

## **Chương 9**

# **Xây dựng ứng dụng mạng : phần Client**

### **9.0 Dẫn nhập**

### **9.1 Tổng quát về lập trình mạng trên Internet**

### **9.2 Các bước hoạt động điển hình của ứng dụng client**

### **9.3 Thí dụ về ứng dụng mạng cơ bản**

### **9.4 Các điểm chính về lập trình ứng dụng client**

### **9.5 Kết chương**



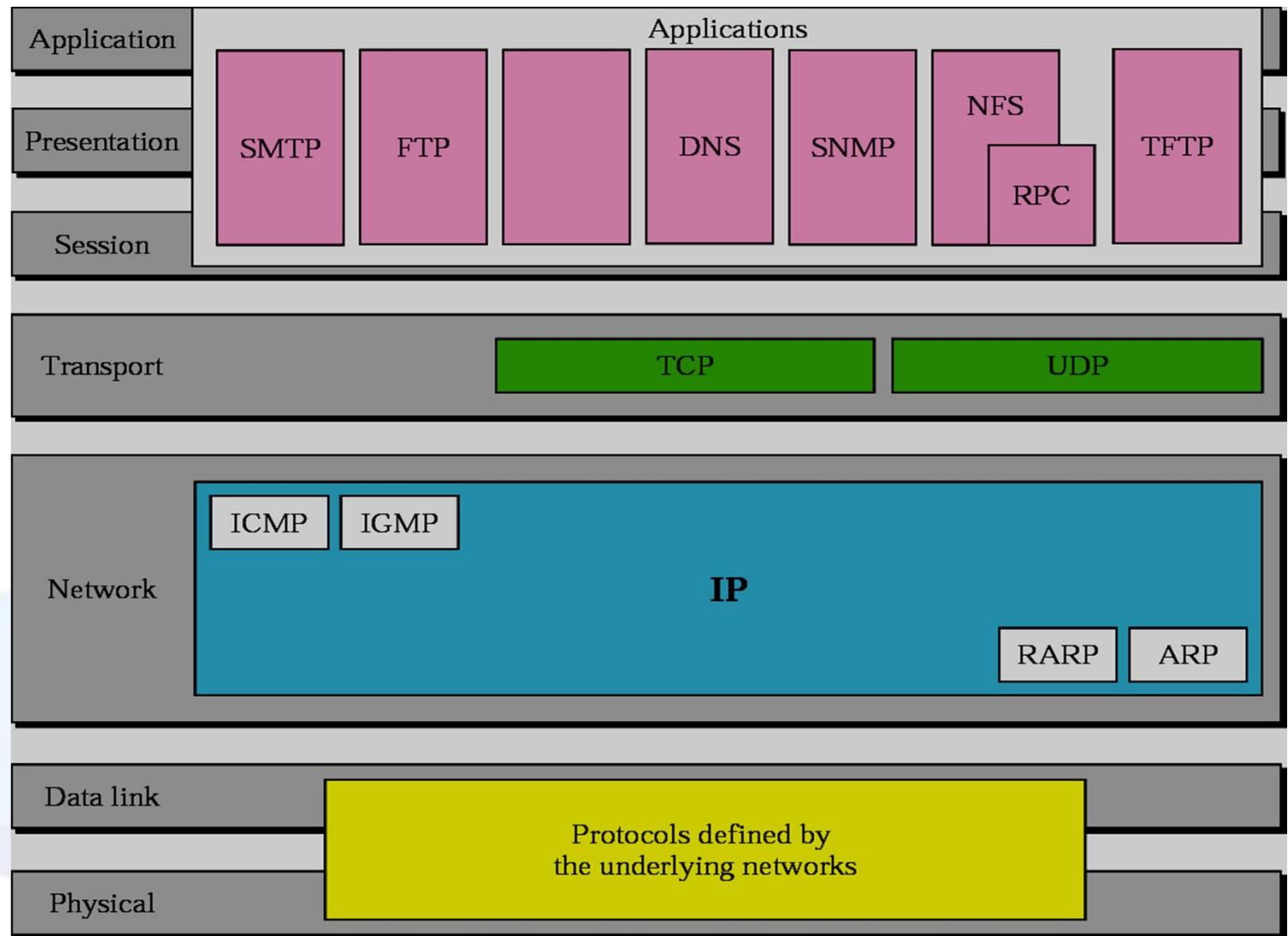
## 9.0 Dẫn nhập

- ❑ Chương này sẽ trình bày kiến trúc hoạt động của mạng Internet, cơ chế hoạt động của module client và server để chúng giao tiếp tốt với nhau.
- ❑ Chương này cũng sẽ giới thiệu qui trình làm việc điển hình của module client và cách lập trình miêu tả từng bước hoạt động này.
- ❑ Chương này cũng sẽ giới thiệu cách xây dựng module client của ứng dụng Minichat cùng các đoạn code thực hiện các công việc chính yếu của module client này.



# 9.1 Tổng quát về lập trình mạng trên Internet

So sánh kiến trúc mạng Internet và mạng chuẩn hóa ISO



## 9.1 Tổng quát về lập trình mạng trên Internet

- ❑ Xem kiến trúc của mạng Internet trong slide trước, ta thấy việc lập trình ứng dụng sẽ dựa vào 1 trong 2 giao thức TCP hay UDP của cấp TCP.
