TRƯỜNG ĐẠI HỌC BÁCH KHOA

KHOA CÔNG NGHỆ THÔNG TIN



ĐỒ ÁN 3

**ĐỀ TÀI : SỬ DỤNG PIPELINE TRONG JAVA XÂY DỰNG CHƯƠNG TRÌNH TRAO ĐỔI THÔNG TIN THEO MÔ HÌNH CLIENT-SERVER**

**Giáo viên hướng dẫn: ThS. TRẦN HỒ THỦY TIÊN**

**Sinh viên thực hiện: NGUYỄN THỊ THƯ**

**Lớp: 16TCLC3**

**MSSV: 102160266**

Đà Nẵng, 12/2019

**LỜI CẢM ƠN**

Nguyên lý hệ điều hành và Lập trình mạng là những kiến thức căn bản là nền tảng mà mỗi lập trình viên phải hiểu rõ để phục vụ cho những kiến thức cao hơn.

Hệ điều hành là tập hợp các chương trình phần mềm chạy trên máy tính, dùng để điều hành, quản lý các thiết bị phần cứng và tài nguyên phần mềm trên máy tính. Hệ điều hành đóng vai trò trung gian trong việc giao tiếp giữa người sử dụng và phần cứng máy tính, cung cấp một môi trường cho phép người sử dụng phát triển và thực hiện các ứng dụng một cách dễ dàng.

Cùng với sự phát triển của mạng Internet hiện nay, thì việc nghiên cứu nằm vững những kiến thức về mạng là rất quan trọng. Dựa trên những hiểu biết của mình và tìm hiểu tài liệu, em đã nghiên cứu và thực hiện đề tài: SỬ DỤNG PIPELINE TRONG JAVA XÂY DỰNG CHƯƠNG TRÌNH TRAO ĐỔI THÔNG TIN THEO MÔ HÌNH CLIENT-SERVER.

Em xin chân thành cảm ơn thầy cô khoa Công Nghệ Thông Tin đã tạo điều kiện để em nghiên cứu kĩ hơn những kiến thức này, và đặc biệt là cô Trần Hồ Thủy Tiên đã nhiệt tình theo dõi, hướng dẫn em trong quá trình thực hiện đề tài này.

Vì kiến thức còn hạn hẹp, nên không thể tránh khỏi những sai sót trong quá trình làm đề tài, rất mong nhận được sự góp ý của thầy cô để sản phầm được hoàn thiện hơn. Em xin chân thành cảm ơn.

Sinh viên thực hiện

Nguyễn Thị Thư

MỤC LỤC

[MỞ ĐẦU 1](#_Toc27760790)

[CHƯƠNG 1: CƠ SỞ LÍ THUYẾT 3](#_Toc27760791)

[1. Giới thiệu về Ống Dẫn 3](#_Toc27760792)

[2. Kĩ Thuật Ống Dẫn Pipe Line Trong Java 4](#_Toc27760793)

[3. Mô Hình Ứng Dụng Client-Server 5](#_Toc27760794)

[CHƯƠNG 2: PHÂN TÍCH THIẾT KẾ HỆ THỐNG 9](#_Toc27760795)

[1. Bài toán: 9](#_Toc27760796)

[3. Sơ đồ hoạt động 10](#_Toc27760797)

[CHƯƠNG 3: TRIỂN KHAI VÀ ĐÁNH GIÁ KẾT QUẢ 11](#_Toc27760798)

[1. Môi trường 11](#_Toc27760799)

[2. Triển khai 11](#_Toc27760800)

[KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN 14](#_Toc27760801)

[1. Kết Quả Đạt Được 14](#_Toc27760802)

[2. Hướng Phát Triển 14](#_Toc27760803)

[KẾT LUẬN CHUNG 15](#_Toc27760804)

[TÀI LIỆU THAM KHẢO 16](#_Toc27760805)

[PHỤ LỤC 17](#_Toc27760806)

