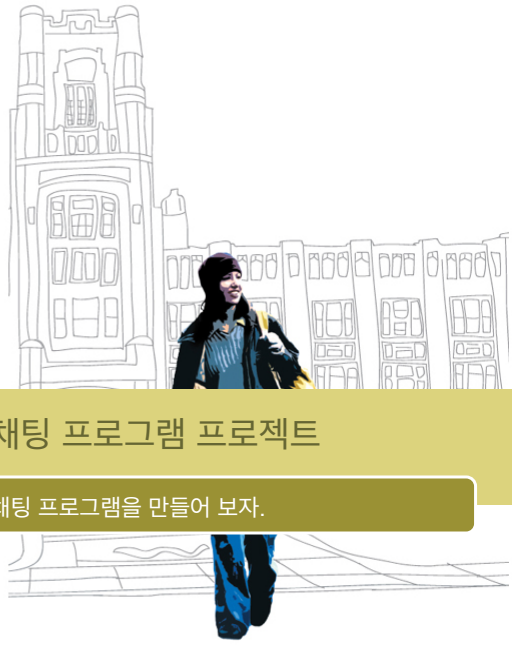


# 멀티스레드 채팅 프로그램 프로젝트

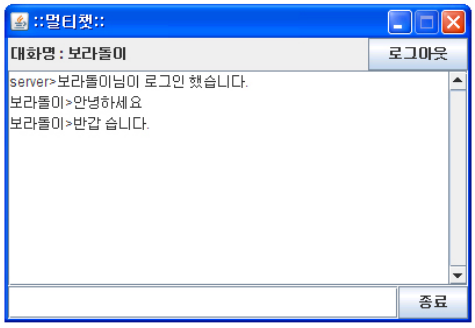
자바 채팅 프로그램을 만들어 보자.

## 목차

## 01. 프로젝트의개요

### ☐ 프로젝트 결과물과 구성요소

- 결과물
  - 다중 사용자 사이에서 동시 채팅을 지원하는 멀티스레드 채팅 프로그램으로, 간단하게 “멀티챗”이라는 이름을 사용하기로 한다.
  - 이 프로젝트의 목적
    - 실제 고급 응용 프로그램을 개발할 때 필요한 기술을 미리 경험할 수 있으므로 조금 어렵게 느껴지더라도 포기하지 말고 끝까지 완성해 본다.



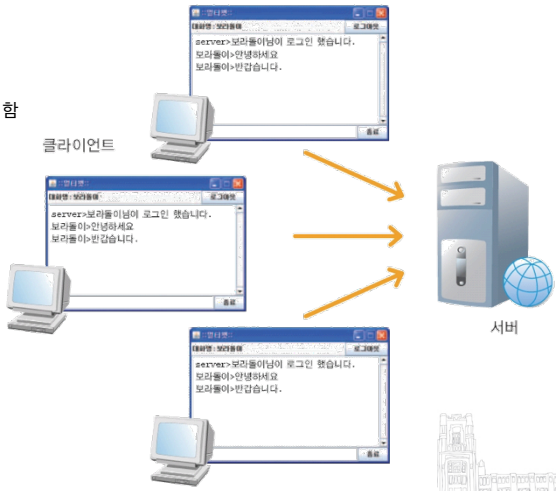

[그림 1] 클라이언트 프로그램의 실행화면



## 01. 프로젝트의개요

### ☐ 프로젝트 결과물과 구성요소

- 구성요소
  - 서버와 클라이언트로 구성
  - 개발된 프로그램을 완전하게 테스트 하려면 세 대 이상의 컴퓨터가 있어야 함
    - 여러 컴퓨터에서 클라이언트를 실행해 로그인과 로그아웃, 채팅을 동시 다발적으로 수행해 보는 게 좋다
  - 컴퓨터 한 대에서도 테스트는 가능




[그림 2] 채팅 프로그램의 구성요소와 실행환경

## 01. 프로젝트의개요

### ☐ 프로젝트 결과물과 구성요소

- 서버 프로그램
  - 채팅 메시지를 중계해 주는 프로그램. 멀티스레드 기법을 이용해 동시에 여러 클라이언트와 네트워크 연결이 가능하도록 함.
  - 모든 메시지는 연결된 모든 사용자에게 전달되는 브로드캐스팅 방식의 서버 프로그램이다.
    - 메시지 프로토콜을 만들고 서버 기능을 보완하면 좀더 기능이 다양한 서버 프로그램으로 확장이 용이
- 클라이언트 프로그램
  - 대화명을 입력하고 로그인하면 별도의 인증절차를 거치지 않아도 된다. 메시지를 입력하고 서버에서 수신되는 메시지를 출력해 주는 창으로 구성
  - 로그아웃, 로그인 가능
    - 서버 기능을 확장하면 클라이언트 역시 확장된 기능을 반영해 로그인, 귓속말 등 다양한 기능을 추가가 용이



## 01. 프로젝트의개요

### ☐ 기능 정의와 클래스 설계

프로그램을 개발할 때 설계가 차지하는 비율은 거의 절반에 가깝다고 해도 과언이 아니다.

