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)
Афанасьев В.В.

Проверил:
Крощенко А.А.

Брест, 2021

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

Вариант: 2

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.

Код программы:

Message

```
package NordChat;

import java.io.*;

public class Message implements Serializable {
    protected static final long serialVersionUID = 1112122200L;

    static final
        int WHOISIN = 0,
            MESSAGE = 1,
            LOGOUT = 2;

    private int type;
    private String message;

    Message(int type, String message) {
        this.type = type;
        this.message = message;
    }

    int getType() {
        return type;
    }

    String getMessage() {
        return message;
    }
}
```

Server

```
package NordChat;

import java.io.*;
import java.net.*;
import java.text.SimpleDateFormat;
import java.util.*;

/*
 * This server can be run as a console application or as a GUI
 * To run as a console application just:
 * > java Server
 * > java Server portNumber
 * If the port number is not specified 1200-port is used
 */

public class Server {
    private static int uniqueId;                // An unique ID-code for each connection
```

```
private ArrayList<ClientThread> clients;    // The list of the Clients

private ServerGUI serverAsGUI;              // An object of ServerGUI (for gui running)

private SimpleDateFormat simpleDateFormat;
private int port;

private boolean isRunning;                  // The state of the server (running/is stop)

public Server(int port) {                   // For a console running
    this(port, null);
}

public Server(int port, ServerGUI serverAsGUI) {
    this.serverAsGUI = serverAsGUI;
    this.port = port;
    simpleDateFormat = new SimpleDateFormat("HH:mm:ss");
    clients = new ArrayList<ClientThread>();
}

public void start() {
    isRunning = true;
    try {
        ServerSocket serverSocket = new ServerSocket(port);
        while (isRunning) {
            display("Server is waiting for Guests on the " + port + " port.");
            Socket socket = serverSocket.accept();           // Accept the connection

            if (!isRunning)
                break;

            ClientThread thread = new ClientThread(socket);  // Make a thread for it
            clients.add(thread);                             // Saving in the Clients list
            thread.start();
        }

        try {
            serverSocket.close();

            for (int i = 0; i < clients.size(); ++i) {
                ClientThread clientThread = clients.get(i);
                try {
                    clientThread.sInput.close();
                    clientThread.sOutput.close();
                    clientThread.socket.close();
                }
                catch (IOException ioException) {}
            }
        }
        catch (Exception exception) {
            display("Exception closing the server and guests: " + exception);
        }
    }

    catch (IOException ioException) {
        String message = simpleDateFormat.format(new Date())
                + " IOException on the new ServerSocket: " + ioException + "\n";
        display(message);
    }
}

protected void stop() {
    isRunning = false;

    try {
        new Socket("localhost", port);
    }
    catch (Exception exception) {}
}

private void display(String message) {                  // Displaying the event
    String time = simpleDateFormat.format(new Date()) + " " + message;
    if (serverAsGUI == null)
```

```
            System.out.println(time);
        else
            serverAsGUI.appendEvent(time + "\n");
    }

    private synchronized void broadcast(String message) {
        String time = simpleDateFormat.format(new Date());
        String messageLf = time + " " + message + "\n";

        if (serverAsGUI == null)
            System.out.print(messageLf);
        else
            serverAsGUI.appendRoom(messageLf);          // Append in the room window

        for (int i = clients.size(); --i >= 0; ) {      // The loop in the reverse order because
of the opportunity
                                                        // to deleting disconected guest
            ClientThread clientThread = clients.get(i);

            if (!clientThread.writeMessage(messageLf)) {
                clients.remove(i);
                display("Disconnected Guest " + clientThread.username);
            }
        }
    }

    synchronized void remove(int id) {
        for (int i = 0; i < clients.size(); ++i) {
            ClientThread clientThread = clients.get(i);

            if (clientThread.id == id) {
                clients.remove(i);
                return;
            }
        }
    }

    // ----

    public static void main(String[] args) {
        int portNumber = 1200;
        switch (args.length) {
            case 1:
                try {
                    portNumber = Integer.parseInt(args[0]);
                }
                catch (Exception exception) {
                    System.out.println("Invalid port number.");
                    System.out.println("Usage is: > java Server [portNumber]");
                    return;
                }
            case 0:
                break;
            default:
                System.out.println("Usage is: > java Server [portNumber]");
                return;
        }

        Server server = new Server(portNumber);
        server.start();
    }

    class ClientThread extends Thread {
        Socket socket;

        ObjectInputStream sInput;
        ObjectOutputStream sOutput;

        int id;
        String username;
        Message clientMessage;
        String currentDate;

        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 + " is connected.");
            }
            catch (IOException exception) {
                display("Exception creating new Input/output Streams: " + exception);
                return;
            }
            catch (ClassNotFoundException exception) {
            }

            currentDate = new Date().toString() + "\n";
        }

        public void run() {
            boolean keepGoing = true;
            while (keepGoing) {
                try {
                    clientMessage = (Message) sInput.readObject();
                }
                catch (IOException exception) {
                    display(username + " Exception reading Streams: " + exception);
                    break;
                }
                catch (ClassNotFoundException exception) {
                    break;
                }

                String message = clientMessage.getMessage();            // The message part

                switch (clientMessage.getType()) {                      // Switcher of the types
                    case Message.MESSAGE:
                        broadcast(username + ": " + message);
                        break;
                    case Message.LOGOUT:
                        display(username + " disconnected with a LOGOUT message.");
                        keepGoing = false;
                        break;
                    case Message.WHOISIN:
                        writeMessage("List of the guests connected at " +
simpleDateFormat.format(new
                                Date()) + "\n");

                        for (int i = 0; i < clients.size(); ++i) {
                            ClientThread clientThread = clients.get(i);
                            writeMessage((i + 1) + ") " + clientThread.username + " since " +
                                    clientThread.currentDate);
                        }
                        break;
                }
            }

            remove(id);             // Removing adm-id
            close();
        }

        private void close() {
            try {
                if (sOutput != null)
                    sOutput.close();
            }
            catch (Exception exception) {}

            try {
                if (sInput != null)
                    sInput.close();
            }
```

```
            catch (Exception exception) {}

            try {
                if (socket != null)
                    socket.close();
            }
            catch (Exception exception) {}
        }

        private boolean writeMessage(String message) {
            if (!socket.isConnected()) {
                close();
                return false;
            }

            try {
                sOutput.writeObject(message);
            }
            catch (IOException exception) {
                display("Error sending message to " + username);
                display(exception.toString());
            }

            return true;
        }
    }
}
```

