**МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

Курсовая работа по дисциплине

«Программирование»

Сетевая программа «On-line конференция».

Группа: АВТ-610

Студент: Дунаев Н.Ю.

Преподаватель: Васюткина И. А.

Новосибирск 2018

Оглавление

[Техническое задание 3](#_Toc515704691)

[Постановка задачи 3](#_Toc515704692)

[Требования к программе 3](#_Toc515704693)

[Проектирование 3](#_Toc515704694)

[Структурное описание разработки 5](#_Toc515704695)

[Функциональное описание 9](#_Toc515704696)

[Описание серверной части 9](#_Toc515704697)

[Описание клиентской части 13](#_Toc515704698)

[Описание пользовательского интерфейса и инструкции к использованию 19](#_Toc515704699)

[Заключение 21](#_Toc515704700)

# Техническое задание

## Постановка задачи

Разработать сетевую многопотоковую программу «On-line конференция». Количество участников конференции неограниченно. Для каждого участника, создается свой рабочий поток. Сервер ведет учет участников конференций и пересылает сообщения всем подключенным к определенной конференции.

Участники могут создавать конференции. Остальные присоединяются по желанию (одновременно можно участвовать в нескольких). Закрыть конференцию может только ее владелец. Сообщения передаются только членам конференции.

## Требования к программе

Требуется разработать два модуля: клиент и сервер. Связь будет осуществляться по TCP протоколу. Клиент должен иметь возможность подключиться к серверу и получить доступ к чатам для дальнейшей передачи сообщений. Каждый пользователь сможет идентифицировать себя с помощью имени.

Сервер должен принимать и обрабатывать различные команды от клиента и осуществлять обратную связь.

Передача сообщений происходит следующим образом:

1. Клиент отправляет сообщение на сервер
2. Сервер принимает сообщение и отправляет всем клиентам в рамках одной конференции
3. Все участники конференции принимают сообщение и отображают в своем окне сообщений

# Проектирование

В результате выполнения работы были созданы следующие классы:

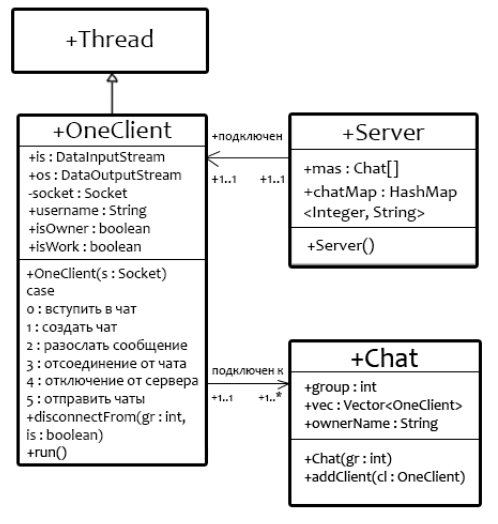
*- Сервер:*

* Main – отвечает за запуск программы (создает объект класса Server)
* Server – отвечает за присоединение клиента к серверу
* ОneClient – слушает команды клиента и осуществляет обратную связь
* Chat – отвечает за создание чата/присоединение к чату

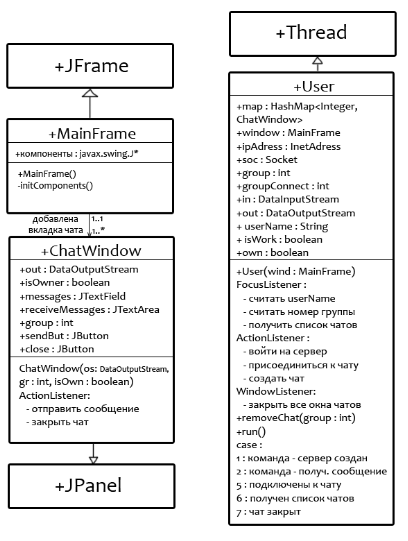
*- Клиент:*

* Main – отвечает за запуск программы (создает объекты классов MainFrame и User)
* MainFrame – пользовательский интерфейс программы (создание окна и добавление компонент)
* User – осуществляет подключение к серверу, передачу и прослушивание сообщений
* ChatWindow – добавляет на окно вкладки с чатами

На рисунках 1,2 изображены диаграммы классов.