DANH SÁCH HÌNH VẼ

*Hình 1: Mô hình ống dẫn 3*

*Hình 2: Mô hình Client-Server sử dụng ống dẫn 4*

*Hình 3: Cách tạo ống dẫn trong java 6*

*Hình 4: Mô hình Client-Server 7*

*Hình 5: Chế độ giao tiếp nghẽn 7*

*Hình 6: Chế độ giao tiếp không nghẽn 8*

*Hình 7: Sơ đồ hoạt động của ứng dụng client-server 10*

*Hình 8: Giao diện chat client 12*

*Hình 9: Giao diện chat client 12*

*Hình 10: Nội dung chat 13*

DANH SÁCH TỪ VIẾT TẮT

|  |  |
| --- | --- |
| **Từ viết tắt** | **Diễn giải** |
| JDT | Java Development Toolkit |
| SDK | Software Development Kit |
| PDE | Plug-in Development Environment |

MỞ ĐẦU

1. **Tổng quan về đề tài**

**2. Mục đích và ý nghĩa của đề tài**

***2.1. Mục đích***

- Đi sâu và nẵm vững một cách có hệ thống kiến thức đã thu nhận được trong quá trình học lí thuyết, làm bài tập và thực hành.

- Từng bước làm quen với các công tác khoa học có định hướng của giáo viên hướng dẫn và hình thành hành vi nghiên cứu độc lập có sự trợ giúp của tài liệu tham khảo.

- Gắn quá trình học lí thuyết với công tác nghiên cứu thực tế.

- Trình bày rõ ràng và khoa học một vấn đề thuộc lĩnh vực nghiện cứu của mình.

***2.2. Ý nghĩa***

- Kết quả của công việc phản ánh công sức, tài năng, trí tuệ của người làm đồ án và phải được trình bày bằng văn bản trong đồ án môn học theo những chuẩn mực và yêu cầu của giáo viên hướng dẫn.

- Đồ án môn học phản ánh công sức nghiên cứu nên cần phải được trình bày trước bộ môn và được đánh giá bằng điểm số. Việc trình bày đúng, rõ ràng, ngắn gọn và khoa học chỉ có được ở những sinh viên có quá trình lao động nghiêm túc cho đề tài và có chuẩn bị đầy đủ kiến thức trong lĩnh vực nghiên cứu đề tài.

**3. Bố cục của đề tài**

Báo cáo đề tài bao gồm các nội dung sau:

*Cơ sở lý thuyết*

*Phân tích thiết kế hệ thống*

*Triển khai và đánh giá kết quả*

*Kết luận chung*

*Tài liệu tham khảo*

*Phụ lục*

# 

**Tổng quan về đề tài**

**Đề tài:** Sử dụng Pipe line trong Java xây dựng chương trình trao đổi thông tin theo mô hình Cient-Server.

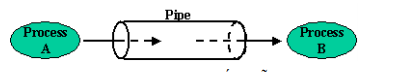
Nội dung đề tài:

* Server:
* Chờ nhận các byte gửi đến từ client.
* Khi có byte đến, nhận byte và gửi về client chính byte đó.
* Client:
* Gửi từng kí tự đến server và chờ nhận kết quả gửi về.
* Chương trình minh họa:
* Sử dụng pipe làm phương tiện giao tiếp giữa Client và Server.
* Server là một thread thực thi song song với Client
* Client nhận các kí tự từ bàn phím và gửi cho server, hiển thị kết quả nhận được ra màn hình.
* Server đổi kí tự nhận được thành kí tự HOA trước khi gửi về cho client.

## CHƯƠNG 1: CƠ SỞ LÍ THUYẾT

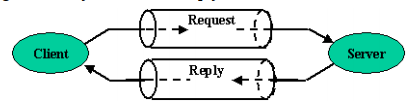
### Giới thiệu về Ống Dẫn

Ống dẫn là một tiện ích được hỗ trợ trong hầu hết các ngôn ngữ lập trình vận hành trên các hệ thống đa nhiệm. Ống dẫn cho phép hai quá trình nằm trên cùng một máy có thể trao đổi dữ liệu với nhau.  
 Dữ liệu đi trên ống dẫn theo một chiều nhất định. Khi sử dụng ống dẫn, người ta dùng một đầu cho việc viết dữ liệu vào và một đầu còn lại cho việc đọc dữ liệu ra.



*Hình 1: Mô hình ống dẫn*

Ống dẫn thích hợp cho trường hợp dữ liệu tạo ra của quá trình này sẽ là dữ liệu đầu vào cho quá trình kia.  
Tuy nhiên ta cũng có thể sử dụng ống dẫn để xây dựng các ứng dụng theo kiến trúc Client- Server bằng cách sử dụng hai ống dẫn: một ống dẫn để truyền các yêu cầu (request), một ống dẫn để truyền các trả lời (reply).