ServerGUI

```
package NordChat;

import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class ServerGUI extends JFrame implements ActionListener, WindowListener {
    private static final long serialVersionUID = 1L;

    private JButton stopStart;
    private JTextArea chat, event;
    private JTextField tPortNumber;
    private Server server;

    ServerGUI(int port) {
        super("NordChat (Server)");
        server = null;

        JPanel nord = new JPanel();

        nord.add(new JLabel("Port: "));

        tPortNumber = new JTextField(" " + port);
        nord.add(tPortNumber);

        stopStart = new JButton("Start");
        stopStart.addActionListener(this);
        nord.add(stopStart);

        add(nord, BorderLayout.NORTH);                              // The event and chat room

        JPanel center = new JPanel(new GridLayout(2, 1));

        chat = new JTextArea(90, 90);
        chat.setEditable(false);
        appendRoom("Chating room\n");
        center.add(new JScrollPane(chat));

        event = new JTextArea(90, 90);
        event.setEditable(false);
        appendEvent("Events log.\n");

        center.add(new JScrollPane(event));
        add(center);
```

```
    addWindowListener(this);
    setSize(400, 500);
    setVisible(true);
}

void appendRoom(String str) {                    // Append a message to the two JTextAreas
    chat.append(str);
    chat.setCaretPosition(chat.getText().length() - 1);
}

void appendEvent(String str) {
    event.append(str);
    event.setCaretPosition(chat.getText().length() - 1);
}

public void actionPerformed(ActionEvent event) {
    if (server != null) {
        server.stop();
        server = null;
        tPortNumber.setEditable(true);
        stopStart.setText("Starting");

        return;
    }

    int port = 0;
    try {
        port = Integer.parseInt(tPortNumber.getText().trim());
    }
    catch (Exception exception) {
        appendEvent("Invalid port number");

        return;
    }

    server = new Server(port, this);

    new ServerRunning().start();

    stopStart.setText("Stop");
    tPortNumber.setEditable(false);
}

public static void main(String[] arg) {             // Starting the server
    new ServerGUI(1200);
}

public void windowClosing(WindowEvent event) {      // Closing by Windows event
    if (server != null) {
        try {
            server.stop();
        }
        catch (Exception eClose) {}
        server = null;
    }

    dispose();
    System.exit(0);
}

public void windowClosed(WindowEvent event) {}
public void windowOpened(WindowEvent event) {}
public void windowIconified(WindowEvent event) {}
public void windowDeiconified(WindowEvent event) {}
public void windowActivated(WindowEvent event) {}
public void windowDeactivated(WindowEvent event) {}

class ServerRunning extends Thread {
    public void run() {
        server.start();                          // Should executing until if fails

        stopStart.setText("Start");
        tPortNumber.setEditable(true);
```

```
            appendEvent("Server is stopped\n");
            server = null;
        }
    }
}
```

