ФЕДЕРАЛЬНОЕ АГЕНТСТВО СВЯЗИ
Федеральное государственное образовательное бюджетное учреждение высшего профессионального образования
«Санкт-Петербургский государственный университет телекоммуникаций им. проф. М. А. Бонч-Бруевича»

Факультет кибербезопасности
Кафедра ЗСС

ОТЧЕТ

По лабораторной работе №11

по дисциплине
«Разработка защищенных сетевых приложений»

«СОКЕТНОЕ СОЕДИНЕНИЕ»

Выполнил студент
группы ИКБ-33

Киреев К. К.

Санкт-Петербург
2025 г

Цель лабораторной работы

Освоить работу с созданием сетевых приложений в Java.

Задание

Написать клиент-серверные программы, организующие передачу текстовых сообщений от одного клиента к другому через сервер. Клиенты должны выводит текстовые сообщения в графическое окно.

ХОД РАБОТЫ

При выполнении лабораторной работы были написаны следующие классы:

1. `public class main.`
   Основной класс приложения, содержащий входную точку исполнения `main.` При запуске программы проверяется количество аргументов, с которыми была запущена программа. Если программа была запущена без аргументов, то запускается клиент `ClientGUI`. Если программа была запущена с аргументом, то этот аргумент интерпретируется как порт и программа запускает сервер на этом порту.

2. `public class ChatServerSocket extends ServerSocket`
   Класс, отвечающий за серверную часть приложения. Имеет поле `session`, являющееся списком объектов `Session` – установленных с пользователями сессий
   Методы:
   a. `start()` - Запускает основной цикл. В этом цикле метод `.accept()` блокирует основной поток до тех пор, пока не будет установлено новое соединение. В результате установки соединения получается экземпляр Socket с помощью которого создается новый `Session,` запускается в новом потоке и добавляется в список сессий.
   b. `broadcast(String msg, Session sender)` - Отправляет сообщение `msg` во все сессии `Session`, кроме `sender`, если он указан

3. `public class Session implements Runnable`
   Внутренний класс `ChatServerSocket`. Конструктор принимает на вход экземпляр класса Socket и создает интерфейсы для чтения и отправки сообщений.
   Методы:
   a. `run()` - Запускает исполнение. Считывает первое сообщение от клиента и интерпретирует его как имя пользователя. Далее запускается цикл, в котором вызов `in.readline()` блокирует поток до тех пор, пока не получит сообщение от клиента. После получения сообщение отсылается остальным клиентом при помощи метода `ChatServerSocket.broadcast`
   b. `sendMsg(String msg)`- Отправляет сообщение `msg` клиенту

4. `public class ClientGUI extends JFrame`
   Основной класс клиента. Обеспечивает взаимодействие остальных компонентов клиента.
   Методы:
   a. `launch()` - Получает от пользователя данные для подключения с помощью компонента `LoginPopup`. При успешном получении запускает `ChatClientSocket`.
   b. `quit()` - Завершает исполнение программы

5. `public class MessageEvent extends EventObject`
   Класс события, обозначающего отправку сообщения
   Методы:
   a. `getContent()` – Возвращает содержание сообщения
   b. `getSender()` – Возвращает отправителя сообщения

6. `public interface MessageListener extends EventListener`
   Интерфейс обработчика событий `MessageEvent`
   Методы:
   a. `newMessage(MessageEvent event)` – оповещает `MessageListener` о новом событии `MessageEvent`

7. `public class ChatClientSocket extends Socket`
   Отвечает за взаимодействие с сервером.
   Конструктор `ChatClientSocket(InetAddress address, int port, StringBuffer username)` устанавливает соединение с сервером, создает интерфейсы для чтения и отправки сообщений, а также определяет `MessageListener` для отправки сообщений на сервер
   Методы:
   a. `addListener(MessageListener l)` – добавляет `MessageListener l` в список для оповещения о новом сообщении с сервера
   b. `getMessageListener()` – возвращает свой `MessageListener`, отправляющий сообщение на сервер.