설계가 잘되면 완성도 있는 프로그램이 나올 수 있음

- 기능 정의
  - 클라이언트와 서버로 구성

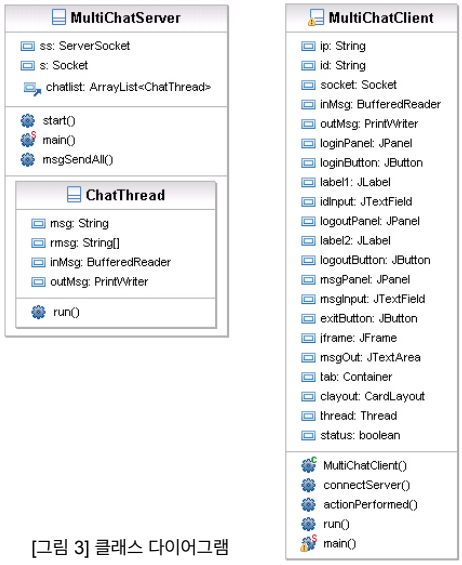
[표 1] 클라이언트와 서버의 기능 정의

| 구분 | 주요 기능 |
|---|---|
| 클라이언트 | • 로그인/로그아웃<br>• 대화명 입력/표시<br>• 채팅 메시지 출력<br>• 프로그램 종료 |
| 서버 | • 클라이언트 대기/연결<br>• 연결된 클라이언트 목록 관리<br>• 채팅 메시지 수신/전달<br>• 로그 출력 |

## 01. 프로젝트의개요

### ☐ 기능 정의와 클래스 설계

- 클래스 설계
  - MultiChatClient, MultiChatServer, ChatThread 로 구성




[그림 3] 클래스 다이어그램



## 01. 프로젝트의개요

### ☐ 기능 정의와 클래스 설계

- MultiChatClient
  - 사용자 UI를 구성하고 서버와의 연결을 만든다. 또한 서버로 보내는 메시지를 입력받고 서버에서 전송되는 메시지 역시 출력하는 기능으로 구성

[표 2] MultiChatClient의 주요 메소드

| 번호 | 메소드명 | 설명 |
|---|---|---|
| 1 | MultiChatClient(String ip) | 생성자로 ip 주소를 문자열로 받아 서버 주소를 설정하고, Swing API를 이용해 프로그램 화면을 구성한다. |
| 2 | void connectServer() | 서버 주소가 있고, 소켓 연결을 만들며, 서버와 메시지를 주고받는 입·출력 스트림을 생성한다. 또 수신 메시지를 처리하려고 스레드를 실행한다. |
| 3 | void run() | 스레드를 실행하면 자동으로 호출되는 메소드로, 상태 정보에 따라 계속 루프를 돌며 입력 스트림에서 수신된 메시지를 파싱해 메시지 창에 출력한다. |
| 4 | void actionPerformed (ActionEvent e) | 이벤트를 처리하는 메소드로, 〈로그인〉 버튼, 〈로그아웃〉 버튼, 메시지 전송, 〈종료〉 버튼 등 모든 이벤트를 발생할 때 자동으로 호출하는 메소드다. |

## 01. 프로젝트의개요

### ☐ 기능 정의와 클래스 설계

- MultiChatServer, ChatThread
  - MultiChatServer 클래스는 동시에 다중 사용자와 연결하는 스레드를 처리하려고 ChatThread 내부 클래스를 만들어 처리

[표 3] MultiChatServer의 주요 메소드

| 번호 | 메소드명 | 설명 |
|---|---|---|
| 1 | void start() | 서버의 메인 실행 메소드로, 서버 소켓을 생성하고 무한루프를 돌며 클라이언트 연결을 기다린다. 클라이언트가 연결되면 스레드를 생성하고 클라이언트 목록에 추가한다. |
| 2 | void msgSendAll(String msg) | 인자로 받은 문자열 메시지를 연결된 모든 클라이언트에 전달한다. |

[표 4] ChatThread 클래스의 주요 메소드

| 번호 | 메소드명 | 설명 |
|---|---|---|
| 1 | void run() | 스레드 클래스의 메인 메소드로, 스레드를 시작할 때 자동으로 호출되는 메소드다. 연결된 클라이언트 소켓을 이용해 입·출력 스트림을 만들어 상태 정보에 따라 루프를 돌며 메시지를 수신한다. |

9/44

## 02. 클라이언트 UI의 프로토타입 개발

### ☐ UI 구성

Swing API를 이용해 필요한 위젯을 정의하고, 패널을 이용해 레이아웃을 설정

- 화면 레이아웃 검토와 프로젝트 생성

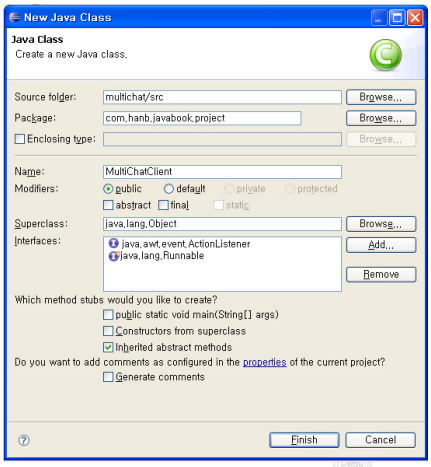
화면 레이아웃은 구현할 기능을 간단하게 그림 형식으로 그려보고 어떤 컴포넌트를 어떻게 배치할 것인지 결정하는 과정


[그림 4] 화면 레이아웃 스케치

## 02. 클라이언트 UI의 프로토타입 개발

☐ UI 구성

- 프로젝트 생성과 클래스 작성
  - 멀티챗 프로젝트를 위해 이클립스 에서 새로운자바 프로젝트를 생성
  - 프로젝트명은 multichat으로 함
  - 자바 클래스를 만들고 패키지명은 이 책에서는 com.hanb.javabook.project 로 하고, 클래스명은 MultiChat Client로 했다. (크게 상관 없음)
  - ActionListener와 Runnable 인터페이스를 구현(스레드와 이벤트 처리를 위해)




[그림 5] 클래스 생성



## 02. 클라이언트 UI의 프로토타입 개발

☐ UI 구성

- 이클립스에서는 자동으로 관련 코드를 생성
- ActionListener 인터페이스 구현에 필요한actionPerfomed 메소드와 Runnable 인터페이스 구현에 필요한 run( ) 메소드를 자동으로 생성

[ MultiChatClient.java의 기본 코드 ]

```
public class MultiChatClient implements ActionListener, Runnable {

   public void actionPerformed(ActionEvent arg0) {
   }

      public void run( ) {
      }

         public static void main(String[ ] args) {
         }
}
```

## 02. 클라이언트 UI의 프로토타입 개발

### UI 구성

- 레이아웃 결정
  - JFrame의 기본 레이아웃인 BorderLayout을 사용(위젯 컴포넌트를 효과적으로 구성하기 위한 전당한 레이아웃 결정)
  - 로그인/로그아웃 패널은 로그인 전에는 대화명을 입력한 뒤 <로그인> 버튼을 누르고, 로그인이 되면 입력한 대화명이 나타나면서 <로그인> 버튼 대신 <로그아웃> 버튼이 표시되어야 함.