*Рисунок 1. Диаграмма классов, сервер*



*Рисунок 2. Диаграмма классов, клиент.*

# Структурное описание разработки

***Сервер***

* **Main**

Класс не имеет полей. Создает объект класса **Server** в методе

public static void main(String[] args)

* **Server**

**Поля:**

static Chat[] mas = new Chat[100]; // Массив чатов. Чат добавляется в массив в том случае, если пользователь пожелал создать новый чат  
static HashMap<Integer, String> chatMap; // Словарь для хранения названий чатов для рассылки доступных чатов пользователям

**Конструкторы:**   
Server() throws Exception // конструктор без параметров

* **Chat**

**Поля:**

static int *group*; // номер чата  
Vector<OneClient> vec; // массив для хранения пользователей, подключенных к одному чату  
String ownerName; // имя владельца чата

**Конструкторы:**

Chat(int gr)// в качестве параметра принимает номер чата для создания

**Методы:**

public void addClient(OneClient client) // добавить клиента к чату

* **OneClient**

**Поля:**  
public DataInputStream is; //потоки вода-вывода  
public DataOutputStream os; //потоки вода-вывода  
private Socket socket; // сокет, отведенный пользователю  
public String userName; // имя пользователя  
boolean isOwner; // является ли пользователь владельцем чата  
boolean isWork = true; // для цикла while метода run()

**Конструкторы:**  
OneClient(Socket s) // в качестве параметра принимает сокет, выделенный пользователю

**Методы:**

public void disconnectFrom(int toDisconnect,Boolean isOwner)// отключение от чата

public void run()// Прослушивание сообщений от клиента и обратная связь по средствам записи данных в поток

***Клиент***

* **Main**

Класс не имеет полей. Создает объект класса **MainFrame** в методе

public static void main(String[] args)

* **MainFrame**

**Поля:**

public java.awt.TextArea groups; // Список чатов  
public javax.swing.JButton jButton1; // Кнопка «Вход»  
public javax.swing.JButton jButton2; // Кнопка «Присоединиться»  
public javax.swing.JButton jButton3; // Кнопка «Создать чат»  
public javax.swing.JLabel jLabel1; // Метка «Username»  
public javax.swing.JLabel jLabel2; // Метка «Список доступных конференций»  
public javax.swing.JLabel jLabel3; // Метка «Номер конференции»  
public javax.swing.JPanel jPanel1;   
public javax.swing.JPanel jPanel2;  
public javax.swing.JScrollPane jScrollPane1;  
public javax.swing.JScrollPane jScrollPane2;  
public javax.swing.JTextArea jTextArea1;  
public javax.swing.JTextArea jTextArea2;  
public static javax.swing.JTabbedPane tabs; // Окно для вкладок

**Конструкторы:**

public MainFrame()

**Методы:**

initComponents(); // Группировка компонент и формирование интерфейса

* **User**

**Поля:**

static HashMap<Integer, ChatWindow> map; // Словарь для сопоставления номера группы и вкладки чата на окне  
MainFrame window;  
InetAddress ipAdress;  
Socket soc;  
int group;  
int groupConnect;  
DataInputStream in;  
DataOutputStream out;  
String userName = "User"; // Имя пользователя, по учмолчанию «User»  
boolean isWork; // Для цикла while метода run()  
boolean own = false; // Является ли пользователь владельцем чата

**Слушатели событий:**

wind.jTextArea1.addFocusListener(new FocusAdapter()) // считывание имени пользователя

wind.jTextArea2.addFocusListener(new FocusAdapter()) // считываем номер группы для подключения

wind.jButton3.addActionListener(new ActionListener()) // вход на сервер

wind.jButton1.addActionListener(new ActionListener()) // присоединение к серверу

wind.jButton2.addActionListener(new ActionListener()) // создание чата

wind.groups.addFocusListener(new FocusAdapter() // получение списка чатов

wind.addWindowListener(new WindowAdapter()) // закрытие всех чатов и согласование их закрытия с сервером