- ❑ Giao thức TCP dùng cầu nối nên rất tin cậy (không mất, không sai, không thay đổi thứ tự truyền/nhận).
- ❑ Giao thức UDP không dùng cầu nối nên không tin cậy, code của ứng dụng cần kiểm soát lỗi trong quá trình gửi/nhận thông tin (nếu muốn).
- ❑ Hiện trên các platform khác nhau, người ta cung cấp giao tiếp lập trình của thư viện socket để lập trình trên cấp TCP. Thư viện socket trên JDK được đóng gói trong package `java.net.*`, trong đó class thiết yếu nhất có tên là `Socket`, đối tượng thuộc class này quản lý 1 cổng giao tiếp (port) của module phần mềm mạng.



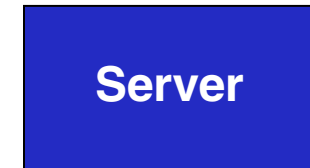
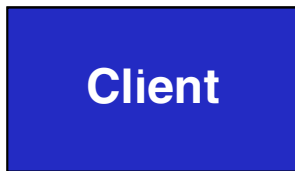
## 9.1 Tổng quát về lập trình mạng trên Internet

- ❑ Trong phần còn lại của chương này, chúng ta sẽ trình bày chi tiết về các tác vụ cơ bản của class Socket và cách sử dụng chúng để lập trình 1 ứng dụng mạng nhỏ.
- ❑ Các thông tin của chương này có thể được áp dụng trên các platform khác với sự thay đổi nhỏ (do có sự khác biệt nhỏ giữa các thư viện socket trên các platform khác nhau).