Client

```
package NordChat;

import java.net.*;
import java.io.*;
import java.util.*;

/*
 * To start the Client-part in console mode use one of the following command
 * > java Client
 * > java Client username
 * > java Client username portNumber
 * > java Client username portNumber serverAddress
 * at the console prompt
 * If the portNumber is not specified 1200 is used
 * If the serverAddress is not specified "localHost" is used
 * If the username is not specified "Guest" is used
 * > java Client
 * > java Client Anonymous 1200 localhost
 */

public class Client {
    private ObjectInputStream sInput;               // To read from the socket
    private ObjectOutputStream sOutput;             // To write to the socket

    private Socket socket;

    private ClientGUI clientGUI;

    private String server, username;
    private int port;

    Client(String server, int port, String username) {
        this(server, port, username, null);
    }

    Client(String server, int port, String username, ClientGUI clientGUI) {
        this.server = server;
        this.port = port;
        this.username = username;
    }

    public boolean start() {
        try {
            socket = new Socket(server, port);
        }

        catch (Exception exception) {
            display("Error connectiong to server:" + exception);
            return false;
        }

        String message = "Connection accepted " + socket.getInetAddress() + ":" +
socket.getPort();
        display(message);

        try {
            sInput = new ObjectInputStream(socket.getInputStream());        // Creating data
streams
            sOutput = new ObjectOutputStream(socket.getOutputStream());
        }
        catch (IOException IOException) {
            display("Exception creating new Input/output Streams: " + IOException);
            return false;
        }

        new ListenFromServer().start();                 // Create the thread to listen from the
```

server

```
        try {
            sOutput.writeObject(username);
        }
        catch (IOException IOException) {
            display("Exception doing login : " + IOException);
            disconnect();
            return false;
        }

        return true;
    }

    private void display(String message) {          // To send a message to the console or to
the GUI
        if (clientGUI == null)
            System.out.println(message);            // Console mode
        else
            clientGUI.append(message + "\n");       // ClientGUI JTextArea
    }

    void sendMessage(Message message) {             // Send to the server
        try {
            sOutput.writeObject(message);
        }
        catch (IOException exception) {
            display("Exception writing to server: " + exception);
        }
    }

    private void disconnect() {
        try {
            if (sInput != null) sInput.close();
        }
        catch (Exception exception) {}
        try {
            if (sOutput != null) sOutput.close();
        } catch (Exception exception) {
        }
        try {
            if (socket != null) socket.close();
        }
        catch (Exception exception) {}

        if (clientGUI != null)                          // Inform the GUI-part
            clientGUI.connectionFailed();
    }

    public static void main(String[] args) {
        int portNumber = 1200;
        String serverAddress = "localhost";
        String userName = "Guest";

        switch (args.length) {
            case 3:
                serverAddress = args[2];
            case 2:
                try {
                    portNumber = Integer.parseInt(args[1]);
                }
                catch (Exception exception) {
                    System.out.println("Invalid port number.");
                    System.out.println("Usage is: > java Client [username] [portNumber]
[serverAddress]");
                    return;
                }
            case 1:
                userName = args[0];
            case 0:
                break;
            default:
                System.out.println("Usage is: > java Client [username] [portNumber]
{serverAddress]");
```

```
                return;
            }

            Client client = new Client(serverAddress, portNumber, userName);

            if (!client.start())
                return;

            Scanner scan = new Scanner(System.in);      // Waiting for messages from the user

            while (true) {
                System.out.print("> ");

                String message = scan.nextLine();           // Reading the message

                if (message.equalsIgnoreCase("LOGOUT")) {
                    client.sendMessage(new Message(Message.LOGOUT, ""));
                    break;
                }
                else if (message.equalsIgnoreCase("WHOISIN")) {
                    client.sendMessage(new Message(Message.WHOISIN, ""));
                }
                else {
                    client.sendMessage(new Message(Message.MESSAGE, message));
                }
            }

            client.disconnect();
        }

    class ListenFromServer extends Thread {     // Waits for the message from the server and
append them to the
                                                // JTextArea or to the console mode
        public void run() {
            while (true) {
                try {
                    String message = (String) sInput.readObject();
                    if (clientGUI == null) {
                        System.out.println(message);
                        System.out.print("> ");
                    }
                    else {
                        clientGUI.append(message);
                    }
                }
                catch (IOException exception) {
                    display("Server has close the connection: " + exception);
                    if (clientGUI != null)
                        clientGUI.connectionFailed();
                    break;
                }
                catch (ClassNotFoundException exception) {}
            }
        }
    }
}
```