[그림 6] 로그인 전과 로그인 후 패널 변경

  - 각각 loginPanel, logoutPanel을 만들어 각 위젯을 구성
  - 별도의 패널에서 CardLayout을 이용해 추가한 뒤 필요에 따라 패널을 바꿔서 보여주는 방식을 이용하기로 함
  - 메시지 패널은 메시지 입력과 <종료> 버튼이 위치하는 패널(Border Layout)



## 02. 클라이언트 UI의 프로토타입 개발

### UI 구성

- 로그인과 로그아웃 패널 구성
  - 로그인 패널 구성을 위한 컴포넌트는 다음과 같다. 멤버변수 선언 부분에서 모두 private로 선언

```
// 로그인 패널
private JPanel loginPanel;
// 로그인 버튼
private JButton loginButton;
// 대화명 라벨
private JLabel label1;
// 대화명 입력 텍스트 필드
private JTextField idInput;
```

## 02. 클라이언트 UI의 프로토타입 개발

### UI 구성

- 화면 구성은 생성자에서 처리할 것이므로 다음과 같이 생성자를 만들어 해당 코드를 입력함

```
public MultiChatClient(String ip) {

    // 로그인 패널 구성
    loginPanel = new JPanel( );
    // 레이아웃 설정
    loginPanel.setLayout(new BorderLayout( ));
    idInput = new JTextField(15);
    loginButton = new JButton("로그인");
    // 이벤트 리스너 등록
    loginButton.addActionListener(this);
    label1 = new JLabel("대화명");
    // 패널에 위젯 구성
    loginPanel.add(label1, BorderLayout.WEST);
    loginPanel.add(idInput, BorderLayout.CENTER);
    loginPanel.add(loginButton, BorderLayout.EAST);

}
```



## 02. 클라이언트 UI의 프로토타입 개발

### UI 구성

- 같은 방법으로 로그아웃 패널을 구성함

```
// 로그아웃 패널 구성
private JPanel logoutPanel;
// 대화명 출력 라벨
private JLabel label2;
// 로그아웃 버튼
private JButton logoutButton;
:
:
// 로그아웃 패널 구성
logoutPanel = new JPanel( );
// 레이아웃 설정
logoutPanel.setLayout(new BorderLayout( ));
label2 = new JLabel( );
logoutButton = new JButton("로그아웃");
// 이벤트 리스너 등록
logoutButton.addActionListener(this);
// 패널에 위젯 구성
logoutPanel.add(label2, BorderLayout.CENTER);
logoutPanel.add(logoutButton, BorderLayout.EAST);
```

## 02. 클라이언트 UI의 프로토타입 개발

### UI 구성

- 입력 패널 구성
  - 이번에는 메시지 입력 패널을 구성한다. 입력 패널은 메시지 입력 창과 <종료> 버튼으로 구성 (로그인/로그아웃 패널과 형태가 동일)
  - 메시지 입력 창은 별도의 버튼 없이 메시지를 입력한 뒤 Enter 를 치는 것으로 바로 메시지를 전송할 수 있도록 리스너를 등록

```
// 입력 패널 구성
private JPanel msgPanel;
// 메시지 입력 텍스트 필드
private JTextField msgInput;
// 종료 버튼
private JButton exitButton;
:
// 입력 패널 구성
msgPanel = new JPanel( );
// 레이아웃 설정
msgPanel.setLayout(new BorderLayout( ));
msgInput = new JTextField(30);
// 이벤트 리스너 등록
msgInput.addActionListener(this);
exitButton = new JButton("종료");
exitButton.addActionListener(this);
// 패널에 위젯 구성
msgPanel.add(msgInput, BorderLayout.CENTER);
msgPanel.add(exitButton, BorderLayout.EAST);
```



## 02. 클라이언트 UI의 프로토타입 개발

### UI 구성

- 로그인/로그아웃 카드 레이아웃 구성
  - 로그인한 뒤 화면 구성을 변화시키려고 카드 레이아웃을 사용함
    카드 레이아웃은 간단하게 패널에 이름을 부여하고 필요할 때마다 특정 패널을 보여주는 레이아웃

```
// 카드 레이아웃 관련
private Container tab;
private CardLayout clayout;
:
// 로그인/로그아웃 패널 선택을 위한 카드 레이아웃 패널
tab = new JPanel( );
clayout = new CardLayout( );
tab.setLayout(clayout);
tab.add(loginPanel, "login");
tab.add(logoutPanel, "logout");
```

  - 컨테이너 컴포넌트에 카드 레이아웃을 설정하고 컨테이너에 loginPanel과 logoutPanel을 각자 login, logout 이름으로 추가
    - 보고 싶은 패널을 화면에 나타낼 때 => clayout.show(tab,“login”);