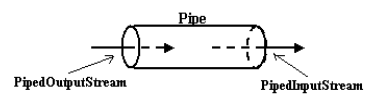
*Hình 2: Dùng ống dẫn trong mô hình Client-Server*

Có hai loại ống dẫn:  
• Ống dẫn bình thường ( Normal Pipe): Giới hạn trong phạm vi không gian địa  
chỉ của một quá trình mà thôi. Nó chỉ cho phép giao tiếp giữa quá trình cha  
với các quá trình con hay giữa các quá trình con của một quá trình với nhau.  
Java hỗ trợ ống dẫn loại này. Trong đó các quá trình con được thay thế bởi  
các luồng.  
• Ống dẫn có tên (Named Pipe): Loại này có thể cho phép hai quá trình có  
không gian địa chỉ khác nhau (trên cùng một máy) giao tiếp với nhau. Thực

chất nó giống như một tập tin với qui định rằng dữ liệu sẽ được lấy ra ở đầu  
tập tin và được thêm vào ở cuối tập tin.

### Kĩ Thuật Ống Dẫn Pipe Line Trong Java

* 1. **Giới thiệu**
* Java hỗ trợ tiên ích ống dẫn thông qua hai lớp java.io.PipedInputStream và java.io.PipedOutputStream. Chúng là hai đầu của một ống dẫn. Trong đó PipedInputStream là đầu đọc dữ liệu và PipedOutputStream là đầu ghi dữ liệu của ống dẫn.
* PipedInputStream là lớp con của InputStream nên nó có tất cả các thuộc tính của InputStream.
* PipedOutputStream là lớp con của OutputStream nên nó có tất cả các thuộc tính của OutputStream.
  1. **Các cách tạo ống dẫn**
* Để tạo một ống dẫn ta chỉ cần tạo ra hai đối tượng thuộc lớp PipedInputStream và PipedOutputStream và nối chúng lại với nhau. Khi đó dữ liệu được ghi vào PipedOutputStream sẽ được đọc ra ở đầu PipedInputStream:



*Hình 3: Cách tạo ống dẫn trong java*

* **Cách 1:**

Bước 1: Tạo đầu đọc:

PipedInputStream readId = new PipedInputStream();

Bước 2: Tạo đầu ghi:

PipedOutputStream writeId = new PipedOutputStream();

Bước 3: Nối đầu đọc với đầu ghi hay ngược lại

readId.connect(writeId);

// hoặc writeId.connect(readId);

* **Cách 2**

Bước 1: Tạo đầu đọc:

PipedInputStream readId = new PipedInputStream();

Bước 2: Tạo đầu ghi và nối vào đầu đọc đã có:

PipedOutputStream writeId = new PipedOutputStream(readId);

Hoặc:  Ta có thể tạo đầu ghi trước rồi tạo đầu đọc sau.

* Lưu ý: Các phương thức khởi tạo của PipedInputStream và PipedOutputStream sử dụng ở trên đòi hỏi phải "bắt" (catch) IOException do chúng có thể “quẳng” ra (throws).

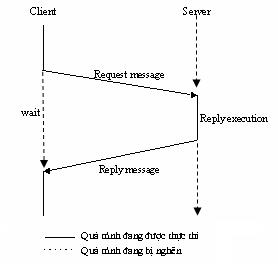
### Mô Hình Ứng Dụng Client-Server

**3.1.** **Thành phần:**

- Quá trình chuyên cung cấp một số phục vụ nào đó, chẳng hạn: phục vụ tập tin, phục vụ máy in, phục vụ thư điện tử, phục vụ Web... Các quá trình này được gọi là các trình phục vụ hay Server.

* Một số quá trình khác có yêu cầu sử dụng các dịch vụ do các server cung cấp được gọi là các quá trình khách hàng hay Client.

Việc giao tiếp giữa client và server được thực hiện dưới hình thức trao đổi các thông điệp (Message). Để được phục vụ, client sẽ gởi một thông điệp yêu cầu (Request Message) mô tả về công việc muốn server thực hiện. Khi nhận được thông điệp yêu cầu, server tiến hành phân tích để xác định công việc cần phải thực thi. Nếu việc thực hiện yêu cầu này có sinh ra kết quả trả về, server sẽ gởi nó cho client trong một thông điệp trả lời (Reply Message). Dạng thức (format) và ý nghĩa của các thông điệp trao đổi giữa client và server được qui định rõ bởi giao thức (protocol) của ứng dụng.



