Министерство образования Республики Беларусь

Учреждение образования

«Брестский государственный технический университет»

Кафедра ИИТ

**Лабораторная работа №12**

По дисциплине «СПП»

за 6-й семестр

Выполнил:

студент 3 курса

группы ПО-3 (1)

Лущ М. Г.

Проверил:

Крощенко А.А.

Брест, 2021

**Цель работы:** освоить приемы разработки оконных клиент-серверных приложений на Java с использованием сокетов.

**Вариант:** 14

**Задание:**

Разработать клиент-серверное оконное приложение на Java с использованием сокетов и JavaFX.  
Можно сделать одну программу с сочетанием функций клиента и сервера либо две отдельных (клиентская часть и серверная часть). Продемонстрировать работу разработанной программы в сети  
либо локально (127.0.0.1). Лабораторную работу разрешается выполнять в команде из 2-х  
человек.

2) Простейший многопользовательский чат. Простой чат с возможностью подключения до 5 пользователей. Все пользователи подключаются к серверу, задача сервера – отображение сообщений  
конкретного пользователя (приват) или общего чата.

**Ход работы**

Код программы:

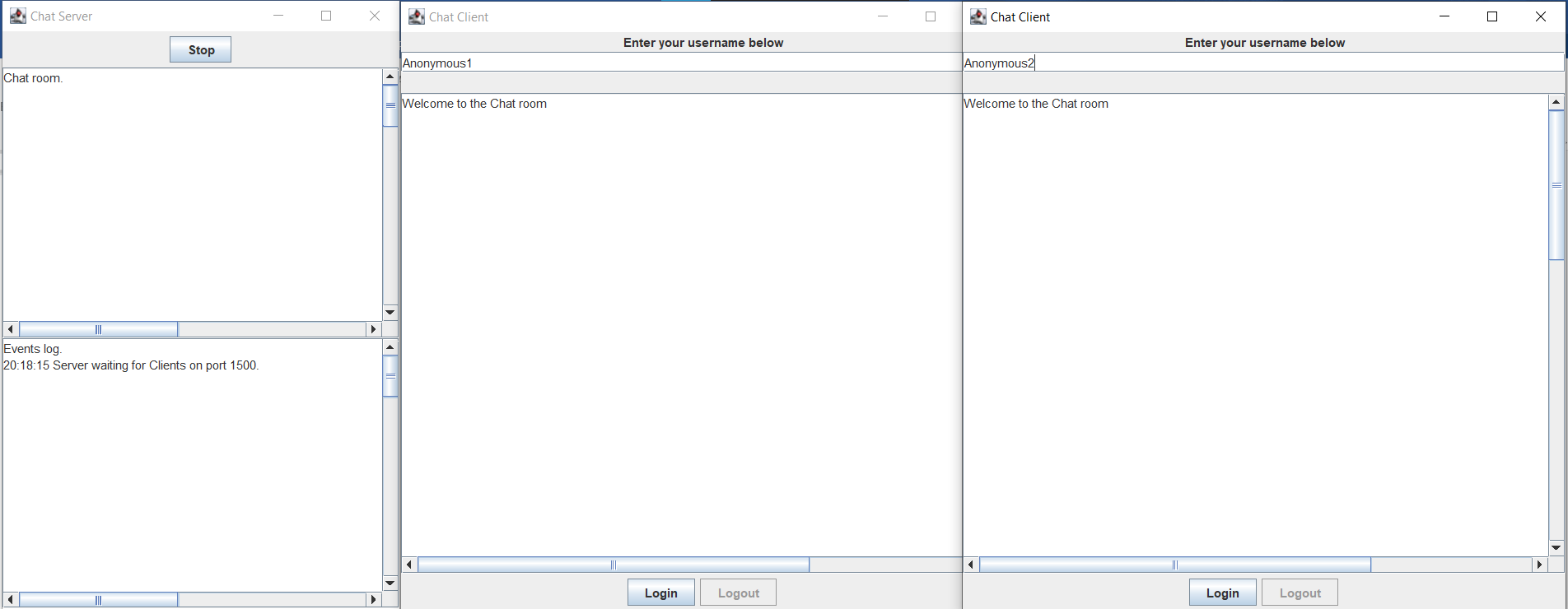
package Chat.Client;  
  
import Chat.Models.Answer;  
  
import java.net.\*;  
import java.io.\*;  
  
public class Client {  
 private ObjectInputStream inputStream;  
 private ObjectOutputStream outputStream;  
 private Socket socket;  
  
 private final ClientGUI client;  
 private final String server;  
 private final String username;  
 private final int port;  
  
