**БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ**

МЕХАНИКО-МАТЕМАТИЧЕСКИЙ ФАКУЛЬТЕТ

Кафедра веб-технологий и компьютерного моделирования

ЛОГАШ Полина Александровна

**РАЗРАБОТКА ЧАТА С ИСПОЛЬЗОВАНИЕМ ТЕХНОЛОГИЙ КЛИЕНТ-СЕРВЕР**

Курсовая работа

студентки II курса специальности   
1-31 03 08 Математика и информационные технологии (по направлениям)

Руководитель

ЛЮЛЬКИН Аркадий Ефимович

кандидат технических наук,   
доцент кафедры веб-технологий   
и компьютерного моделирования

Минск, 2022

# СОДЕРЖАНИЕ

ВВЕДЕНИЕ 3

1.Средства Java для разработки распределённых приложений 5

2. Разработка чата на основе сокетов 7

2.1 Клиентская часть 7

2.2 Серверная часть 13

3. Инструкция пользователю 17

ЗАКЛЮЧЕНИЕ 19

СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ 20

ПРИЛОЖЕНИЕ. ЛИСТИНГИ ПРОГРАММ 21

# ВВЕДЕНИЕ

Зачастую для предоставления телекоммуникационных услуг требуется создание распределённых приложений. Распределённое приложение – это программа, состоящая из нескольких взаимодействующих частей, каждая из которых, как правило, выполняется на отдельном компьютере (или другом устройстве) сети. Например, одна часть приложения, выполняющаяся на компьютере пользователя, может поддерживать специализированный графический интерфейс, а вторая работать на мощном выделенном компьютере и заниматься статистической обработкой введенных пользователем данных.

Одной из типовых архитектур распределённого приложения является клиент-серверная архитектура. Данный вид архитектуры распределённых приложений является в настоящее время наиболее распространённым для информационных систем. Существует мнение, что все остальные архитектуры могут быть представлены вариациями данной базовой архитектуры.

Данная информационная система представляет собой совокупность взаимодействующих компонент двух типов: клиентов и серверов.

Клиенты обращаются к серверам с запросами, которые те обрабатывают и возвращают результат. Один клиент может обращаться с запросами к нескольким серверам. Серверы также могут обращаться с запросами друг к другу. Таким образом, типичный протокол взаимодействия может быть представлен в виде обмена сообщениями: запрос клиента – ответ сервера.

Для общения таких компонентов применяется наиболее широко распространённый набор протоколов под общим названием TCP/IP. Этот протокол обеспечивает надежное соединение для передачи данных от одного приложения к другому. IP (Internet Protocol) – межсетевой протокол, который разделяет данные на отдельные пакеты, к которым присоединяет заголовки с адресом получателя. TCP (Transmission Control Protocol) – протокол управления передачей, который отвечает за правильную доставку таких пакетов.

Для работы с сетью используются сокеты – объекты, позволяющие записывать и считывать сообщения, пересылаемые от одного приложения к другому. Сокет – это комбинация IP адреса и номера порта, которые представляют собой способ адресации. Адрес и порт обеспечивают нормальное взаимодействие большого количества приложений в рамках одной системы.

Целью данной работы является изучение разработки распределённых приложений типа клиент-сервер. Для достижения данной цели были поставлены следующие задачи:

* изучить библиотеку Swing для создание графического интерфейса для программ на языке Java;
* изучить средства Java для разработки распределённых приложений
* разработать многопользовательский многопоточный клиент-серверный чат на основе сокетов

1. **Средства Java для разработки распределённых приложений**

**Swing**

Swing в Java является частью базового класса Java. Это один из популярных графических интерфейсов на этом языке. Практически все популярные среды программирования Java включают графические редакторы для Swing-форм, поэтому разрабатывать графический интерфейс для приложений на Swing довольно просто и доступно.

С его помощью создаются оконные приложения, которые включают в себя такие компоненты, как кнопка, полоса прокрутки, текстовое поле и т. д.  
Объединение всех этих компонентов создает графический интерфейс пользователя.