ClientGUI

```
package NordChat;

import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class ClientGUI extends JFrame implements ActionListener {
    private static final long serialVersionUID = 1L;

    private JLabel label;               // For username/enter message

    private JTextField textField;

    private JTextField tfServer, tfPort;
```

```
private JButton login, logout, whoIsIn;

private JTextArea textArea;

private boolean connected;

private Client client;

private int defaultPort;
private String defaultHost;

ClientGUI(String host, int port) {                  // Receive a socket number
    super("Chat Client");
    defaultPort = port;
    defaultHost = host;

    JPanel nordPanel = new JPanel(new GridLayout(3,1));

    JPanel serverAndPort = new JPanel(new GridLayout(1,5, 1, 3));

    tfServer = new JTextField(host);
    tfPort = new JTextField("" + port);
    tfPort.setHorizontalAlignment(SwingConstants.RIGHT);

    serverAndPort.add(new JLabel("Address: "));
    serverAndPort.add(tfServer);

    serverAndPort.add(new JLabel("Port: "));
    serverAndPort.add(tfPort);

    serverAndPort.add(new JLabel(""));

    nordPanel.add(serverAndPort);

    label = new JLabel("Enter your username ", SwingConstants.CENTER);
    nordPanel.add(label);

    textField = new JTextField("Guest");
    textField.setBackground(Color.WHITE);
    nordPanel.add(textField);

    add(nordPanel, BorderLayout.NORTH);

    textArea = new JTextArea("Welcome to the chat room\n", 70, 70);
    JPanel centralPanel = new JPanel(new GridLayout(1,1));
    centralPanel.add(new JScrollPane(textArea));

    textArea.setEditable(false);
    add(centralPanel, BorderLayout.CENTER);

    login = new JButton("Login");
    login.addActionListener(this);

    logout = new JButton("Logout");
    logout.addActionListener(this);
    logout.setEnabled(false);

    whoIsIn = new JButton("Who is in");
    whoIsIn.addActionListener(this);
    whoIsIn.setEnabled(false);

    JPanel downPanel = new JPanel();
    downPanel.add(login);
    downPanel.add(logout);
    downPanel.add(whoIsIn);

    add(downPanel, BorderLayout.SOUTH);
    setDefaultCloseOperation(EXIT_ON_CLOSE);

    setSize(400, 400);
    setVisible(true);

    textField.requestFocus();
}
```

```
void append(String str) {           // Appending the text in the TextArea
    textArea.append(str);
    textArea.setCaretPosition(textArea.getText().length() - 1);
}

void connectionFailed() {
    login.setEnabled(true);
    logout.setEnabled(false);
    whoIsIn.setEnabled(false);

    label.setText("Enter your username");
    textField.setText("Guest");

    tfPort.setText("" + defaultPort);tfServer.setText(defaultHost);

    tfServer.setEditable(false);
    tfPort.setEditable(false);

    textField.removeActionListener(this);
    connected = false;
}

public void actionPerformed(ActionEvent event) {
    Object tempObject = event.getSource();

    if (tempObject == logout) {
        connectionFailed();
        client.sendMessage(new Message(Message.LOGOUT, ""));
        return;
    }

    if (tempObject == whoIsIn) {
        client.sendMessage(new Message(Message.WHOISIN, ""));
        return;
    }

    if (connected) {                  // Come from the JTextField
        client.sendMessage(new Message(Message.MESSAGE, textField.getText()));
        textField.setText("");
        return;
    }

    if (tempObject == login) {
        String username = textField.getText().trim();           // A connection request

        if (username.length() == 0)
            return;

        String server = tfServer.getText().trim();
        if (server.length() == 0)
            return;

        String portNumber = tfPort.getText().trim();
        if (portNumber.length() == 0)
            return;

        int port = 0;
        try {
            port = Integer.parseInt(portNumber);
        }
        catch(Exception exception) {
            return;
        }

        client = new Client(server, port, username, this);

        if (!client.start())
            return;

        textField.setText("");label.setText("Enter your message below");
        connected = true;

        login.setEnabled(false);
```