 Client(String server, int port, String username, ClientGUI client) {  
 this.server = server;  
 this.port = port;  
 this.username = username;  
 this.client = client;  
 }  
  
 public boolean start() {  
 try {  
 socket = new Socket(server, port);  
 } catch (Exception ec) {  
 display("Error connecting to server:" + ec);  
 return false;  
 }  
 String msg = "Connection accepted " + socket.getInetAddress() + ":" + socket.getPort();  
 display(msg);  
  
 try {  
 inputStream = new ObjectInputStream(socket.getInputStream());  
 outputStream = new ObjectOutputStream(socket.getOutputStream());  
 } catch (IOException eIO) {  
 display("Exception creating new Input/output Streams: " + eIO);  
 return false;  
 }  
  
 new ListenFromServer().start();  
 try {  
 outputStream.writeObject(username);  
 } catch (IOException eIO) {  
 display("Exception doing login : " + eIO);  
 disconnect();  
 return false;  
 }  
  
 return true;  
 }  
  
 private void display(String msg) {  
 client.append(msg + "\n");  
 }  
  
 void sendMessage(Answer msg) {  
 try {  
 outputStream.writeObject(msg);  
 } catch (IOException e) {  
 display("Exception writing to server: " + e);  
 }  
 }  
  
 private void disconnect() {  
 try {  
 if (inputStream != null)  
 inputStream.close();  
  
 if (outputStream != null)  
 outputStream.close();  
  
 if (socket != null)  
 socket.close();  
 } catch (Exception ignored) {  
 }  
  
 if (client != null)  
 client.connectionFailed();  
 }  
  
 class ListenFromServer extends Thread {  
 public void run() {  
 while (true) {  
 try {  
 String msg = (String) inputStream.readObject();  
 if (client == null) {  
 System.*out*.println(msg);  
 System.*out*.print("> ");  
 } else {  
 client.append(msg);  
 }  
 } catch (IOException e) {  
 display("Server has close the connection: " + e);  
 if (client != null)  
 client.connectionFailed();  
 break;  
 } catch (ClassNotFoundException ignored) {  
 }  
 }  
 }  
 }  
}

package Chat.Client;  
  
import Chat.Models.Answer;  
import Chat.Models.MessageType;  
  
import javax.swing.\*;  
import java.awt.\*;  
import java.awt.event.\*;  
import java.io.Serial;  
  
public class ClientGUI extends JFrame implements ActionListener {  
 @Serial  
 private static final long *serialVersionUID* = 1L;  
  
 private final JLabel label;  
 private final JTextField tf;  
 private final JButton login;  
 private final JButton logout;  
 private final JTextArea ta;  
 private final int defaultPort;  
 private final String defaultHost;  
  
 private boolean connected;  
 private Client client;  
  
 ClientGUI(String host, int port) {  
 super("Chat Client");  
 defaultPort = port;  
 defaultHost = host;  
 JPanel northPanel = new JPanel(new GridLayout(3, 1));  
 label = new JLabel("Enter your username below", SwingConstants.*CENTER*);  
 northPanel.add(label);  
 tf = new JTextField("Anonymous");  
 tf.setBackground(Color.*WHITE*);  
 northPanel.add(tf);  
 add(northPanel, BorderLayout.*NORTH*);  
 ta = new JTextArea("Welcome to the Chat room\n", 80, 80);  
 JPanel centerPanel = new JPanel(new GridLayout(1, 1));  
 centerPanel.add(new JScrollPane(ta));  
 ta.setEditable(false);  
 add(centerPanel, BorderLayout.*CENTER*);  
 login = new JButton("Login");  
 login.addActionListener(this);  
 logout = new JButton("Logout");  
 logout.addActionListener(this);  
 logout.setEnabled(false);  
 JPanel southPanel = new JPanel();  
 southPanel.add(login);  
 southPanel.add(logout);  
 add(southPanel, BorderLayout.*SOUTH*);  
 setDefaultCloseOperation(*EXIT\_ON\_CLOSE*);  
 setSize(600, 600);  
 setVisible(true);  
 tf.requestFocus();  
 }  
  
 void append(String str) {  
 ta.append(str);  
 ta.setCaretPosition(ta.getText().length() - 1);  
 }  
  