8. `private class InputHandler implements Runnable`
   Внутренний класс `ChatClientSocket`. Отвечает за прием сообщений с сервера. Запускается в отдельном потоке
   Методы:
   a. `run()` - Запускает исполнение. Содержит цикл, в котором вызов `in.readline()` блокирует поток до тех пор, пока не получит сообщение от сервера. При получении сообщения формирует `MessageEvent` и оповещает о нем всех зарегистрированных `MessageListener.`

9. `public class ChatBox extends JPanel`
   Компонент, осуществляющий вывод полученных сообщений на экран
   Методы:
   a. `getMessageListener()` – возвращает свой `MessageListener`, выводящий сообщение на экран.

10. `public class SendBox extends JPanel`
    Компонент, отвечающий за прием сообщений от пользователя.
    Методы:
    a. `addListener(MessageListener l)` - добавляет `MessageListener l` в список для оповещения о новом сообщении от пользователя

11. `public class LoginPopup`

    Компонент, отвечающий за получение от пользователя данных для подключения.
    Методы:

    a. run() – Запускает исполнение. Раз за разом показывает пользователю форму для ввода данных, пока он не введет правильные данные, или не захочет закрыть окно.
    b. getAddress() – возвращает адрес сервера, введенный пользователем
    c. getPort() – возвращает номер порта, введенный пользователем
    d. getUsername() – возвращает юзернейм, введенный пользователем