## 02. 클라이언트 UI의 프로토타입 개발

### UI 구성

- 메인 윈도우 구성과 패널 배치
  - jFrame 윈도우를 생성하고 앞에서 만든 패널을 배치함. 이 과정에서 메시지 출력 창은 JTextArea와 JScrollPane을 이용

```
// 메인 프레임 구성
jframe = new JFrame("::멀티챗::");
msgOut = new JTextArea(" ", 10, 30);
// JTextArea의 내용을 수정하지 못하게 함. 즉, 출력 전용으로 사용
msgOut.setEditable(false);
// 수직 스크롤바는 항상 나타내고, 수평 스크롤바는 필요할 때만 나타나게 함
JScrollPane jsp = new JScrollPane(msgOut, JScrollPane.VERTICAL_SCROLLBAR_ALWAYS,
                                  JScrollPane.HORIZONTAL_SCROLLBAR_NEVER);
jframe.add(tab, BorderLayout.NORTH);
jframe.add(jsp, BorderLayout.CENTER);
jframe.add(msgPanel, BorderLayout.SOUTH);
// 로그인 패널을 우선 표시
clayout.show(tab, "login");
// 프레임 크기 자동 설정
jframe.pack( );
// 프레임 크기 조정 불가 설정
jframe.setResizable(false);
// 프레임 표시
jframe.setVisible(true);
```



## 02. 클라이언트 UI의 프로토타입 개발

### UI 구성

- 메인 메소드 구현과 테스트
  - main 메소드는 MultiChatClient 클래스를 생성하는 부분만 들어가도록 간단하게 구성

    접속할 서버의 IP 주소를 인스턴스에 전달하는 인자는 추가해 두었다.
    - 아직은 해당 인자로 전달된 IP 주소를 활용하는 부분은 다루지 않기 때문.
    - 실행을 위해 private String ip를 멤버변수 선언부에 추가해 주고, MultiChatClient(String ip) 생성자 첫 행에 this.ip = ip; 코드를 한 줄 추가한다.

```
public static void main(String[ ] args) {
    MultiChatClient mcc = new MultiChatClient("127.0.0.1");
}
```

  - 실행해보면 아무런 기능을 하지 않지만 최종 산출물과 동일한 화면을 나타냄

## 02. 클라이언트 UI의 프로토타입 개발

### ☐ 이벤트 처리

멀티챗 클라이언트에는 이벤트가 많지 않아 각 위젯에 리스너 클래스를 두지 않고 하나의 이벤트 클래스에서 모든 이벤트를 처리하도록 함

- actionPerformed 메소드 구현
  - MultiChatClient 클래스는 ActionListener 인터페이스를 구현하므로 actionPerformed ( ) 메소드를 반드시 구현. 각 위젯에서 발생하는 이벤트 처리 코드를 작성하면 된다.
    - 채팅 서버와의 연동을 위한 비즈니스 로직이 만들어지지 않은 관계로 각 이벤트를 처리하는 코드만 구성하고, 임시로 이벤트 처리를 테스트 하려고 콘솔로 메시지를 출력 하도록 구성

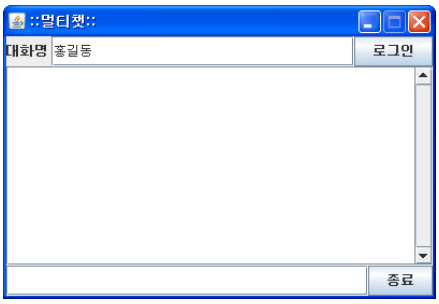
```
public void actionPerformed(ActionEvent arg0) {
  Object obj = arg0.getSource( );

  public void actionPerformed(ActionEvent arg0) {
    Object obj = arg0.getSource( );
    // 종료 버튼 처리
    if(obj == exitButton) {
      System.exit(0);
    } else if(obj == loginButton) {
      clayout.show(tab, "logout");
    } else if(obj == logoutButton) {
      clayout.show(tab, "login");
    } else if(obj == msgInput) {
      msgInput.setText(" ");
```



## 02. 클라이언트 UI의 프로토타입 개발

### ☐ 이벤트 처리

- 멀티챗 클라이언트의 UI 프로토타입과 이벤트 모델이 완성
- 이제 각 이벤트에 프로그램이 반응함을 확인할 수 있음




[그림 7] 프로토타입 실행화면

- 서버를 먼저 완성 후, 프로그램의 나머지 부분을 보강하도록 함

## 03. 서버와 클라이언트 통신 모듈 개발

☐ 멀티챗 서버 구현

- 서버 프로그램은 서비스를 제공하는 프로그램을 뜻함
- 멀티챗 서버는 다중 사용자 연결을 관리하고 서로 간의 대화를 중계해 주는 역할

- MultiChatServer 클래스 구현
  - MultiChatServer 클래스는 서버 소켓을 생성후 무한루프를 돌면서 연결된 클라이언트가 있으면 새로운 스레드 클래스인 ChatThread 클래스를 생성
  - 생성된 인스턴스를 chatlist 이름으로 ArrayList에 추가
  - 프로그램은 main 메소드에서 인스턴스를 생성하고 start( ) 메소드를 호출하는 것으로 시작
  - MultiChatServer 클래스를 구현할 때 가장 중요한 부분
    - 스레드 처리



## 03. 서버와 클라이언트 통신 모듈 개발

☐ 멀티챗 서버 구현

```
public class MultiChatServer {
    // 서버 소켓과 클라이언트 연결 소켓
    private ServerSocket ss = null;
    private Socket s = null;
    // 연결된 클라이언트 스레드를 관리하는 ArrayList
    ArrayList <ChatThread> chatlist = new ArrayList <ChatThread>( );
    // 멀티챗 메인 프로그램부
    public void start( ) {
        try {
            // 서버 소켓 생성
            ss = new ServerSocket(8888);
            System.out.println("server start");
            // 무한루프를 돌면서 클라이언트 연결을 기다림
            while(true) {
                s = ss.accept( );
                // 연결된 클라이언트에서 스레드 클래스 생성
                ChatThread chat = new ChatThread( );
                // 클라이언트 리스트 추가
                chatlist.add(chat);
                // 스레드 시작
                chat.start( );
            }
        } catch(Exception e) {
            System.out.println("[MultiChatServer]start( ) Exception 발생!!");
        }
    }
    public static void main(String[ ] args) {
        MultiChatServer server = new MultiChatServer( );
        server.start( );
    }
}
```