## Преимущества библиотеки Swing:

* содержит богатый и удобный набор элементов пользовательского интерфейса
* намного меньше зависит от той платформы, на которой должна выполняться программа, следовательно, меньше подвержена ошибкам, характерным для конкретной платформы
* обеспечивает одинаковое восприятие конечными пользователями приложений с графическим пользовательским интерфейсом на разных платформах
* встроенный редактор форм почти во всех средах разработки
* поддержка различных стилей (Look and feel)

## Недостатки:

* окно с множеством компонентов начинает подтормаживать;
* работа с менеджерами компоновки может стать настоящим кошмаром в сложных интерфейсах

**Сокеты**

Java для работы в сети имеет специальный пакет java.net, содержащий класс Socket. Ключевыми классами для реализации взаимодействия программ по протоколу TCP являются:

* java.net.ServerSocket - класс реализует серверный сокет, который ожидает запросы, приходящие от клиентов по сети, и может отправлять ответ.
* java.net.Socket - класс реализует клиентский сокет.

Есть несколько конструкторов для создания серверного сокета ServerSocket, но в своём приложении я использую конструктор с параметром порта. Этот параметр будет привязан к серверному сокету.

У клиентского сокета Socket тоже не один конструктор, но я использую c параметрами хоста и порта.

# Разработка чата на основе сокетов

Для разработки клиент-серверного приложения необходимо разделить его на 2 части — клиентскую и серверную. Структура приложения представлена на рис.1.

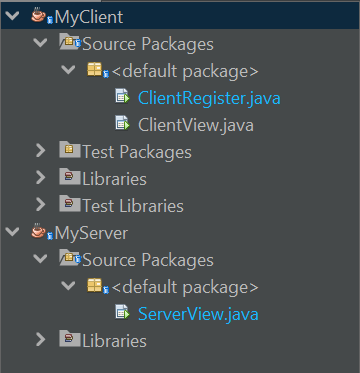
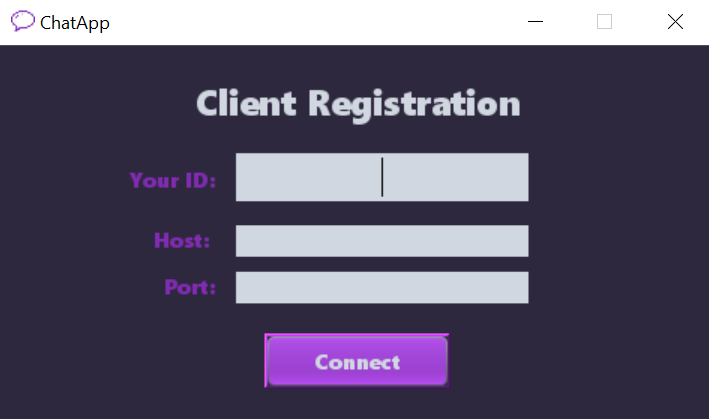


Рис. 1 Структура приложения

## 2.1 Клиентская часть

Для разработки чата на основе TCP/IP сперва нужно создать клиентскую часть. Для этого создаётся новое приложение MyClient. Сначала создадим новую форму JFrameForm для регистрации клиента ClientRegister. В этой форме на панель выносится компонент jPanel, в котором размещаются 3 компонента jTextField, в которые клиент будет вводить своё имя, хост и порт, к которому он хочет присоединиться. Также в jPanel помещается кнопка jButton, при нажатии на которую введённые клиентом данные отправляются на сервер. Для этой кнопки задаётся свойство Variable Name “connectButton”.

Форма ClientRegister после размещения всех компонентов на показана на рис. 2.



*Рис. 2* Окно регистрации клиентской части приложения

Затем для кнопки с заголовком “Connect” описываем обработчик события ActionPerformed:

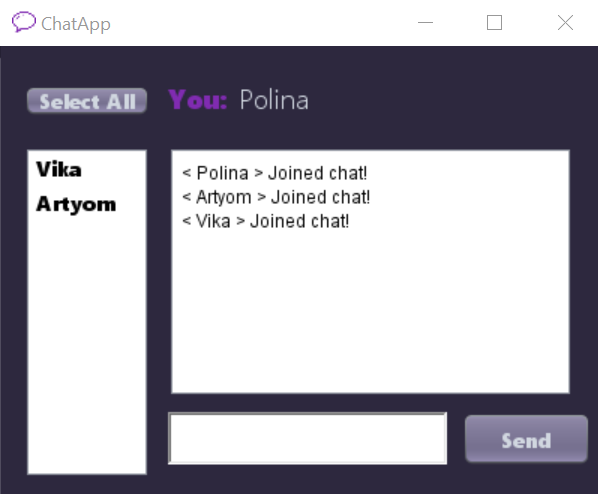


Обработчик собирает введённые данные: id клиента, номер порта и хоста. Данные о порте и хосте используются для активации сокета и попытке подключения к серверу. Если с сервера приходит сообщение о том, что клиент с таким id уже подключён к серверу, то приложение просит поменять свой id. При правильно введённых данных и успешном подключении к серверу панель ClientRegister уничтожится и инициализируется новая панель ClientView.