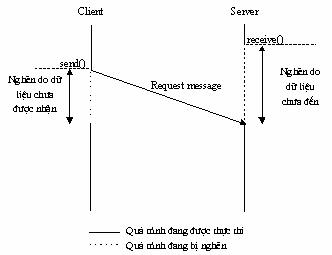
*Hình 4: Mô hình Client-Server*

**3.2. Các chế độ giao tiếp**

Quá trình giao tiếp giữa client và server có thể diễn ra theo hai chế độ là nghẽn (blocked) hay không nghẽn (Non blocked).

* Chế độ nghẽn:

Trong chế độ này, khi quá trình client hay server phát ra lệnh gởi dữ liệu, (thông thường bằng lệnh send), sự thực thi của nó sẽ bị tạm dừng cho đến khi quá trình nhận phát ra lệnh nhận số dữ liệu đó (thường là lệnh receive).

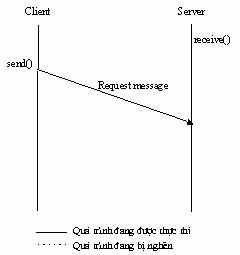


*Hình 5: Chế độ giao tiếp nghẽn*

Tương tự cho trường hợp nhận dữ liệu, nếu  quá trình nào đó, client hay server, phát ra lệnh nhận dữ liệu, mà ở thời điểm đó chưa có dữ liệu gởi đến, sự thực thi của nó cũng tạm dừng cho đến khi có dữ liệu gởi đến.

* Chế độ không nghẽn:

Trong chế độ này, khi quá trình client hay server phát ra lệnh gởi dữ liệu, sự thực thi của nó vẫn được tiếp tục mà không quan tâm đến việc có quá trình nào phát ra lệnh nhận số dữ liệu đó hay không.



*Hình 6: Chế độ giao tiếp không nghẽn*

Tượng tự cho trường hợp nhận dữ liệu, khi quá trình phát ra lệnh nhận dữ liệu, nó sẽ nhận được số lượng dữ liệu hiện có (bằng 0 nếu chưa có quá trình nào gởi dữ liệu đến). Sự thực thi của quá trình vẫn được tiếp tục.

Trong thực tế cần chú ý đến chế độ giao tiếp nghẽn khi lập trình, vì nó có thể dẫn đến trường hợp chương trình bị "treo" do số lần gởi và nhận giữ liệu không bằng nhau giữa hai bên giao tiếp.

## CHƯƠNG 2: PHÂN TÍCH THIẾT KẾ HỆ THỐNG

### Bài toán:

Sử dụng Pipe line trong Java xây dựng chương trình trao đổi thông tin theo mô hình Cient-Server.

1. **Thiết kế:**

Sử dụng kĩ thuật pipeline trong java:

* Tạo 2 tuyến con độc lập, tuyến Client và tuyến Server, đầu đọc của Client sẽ nối với đầu ghi của Server và ngược lại, đầu ghi của Client sẽ nối với đầu đọc của Server- sử dụng phương thức readId.connect(writeId);
* Override phương thức run() để lấy dữ liệu từ đầu đọc (read) khi có dữ liệu gửi từ đầu ghi (write)

@Override

**public** **void** run() {

**while**(**true**){

readMessage();

}

}

**private** **void** readMessage() {

**byte**[] buf = **new** **byte**[1024];

**try** {

**int** len = in.read(buf);

System.***out***.println("Client: " + **new** String(buf, 0, len));

receiveDataFromClient(**new** String(buf, 0, len));

} **catch** (IOException e) {

e.printStackTrace();

} **finally** {

}

}

* Tạo action khi nhấn nút send, tiến hành ghi dữ liệu vào đầu ghi của đường ống gửi sang đầu đọc – sử dụng lệnh out.write(Byte[] data);

@Override

**public** **void** actionPerformed(ActionEvent e) {

**if**(e.getActionCommand()=="Send"){

**try** {

writeMessage(txtMessage.getText());

sendMessageToClient();