 void connectionFailed() {  
 login.setEnabled(true);  
 logout.setEnabled(false);  
 label.setText("Enter your username below");  
 tf.setText("Anonymous");  
 tf.removeActionListener(this);  
 connected = false;  
 }  
  
 public void actionPerformed(ActionEvent e) {  
 Object o = e.getSource();  
 if (o == logout) {  
 client.sendMessage(new Answer(MessageType.*Logout*, ""));  
 return;  
 }  
  
 if (connected) {  
 client.sendMessage(new Answer(MessageType.*Answer*, tf.getText()));  
 tf.setText("");  
 return;  
 }  
 if (o == login) {  
 String username = tf.getText().trim();  
 if (username.length() == 0)  
 return;  
  
 client = new Client(defaultHost, defaultPort, username, this);  
  
 if (!client.start())  
 return;  
 tf.setText("");  
 label.setText("Enter your message below");  
 connected = true;  
  
 login.setEnabled(false);  
  
 logout.setEnabled(true);  
  
 tf.addActionListener(this);  
 }  
 }  
  
 public static void main(String[] args) {  
 new ClientGUI("localhost", 1500);  
 }  
}

package Chat.Server;  
  
import javax.swing.\*;  
import java.awt.\*;  
import java.awt.event.\*;  
import java.io.Serial;  
  
  
public class ServerGUI extends JFrame implements ActionListener, WindowListener {  
 @Serial  
 private static final long *serialVersionUID* = 1L;  
  
 private final JButton serverSwitcher;  
 private final JTextArea chatTextArea;  
 private final JTextArea eventTextArea;  
 private Server server;  
  
 ServerGUI() {  
 super("Chat Server");  
 server = null;  
  
 JPanel north = new JPanel();  
 serverSwitcher = new JButton("Start");  
 serverSwitcher.addActionListener(this);  
 north.add(serverSwitcher);  
 add(north, BorderLayout.*NORTH*);  
 JPanel center = new JPanel(new GridLayout(2, 1));  
 chatTextArea = new JTextArea(80, 80);  
 chatTextArea.setEditable(false);  
 appendRoom("Chat room.\n");  
 center.add(new JScrollPane(chatTextArea));  
 eventTextArea = new JTextArea(80, 80);  
 eventTextArea.setEditable(false);  
 appendEvent("Events log.\n");  
 center.add(new JScrollPane(eventTextArea));  
 add(center);  
 addWindowListener(this);  
 setSize(400, 600);  
 setVisible(true);  
 }  
  
 void appendRoom(String str) {  
 chatTextArea.append(str);  
 chatTextArea.setCaretPosition(chatTextArea.getText().length() - 1);  
 }  
  
 void appendEvent(String str) {  
 eventTextArea.append(str);  
 eventTextArea.setCaretPosition(chatTextArea.getText().length() - 1);  
 }  
  
 public void actionPerformed(ActionEvent e) {  
 if (server != null) {  
 server.stop();  
 server = null;  
 serverSwitcher.setText("Start");  
 return;  
 }  
  
 int defaultPort = 1500;  
 server = new Server(defaultPort, this);  
  
 new ServerRunning().start();  
 serverSwitcher.setText("Stop");  
 }  
  
 public static void main(String[] arg) {  
 new ServerGUI();  
 }  
  
 public void windowClosing(WindowEvent e) {  
 if (server != null) {  
 try {  
 server.stop();  
 } catch (Exception ignored) {  
 }  
 server = null;  
 }  
  
 dispose();  
 System.*exit*(0);  
 }  
  
 public void windowClosed(WindowEvent e) {  
 }  
  
 public void windowOpened(WindowEvent e) {  
 }  
  
 public void windowIconified(WindowEvent e) {  
 }  
  
 public void windowDeiconified(WindowEvent e) {  
 }  
  
 public void windowActivated(WindowEvent e) {  
 }  
  
 public void windowDeactivated(WindowEvent e) {  
 }  
  
 class ServerRunning extends Thread {  
 public void run() {  
 server.start();  
 serverSwitcher.setText("Start");  
 appendEvent("Server crashed\n");  
 server = null;  
 }  
 }  
}

package Chat.Server;  
  
import Chat.Models.Answer;  
  
import java.io.\*;  
import java.net.\*;  
import java.text.SimpleDateFormat;  
import java.util.\*;  
  
  
public class Server {  
 private static int *uniqueId*;  
 private final ArrayList<ClientThread> al;  
 private final ServerGUI sg;  
 private final SimpleDateFormat sdf;  
 private final int port;  
 private boolean keepGoing;  
  