Затем создаётся новая форма ClientView уже для панели самого чата, которая будет появляться после ввода в окне регистрации клиентом имени (id), хоста и порта и нажатия кнопки “Connect”. В новом классе на форму выносится jTextArea c Variable Name “сlientChatArea”, на котором клиент будет видеть текст сообщений вместе с именами от других клиентов и сервера. Далее добавляется компонент jField c Variable Name clientMessageArea, куда клиент будет вводить текст отправляемого сообщения, добавляется кнопка jButton “sendButton” для отправки сообщений. Далее добавляется компонент jList “usersIdList”, в котором клиент может видеть имена активных пользователей, т.е. тех клиентов, которые в данный момент подключены к этому же хосту и порту. В списке ”usersIdList” клиент может выбрать только одного подключённому клиента для отправки ему сообщения.

Форма ClientView после размещения всех компонентов на показана на рис. 3.

Для того, чтобы сообщение пришло всем пользователям, нужно выбрать каждого клиента из списка. Для этого создаётся кнопка “SelectAll”, которую необходимо нажать перед отправкой сообщения.

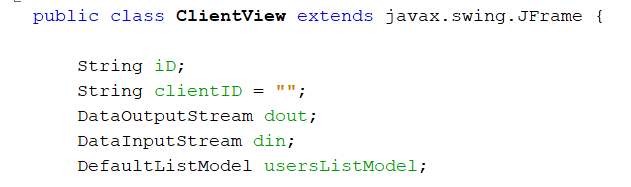


*Рис. 3* Окно чата клиентского приложения

Поле iD класса ClientView содержит данные об id клиента, поле clientID используется как переменная, в которой будет хранится информация о id клиента, которому посылается сообщение, и, если длина этой строки равна нулю, значит сообщение адресовано всем подключённым к серверу клиентам.

В модели данных usersListModel содержатся id всех подключённых к серверу клиентов (Листинг 1).

Листинг 1



Создаётся класс Read (Листинг 2) для чтения сообщений с сервера. Если в полученном сообщении в начале присутствует пометка о том, что это сообщение со строкой id всех подключённых к серверу клиентов, то этот строка разбивается на части с помощью StringTokenizer, параллельно помещая id клиентов в модель usersListModel. Если же в сообщении нет такой пометки, то значит это просто текстовое сообщение от другого пользователя, которое отображается в clientChatArea.

Для удобной отправки сообщения создаётся два обработчика — для кнопки “sendButton” обработчик actionPerformed (Листинг 3) и точно такой же для текстового поля “clientMessageArea”.

Если клиент перед отправкой сообщения выбрал нажал на id пользователя из списка, то при его отправке на сервер вместе с текстом сообщения придёт пометка о том, что сообщение адресовано только одному клиенту и id адресата.

Листинг 2



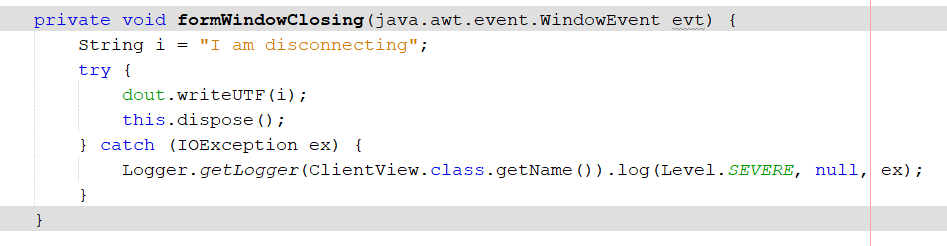
Листинг 3



Если клиент перед отправкой нажал на кнопку “Select All”, то на сервер отправится текст сообщения и пометка о том, что сообщение адресовано всем пользователям.

Если клиент закрывает окно, то на сервер отправляется сообщение “I’m disconnecting” (Листинг 4).