## 03. 서버와 클라이언트 통신 모듈 개발

☐ 멀티챗 서버 구현

- ChatThread 클래스 구현
  - 각 클라이언트를 스레드로 처리하는 클래스다. 스레드를 구현하려고 Thread 클래스를 상속하고, MultiChatServer 클래스와 더 쉽게 연결해 사용할 수 있도록 내부 클래스 형태로 구현
  - ChatThread 내부 클래스 선언
    - 앞에서 말한 것처럼 ChatThread 클래스는 내부 클래스 형태로 선언

```
public class MultiChatServer {

    ...
    ...
    public static void main(String[ ] args) {
        ...
    }

    class ChatThread extends Thread {
    }
}
```



## 03. 서버와 클라이언트 통신 모듈 개발

☐ 멀티챗 서버 구현

- ChatThread 클래스에 들어가는 내용임
  - 코드의 위치에 신경써야 함
- 관련 변수 선언
  - ChatThread 클래스에는 각 소켓 연결을 입·출력 스트림으로 생성하고 메시지를 수신하거나 중계하는 변수

```
// 수신 메시지와 파싱 메시지를 처리하는 변수 선언
String msg;
String[ ] rmsg;

// 입·출력 스트림 생성
private BufferedReader inMsg = null;
private PrintWriter outMsg = null;
```

## 03. 서버와 클라이언트 통신 모듈 개발

### ☐ 멀티챗 서버 구현

- run( ) 메소드 구현
  - 스레드 수행 메소드로, ChatThread 클래스의 메인 부분이 된다.
  - 입·출력 스트림을 생성하는 부분

```
// 입·출력 스트림 생성
inMsg = new BufferedReader(new InputStreamReader(s.getInputStream( )));
outMsg = new PrintWriter(s.getOutputStream( ), true);
```

버퍼를 flush하는 옵션, true로 설정시 매 출력마다 버퍼를 자동으로 비워 메시지를 전달

- 클라이언트와 서버 간에 사용하는 메시지 규격이다.

| ID | 구분자 | 유형/메시지 |
|---|---|---|

  - 메시지는 전달할 메시지의 내용으로 단순 문자열로 구분자인'/'를 사용
  - 메시지 형태를 정의한 것을 보통 프로토콜이라고 함



## 03. 서버와 클라이언트 통신 모듈 개발

### ☐ 멀티챗 서버 구현

- status 변수에 따라 루프를 돌면서 수신된 메시지를 처리하는 while 블록이다. 수신된 메시지 유형에 따라 몇 가지 처리 로직을 다르게 구성
- 메시지를 파싱해 문자열 배열로 자동 생성해 주는 String 클래스의 split 메소드를 사용

```
while(status) {
   // 수신된 메시지를 msg 변수에 저장
   msg = inMsg.readLine( );
   // '/' 구분자를 기준으로 메시지를 문자열 배열로 파싱
   rmsg = msg.split("/");
   ...
}
```

- rmsg
  - 문자열 배열로 수신된 메시지에서 구분자를 제외한 채 메시지, 유형의 순서로 저장

| 인덱스 | 0 | 1 |
|---|---|---|
| 값 | 홍길동 | login |

## 03. 서버와 클라이언트 통신 모듈 개발

☐ 멀티챗 서버 구현

- 메시지 유형에 따라 처리하는 블록 구분

```
    // 파싱된 문자열 배열의 두 번째 요소값에 따라 처리
    // 로그아웃 메시지일 때
    if(rmsg[1].equals("logout")) {
        chatlist.remove(this);
        msgSendAll("server/" + rmsg[0] + "님이 종료했습니다.");
        // 해당 클라이언트 스레드 종료로 status를 false로 설정
        status = false;
    }
    // 로그인 메시지일 때
    else if(rmsg[1].equals("login")) {
        msgSendAll("server/" + rmsg[0] + "님이 로그인했습니다.");
    }
    // 그 밖의 일반 메시지일 때
    else {
        msgSendAll(msg);
    }
} // while 종료
```



## 03. 서버와 클라이언트 통신 모듈 개발

☐ 멀티챗 서버 구현

- 메시지 유형에 따라 처리하는 블록 구분
- 로그아웃
  - chatlist에서 현재 스레드를 삭제하고 msgSendAll( ) 메소드를 이용해 모든 사용자에게 사용자의 종료 메시지를 전달이때, 메시지를 보내는 아이디는 server로 지정 후 status를 false
- 로그인
  - 모든 사용자에게 로그인 메시지를 전달하고, 일반 메시지일 때는 별도의 추가 작업없이 메시지만 전달
  - while 블록을 빠져나오면
  - 사용자가 종료한 것이므로 해당 스레드를 인터럽트하고 메시지를 출력

```
// 루프를 벗어나면 클라이언트 연결이 종료되므로 스레드 인터럽트됨
this.interrupt( );
System.out.println("##" + this.getName( ) + "stop!!");
```

## 03. 서버와 클라이언트 통신 모듈 개발

### ☐ 멀티챗 서버 구현

- msgSendAll( ) 메소드 구현
  - 연결된 모든 사용자에게 메시지를 전달하는 메소드
  - chatlist ArrayList에서 ChatThread 클래스 인스턴스를 가져와 출력 스트림을 이용해 메시지를 출력하는 형식으로 구성

```
// 연결된 모든 클라이언트에 메시지 중계
void msgSendAll(String msg) {
    for(ChatThread ct : chatlist) {
        ct.outMsg.println(msg);
    }
}
```



## 03. 서버와 클라이언트 통신 모듈 개발

### ☐ 클라이언트 통신 모듈 구현

- 프로토타입에 통신 모듈을 추가로 구현해 프로그램을 완성한다.
- connectServer( ) 메소드 구현
  - 멀티챗 서버에 접속해 주는 메소드로, actionPerformed( ) 메소드에서 loginButton 이벤트를 처리하는 부분에서 호출됨
  - 다음과 같이 수정한다.

```
else if(obj == loginButton) {
    id = idInput.getText( );

    label2.setText("대화명 : " + id);
    clayout.show(tab, "logout");
    connectServer( );
}
```