Методы:

public static void removeChat(int group) // удаление чата из словаря одного пользователя

public void run() // слушает команды от сервера

* **ChatWindow**

**Поля:**

DataOutputStream out;  
boolean isOwner; // является ли пользователь владельцем чата  
JTextField messages = new JTextField(35); // для ввода сообщений  
JTextArea receiveMessages= new JTextArea(17,45); // полученные сообщения  
int group; // номер группы чата  
JButton sendBut = new JButton("Отправить");  
JButton close = new JButton("Закрыть");

**Конструкторы:**

ChatWindow(DataOutputStream os, int gr, boolean isOwn)// принимает на вход поток вывода, номер группы чата и информацию о наших правах на чат

**Слушатели событий:**

sendBut.addActionListener(new ActionListener()) // Считывание из формы и отрправка сообщения

close.addActionListener(new ActionListener()) // закрыть чат

**Методы:**

public ChatWindow getWind() // получить ссылку на данную вкладку

# Функциональное описание

## Описание серверной части

Сервер постоянно ожидает подключения клиентов. В случае если клиент подключился, сервер выделяет ему сокет и создает объект класса, описывающего клиента. В классе одного клиента создаются потоки ввода-вывода для связи с пользователем по TCP протоколу. Класс клиента унаследован от Thread, в методе run() реализована обработка всех возможных команд, которые может послать клиент. В зависимости от полученной команды сервер выполняет определенные действия, после чего отправляет клиенту команду в качестве обратной связи.

Клиент может пожелать создать свой собственный чат. Для этого на сервере предусмотрен массив, хранящий все чаты, созданные клиентами.

Ниже представлено подробное описание классов сервера.

* **Server**

Server() throws Exception // Конструктор класса Server  
{  
 *chatMap* = new HashMap<>();  
 ServerSocket serverSocket = new ServerSocket(6538);  
 while (true)  
 {

// Слушаем порт 6538, в случае подключения создаем объект класса одного клиента и передаем в качестве параметра сокет, выделенный клиенту  
 Socket socket = serverSocket.accept();  
 new OneClient(socket);  
 }  
}

* **Chat**

Если пользователь решил создать чат, вызывается конструктор класса **Chat**.

В качестве параметра будет передан номер чата. Номер чата соответствует индексу в массиве чатов Chat[] mas.

Chat(int gr)   
{  
 vec = new Vector<>();  
 *group* = gr;  
}

Если пользователь решил подключиться к чату, он должен будет отправить номер чата для подключения. Если в массиве чатов по данному индексу существует чат, пользователь будет добавлен в него методом:

**public void** addClient(OneClient client)  
{  
 **vec**.add(client);  
}

* **OneClient**

OneClient(Socket s)   
{

// Получаем потоки ввода-вывода от сокета, полученного в качестве параметра  
 try {  
 socket = s;  
 os = new DataOutputStream(socket.getOutputStream());  
 os.flush();  
 is = new DataInputStream(socket.getInputStream());  
 userName = is.readUTF(); // считываем имя пользователя и сохраняем его в поле класса  
 start(); // запускаем поток, слушающий команды от клиента  
 }catch (Exception e)  
 {  
 e.printStackTrace();  
 }  
}

В методе run() сервер слушает команды от клиента, в случае, если команда поступила, сохраняет ее в переменную и при помощи оператора ветвления обрабатывает поступившую команду.

int sw = is.readInt();

switch (sw) {

**Вступить в чат под заданным номером**  
 case 0:  
 int group = is.readInt(); // считываем номер чата для вступления  
 os.writeInt(5); // передаем команду «5» клиенту  
 if (Server.mas[group] != null) { // если такой чат существует  
 isOwner = false;  
 Server.mas[group].addClient(this); // добавляем клиента к чату  
 os.writeInt(1); // оповещаем клиента о том, что подключение успешно  
 } else {  
 os.writeInt(-1); // если группы нет, оповещаем клиента о неудачном присоединении  
 }  
  
 break;

**Создать чат**

case 1:  
 while (Server.mas[i] != null) // ищем свободное место в массиве чатов  
 i++;  
 isOwner = true; // метка о том, что пользователь владеет чатом  
 Server.mas[i] = new Chat(i); // создаем объект класса Chat  
 Server.mas[i].addClient(this); // добавляем в чат клиента  
 Server.mas[i].ownerName = userName; // даем чату имя – имя владельца  
   