Листинг 4



**2.1 Серверная часть**

Для создания серверной части создаётся новое приложение MyServer. Далее создаётся форма JFrameForm ServerView, после этого добавляется компонент JTextArea “serverChatArea”, где будет отображаться состояние подключения клиентов (рис. 4).

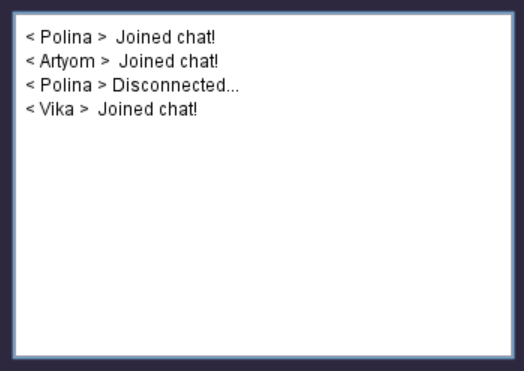


Рис. 4 Область отображения состояния клиентов

Также добавляется компонент JLabel “sStatusLabel”, в котором будет отображаться порт, на котором запущен сервер (рис. 5).

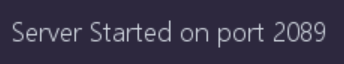


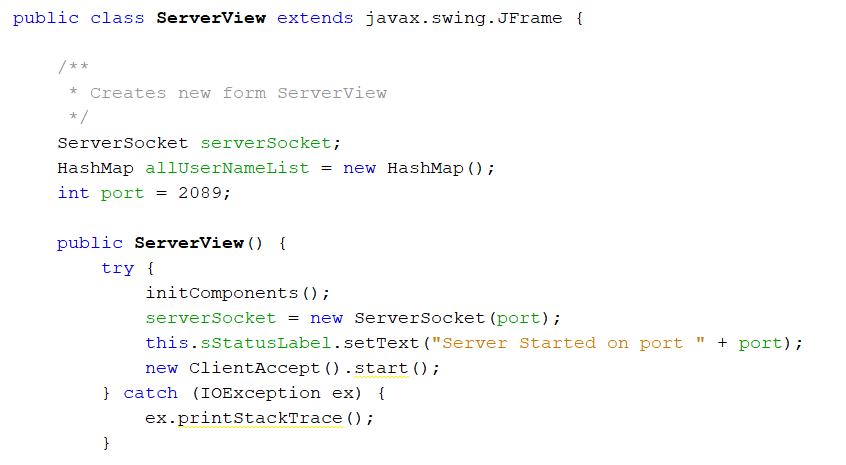
Рис. 5 Область отображения состояния сервера

Форма ServerView после размещения всех компонентов на показана на рис. 6.

В классе ServerView создаём серверный сокет на порту 2089 (Листинг 5)

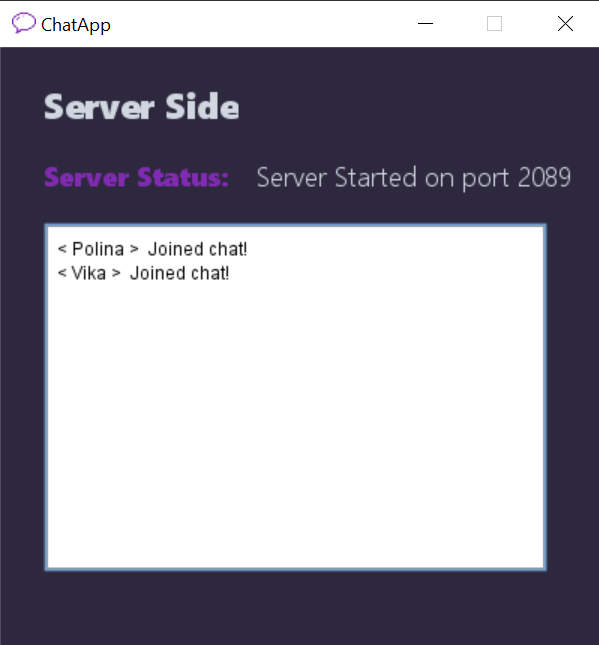
Создаём класс ClientAccept (Листинг 6) для подключения клиентов. На сервер приходит сообщение от клиента с id клиента. Если такое id уже присутствует среди подключённых к серверу клиентов, то сервер посылает обратно сообщение с информацией о том, что с таким id к серверу подключится уже нельзя. Если это новое id, то сервер отправляет сообщение всем клиентам о том, что новый пользователь вошёл в чат.