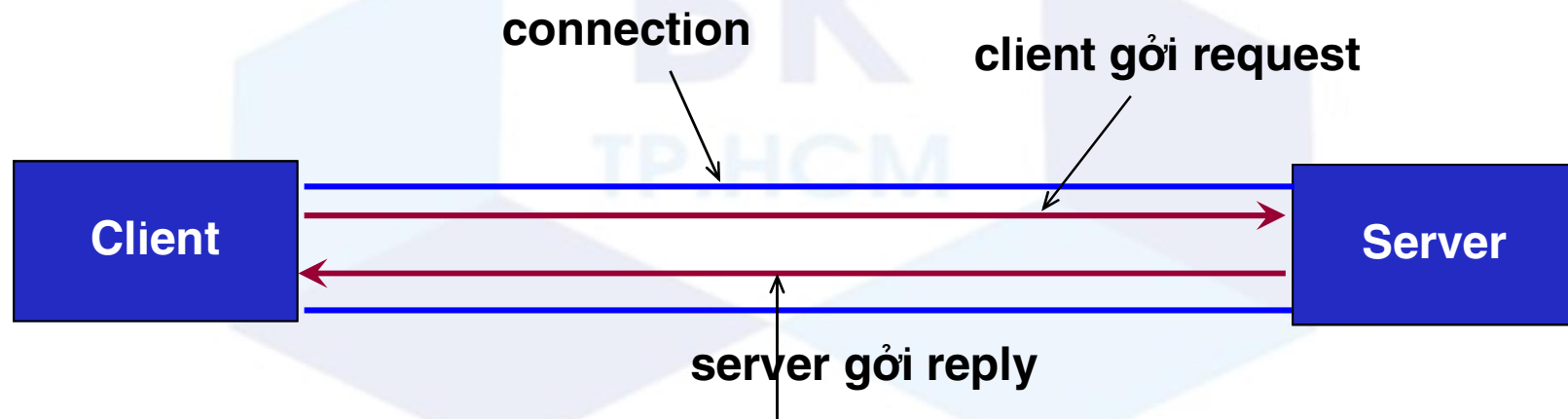


## 9.1 Tổng quát về lập trình mạng trên Internet

Hai ứng dụng client/server lúc còn độc lập nhau



Hai ứng dụng client/server lúc giao tiếp nhau (dùng cầu nối TCP và giao thức request/reply)



## 9.1 Tổng quát về lập trình mạng trên Internet

- ❑ Trong slide trước, ta thấy 2 module client/server sẽ giao tiếp với nhau thông qua 1 connection (cầu nối). Trên cầu nối này, client có quyền gửi bất kỳ thông tin gì (chuỗi bit/byte) đến server bất kỳ khi nào nó muốn. Tương tự, server cũng có quyền gửi bất kỳ thông tin gì về client bất kỳ khi nào nó muốn. Tuy nhiên để kiểm soát trật tự của việc gửi/nhận và để 2 module client/server thông hiểu lẫn nhau, chúng cần tuân thủ 1 giao thức làm việc rõ ràng. Giao thức của phần mềm mạng thường là dạng request/reply, cụ thể giao thức request/reply bao gồm :
  - tập các thông báo request và định dạng cụ thể của chúng mà client có quyền dùng để gửi đến server.
  - tập các thông báo reply và định dạng cụ thể của chúng mà server sẽ xây dựng và gửi về client sau khi thực hiện request tương ứng.



## 9.1 Tổng quát về lập trình mạng trên Internet

### Client

tạo Socket

cho Socket địa chỉ TCP của server để nó connect tới

gửi thông báo TCP yêu cầu tạo cầu nối

gửi thông báo TCP chấp nhận tạo cầu nối

send request yêu cầu 1 chức năng

chờ nhận reply chứa kết quả và sử dụng

đóng Socket

### Server

tạo Socket

thiết lập địa chỉ TCP của port giao tiếp

chờ và chấp nhận kết nối từ Client

chờ nhận request và xử lý

send reply chứa kết quả

đóng Socket





## 9.2 Các bước hoạt động điển hình của ứng dụng client

□ Theo mô hình giao tiếp giữa module client và module server ở slide trước thì module client cần thực hiện các thao tác sau để nối kết đến server hầu gửi thông báo request nhờ server thực hiện 1 chức năng nào đó :

1. tạo đối tượng Socket để quản lý port giao tiếp của mình

```
Socket socket = new Socket();
```

2. thiết lập địa chỉ TCP của server để đối tượng Socket biết và cố gắng yêu cầu nối kết đến server. Thường trong lập trình Java, ta thực hiện bước 2 này luôn trong bước 1 :

```
socket = new Socket("vnn.vn", 80);
```

Câu lệnh trên sẽ tạo đối tượng socket, thiết lập địa chỉ TCP của server là "vnn.vn":80 để nó cố gắng nối kết đến server này. Nếu thành công, ta qua bước 3.