 Server.chatMap.put(i, userName); // Сохранение чата в словарь для отправки  
 os.writeInt(1);  
 os.writeInt(i); // отсылаем клиенту номер группы – номер чата, к которому он подключен  
 break;

**Рассылка сообщений**

case 2:  
 int groupToSend = is.readInt(); //читаем группу, в которую требуется отправить сообщение  
 String message = is.readUTF(); // читаем само сообщение   
 for (int j = 0; j < Server.mas[groupToSend].vec.size(); j++) {  
 synchronized (os) {

// рассылаем сообщение всей группе  
 Server.mas[groupToSend].vec.get(j).os.writeInt(2);  
 Server.mas[groupToSend].vec.get(j).os.writeInt(groupToSend);  
 Server.mas[groupToSend].vec.get(j).os.writeUTF(userName);  
 Server.mas[groupToSend].vec.get(j).os.writeUTF(message);  
 Server.mas[groupToSend].vec.get(j).os.flush();  
 }  
 }  
 break;

**Отсоединение от чата**

case 3:  
 int toDisconnect = is.readInt(); // читаем номер чата для отсоединения  
 disconnectFrom(toDisconnect,true); // вызов метода отключения от чата. Помимо номера чата, передаем в качестве параметра информацию о том, что было закрыто не главное окно  
 break;

**Закрытие сокета при выключении программы**

Если пользователь закрыл главное окно, требуется отключить его от всех чатов, чтобы чаты, которыми он владел, были закрыты.

case 4:  
 try {  
 int val = is.readInt(); // читаем количество чатов, в которых участвует пользователь   
 int gr = 0;  
 for(int k = 0; k < val; k++)  
 {

// читаем номер группы для отсоединения и для каждой вызываем метод отключения от чата для каждой группы  
 gr = is.readInt();  
 disconnectFrom(gr, false);  
 }  
 isWork = false;

//закрываем потоки и сокет  
 is.close();  
 os.close();  
 socket.close();  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 break;

**Рассылка текущих чатов**

case 5:  
 try {  
 synchronized (os) {  
 os.writeInt(6); // записываем команду 6

// записываем кол-во чатов, группу и имя владельца чата  
 os.writeInt(Server.chatMap.size());  
 for (Map.Entry<Integer, String> entry : Server.chatMap.entrySet()) {  
 os.writeInt(entry.getKey());  
 os.writeUTF(entry.getValue());  
 }  
 }  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
  
 break;

**Отсоединение от чата**

public void disconnectFrom(int toDisconnect, boolean isOwn)  
{

// Если пользователь является владельцем чата, всем участникам рассылается команда о том, что владелец отключился от чата  
 if (isOwner) {

for (int j = 0; j < Server.mas[toDisconnect].vec.size(); j++) {

synchronized (os)

{

try {

if(Server.mas[toDisconnect].vec.get(j) != Server.mas[toDisconnect].ownerObject) {

Server.mas[toDisconnect].vec.get(j).os.writeInt(7);

Server.mas[toDisconnect].vec.get(j).os.writeInt(toDisconnect);

}

} catch (Exception x) {

x.printStackTrace();

}

}

}

Server.mas[toDisconnect] = null;

Server.chatMap.remove(toDisconnect);

// Если отключен не владелец, удаляем его из массива одного чата

} else if(Server.mas[toDisconnect] != null){

for (int j = 0; j < Server.mas[toDisconnect].vec.size(); j++)

if (this.socket == Server.mas[toDisconnect].vec.get(j).socket)

Server.mas[toDisconnect].vec.remove(j);

}

}

## Описание клиентской части

Класс **MainFrame** создает окно – интерфейс пользователя. При нажатии на кнопку «Вход», находящуюся на окне, осуществляется подключение к серверу и вызывается метод run(), слушающий команды от сервера. При подключении клиента к чату или при создании чата, на панель, находящуюся на окне, добавляется вкладка чата – объект класса **ChatWindow**.