} **catch** (Exception e1) {

e1.printStackTrace();

}

}

}

**private** **void** writeMessage(String tem) {

**try** {

out.write(tem.getBytes());

} **catch** (IOException e) {

e.printStackTrace();

}

}

### Sơ đồ hoạt động

**readMessage()**

**readMessage()**

**in**

**out**

Client

Server

**readMessage()**

**writeMessage(msg)**

**send()**

toUpperCaseMsg()

**writeMessage(msg)**

**send()**

**Connect()**

**Connect()**

**in**

**out**

*Hình 7: Sơ đồ hoạt động của ứng dụng client-server*

## CHƯƠNG 3: TRIỂN KHAI VÀ ĐÁNH GIÁ KẾT QUẢ

### Môi trường

* 1. **Eclipse IDE**

Eclipse là phần mềm miễn phí, được các nhà phát triển sử dụng để xây dựng những ứng dụng J2EE, sử dụng Eclipse nhà phát triển có thể tích hợp với nhiều công cụ hỗ trợ khác để có được một bộ công cụ hòan chỉnh mà không cần dùng đến phần mềm riêng nào khác. Eclipse SDK bao gồm 3 phần chính: Platform, Java Development Toolkit (JDT), Plug-in Development Environment (PDE). Với JDT, Eclipse được xem như là một môi trường hỗ trợ phát triển Java mạnh mẽ. PDE hỗ trợ việc mở rộng Eclipse, tích hợp các Plug-in vào Eclipse Platform. Eclipse Platform là nền tảng của toàn bộ phần mềm Eclipse, mục đích của nó là cung cấp những dịch vụ cần thiết cho việc tích hợp những bộ công cụ phát triển phần mếm khách dưới dạng Plug-in, bản thân JDT cũng có thể được coi như là một Plug-in làm cho Eclipse như là một Java IDE (Integrated Development Enviroment).

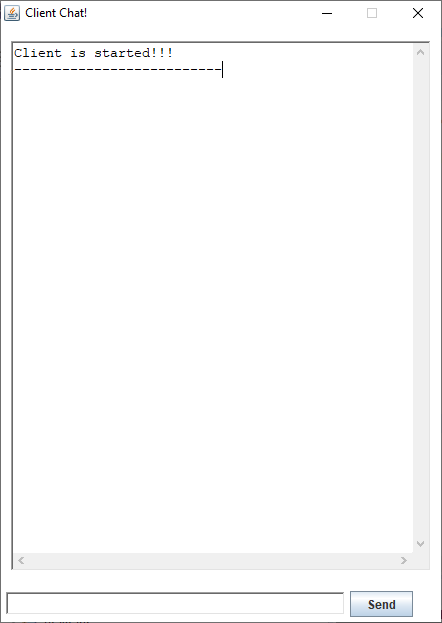
* 1. **JRE**

JRE (là viết tắt của Java Runtime Environment) được sử dụng để cung cấp môi trường runtime. Nó là trình triển khai của JVM. JRE bao gồm tập hợp các thư viện và các file khác mà JVM sử dụng tại runtime. Trình triển khai của JVM cũng được công bố bởi các công ty khác ngoài Sun Micro Systems.

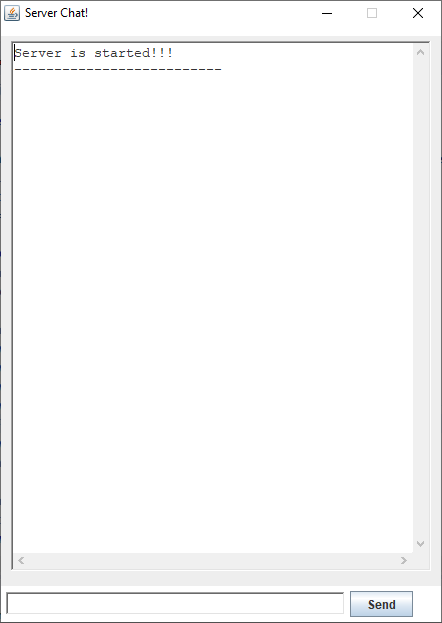
### Triển khai

* Class ClientUI.java: chứa giao diện chat client và các phương thức gửi, nhận dữ liệu
* Class ServerUI.java: chứa giao diện chat Server và các phương thức gửi, nhận dữ liệu từ server.
* Class PipedTest.java: tạo 2 tuyến client và server chạy song song, kết nối đầu đọc với đầu ghi.