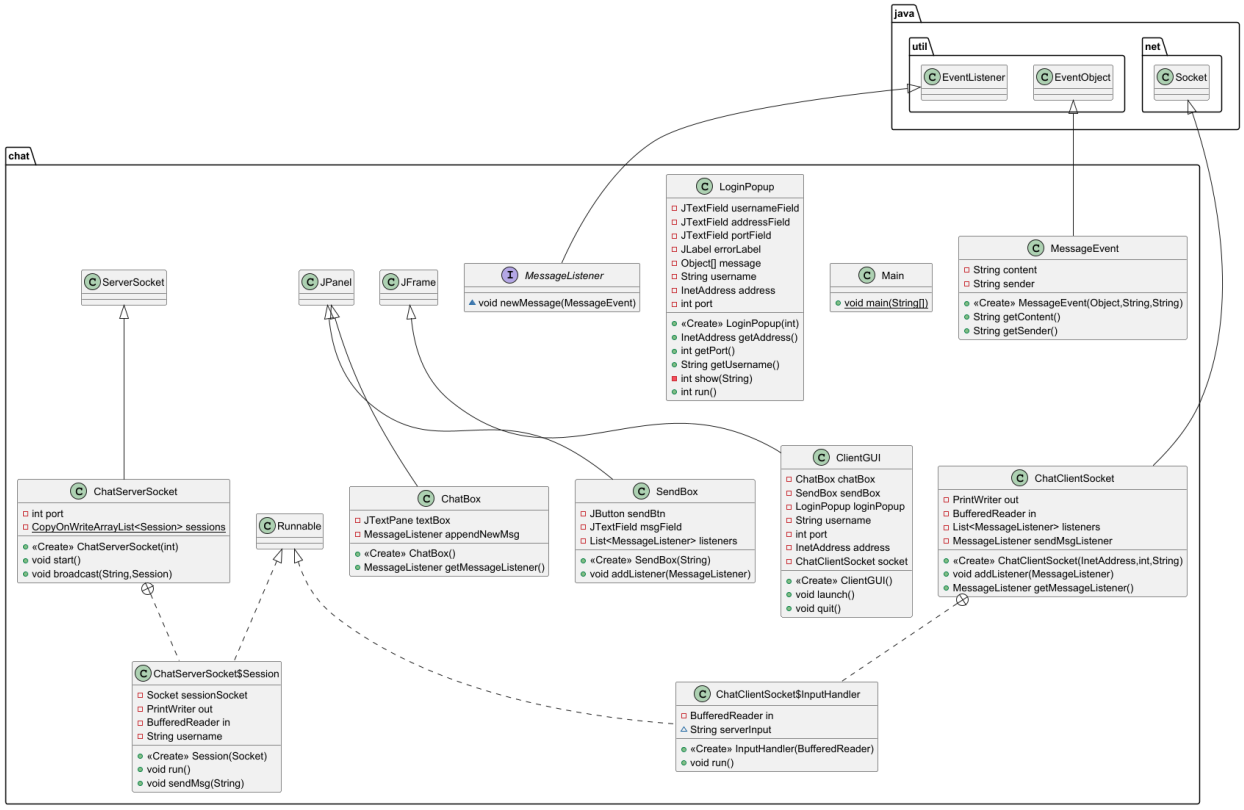

Рисунок 1. UML-диаграмма классов

Таким образом было реализовано клиент-серверное приложение для обмена сообщениями, поддерживающее несколько клиентов

Примеры работы программы:

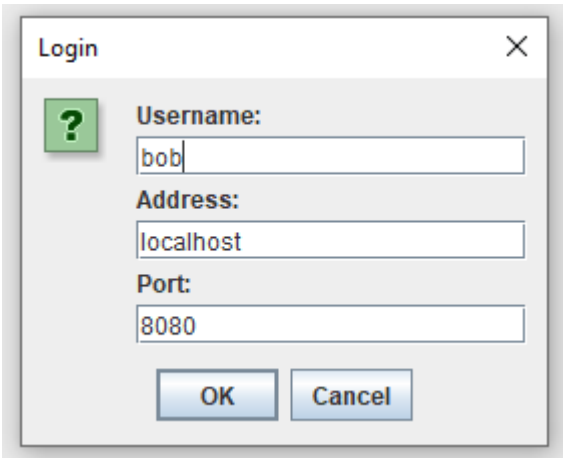

Рисунок 2. Окно ввода данных для подключения

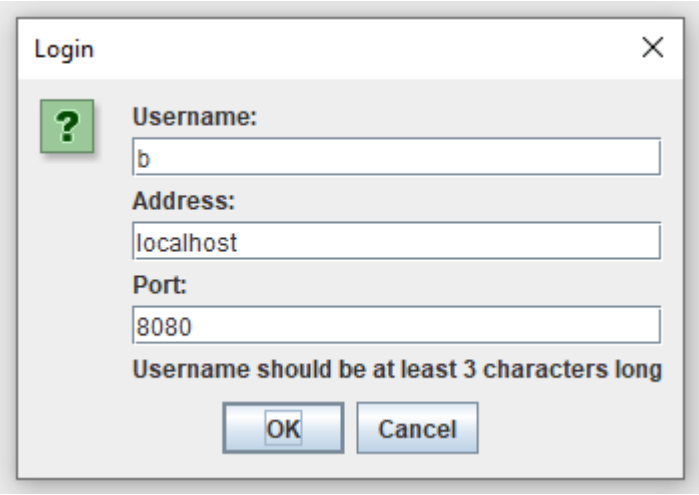

Рисунок 3. Неправильные данные

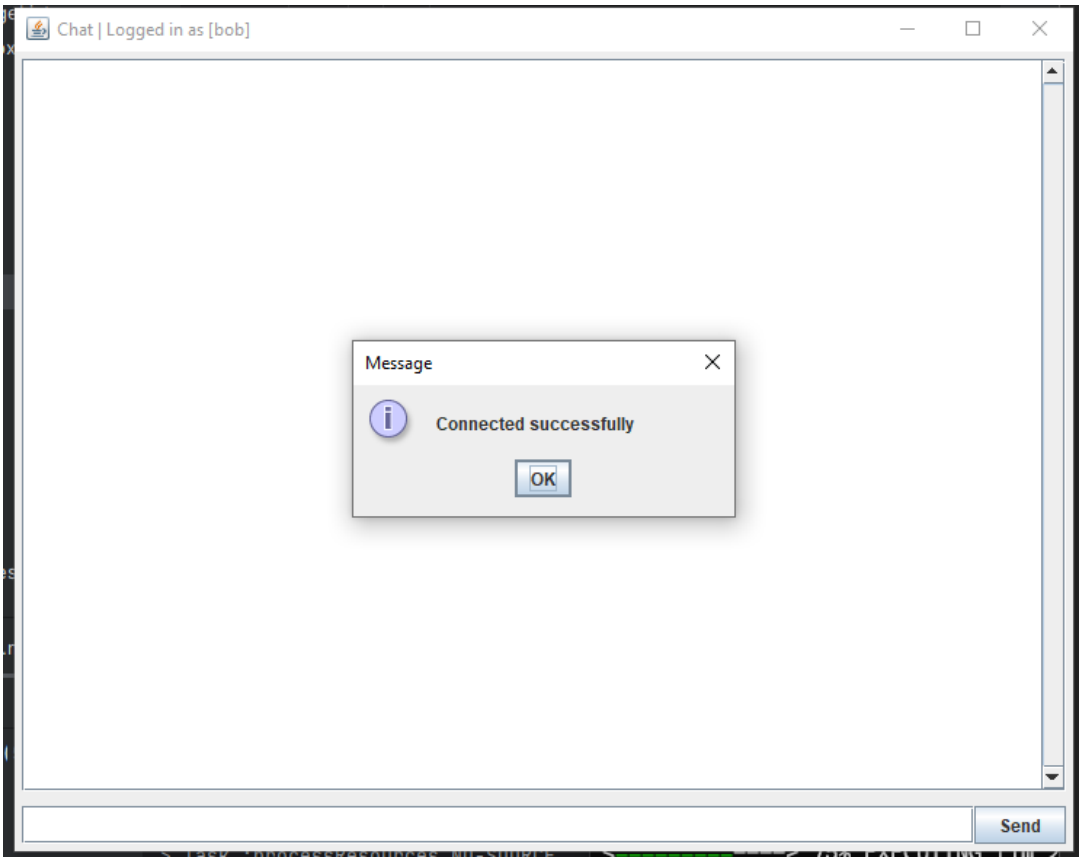

Рисунок 4. Успешное подключение

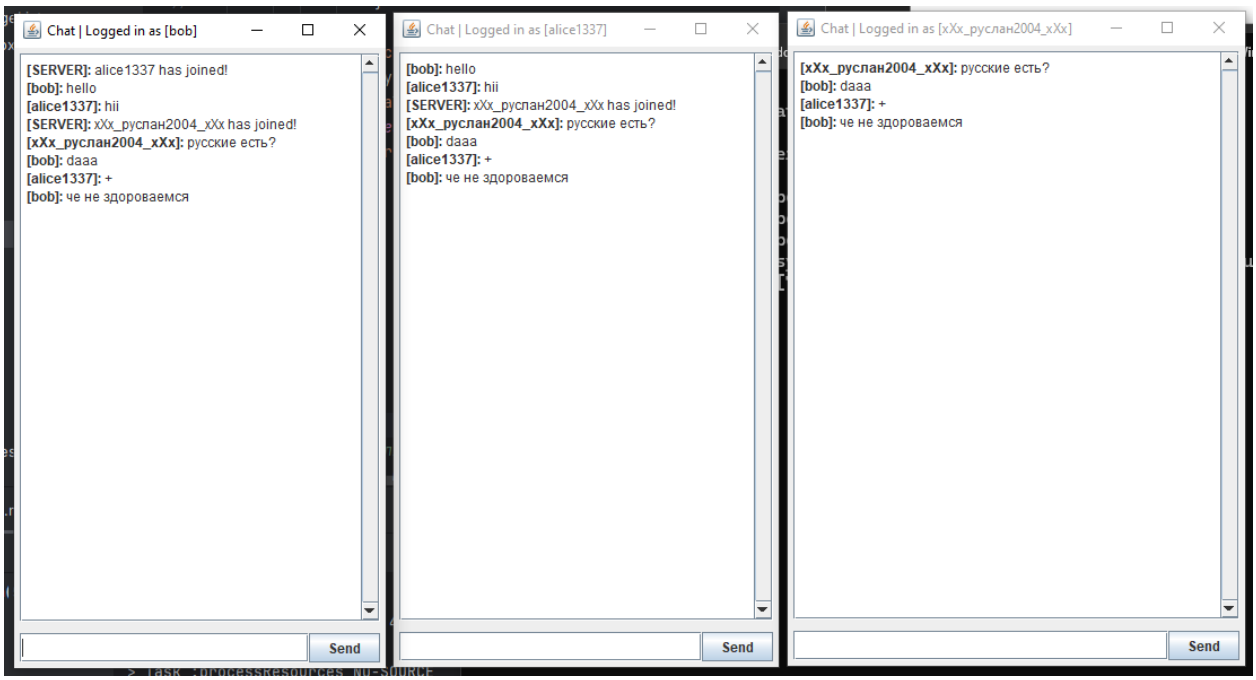

Рисунок 5. Пример взаимодействия нескольких клиентов

```
PS C:\Users\admin\Desktop\3 курс 5 сем\разница\РЗСП\Lab11> ./gradlew run --args="8080"
Starting a Gradle Daemon, 1 incompatible and 5 stopped Daemons could not be reused, use --status for details

> Task :run
Server started on port 8080
Client [127.0.0.1] connected
[SERVER]bob has joined!
Client [127.0.0.1] connected
[SERVER]alice1337 has joined!
[bob]hello
[alice1337]hii
Client [127.0.0.1] connected
[SERVER]xXx_руслан2004_xXx has joined!
[xXx_руслан2004_xXx]русские есть?
[bob]daaa
[alice1337]+
[bob]че не здороваемся
<=========----> 75% EXECUTING [16m 40s]
> :run
```

Рисунок 6. Сервер

Вывод: В этой лабораторной работе, при помощи WebSockets, было реализовано приложение с графическим пользовательским интерфейсом, позволяющее нескольким клиентам обмениваться сообщениями через сервер.

ПРИЛОЖЕНИЕ №1
Исходный код

Листинг 1. Main.java

```
package chat;

import java.io.IOException;

/**
 * Основной класс
 * Для запуска клиента: Lab11.jar
 * Для запуска сервера: Lab11.jar [port]
 */
public class Main {
    public static void main(String[] args) {
        // Если нет аргументов, то запускаем клиент
        if(args.length == 0) {
            ClientGUI gui = new ClientGUI();
            gui.launch();
        }
        // Иначе, запускаем сервер
        else {
            int port;
            // Валидация номера порта, переданного пользователем
            try {
                port = Integer.parseInt(args[0]);
            } catch (NumberFormatException e) {
                System.out.println("Incorrect port number");
                System.out.println("Usage:");
                System.out.println("    To start client: Lab11.jar");
                System.out.println("    To start server: Lab11.jar [port]");
                return;
            }
            // Запуск сервера
            try (ChatServerSocket server = new ChatServerSocket(port)) {
                server.start();
            } catch (IOException e) {
                System.out.println(e);
            }
        }
    }
}
```