## 03. 서버와 클라이언트 통신 모듈 개발

### ☐ 클라이언트 통신 모듈 구현

- connectServer( ) 메소드에서는 서버 IP로 연결을 만들고 입·출력 스트림을 생성. 연결되면 서버에 ID/login 형태로 구성된 메시지를 만들어 서버에 전달. 프로그램이 실행된 상태에서 수신 메시지를 계속 처리하는 스레드를 생성하고 실행함.
  - id 는 이벤트가 발생할 때 idInput.getText( )를 이용해 가져온 값

```
public void connectServer( ) {
    try {
        // 소켓 생성
        socket = new Socket(ip, 8888);
        System.out.println("[Client]Server 연결 성공!!");
        // 입·출력 스트림 생성
        inMsg = new BufferedReader
                (new InputStreamReader(socket.getInputStream( )));
        outMsg = new PrintWriter(socket.getOutputStream( ), true);
        // 서버에 로그인 메시지 전달
        outMsg.println(id + "/" + "login");
        // 메시지 수신을 위한 스레드 생성
        thread = new Thread(this);
        thread.start( );
    } catch(Exception e) {
        System.out.println("[MultiChatServer]start( ) Exception 발생!!");
    }
}
```



## 03. 서버와 클라이언트 통신 모듈 개발

### ☐ 클라이언트 통신 모듈 구현

- run( ) 메소드 구현
  - MultiChatClient 클래스 는 Runnable 인터페이스 를 구현하므로 run( ) 메소드에서 스레드 처리를 구현해 주면 됨

```
public void run( ) {
    // 수신 메시지를 처리하는 변수
    String msg;
    String[ ] rmsg;

    // 상태 플래그
    boolean status = true;
    while(status) {
        try {
            // 메시지 수신과 파싱
            msg = inMsg.readLine( );
            rmsg = msg.split("/");
            // JTextArea에 수신된 메시지 추가
            msgOut.append(rmsg[0] + ">" + rmsg[1] + "\n");
            // 커서를 현재 대화 메시지에 표시
            msgOut.setCaretPosition(msgOut.getDocument( ).getLength( ));
        } catch(Exception e) {
            // e.printStackTrace( );
            status = false;
        }
    }
    System.out.println("[MultiChatClient]" + thread.getName( ) + " 종료됨");
}
```

## 03. 서버와 클라이언트 통신 모듈 개발

☐ 클라이언트 통신 모듈 구현

- 로그아웃과 메시지 전달 처리
  - 로그아웃과 채팅 메시지 전달과 관련된 이벤트 부분을 최종 업그레이드 함
  - 로그아웃은 서버에 ID/logout 형태의 메시지를 전송하고 대화 창을 초기화시킨 뒤 관련 리소스를 정리함 그리고 스레드를 인터럽트하는 구조. 로그인 패널로 바뀌는 부분이 포함됨

```
else if(obj == logoutButton) {
    // 로그아웃 메시지 전송
    outMsg.println(id + "/" + "logout");
    // 대화 창 클리어
    msgOut.setText(" ");
    // 로그인 패널로 전환
    clayout.show(tab, "login");
    outMsg.close( );
        try {
            inMsg.close( );
            socket.close( );
        } catch(IOException e) {
            e.printStackTrace( );
        }

    status  = false;
```

35/44

## 03. 서버와 클라이언트 통신 모듈 개발

☐ 클라이언트 통신 모듈 구현

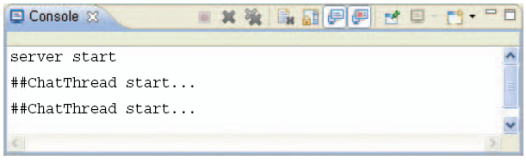
- 일반 메시지는 서버에'ID/메시지'형태의 메시지를 전송
- 메시지 입력 창을 초기화해 새로운 메시지를 입력할 수 있도록 함

```
else if(obj == msgInput) {
    // 메시지 전송
    outMsg.println(id + "/" + msgInput.getText( ));
    // 입력 창 클리어
    msgInput.setText(" ");
}
```

36/44

## 03. 서버와 클라이언트 통신 모듈 개발

- 프로그램 완성과 통합 테스트
  - 가능한 여러 PC에 클라이언트를 설치해 테스트하는 게 좋음
  - 서버는 별도의 UI가 없으므로 실행하면 콘솔에 메시지만 출력됨




[그림 8] 서버 실행화면

  - 서버의 IP 주소를 적어 놓고 클라이언트 프로그램을 서로 다른 PC에 설치한 뒤 서버의 IP 주소를 main 메소드에서 클래스 인스턴스를 생성할 때 인자로 전달하도록 함
    (사정상 PC를 여러 대 확보하기가 어려울 때는 서버의 IP주소를 localhost인 127.0.0.1로 사용)
  - 각 클라이언트에서 로그인과 로그아웃을 반복하면서 메시지를 자유롭게 주고받을 수 있으면 정상적으로 구현 완료