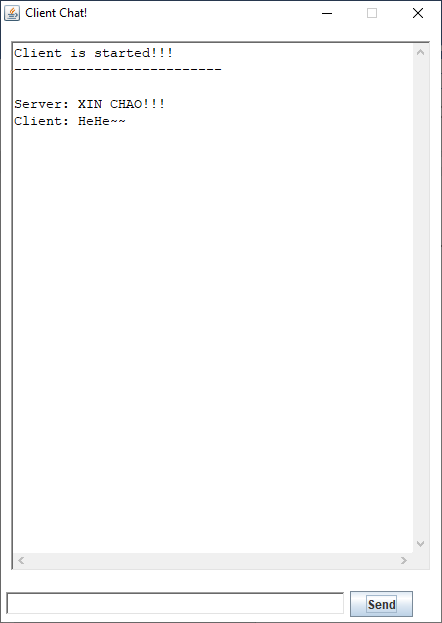
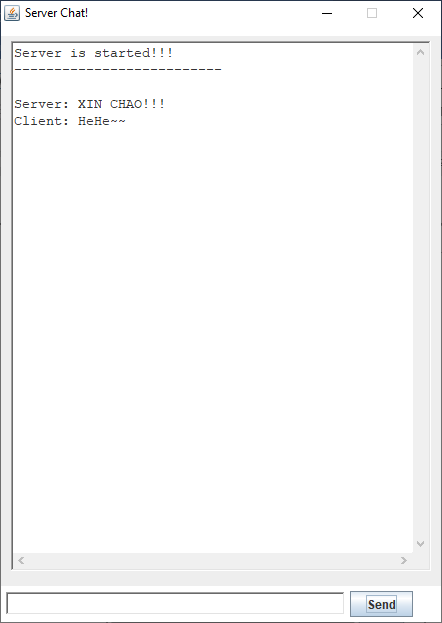
1. **Kết quả**



*Hình 8: Giao diện chat client*



*Hình 9: Giao diện chat Server*

*Hình 10: Nội dung chat*

KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

### Kết Quả Đạt Được

Trong thời gian tìm hiểu, nghiên cứu cơ sở lý thuyết và triển khai ứng dụng công nghệ, đề tài đã đạt được những kết quả sau:

*Về mặt lý thuyết,* hiểu được cách thức hoạt động của Pipeline, client-server và đa luồng.

*Về mặt thực tiễn ứng dụng*, ứng dụng vào việc giao tiếp giữa client-server

Tuy nhiên, đề tài còn tồn tại các vấn đề như sau:

* Vấn đề 1: Pipeline thường không được ưu tiên sử dụng trong trao đổi client-server vì nó không kết nối các máy khác IP.
* Vấn đề 2: Giao diện còn đơn giản

### Hướng Phát Triển

Một số số hướng nghiên cứu và phát triển của đề tài như sau:

* Bổ sung các chức năng trao đổi dữ liệu trong trao đổi client-server.
* Hoàn thiện giao diện đẹp và phù hợp với người dùng

**KẾT LUẬN CHUNG**

Dựa trên những hiểu biết của mình và tìm hiểu tài liệu, em đã nghiên cứu và thực hiện đề tài này. Từ đó em đã hiểu hơn về cách thức hoạt động của Pipeline, mô hình Client-Server, em đã trình bày trên báo cáo. Vì kiến thức còn hạn hẹp,nên không tránh được sai sót , em rất mong nhận được sự góp ý của thầy cô để sản phầm được hoàn thiện hơn.

Em xin chân thành cảm ơn.