Листинг 5



Листинг 6





*Рис.6* Окно серверного приложения

Класс PrepareClientList (Листинг 7) создаёт список клиентов, подключенных к серверу и отсылает его всем клиентам.

Класс MessangerReader проверяет, какое сообщение пришло от клиента. Если сообщение состоит из “I’m disconnecting”, то сервер удаляет id клиента из списка подключённых к серверу клиентов, отсылает всем клиентам сообщение что пользователь отключился (Листинг 8), если в сообщение есть пометка о том, что оно для всех подключённых клиентов, то оно отсылается всем, и клиенты видят текст сообщения, а перед ним “< <id отравителя> to All>”. Если приходит сообщение с пометкой, что оно для определённого адресата, то сообщение пересылается определённому клиенту, а перед сообщением: “<id отправителя> to <id получателя>”.

Листинг 7



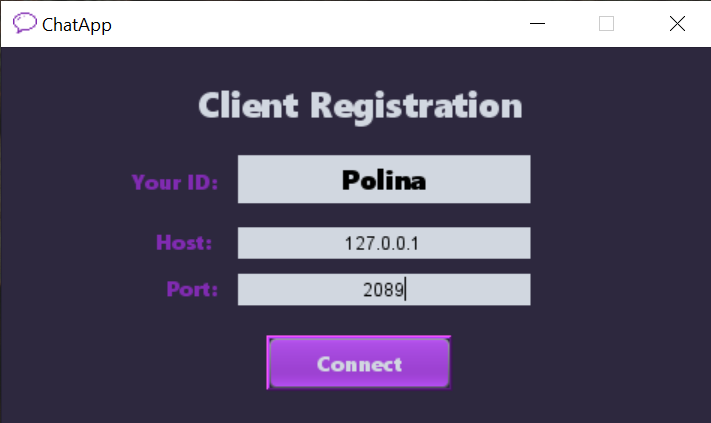
Листинг 8

****

# 3. Инструкции пользователю

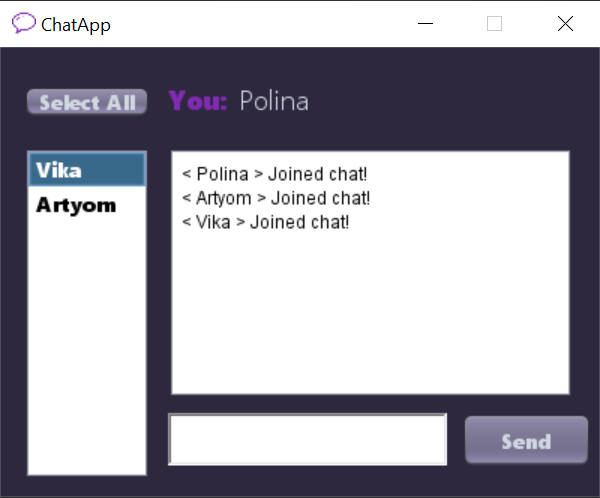
Для запуска и работы с клиентским приложением необходимо:

1. Запустить приложение MyClient
2. Для соединения с сервером ввести своё имя, хост и порт как показано на рис. 7



*Рис. 7 Окно регистрации клента*

1. Для подключения к серверу нажать кнопку “Connect” (рис. 7)
2. В появившемся окне для отправки сообщения одному пользователю, выбрать имя получателя из списка слева в окне (рис. 8)



*Рис. 8 Окно чата клиента*

1. Для отправки сообщения всем подключённым пользователям нажать кнопку “SelectAll” (рис.8)
2. Для отправки сообщения ввести его в текстовое поле расположенное внизу окна и нажать на кнопку “Send” или клавишу Enter (рис.8)

# 

# ЗАКЛЮЧЕНИЕ

В результате выполнения курсовой работы были изучены механизмы и компоненты, которые могут быть использованы для построений распределённых приложений типа клиент-сервер, а также инструменты для создания графического интерфейса на языке Java.

# СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

Видеоплатформа Udemy [Электронный ресурс] /Java. От простого к сложному. Электрон. дан. — М., Александр Васько, 2022. — URL : [https://www.udemy.com/course/java-simple2advanced/ —](https://www.udemy.com/course/java-simple2advanced/%20%E2%80%94) Дата доступа : 10.04.2022.