## 9.2 Các bước hoạt động điển hình của ứng dụng client

3. tạo đối tượng quản lý gửi/nhận dữ liệu tới/từ server. Đối tượng Socket chứa 2 đối tượng gửi/nhận dữ liệu : OutputStream để gửi dữ liệu theo cơ chế từng byte nhị phân, InputStream để nhận dữ liệu theo cơ chế từng byte nhị phân (xem lại Chương 4). Mỗi thông báo request/reply thường là 1 dòng văn bản (được kết thúc bởi ký tự CRLF), để thuận lợi trong việc nhận/gửi từng dòng văn bản, ta thường tạo 2 đối tượng tương ứng như sau :

```
//tạo đối tượng hỗ trợ việc gửi từng dòng văn bản (request)
PrintWriter writer = new PrintWriter(sock.getOutputStream());
//tạo đối tượng hỗ trợ việc nhận từng dòng văn bản (reply)
socket.setSoTimeout(5000);
input = new BufferedReader(new InputStreamReader(
    socket.getInputStream()));
```



## 9.2 Các bước hoạt động điển hình của ứng dụng client

4. Khi cần 1 chức năng nào đó, client xây dựng 1 thông báo request theo đúng qui định của giao thức làm việc rồi gửi request này đến server :

```
String req_Msg = "GLIST "; //xây dựng chuỗi request  
writer.println(req_Msg); //gửi dòng request  
writer.flush(); //buộc hệ thống gửi ngay, không đệm
```

5. chờ nhận thông báo reply chứa kết quả từ server gửi về rồi dùng kết quả này.

```
String message = input.readLine();
```

6. nếu cần chức năng khác của server (thường là vậy), quay về bước 4.

7. khi biết chắc là không cần server nữa, đóng Socket lại.

```
socket.close();
```



## 9.2 Các bước hoạt động điển hình của ứng dụng client

- ❑ Trong bước 5 ở slide trước, lệnh gọi `input.readLine()` có thể bị kẹt lâu dài (tùy thuộc vào thời điểm đối tác gửi dữ liệu cho mình). Nếu lệnh này nằm trong chương trình cần tương tác trực tiếp với người dùng thì trong khoảng thời gian tác vụ `readLine()` chạy, chương trình sẽ không thể tương tác với người dùng. Để khắc phục vấn đề này, ta thường tạo thread con chạy song hành với chương trình chính, thread con này chỉ có nhiệm vụ thực hiện tác vụ `readLine()`, khi nào nhận được dữ liệu thì nó sẽ cảnh báo cho chương trình biết để xử lý (bằng cách gọi thông điệp kích hoạt 1 tác vụ xác định trước của chương trình chính để tác vụ đó xử lý dữ liệu nhận được).



## 9.3 Thí dụ về ứng dụng mạng cơ bản

- ❑ Để thấy rõ chi tiết lập trình 1 ứng dụng mạng, ta thử xây dựng hệ thống MiniChatter với 1 số tính chất sau :
  - Chức năng: cho phép nhiều user đăng ký vào các nhóm để trò chuyện với nhau.
  - Mô hình chọn lựa : client/server
  - Server: quản lý các nhóm và các user thuộc từng nhóm, phân phối các chuỗi thông tin từ một user đến các user khác cùng nhóm...
  - Client: giao tiếp với user, cho phép họ đăng ký nhóm, gửi/nhận thông tin lẫn nhau.



## 9.3 Thí dụ về ứng dụng mạng cơ bản

- ❑ Sau khi phân tích chức năng của chương trình chat, ta định nghĩa giao thức được dùng bởi hệ thống MiniChatter gồm 5 thông báo request sau :

1. Lệnh **GLIST** <LF>

Nhận danh sách các nhóm đang được server quản lý

2. Lệnh **ULIST** <LF>

Nhận danh sách user thuộc nhóm mà mình đang ở

3. Lệnh **LOGIN** <tên group> "," <tên user> <LF>

Đăng ký thành viên mới vào nhóm xác định

4. Lệnh **SEND** <string> <LF>

Gửi 1 dòng dữ liệu đến các user trong cùng nhóm

5. Lệnh **LOGOU** <LF>

Logout khỏi nhóm hiện hành (để đăng ký vào nhóm khác)



## 9.3 Thí dụ về ứng dụng mạng cơ bản

Và định dạng thông báo reply cho tất cả các request :

- $n$  <chuỗi dữ liệu phụ trợ kèm theo> <LF>

với  $n = 1$  : thành công,  $n = 0$  : thất bại.