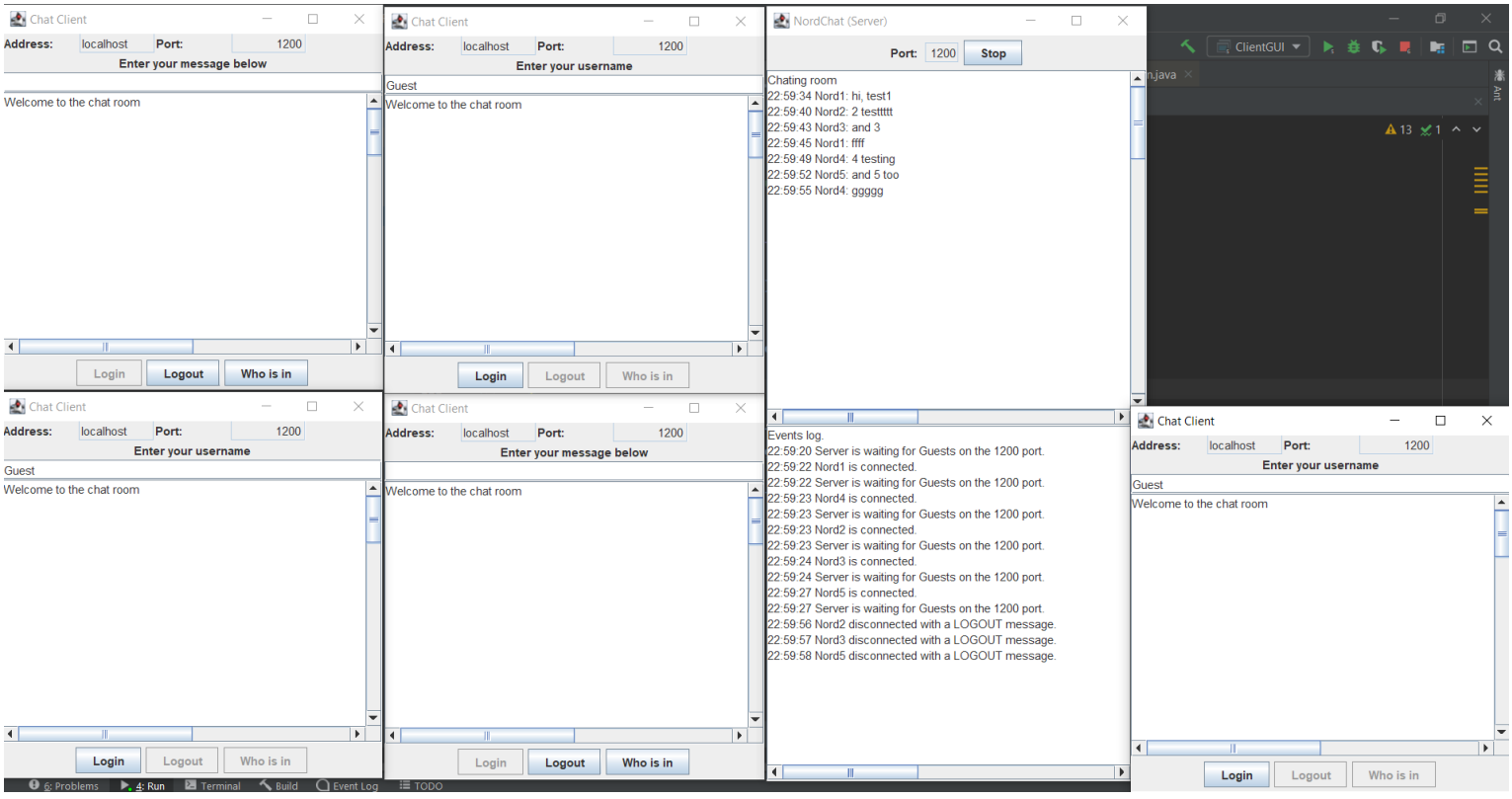
```
            logout.setEnabled(true);
            whoIsIn.setEnabled(true);

            tfServer.setEditable(false);
            tfPort.setEditable(false);

            textField.addActionListener(this);      // Action listener for when the user enter a
message
        }
    }

    public static void main(String[] args) {
        new ClientGUI("localhost", 1200);
    }    // Start the whole server
    @SuppressWarnings("unchecked")
    private void initComponents() {
        setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);
        javax.swing.GroupLayout layout = new javax.swing.GroupLayout(getContentPane());
        getContentPane().setLayout(layout);
        layout.setHorizontalGroup(
                layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                        .addGap(0, 400, Short.MAX_VALUE)
        );
        layout.setVerticalGroup(
                layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                        .addGap(0, 300, Short.MAX_VALUE)
        );
        pack();
    }
}
```

Результаты работы:



Выводы: в ходе выполнения лабораторной работы были освоены приемы разработки оконных клиент-серверных приложений на Java с использованием сокетов.