Блинов, И.Н., Романчик, В. С. Java. Методы программирования : уч.-мет. пособие / И.Н. Блинов, В.С. Романчик. — Минск : издательство «Четыре четверти», 2013. — 896 с

# ПРИЛОЖЕНИЕ. ЛИСТИНГИ ПРОГРАММ

**MyClient (Клиентское приложение) ClientRegister.java**

import java.awt.HeadlessException;

import java.io.DataInputStream;

import java.io.DataOutputStream;

import java.io.File;

import java.io.IOException;

import java.net.Socket;

import javax.imageio.ImageIO;

import javax.swing.JOptionPane;

public class ClientRegister extends javax.swing.JFrame {

public ClientRegister() {

initComponents();

try {

setIconImage(ImageIO.read(new File("../img/msg.png")));

setTitle("ChatApp");

} catch (IOException e) {

e.printStackTrace();

}

}

@SuppressWarnings("unchecked")

Generated Code

private void connectButtonActionPerformed(java.awt.event.ActionEvent evt) {

try {

String id = clientIdField.getText();

String host = hostField.getText();

int port = Integer.parseInt(postField.getText());

Socket s = new Socket(host, port);

DataInputStream din = new DataInputStream(s.getInputStream());

DataOutputStream dout = new DataOutputStream(s.getOutputStream());

dout.writeUTF(id);

String msgFromServer = new DataInputStream(s.getInputStream()).readUTF();

if (msgFromServer.equals("User with such ID already entered!")) {

JOptionPane.showMessageDialog(this, "This ID already registrated!\nPlease, change your ID and try again");

} else if (id.equals("")) {

JOptionPane.showMessageDialog(this, "Please, enter your ID");

} else {

new ClientView(id, s).setVisible(true);

this.dispose();

}

} catch (NumberFormatException | IOException ex) {

JOptionPane.showMessageDialog(this, "Incorrect data entered!\nPlease, check host and port and try again");

}

} ы

public static void main(String args[]) {

try {

for (javax.swing.UIManager.LookAndFeelInfo info : javax.swing.UIManager.getInstalledLookAndFeels()) {

if ("Nimbus".equals(info.getName())) {

javax.swing.UIManager.setLookAndFeel(info.getClassName());

break;

}

}

} catch (ClassNotFoundException | InstantiationException | IllegalAccessException | javax.swing.UnsupportedLookAndFeelException ex) { java.util.logging.Logger.getLogger(ClientRegister.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);

}

java.awt.EventQueue.invokeLater(new Runnable() {

@Override

public void run() {

new ClientRegister().setVisible(true);

}

});

}

private javax.swing.JTextField clientIdField;

private javax.swing.JButton connectButton;

private javax.swing.JTextField hostField;

private javax.swing.JLabel hostLabel;

private javax.swing.JPanel jPanel1;

private javax.swing.JLabel portLabel;

private javax.swing.JTextField postField;

private javax.swing.JLabel registrationLabel;

private javax.swing.JLabel yourIDLabel;

}

**MyClient (Клиентское приложение) ClientView.java**

import java.io.DataInputStream;

import java.io.DataOutputStream;

import java.io.File;

import java.io.IOException;

import java.net.Socket;

import java.util.StringTokenizer;

import java.util.logging.Level;

import java.util.logging.Logger;

import javax.imageio.ImageIO;

import javax.swing.DefaultListModel;

import javax.swing.JOptionPane;

public class ClientView extends javax.swing.JFrame {

String iD;

String clientID = "";

DataOutputStream dout;

DataInputStream din;

DefaultListModel usersListModel;

public ClientView() {

initComponents();

}

public ClientView(String id, Socket s) {

this.iD = id;

try {

initComponents();

try {

setIconImage(ImageIO.read(new File("../img/msg.png")));

setTitle("ChatApp");

} catch (IOException e) {

e.printStackTrace();

}

usersListModel = new DefaultListModel();

usersIdList.setModel(usersListModel);

clientIdLabel.setText(id);

din = new DataInputStream(s.getInputStream());

dout = new DataOutputStream(s.getOutputStream());

new Read().start();

} catch (IOException ex) {

ex.printStackTrace();