* **Main**

public class Main {  
  
 public static void main(String[] args) {  
 MainFrame fr = new MainFrame(); // создаем окно  
 new User(fr); // создаем объект класса User  
 }  
}

* **User**

User(MainFrame wind)  
{  
 window = wind;  
 *map* = new HashMap<>();  
 isWork = true;  
  
 Получаем имя пользователя и сохраняем его в поле классаwind.jTextArea1.addFocusListener(new FocusAdapter() {  
 @Override  
 public void focusLost(FocusEvent e) {  
 userName = wind.jTextArea1.getText();  
 }  
 });  
  
 Считывание номера группы для подключенияwind.jTextArea2.addFocusListener(new FocusAdapter() {  
 @Override  
 public void focusLost(FocusEvent e) {  
 super.focusLost(e);  
 try {  
 groupConnect = Integer.*parseInt*(wind.jTextArea2.getText());  
 }catch (Exception t){  
 wind.jTextArea2.setText("0");  
 }  
  
 }  
 });  
 Вход на сервер: подключаемся к порту 6538 и передаем серверу имя пользователя, после чего отправляем серверу команду о том, что нам нужен список чатовwind.jButton3.addActionListener(new ActionListener() {  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 try {  
 ipAdress = InetAddress.*getByName*("127.0.0.1");  
 soc = new Socket(ipAdress, 6538);  
 out = new DataOutputStream(soc.getOutputStream());  
 out.flush();  
 in = new DataInputStream(soc.getInputStream());  
 start();  
 out.writeUTF(userName);  
 System.*out*.println(userName + " присоеднен к серверу!");  
 wind.jButton3.setEnabled(false);  
 isWork = true;  
 out.writeInt(5);  
 } catch (Exception ex) {  
 ex.printStackTrace();  
 }  
 }  
 });  
  
 Обработка нажатия на кнопку "Присоединиться". Отправляем серверу соответствующую команду и номер чата к присоединениюwind.jButton1.addActionListener(new ActionListener() {  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 if(!*map*.containsKey(groupConnect)) {  
 try {  
 out.writeInt(0);  
 out.writeInt(groupConnect);  
 own = false;  
 } catch (IOException e1) {  
 e1.printStackTrace();  
 }  
 }  
 }  
 });  
  
 Обработка нажатия на кнопку "Создать конференцию". Отправляем серверу сообщение о том, что хотим создать серверwind.jButton2.addActionListener(new ActionListener() {  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 try {  
 out.writeInt(1);  
 }catch (Exception e1)  
 {  
 e1.printStackTrace();  
 }  
 own = true;  
 }  
 });  
  
 Отправить запрос на получение чатов доступных к подключению  
wind.groups.addFocusListener(new FocusAdapter() {  
 @Override  
 public void focusGained(FocusEvent e) {  
 super.focusGained(e);  
 try {  
 out.writeInt(5);  
 } catch (IOException e1) {  
 e1.printStackTrace();  
 }  
  
 }  
 });  
  
 // Закрытие сокета и потоков при закрытии окна. Отправляем серверу команду о том, что хотим отключиться от сервера и отправляем все чаты, к которым подключены.  
 wind.addWindowListener(new WindowAdapter() {  
 @Override  
 public void windowClosing(WindowEvent e) {  
 if(soc != null) {  
 try {  
 out.writeInt(4);  
 out.writeInt(map.size());  
 for (Map.Entry<Integer, ChatWindow> entry : map.entrySet())  
 {  
 out.writeInt(entry.getKey());  
 }  
 out.flush();  
 isWork = false;  
 in.close();  
 out.close();  
 soc.close();  
 } catch (IOException e1) {  
 e1.printStackTrace();  
 }  
 isWork = false;  
 }  
 super.windowClosing(e);  
 }  
 });  
}

**// Отключить клиента от одного чата**  
public static void removeChat(int group)  
{  
 map.remove(group);  
}

Метод run() слушает команды, полученные от сервера

public void run() {  
 int command;  
 while (isWork) {  
 try {  
 command = in.readInt();  
 switch (command) {  
  **Читает команду о том, что сервер создан**  
 case 1:  
 group = in.readInt(); //читаем номер чата, созданного сервером  
 ChatWindow chat = new ChatWindow(out, group, own); // создаем объект класса ChatWindow, передавая в качетве параметров поток ввода, номер группы и права на чат  
 window.tabs.add(chat.getWind(),"Chat " + group); // добавляем вкладку чата на панель вкладок  
 map.put(group, chat); // сохраняем информацию о том, что пользователь теперь состоит в данной группе  
 out.writeInt(5); // получаем обновленный список чатов  
 break;  
  