## 04. 전체 소스

**기본예제 1** 클라이언트 UI 프로토타입 MultiChatClient.java

```
001 package javabook.ch14;
002 import java.awt.BorderLayout;
003 import java.awt.CardLayout;
004 import java.awt.Container;
005 import java.awt.event.ActionEvent;
006 import java.awt.event.ActionListener;
007 import java.io.BufferedReader;
008 import java.io.IOException;
009 import java.io.InputStreamReader;
010 import java.io.PrintWriter;
011 import java.net.Socket;
012
013 import javax.swing.JButton;
014 import javax.swing.JFrame;
015 import javax.swing.JLabel;
016 import javax.swing.JPanel;
017 import javax.swing.JScrollPane;
018 import javax.swing.JTextArea;
019 import javax.swing.JTextField;
020
021 public class MultiChatClient implements ActionListener, Runnable {
022   private String ip;
023   private String id;
024   private Socket socket;
025   private BufferedReader inMsg = null;
026   private PrintWriter outMsg = null;
027
028   // 로그인 패널
029   private JPanel loginPanel;
030   // 로그인 버튼
031   private JButton loginButton;
032   // 대화명 라벨
033   private JLabel label1;
034   // 대화명 입력 텍스트 필드
035   private JTextField idInput;
036
037   // 로그아웃 패널 구성
038   private JPanel logoutPanel;
039   // 대화명 출력 라벨
040   private JLabel label2;
041   // 로그아웃 버튼
042   private JButton logoutButton;
043
044   // 입력 패널 구성
045   private JPanel msgPanel;
046   // 메시지 입력 텍스트 필드
047   private JTextField msgInput;
048   // 종료 버튼
049   private JButton exitButton;
050
051   // 메인 윈도우
052   private JFrame jframe;
053   // 채팅 내용 출력 창
054   private JTextArea msgOut;
055
056   // 카드 레이아웃 관련
```

## 04. 전체 소스

**기본예제 1** 클라이언트 UI 프로토타입 — MultiChatClient.java

```
057    private Container tab;
058    private CardLayout clayout;
059    private Thread thread;
060
061    // 상태 플래그
062    boolean status;
063
064    public MultiChatClient(String ip) {
065      this.ip = ip;
066
067      // 로그인 패널 구성
068      loginPanel = new JPanel( );
069      // 레이아웃 설정
070      loginPanel.setLayout(new BorderLayout( ));
071      idInput = new JTextField(15);
072      loginButton = new JButton("로그인");
073      // 이벤트 리스너 등록
074      loginButton.addActionListener(this);
075      label1 = new JLabel("대화명");
076      // 패널에 위젯 구성
077      loginPanel.add(label1, BorderLayout.WEST);
078      loginPanel.add(idInput, BorderLayout.CENTER);
079      loginPanel.add(loginButton, BorderLayout.EAST);
080
081      // 로그아웃 패널 구성
082      logoutPanel = new JPanel( );
083      // 레이아웃 설정
084      logoutPanel.setLayout(new BorderLayout( ));
085      label2 = new JLabel( );
086      logoutButton = new JButton("로그아웃");
087      // 이벤트 리스너 등록
088      logoutButton.addActionListener(this);
089      // 패널에 위젯 구성
090      logoutPanel.add(label2, BorderLayout.CENTER);
091      logoutPanel.add(logoutButton, BorderLayout.EAST);
092
093      // 입력 패널 구성
094      msgPanel = new JPanel( );
095      // 레이아웃 설정
096      msgPanel.setLayout(new BorderLayout( ));
097      msgInput = new JTextField(30);
098      // 이벤트 리스너 등록
099      msgInput.addActionListener(this);
100      exitButton = new JButton("종료");
101      exitButton.addActionListener(this);
102      // 패널에 위젯 구성
103      msgPanel.add(msgInput, BorderLayout.CENTER);
104      msgPanel.add(exitButton, BorderLayout.EAST);
105
106    // 로그인/로그아웃 패널 선택을 위한 카드 레이아웃 패널
107      tab = new JPanel( );
108      clayout = new CardLayout( );
109      tab.setLayout(clayout);
110      tab.add(loginPanel, "login");
111      tab.add(logoutPanel, "logout");
112
```

39/44

## 04. 전체 소스

**기본예제 1** 클라이언트 UI 프로토타입 — MultiChatClient.java

```
113    // 메인 프레임 구성
114    jframe = new JFrame("::멀티챗::");
115    msgOut = new JTextArea(" ", 10, 30);
116    // JTextArea의 내용을 수정하지 못하게 함. 즉, 출력 전용으로 사용
117    msgOut.setEditable(false);
118    // 수직스크롤바는 항상 나타내고, 수평스크롤바는 필요할 때만 나타나게 함
119    JScrollPane jsp = new JScrollPane(msgOut,
           JScrollPane.VERTICAL_SCROLLBAR_ALWAYS,
           JScrollPane.HORIZONTAL_SCROLLBAR_NEVER);
120    jframe.add(tab, BorderLayout.NORTH);
121    jframe.add(jsp, BorderLayout.CENTER);
122    jframe.add(msgPanel, BorderLayout.SOUTH);
123    // 로그인 패널을 우선 표시
124    clayout.show(tab, "login");
125    // 프레임 크기 자동 설정
126    jframe.pack( );
127    // 프레임 크기 조정 불가 설정
128    jframe.setResizable(false);
129    // 프레임 표시
130    jframe.setVisible(true);
131
132  }
133
134  public void connectServer( ) {
135    try {
136      // 소켓 생성
137      socket = new Socket(ip, 8888);
138      System.out.println("[Client]Server 연결 성공!!");
139
140      // 입·출력 스트림 생성
141      inMsg = new BufferedReader
             (new InputStreamReader(socket.getInputStream( )));
142      outMsg = new PrintWriter(socket.getOutputStream( ), true);
143
144      // 서버에 로그인 메시지 전달
145      outMsg.println(id + "/" + "login");
146
147      // 메시지 수신을 위한 스레드 생성
148      thread = new Thread(this);
149      thread.start( );
150    } catch(Exception e) {
151      // e.printStackTrace( );
152      System.out.println("[MultiChatClient]connectServer( )Exp발생!");
153    }
154  }
155
156  // 이벤트 처리
157  public void actionPerformed(ActionEvent arg0) {
158    Object obj = arg0.getSource( );
159
160    // 종료 버튼 처리
161    if(obj == exitButton) {
162      System.exit(0);
163    } else if(obj == loginButton) {
164      id = idInput.getText( );
165
```