}

}

class Read extends Thread {

@Override

public void run() {

while (true) {

try {

String msgFromServer = din.readUTF();

if (msgFromServer.contains("===userIds===")) {

msgFromServer = msgFromServer.substring(13);

usersListModel.clear();

StringTokenizer st = new StringTokenizer(msgFromServer, ",");

while (st.hasMoreTokens()) {

String buffID = st.nextToken();

if (!iD.equals(buffID)) {

usersListModel.addElement(buffID);

}

}

} else {

clientChatArea.append(msgFromServer + "\n");

}

} catch (IOException ex) {

ex.printStackTrace();

break;

}

}

}

}

@SuppressWarnings("unchecked")

Generated Code

private void connectButtonActionPerformed(java.awt.event.ActionEvent evt) {

try {

String id = clientIdField.getText();

String host = hostField.getText();

int port = Integer.parseInt(postField.getText());

Socket s = new Socket(host, port);

DataInputStream din = new DataInputStream(s.getInputStream());

DataOutputStream dout = new DataOutputStream(s.getOutputStream());

dout.writeUTF(id);

String msgFromServer = new DataInputStream(s.getInputStream()).readUTF();

if (msgFromServer.equals("User with such ID already entered!")) {

JOptionPane.showMessageDialog(this, "This ID already registrated!\nPlease, change your ID and try again");

} else if (id.equals("")) {

JOptionPane.showMessageDialog(this, "Please, enter your ID");

} else {

new ClientView(id, s).setVisible(true);

this.dispose();

}

} catch (NumberFormatException | IOException ex) {

JOptionPane.showMessageDialog(this, "Incorrect data entered!\nPlease, check host and port and try again");

}

}

public static void main(String args[]) {

try {

for (javax.swing.UIManager.LookAndFeelInfo info : javax.swing.UIManager.getInstalledLookAndFeels()) {

if ("Nimbus".equals(info.getName())) {

javax.swing.UIManager.setLookAndFeel(info.getClassName());

break;

}

}

} catch (ClassNotFoundException | InstantiationException | IllegalAccessException | javax.swing.UnsupportedLookAndFeelException ex) {

java.util.logging.Logger.getLogger(ClientRegister.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);

}

java.awt.EventQueue.invokeLater(new Runnable() {

@Override

public void run() {

new ClientRegister().setVisible(true);

}

});

}

private javax.swing.JTextField clientIdField;

private javax.swing.JButton connectButton;

private javax.swing.JTextField hostField;

private javax.swing.JLabel hostLabel;

private javax.swing.JPanel jPanel1;

private javax.swing.JLabel portLabel;

private javax.swing.JTextField postField;

private javax.swing.JLabel registrationLabel;

private javax.swing.JLabel yourIDLabel;

// End of variables declaration

}

**MyServer (Серверное приложение) ServerView.java**

import java.io.DataInputStream;

import java.io.DataOutputStream;

import java.io.File;

import java.io.IOException;

import java.net.ServerSocket;

import java.net.Socket;

import java.util.HashMap;

import java.util.Iterator;

import java.util.Set;

import java.util.StringTokenizer;

import javax.imageio.ImageIO;

public class ServerView extends javax.swing.JFrame {

ServerSocket serverSocket;

HashMap allUserNameList = new HashMap();

int port = 2089;

public ServerView() {

try {

initComponents();

serverSocket = new ServerSocket(port);

this.sStatusLabel.setText("Server Started on port " + port);

new ClientAccept().start();

} catch (IOException ex) {

ex.printStackTrace();

}

try {

setIconImage(ImageIO.read(new File("../img/msg.png")));

setTitle("ChatApp");

} catch (IOException e) {

e.printStackTrace();

}

}

class ClientAccept extends Thread {

@Override

public void run() {

while (true) {

try {

Socket clientSocket = serverSocket.accept();

String userIdMsg = new DataInputStream(clientSocket.getInputStream()).readUTF();

if (allUserNameList.containsKey(userIdMsg)) {

DataOutputStream dout = new DataOutputStream(clientSocket.getOutputStream());

dout.writeUTF("User with such ID already entered!");

} else {

allUserNameList.put(userIdMsg, clientSocket);

serverChatArea.append("< " + userIdMsg + " > " + " Joined chat!\n");

DataOutputStream dout = new DataOutputStream(clientSocket.getOutputStream());

dout.writeUTF("");

new MessageReader(clientSocket, userIdMsg).start();

new PrepareClientList().start();