 // Читает команду (получить сообщение) - читает группу - читает username + сообщение - находит окно по групе в словаре map и пишет в поле чата сообщение  
 case 2:  
 int sendGr;  
 sendGr = in.readInt();  
 StringBuffer s = new StringBuffer(2048);  
 s.append(map.get(sendGr).receiveMessages.getText() + "\n" + in.readUTF() + ": " + in.readUTF());  
 map.get(sendGr).receiveMessages.setText(s.toString());  
 break;  
 **// Если хотим подключиться к чату из списка**  
 case 5:

// В случае, если сервер ответил нам, что подключение к серверу успешно, создаем окно чата и добавляем его в массив окон  
 if ((in.readInt()) != -1) {  
 ChatWindow chatt = new ChatWindow(out, groupConnect, false);  
 window.tabs.add(chatt.getWind(),"Chat " + groupConnect);  
 map.put(groupConnect, chatt);  
 out.writeInt(5);  
 break;  
  
 **// Получение списка чатов**  
 case 6:

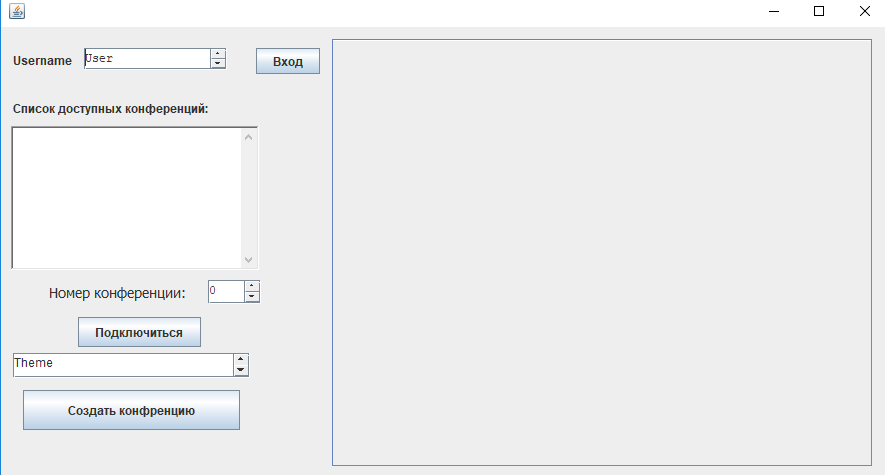
//Читаем кол-во доступных чатов, после чего читаем название чата + имя владельца и склеиваем всю информацию в одну строку, после чего выводим ее в окно чатов  
 StringBuffer strChats = new StringBuffer(4096);  
 int size = in.readInt();  
 for (int i = 0; i < size; i++) {  
 strChats.append(in.readInt() + ". " + in.readUTF() + "'s chat \n");  
 }  
 window.groups.setText(strChats.toString());  
 break;  
  
 // Если закрыли чат  
 case 7:  
 int dis = in.readInt();  
 if(in.readInt() == -1)  
 map.get(dis).receiveMessages.setText("Владелец покинул беседу");  
 map.remove(dis);  
 out.writeInt(5);  
 break;  
 }

* **ChatWindow**

ChatWindow(DataOutputStream os, int gr, boolean isOwn)  
{  
 // Считывание из формы и отправка сообщения  
 sendBut.addActionListener(new ActionListener() {  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 try {  
 synchronized (os)  
 {  
 os.writeInt(2);  
 os.writeInt(group);  
 os.writeUTF(messages.getText());  
 os.flush();  
 }  
 messages.setText("");  
 } catch (IOException e1) {  
 e1.printStackTrace();  
 }  
 }  
 });  
 // закрыть чат  
 close.addActionListener(new ActionListener() {  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 try {  
 os.writeInt(3);  
 os.writeInt(group);  
 } catch (IOException e1) {  
 e1.printStackTrace();  
 }  
 User.removeChat(group);  
 System.out.println("Чат удален " + group);  
 MainFrame.tabs.removeTabAt(MainFrame.tabs.getSelectedIndex());  
 }  
 });  
}