TÀI LIỆU THAM KHẢO

*[1]* Giáo trình lập trình truyền thông (Ngô Bá Hùng - Nguyễn Công Huy)

PHỤ LỤC

**1. Pipeline Client-Server**

**- ClientUI.java**

**package** chatPipeline;

**import** java.awt.Button;

**import** java.awt.FlowLayout;

**import** java.awt.Frame;

**import** java.awt.Label;

**import** java.awt.Panel;

**import** java.awt.TextArea;

**import** java.awt.TextField;

**import** java.awt.event.ActionEvent;

**import** java.awt.event.ActionListener;

**import** java.awt.event.WindowAdapter;

**import** java.awt.event.WindowEvent;

**import** java.io.DataInputStream;

**import** java.io.DataOutputStream;

**import** java.io.IOException;

**import** java.io.PipedInputStream;

**import** java.io.PipedOutputStream;

**import** java.net.Socket;

**import** java.net.UnknownHostException;

**import** java.util.Scanner;

**public** **class** ClientUI **extends** Frame **implements** Runnable,ActionListener {

**public** **static** TextField *txtMessage*;

**public** **static** TextArea *taContent*;

**public** JButton btnSend;

**boolean** isConnected;

**private** PipedOutputStream out = **new** PipedOutputStream();

**private** PipedInputStream in = **new** PipedInputStream();

**static** String *msgOut*;

String tem;

**boolean** isSent= **false**;

**public** ClientUI(String clientTitle,PipedOutputStream out,PipedInputStream in) **throws** Exception {

**super**(clientTitle);

**this**.out = out;

**this**.in= in;

System.***out***.println("Client creation");

Panel panelTop = **new** Panel();

panelTop.setLayout(**new** FlowLayout(FlowLayout.***LEFT***));

Panel panelContent = **new** Panel();

*taContent* = **new** TextArea(15, 35);

*taContent*.setEditable(**false**);

*taContent*.setBackground(java.awt.Color.***white***);

*taContent*.setText("Client is started!!!\n--------------------------\n");

Font myFont1 = **new** Font(Font.***MONOSPACED***, Font.***PLAIN***, 14);

*taContent*.setFont(myFont1);

panelContent.add(*taContent*);

Panel panelBottom = **new** Panel();

panelBottom.setLayout(**new** FlowLayout(FlowLayout.***LEFT***));

*txtMessage* = **new** TextField(25);

btnSend = **new** JButton("Send");

panelBottom.add(*txtMessage*);

panelBottom.add(btnSend);

add(panelTop, "North");

add(panelContent);

add(panelBottom, "South");

addEventForButton();

setVisible(**true**);

pack();

addWindowListener(**new** WindowAdapter() {

**public** **void** windowClosing(WindowEvent we) {

System.*exit*(0);

}

});

setResizable(**false**);

}

**public** **void** addEventForButton() {

btnSend.addActionListener(**this**);

}

**public** **void** sendMessageToServer() {

*msgOut* = *txtMessage*.getText()+"";

*pushMsgToScreen*(*msgOut*);

*txtMessage*.setText("");

}

**public** **static** **void** pushMsgToScreen(String msg) {

*taContent*.append("\nClient: " + msg);

}

**public** **static** **void** receiveDataFromServer(String msg) {

*taContent*.append("\nServer: "+msg );

}

@Override

**public** **void** actionPerformed(ActionEvent e) {

**if** (e.getActionCommand() == "Send") {

isSent = **true**;

**try** {

writeMessage(*txtMessage*.getText());

sendMessageToServer();

} **catch** (Exception e1) {

e1.printStackTrace();

}

}

}

//----------------------------------------------

@Override

**public** **void** run() {

**while**(**true**){

readMessage();

}

}

**private** **void** writeMessage(String tem) {

**try** {

**if**(isSent){

out.write(tem.getBytes());

}

isSent = **false**;

} **catch** (IOException e) {

e.printStackTrace();

}

}

**private** **void** readMessage() {

**byte**[] buf = **new** **byte**[1024];

**try** {

**int** len = in.read(buf);

System.***out***.println("Server: " + **new** String(buf, 0, len));

*receiveDataFromServer*(**new** String(buf, 0, len));

} **catch** (IOException e) {

e.printStackTrace();