Set<String> k = allUserNameList.keySet();

Iterator itr = k.iterator();

while (itr.hasNext()) {

String key = (String) itr.next();

try {

new DataOutputStream(((Socket) allUserNameList.get(key)).getOutputStream()).writeUTF("< " + userIdMsg + " > Joined chat!");

} catch (IOException ex) {

ex.printStackTrace();

}

}

}

} catch (Exception ex) {

ex.printStackTrace();

}

}

}

}

class MessageReader extends Thread {

Socket s;

String ID;

private MessageReader(Socket s, String userName) {

this.s = s;

this.ID = userName;

}

@Override

public void run() {

while (!allUserNameList.isEmpty() && allUserNameList != null) {

try {

String message = new DataInputStream(s.getInputStream()).readUTF();

if (message.equals("I am disconnecting")) {

allUserNameList.remove(ID);

serverChatArea.append("< " + ID + " > Disconnected...\n");

new PrepareClientList().start();

Set<String> k = allUserNameList.keySet();

Iterator itr = k.iterator();

while (itr.hasNext()) {

String key = (String) itr.next();

if (!key.equals(ID)) {

try {

new DataOutputStream(((Socket) allUserNameList.get(key)).getOutputStream()).writeUTF("< " + ID + " > Disconnected...");

} catch (Exception ex) {

allUserNameList.remove(key);

serverChatArea.append(key + ": Disconnected...\n");

new PrepareClientList().start();

}

}

}

} else if (message.contains("Sending to one person")) { message = message.substring(21);

StringTokenizer st = new StringTokenizer(message, ":");

String id = st.nextToken();

message = st.nextToken();

try {

new DataOutputStream(((Socket) allUserNameList.get(id)).getOutputStream())

.writeUTF("< " + ID + " to " + id + " > " + message);

} catch (Exception ex) {

allUserNameList.remove(id);

serverChatArea.append(id + " Removed!\n");

new PrepareClientList().start();

}

} else if (message.contains("Sending to All")) {

message = message.substring(14);

Set<String> k = allUserNameList.keySet();

Iterator itr = k.iterator();

while (itr.hasNext()) {

String key = (String) itr.next();

if (!key.equals(ID)) {

try {

new DataOutputStream(((Socket) allUserNameList.get(key)).getOutputStream()).writeUTF("< " + ID + " to All > " + message);

} catch (IOException ex) {

ex.printStackTrace();

}

}

}

}

} catch (Exception ex) {

ex.printStackTrace();

}

}

}

}

class PrepareClientList extends Thread {

@Override

public void run() {

try {

String ids = "";

Set k = allUserNameList.keySet();

Iterator itr = k.iterator();

while (itr.hasNext()) {

String key = (String) itr.next();

ids += key + ",";

}

if (ids.length() != 0) {

ids = ids.substring(0, ids.length() - 1);

itr = k.iterator();

while (itr.hasNext()) {

String key = (String) itr.next();

try {

new DataOutputStream(((Socket) allUserNameList.get(key)).getOutputStream()).writeUTF("===userIds===" + ids);

} catch (IOException ex) { allUserNameList.remove(key);

serverChatArea.append(key + " Removed!");

}

}

} catch (Exception ex) {

ex.printStackTrace();

}

}

}

@SuppressWarnings("unchecked")

Generated Code

public static void main(String args[]) {

try {

for (javax.swing.UIManager.LookAndFeelInfo info : javax.swing.UIManager.getInstalledLookAndFeels()) {

if ("Nimbus".equals(info.getName())) {

javax.swing.UIManager.setLookAndFeel(info.getClassName());

break;

}

}

} catch (ClassNotFoundException | InstantiationException | IllegalAccessException | javax.swing.UnsupportedLookAndFeelException ex) {

java.util.logging.Logger.getLogger(ServerView.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);

}

java.awt.EventQueue.invokeLater(new Runnable() {

@Override

public void run() {

new ServerView().setVisible(true);

}

});

}

private javax.swing.JPanel jPanel1;

private javax.swing.JScrollPane jScrollPane1;

private javax.swing.JLabel sStatusLabel;

private javax.swing.JTextArea serverChatArea;

private javax.swing.JLabel serverSideLabel;

private javax.swing.JLabel serverStatusLabel;

}