- Bài thực hành số 9.1 sẽ trình bày qui trình viết module MiniChatter client bằng NetBeans và dùng kỹ thuật xử lý multi-thread để giúp module client có thể tương tác với người dùng trong khi chờ nhận reply của từng request.
- Chương trình client gồm 3 form giao diện : form giao diện chính, form nhập thông tin server cần connect tới và form để nhập thông tin login vào 1 nhóm tán gẫu.



## 9.3 Thí dụ về ứng dụng mạng cơ bản

Form giao  
diện chính.

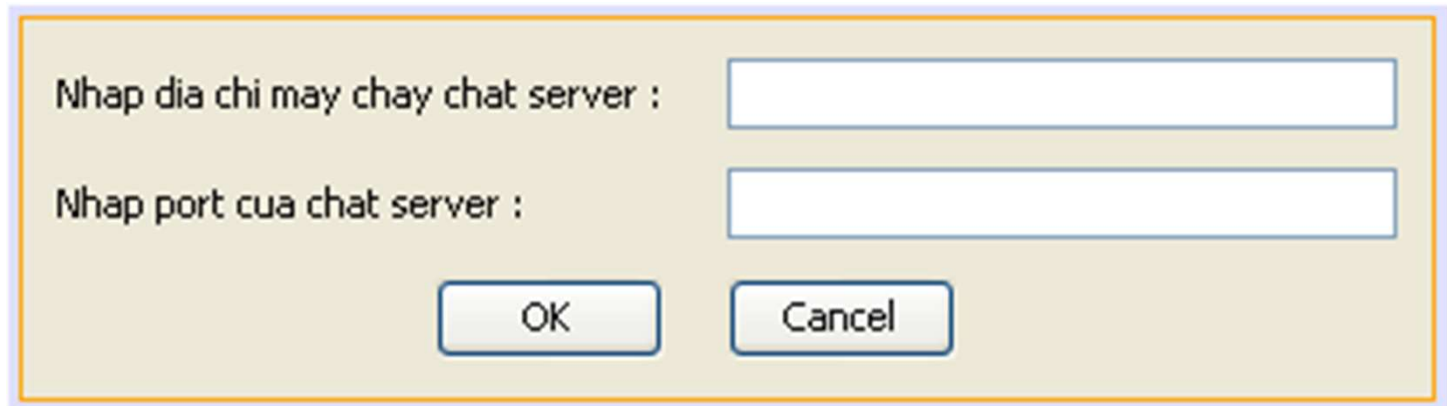
The screenshot shows a Java Swing window titled "Basic Application Example". It has a menu bar with "File" and "Help". Below the menu bar is a row of buttons: "Connect", "Groups", "Login", "Users", "Send", "Logout", and "Disconnect". The main area is divided into three sections. On the left, there are two labels "Danh sach cac nhom :" followed by empty rectangular boxes. On the right, there is a label "Noi dung can goi :" followed by a text input field, and below that, a label "Lich su chat trong nhom :" followed by a large empty rectangular area for chat history.





## 9.3 Thí dụ về ứng dụng mạng cơ bản

Form nhập địa chỉ của chat server.