} **finally** {

}

}

}

**- ServerUI.java**

**package** chatPipeline;

**import** java.awt.Button;

**import** java.awt.ComponentOrientation;

**import** java.awt.FlowLayout;

**import** java.awt.Font;

**import** java.awt.Frame;

**import** java.awt.Label;

**import** java.awt.Panel;

**import** java.awt.TextArea;

**import** java.awt.TextField;

**import** java.awt.event.ActionEvent;

**import** java.awt.event.ActionListener;

**import** java.awt.event.WindowAdapter;

**import** java.awt.event.WindowEvent;

**import** java.io.DataInputStream;

**import** java.io.DataOutputStream;

**import** java.io.IOException;

**import** java.io.PipedInputStream;

**import** java.io.PipedOutputStream;

**import** java.net.Socket;

**import** java.net.UnknownHostException;

**import** java.util.Scanner;

**public** **class** ServerUI **extends** Frame **implements** Runnable,ActionListener {

**public** TextField txtMessage;

**public** **static** TextArea *taContent*;

**public** JButton btnSend;

**boolean** isConnected;

**private** PipedInputStream in = **new** PipedInputStream();

**private** PipedOutputStream out = **new** PipedOutputStream();

**public** ServerUI(String clientTitle, PipedInputStream in,PipedOutputStream out) **throws** Exception{

**super**(clientTitle);

**this**.in = in;

**this**.out = out;

System.***out***.println("Server is starting...");

Panel panelTop = **new** Panel();

panelTop.setLayout(**new** FlowLayout(FlowLayout.***LEFT***));

Panel panelContent = **new** Panel();

*taContent* = **new** TextArea(15,35);

*taContent*.setEditable(**false**);

*taContent*.setBackground(java.awt.Color.***white***);

Font myFont1 = **new** Font(Font.***MONOSPACED***, Font.***PLAIN***, 14);

*taContent*.setFont(myFont1);

*taContent*.setText("Server is started!!!\n------------------------- -\n");

panelContent.add(*taContent*);

Panel panelBottom = **new** Panel();

panelBottom.setLayout(**new** FlowLayout(FlowLayout.***LEFT***));

txtMessage = **new** TextField(25);

btnSend = **new** JButton("Send");

panelBottom.add(txtMessage);

panelBottom.add(btnSend);

add(panelTop,"North");

add(panelContent);

add(panelBottom,"South");

addEventForButton();

setVisible(**true**);

pack();

addWindowListener(**new** WindowAdapter(){

**public** **void** windowClosing(WindowEvent we){

System.*exit*(0);

}

});

setResizable(**false**);

}

**public** **void** addEventForButton(){

btnSend.addActionListener(**this**);

}

**public** **void** sendMessageToClient() {

String msgOut = txtMessage.getText()+"";

pushMsgToScreen(msgOut);

txtMessage.setText("");

}

**public** **void** pushMsgToScreen(String msg){

*taContent*.setComponentOrientation(ComponentOrientation.***RIGHT\_TO\_LEFT***);

*taContent*.append("\nServer: "+msg);

}

**public** **void** receiveDataFromClient(String msg) {

*taContent*.append("\nClient: "+msg);

}

@Override

**public** **void** actionPerformed(ActionEvent e) {

**if**(e.getActionCommand()=="Send"){

**try** {

writeMessage(txtMessage.getText());

sendMessageToClient();

} **catch** (Exception e1) {

e1.printStackTrace();

}

}

}

@Override

**public** **void** run() {

**while**(**true**){

readMessage();

}

}

**private** **void** readMessage() {

**byte**[] buf = **new** **byte**[1024];

**try** {

**int** len = in.read(buf);

System.***out***.println("Client: " + **new** String(buf, 0, len));

receiveDataFromClient(**new** String(buf, 0, len));

} **catch** (IOException e) {

e.printStackTrace();

} **finally** {

}

}

**private** **void** writeMessage(String tem) {

**try** {

out.write(tem.toUpperCase().getBytes());

} **catch** (IOException e) {

e.printStackTrace();

}

}

}

**- PipedTest.java**

**package** chatPipeline;

**import** java.io.IOException;

**import** java.io.PipedInputStream;

**import** java.io.PipedOutputStream;

**public** **class** PipedTest {

**public** **static** **void** main(String[] args) **throws** Throwable {

PipedOutputStream c\_out = **new** PipedOutputStream();

PipedInputStream s\_in = **new** PipedInputStream();

PipedOutputStream s\_out = **new** PipedOutputStream();

PipedInputStream c\_in = **new** PipedInputStream();

Thread client = **new** Thread(**new** ClientUI("Client Chat!",c\_out,c\_in));

Thread server = **new** Thread(**new** ServerUI("Server Chat!",s\_in,s\_out));

**try** {

c\_out.connect(s\_in);

s\_out.connect(c\_in);

client.start();

server.start();

} **catch** (IOException e) {

e.printStackTrace();

}

}

}