40/44

## 04. 전체 소스

**기본예제 1** 클라이언트 UI 프로토타입 — MultiChatClient.java

```
166         label2.setText("대화명 : " + id);
167         clayout.show(tab, "logout");
168         connectServer( );
169       } else if(obj == logoutButton) {
170         // 로그아웃 메시지 전송
171         outMsg.println(id + "/" + "logout");
172         // 대화 창 클리어
173         msgOut.setText(" ");
174         // 로그인 패널로 전환
175         clayout.show(tab, "login");
176         outMsg.close( );
177         try {
178           inMsg.close( );
179           socket.close( );
180         } catch(IOException e) {
181           e.printStackTrace( );
182         }
183
184         status = false;
185       } else if(obj == msgInput) {
186         // 메시지 전송
187         outMsg.println(id + "/" + msgInput.getText( ));
188         // 입력 창 클리어
189         msgInput.setText(" ");
190       }
191     }
192
193     public void run( ) {
194       // 수신 메시지를 처리하는 변수
195       String msg;
196       String[ ] rmsg;
197
198       status = true;
199       while(status) {
200         try {
201           // 메시지 수신과 파싱
202           msg = inMsg.readLine( );
203           rmsg = msg.split("/");
204
205           // JTextArea에 수신된 메시지 추가
206           msgOut.append(rmsg[0] + ">" + rmsg[1] + "\n");
207
208           // 커서를 현재 대화 메시지에 표시
209           msgOut.setCaretPosition(msgOut.getDocument( ).getLength( ));
210         } catch(IOException e) {
211           // e.printStackTrace( );
212           status = false;
213         }
214       }
215       System.out.println("[MultiChatClient]" + thread.getName( ) + "종료됨");
216     }
217
218     public static void main(String[ ] args) {
219       MultiChatClient mcc = new MultiChatClient("127.0.0.1");
220     }
221   }
223
```

41/44

## 04. 전체 소스

**기본예제 2** 서버와 클라이언트 간의 통신 모듈 — MultiChatServer.java

```
001 package javabook.ch14;
002 import java.io.BufferedReader;
003 import java.io.IOException;
004 import java.io.InputStreamReader;
005 import java.io.PrintWriter;
006 import java.net.ServerSocket;
007 import java.net.Socket;
008 import java.util.ArrayList;
009
010 public class MultiChatServer {
011
012   // 서버 소켓과 클라이언트 연결 소켓
013   private ServerSocket ss = null;
014   private Socket s = null;
015
016   // 연결된 클라이언트 스레드를 관리하는 ArrayList
017   ArrayList <ChatThread> chatlist = new ArrayList
                                        <ChatThread>( );
018
019   // 멀티챗 메인 프로그램부
020   public void start( ) {
021     try {
022       // 서버 소켓 생성
023       ss = new ServerSocket(8888);
024       System.out.println("server start");
025
026       // 무한루프를 돌면서 클라이언트 연결을 기다림
027       while(true) {
028         s = ss.accept( );
029         // 연결된 클라이언트에서 스레드 클래스 생성
030         ChatThread chat = new ChatThread( );
031         // 클라이언트 리스트 추가
032         chatlist.add(chat);
033         // 스레드 시작
034         chat.start( );
035       }
036     } catch(Exception e) {
037       // System.out.println(e);
038       System.out.println("[MultiChatServer]start( )Exp 발생!!");
039     }
040   }
041
042   public static void main(String[ ] args) {
043     MultiChatServer server = new MultiChatServer( );
044     server.start( );
045   }
046
047   // 연결된 모든 클라이언트에 메시지 중계
048   void msgSendAll(String msg) {
049     for(ChatThread ct : chatlist) {
050       ct.outMsg.println(msg);
051     }
052   }
053
054   // 각 클라이언트 관리를 위한 스레드 클래스
055   class ChatThread extends Thread {
```

42/44

## 04. 전체 소스

**기본예제 2** 서버와 클라이언트 간의 통신 모듈 — MultiChatServer.java

```
056
057     // 수신 메시지와 파싱 메시지 처리하는 변수 선언
058     String msg;
059     String[ ] rmsg;
060
061     // 입·출력 스트림 생성
062     private BufferedReader inMsg = null;
063     private PrintWriter outMsg = null;
064
065     public void run( ) {
066
067       boolean status = true;
068       System.out.println("##ChatThread start...");
069       try {
070         // 입·출력 스트림 생성
071         inMsg = new BufferedReader
                (new InputStreamReader(s.getInputStream( )));
072         outMsg = new PrintWriter(s.getOutputStream( ), true);
073
074         // 상태정보가 true면 루프를 돌면서 사용자한테서 수신된 메시지
075       처리
076         while(status) {
077           // 수신된 메시지를 msg 변수에 저장
078           msg = inMsg.readLine( );
079           // '/' 구분자를 기준으로 메시지를 문자열 배열로 파싱
080           rmsg = msg.split("/");
081
082           // 파싱된 문자열 배열의 두 번째 요소값에 따라 처리
              // 로그아웃 메시지일 때
083           if(rmsg[1].equals("logout")) {
084             chatlist.remove(this);
085             msgSendAll("server/" + rmsg[0] + "님이 종료했습니다.");
086             // 해당 클라이언트 스레드 종료로 status를 false로 설정
087             status = false;
088           }
089           // 로그인 메시지일 때
090           else if(rmsg[1].equals("login")) {
091             msgSendAll("server/" + rmsg[0] + "님이 로그인했습니다.");
092           }
093           // 그 밖의 일반 메시지일 때
094           else {
095             msgSendAll(msg);
096           }
097         } // while 종료
098   // 루프를 벗어나면 클라이언트 연결이 종료되므로 스레드 인터럽트됨
099         this.interrupt( );
100         System.out.println("##" + this.getName( ) + "stop!!");
101       } catch(IOException e) {
102         chatlist.remove(this);
103         // e.printStackTrace( );
104         System.out.println("[ChatThread]run( ) IOException 발생!!");
105       }
106     }
107   }
108
109 }
```

43/44

# Thank You !