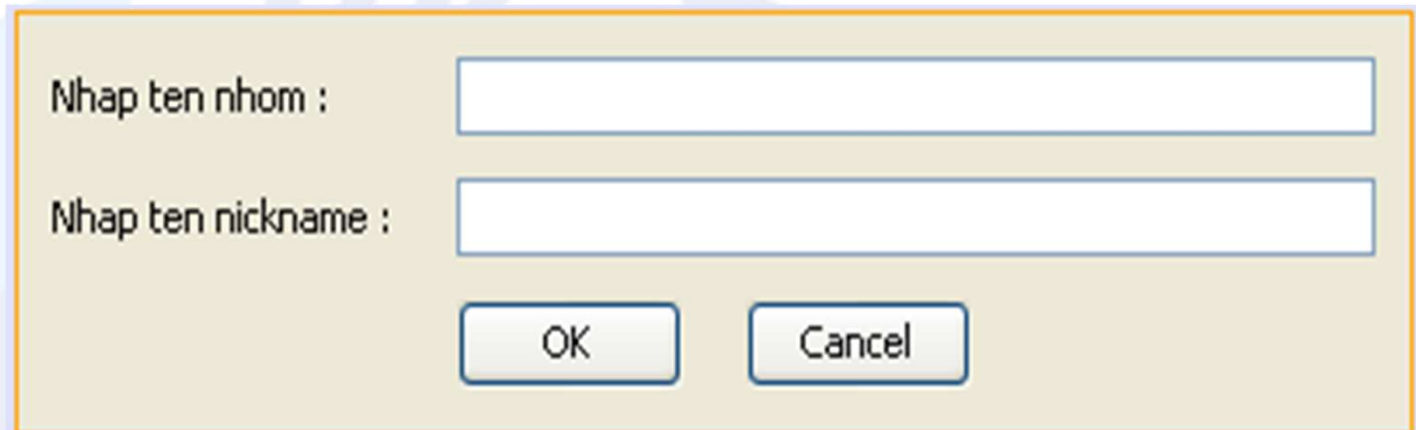


Nhap dia chi may chay chat server :

Nhap port cua chat server :

OK Cancel

Form đăng ký thành viên.



Nhap ten nhom :

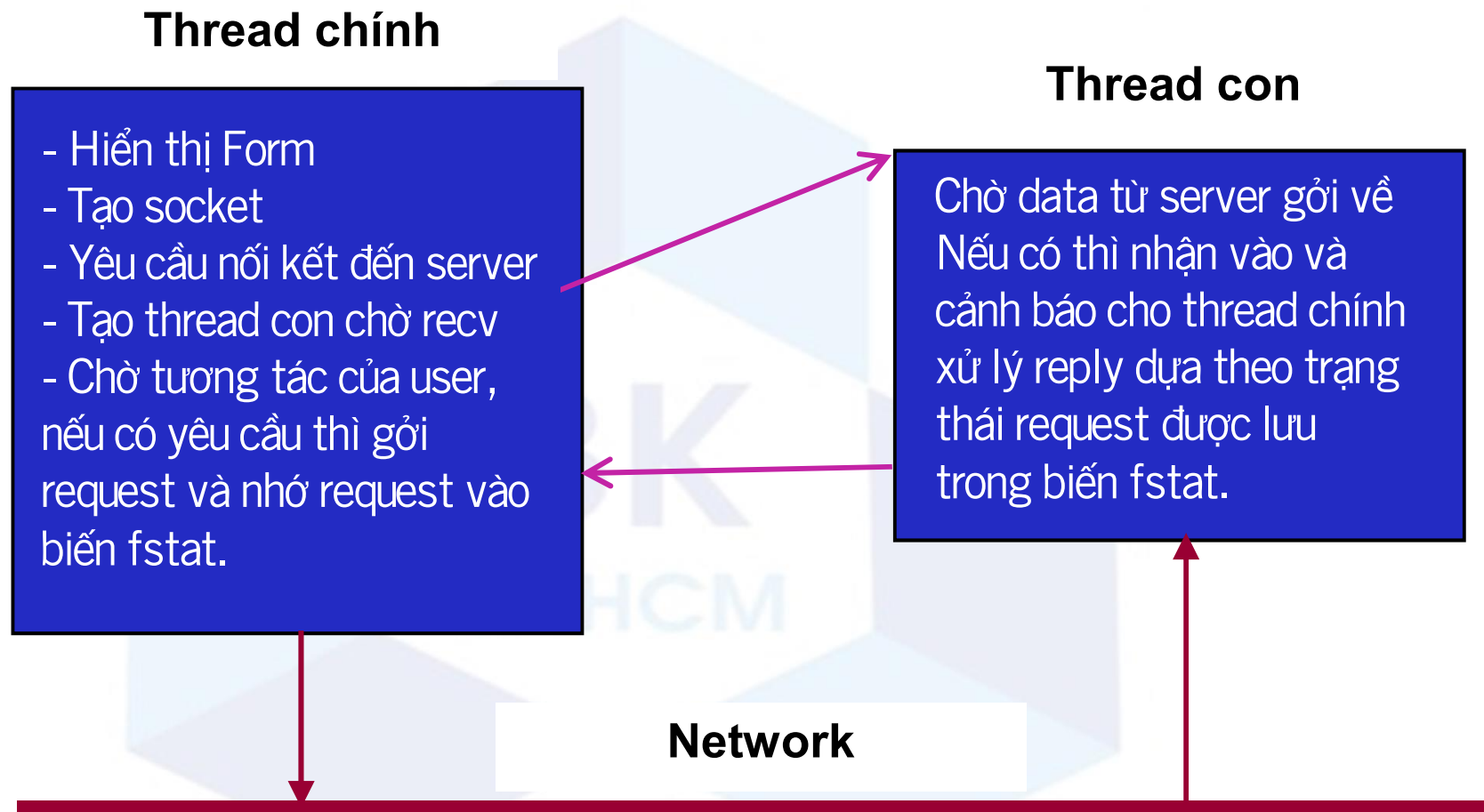
Nhap ten nickname :