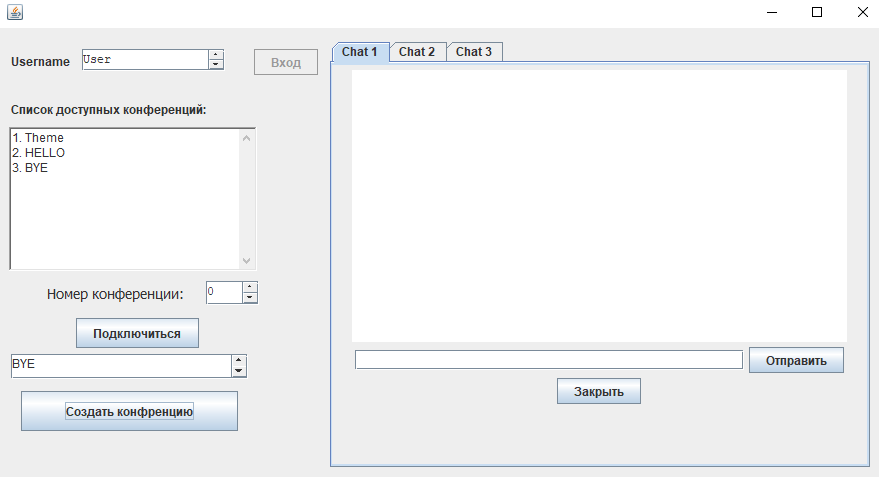
# Описание пользовательского интерфейса и инструкции к использованию

Имеется возможность ввести имя пользователя. Для подключения к серверу следует нажать кнопку «Вход», после чего будет получен список чатов, если чаты имеются. (рис.3)



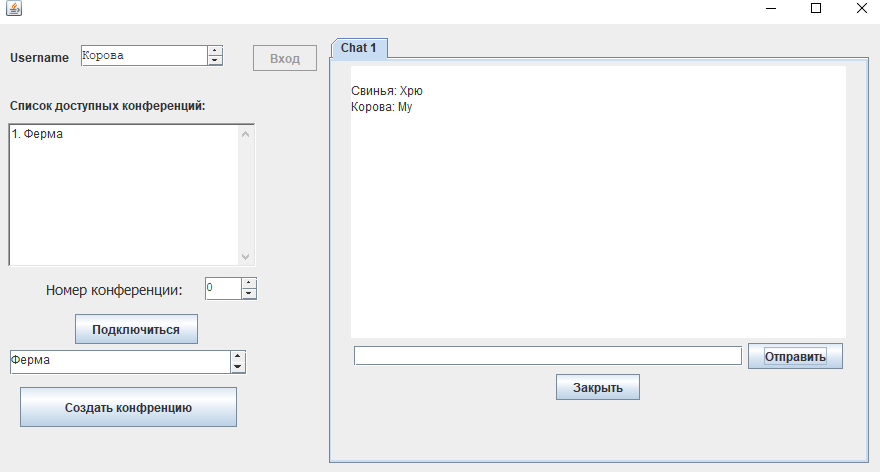
*Рисунок 3. Пользовательский интерфейс*

Пользователь может подключиться к любому из доступных чатов. Для этого требуется ввести номер чата в поле «Номер конференции», после чего нажать на кнопку «Подключиться». Также имеется возможность создать свой чат. Для этого требуется нажать на кнопку «Создать конференцию». При подключении к чату или при создании чата в правой части окна появится набор вкладок с текущими чатами, в которых состоит пользователь. (рис.4)



*Рисунок 4. Пользовательский интерфейс*

Теперь клиент имеет возможность отправить сообщение и закрыть чат. Чтобы отправить сообщение, требуется ввести текст в форму, находящуюся слева от кнопки «Отправить», после чего нажать на саму кнопку. Сообщение получат только пользователи, находящиеся в одном чате с отправителем.(рис.5)



*Рисунок 5. Пользовательский интерфейс*

Чтобы закрыть чат, пользователь должен нажать на кнопку «Закрыть».

# Заключение

В результате выполнения курсовой работы была разработана конференция для общения между двумя или более пользователями. Программа имеет простой графический интерфейс и краткое руководство пользователя.

# Список литературы

**Хорстманн, Кей. 2016.** *Java. Библиотека профессионала. Том 1.* б.м. : Вильямс, 2016.

**Шилдт, Герберт. 2015.** *Java 8. Полное руководство 9-е издание.* 2015.