 public Server(int port, ServerGUI sg) {  
 this.sg = sg;  
 this.port = port;  
 sdf = new SimpleDateFormat("HH:mm:ss");  
 al = new ArrayList<>();  
 }  
  
 public void start() {  
 keepGoing = true;  
 try {  
 ServerSocket serverSocket = new ServerSocket(port);  
 while (keepGoing) {  
 display("Server waiting for Clients on port " + port + ".");  
 Socket socket = serverSocket.accept();  
 if (!keepGoing)  
 break;  
 ClientThread t = new ClientThread(socket);  
 al.add(t);  
 t.start();  
 }  
  
 try {  
 serverSocket.close();  
 for (ClientThread tc : al) {  
 try {  
 tc.sInput.close();  
 tc.sOutput.close();  
 tc.socket.close();  
 } catch (IOException ignored) {  
 }  
 }  
 } catch (Exception e) {  
 display("Exception closing the server and clients: " + e);  
 }  
 } catch (IOException e) {  
 String msg = sdf.format(new Date()) + " Exception on new ServerSocket: "  
 + e + "\n";  
 display(msg);  
 }  
 }  
  
 protected void stop() {  
 keepGoing = false;  
 try {  
 new Socket("localhost", port);  
 } catch (Exception ignored) {  
 }  
 }  
  
 private void display(String msg) {  
 String time = sdf.format(new Date()) + " " + msg;  
 sg.appendEvent(time + "\n");  
 }  
  
 private synchronized void broadcast(String message) {  
 String time = sdf.format(new Date());  
 String messageLf = time + " " + message + "\n";  
 if (sg == null)  
 System.*out*.print(messageLf);  
 else  
 sg.appendRoom(messageLf);  
  
 for (int i = al.size(); --i >= 0; ) {  
 ClientThread ct = al.get(i);  
 if (!ct.writeMsg(messageLf)) {  
 al.remove(i);  
 display("Disconnected Client " + ct.username + " removed from list.");  
 }  
 }  
 }  
  
 synchronized void remove(int id) {  
 for (int i = 0; i < al.size(); ++i) {  
 ClientThread ct = al.get(i);  
 if (ct.id == id) {  
 al.remove(i);  
 return;  
 }  
 }  
 }  
  
 class ClientThread extends Thread {  
 Socket socket;  
 ObjectInputStream sInput;  
 ObjectOutputStream sOutput;  
  
 int id;  
 String username;  
 Answer cm;  
 String date;  
  
 ClientThread(Socket socket) {  
 id = ++*uniqueId*;  
 this.socket = socket;  
 System.*out*.println("Thread trying to create Object Input/Output Streams");  
 try {  
 sOutput = new ObjectOutputStream(socket.getOutputStream());  
 sInput = new ObjectInputStream(socket.getInputStream());  
 username = (String) sInput.readObject();  
 display(username + " just connected.");  
 } catch (IOException e) {  
 display("Exception creating new Input/output Streams: " + e);  
 return;  
 } catch (ClassNotFoundException ignored) {  
 }  
 date = new Date().toString() + "\n";  
 }  
  
 public void run() {  
 boolean keepGoing = true;  
 while (keepGoing) {  
 try {  
 cm = (Answer) sInput.readObject();  
 } catch (IOException e) {  
 display(username + " Exception reading Streams: " + e);  
 break;  
 } catch (ClassNotFoundException e2) {  
 break;  
 }  
  
 String message = cm.getMessage();  
 switch (cm.getType()) {  
 case *Answer* -> broadcast(username + ": " + message);  
 case *Logout* -> {  
 display(username + " disconnected with a LOGOUT message.");  
 keepGoing = false;  
 }  
 }  
 }  
 remove(id);  
 close();  
 }  
  
 private void close() {  
 try {  
 if (sOutput != null) sOutput.close();  
 if (sInput != null) sInput.close();  
 if (socket != null) socket.close();  
 } catch (Exception e) {  
 display("Exception while close: " + e);  
 }  
 }  
  
 private boolean writeMsg(String msg) {  
 if (!socket.isConnected()) {  
 close();  
 return false;  
 }  
 try {  
 sOutput.writeObject(msg);  
 } catch (IOException e) {  
 display("Error sending message to " + username);  
 display(e.toString());  
 }  
 return true;  
 }  
 }  
}

Результат работы:

****