OK Cancel



## 9.3 Thí dụ về ứng dụng mạng cơ bản

Mô hình multi-thread ở client :



## 9.4 Các điểm chính về lập trình ứng dụng client

### Lập trình tạo cầu nối đến server :

//hàm thực hiện nối kết đến ChatServer

```
private void btnConnectActionPerformed(java.awt.event.ActionEvent evt) { //tạo và  
hiển thị form yêu cầu nhập địa chỉ server
```

```
    ConnectDlg dlg = new ConnectDlg(new javax.swing.JFrame(), "Nhập thông tin về  
server", true);
```

```
    dlg.show();
```

```
    if (dlg.fOk) { //người dùng thực sự yêu cầu connect  
        try {
```

```
            //tạo Socket với địa chỉ của Server
```

```
            socket = new Socket(InetAddress.getByName(dlg.strAddress), dlg.intPort);
```

```
            //tạo và chạy thread chờ nhận các thông báo từ server gửi về
```

```
            receivingThread = new ReceivingThread(this, socket);
```

```
            receivingThread.start();
```



## 9.4 Các điểm chính về lập trình ứng dụng client

Lập trình tạo cầu nối đến server :

```
//tạo đối tượng hỗ trợ việc gửi từng dòng văn bản (request)
writer = new PrintWriter(socket.getOutputStream());
} catch (Exception e) {
    //xử lý lỗi
    e.printStackTrace();
}
//ghi nhận trạng thái đã nối kết
connected = true;
}
} //kết thúc hàm btnConnectActionPerformed()
```



## 9.4 Các điểm chính về lập trình ứng dụng client

Lập trình gửi request đến server :

//hàm gửi request GLIST theo đui định của giao thức đến server

```
private void btnGroupsActionPerformed(java.awt.event.ActionEvent evt) {
```

```
    //xây dựng thông báo request GLIST theo qui định của giao thức
```

```
    String mesg = "GLIST ";
```

```
    //gửi thông báo request GLIST đến server
```

```
    SendMessage(socket, mesg);
```

```
    //ghi nhận trạng thái đã gửi request GLIST để xử lý reply sau
```

```
    fstate = FSGLIST;
```

```
}
```



## 9.4 Các điểm chính về lập trình ứng dụng client

Lập trình gửi request đến server :

//hàm gửi thông báo theo qui định của giao thức về server

```
public void SendMessage(Socket sock, String mesg) {  
    try {  
        //xác định đối tượng phục vụ gửi thông báo  
        PrintWriter writer = new PrintWriter(sock.getOutputStream());  
        //gửi thông báo  
        writer.println(mesg);  
        //yêu cầu hệ thống gửi ngay  
        writer.flush();  
    } catch (IOException ioException) {  
        ioException.printStackTrace();  
    }  
}
```



