅 프로그램을 자주 사용하는데, 디스코드는 파이썬 혹은 타입스크립트를 통해 서버에 자신이 만든 채팅봇을 추가할 수 있습니다. 마찬가지로 이 채팅 프로그램에도 채팅봇을 만들어 간단한 게임이나, JSOUP같은 라이브러리를 통한 웹

사이트에서 정보를 크롤링하여 재밌는 채팅봇을 만들 수 있을 것 같습니다. 예를 들어 어떤 사용자가 채팅창에 "!학식" 이라는 명령어를 입력하면 서버에 내장된 채팅 봇이 오늘의 학식을 홈페이지에서 크롤링하여 가져와 출력해주는 것입니다.

- JTextArea가 아닌 JTextPane 사용

현재 채팅 프로그램의 채팅기록 창은 JTextArea를 사용하고 있습니다. 하지만 JTextArea는 세부적인 설정을 블록마다 할 수 없습니다. JTextPane을 사용하여 사용자의 ID는 bold 처리하는 등의 세부적인 설정을 할 수 있으면 좋겠습니다.

- JTextArea 안의 채팅 기록을 txt 파일로 저장

채팅 기록을 txt 파일로 백업하여 저장할 수 있는 기능을 추가하고 싶습니다.