## 9.4 Các điểm chính về lập trình ứng dụng client

Lập trình phân tích và xử lý reply :

```
public void messageReceived(Socket sock, String mesg) {
int status;
    switch (fstate) {           //kiểm tra xem reply của request nào
    case FSLOGIN :              //reply của request LOGIN
        Do_login(mesg); break;
    case FSLOGOUT:              //reply của request LOGOUT
        Do_logout(mesg); break;
    case FSMESG:                //reply của request MSG
        Do_receive(mesg); break;
    case FSGLIST:               //reply của request GLIST
        Do_glist(mesg); break;
    ....
    }
    fstate = FSMESG; //ghi nhận trạng thái reply mặc định là nhận mesg
}
```



## 9.4 Các điểm chính về lập trình ứng dụng client

Lập trình thread chuyên chờ nhận reply :

//class miêu tả thread chuyên chờ nhận reply

```
public class ReceivingThread extends Thread {
```

```
    //định nghĩa các thuộc tính cần dùng
```

```
    private MessageListener messageListener;
```

```
    Socket socket;
```

```
    private BufferedReader input;
```

```
    private boolean keepListening = true;
```

```
    //định nghĩa hàm khởi tạo class
```

```
    public ReceivingThread(MessageListener listener, Socket clientSocket) {
```

```
        //kích hoạt cha chạy trước
```

```
        super("ReceivingThread: " + clientSocket);
```

```
        //ghi nhận khách hàng và Socket cần chờ
```

```
        messageListener = listener;
```

```
        socket = clientSocket;
```





## 9.4 Các điểm chính về lập trình ứng dụng client

```
try {  
    //thiết lập thời gian timeout cho socket  
    socket.setSoTimeout(0);  
    //tạo đối tượng nhận dữ liệu từ socket  
    input = new BufferedReader (new InputStreamReader (  
        socket.getInputStream()));  
} catch (IOException ioException) { ioException.printStackTrace(); }  
}
```



## 9.4 Các điểm chính về lập trình ứng dụng client

//thuật giải hoạt động của thread chờ nhận dữ liệu

```
public void run() {
```

```
    String message;
```

```
    //lập các hoạt động dưới đây cho đến khi hết yêu cầu
```

```
    while (keepListening) {
```

```
        try {
```

```
            //chờ nhận 1 dòng văn bản miêu tả reply được gửi đến
```

```
            message = input.readLine();
```

```
            //kiểm tra nếu chuỗi != rỗng thì gọi khách hàng xử lý
```

```
            if (message != null) //nếu != rỗng thì gọi khách hàng xử lý
```

```
                messageListener.messageReceived (socket, message);
```

```
        }
```

```
        catch (InterruptedException interruptedIOException) {
```

```
            continue; // tiếp tục chờ nhận reply
```

```
        }
```



## 9.4 Các điểm chính về lập trình ứng dụng client

```
catch (IOException ioException) {  
    keepListening = false; //yêu cầu kết thúc vòng lặp while  
    //tạo reply đặc biệt và nhờ khách hàng xử lý  
    messageListener.messageReceived (socket, "CLOSE ");  
    break;  
}  
} //kết thúc vòng lặp while  
try {  
    input.close(); //đóng đối tượng chờ nhận reply và đóng socket  
} catch (IOException ioException) {  
    ioException.printStackTrace();  
}  
} //kết thúc hàm run  
} //kết thúc class ReceivingThread
```



## 9.5 Kết chương

- ❑ Chương này đã trình bày kiến trúc hoạt động của mạng Internet, cơ chế hoạt động của module client và server để chúng giao tiếp tốt với nhau.
- ❑ Chương này cũng đã giới thiệu qui trình làm việc điển hình của module client và cách lập trình miêu tả từng bước hoạt động này.
- ❑ Chương này cũng đã giới thiệu cách xây dựng module client của ứng dụng Minichat cùng các đoạn code thực hiện các công việc chính yếu của module client này.