Листинг 2. ChatServerSocket.java

```
package chat;

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

public class ChatServerSocket extends ServerSocket{
    private final int port;

    private static final CopyOnWriteArrayList<Session> sessions = new
CopyOnWriteArrayList<>();

    public ChatServerSocket(int port) throws IOException {
        super(port);
        this.port = port;
        System.out.printf("Server started on port %d\n", this.port);
    }

    /**
     * Запускает основной цикл. Метод <code>.accept()</code>
     * блокирует поток, пока не будет установлено новое соединение.
     * В результате установки соединения получается экземпляр
<code>Socket</code>,
     * с помощью которого создается новый <code>Session</code> и добавляется
в список сессий
     *
     * @throws IOException
     */
    public void start() throws IOException{
        while(true) {
            Socket sessionSocket = this.accept();
            System.out.printf("Client [%s] connected\n",
sessionSocket.getInetAddress().getHostAddress());

            Session session = new Session(sessionSocket);
            new Thread(session).start();
            sessions.add(session);

        }
    }

    /**
     * Отправляет сообщение <code>msg</code> во все сессии
<code>Session</code>,
     * кроме <code>sender</code>, если он указан
     * @param msg текст сообщения
     * @param sender отправитель. Если не null, то ему сообщение не
отправляется
     */
    public void broadcast(String msg, Session sender) {
        System.out.println(msg);
        for (Session session : sessions) {
            if (session != sender)
                session.sendMsg(msg);
        }
```

```
    }

    /**
     * Код, реализующий взаимодействие с клиентом
     */
     public class Session implements Runnable {
        private final Socket sessionSocket;
        private final PrintWriter out;
        private final BufferedReader in;
        private String username;

        /**
         *
         * @param socket установленное с клиентом подключение, полученное в
результате <code>.accept()</code>
         * @see ServerSocket
         * @throws IOException
         */
        public Session(Socket socket) throws IOException {
            this.sessionSocket = socket;
            this.out = new PrintWriter(
                    this.sessionSocket.getOutputStream(), true
            );
            this.in = new BufferedReader(
                    new
InputStreamReader(this.sessionSocket.getInputStream())
            );

        }

        @Override
        public void run() {
            try {
                // Считывание имени пользователя
                this.username = in.readLine();
                // Оповещение остальных клиентов о подключении
                broadcast(
                        String.format("[SERVER]%s has joined!",
this.username),
                        this
                );

                String clientInput;
                // Цикл, обрабатывающий входящие сообщения
                // .readLine() блокирует поток до тех пор, пока от клиента не
придет сообщение
                while((clientInput = this.in.readLine()) != null) {
                    broadcast(
                            String.format("[%s]%s",
this.username,clientInput),
                            this
                    );
                }

            } catch (IOException e) {
                System.out.println(e);
```

```
            } finally {
                sessions.remove(this);
                try {
                    in.close();
                    out.close();
                    sessionSocket.close();
                    System.out.printf("Client [%s] disconnected\n",
username);
                } catch (IOException e) {
                    System.out.println(e);
                }
            }
        }

        /**
         * Отправляет сообщение <code>msg</code> клиенту
         * @param msg текст сообщения
         */
        public void sendMsg(String msg) {
            out.println(msg);
        }
    }
}
```

Листинг 3. ClientGUI.java

```
package chat;

import javax.swing.*;
import java.awt.*;
import java.io.IOException;
import java.net.InetAddress;


/**
 * Основной класс интерфейса клиентской программы
 */
public class ClientGUI extends JFrame {


    private ChatBox chatBox;
    private SendBox sendBox;
    private LoginPopup loginPopup;


    private String username;
    private int port;
    private InetAddress address;


    private ChatClientSocket socket;

    public ClientGUI() {
        super();
```

```
    }

    /**
     * Запускает клиент
     */
    public void launch() {

        loginPopup = new LoginPopup(8080);
        socket = null;
        // Настройка основного окна
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setSize(800,600);
        setTitle("Chat");
        setLocationByPlatform(true);
        this.setVisible(true);

        // Получаем от пользователя данные для подключения
        int loginResult = loginPopup.run();

        // Если данные не были получены,
        // например пользователь закрыл окно,
        // то завершаем программу
        if (loginResult != JOptionPane.OK_OPTION) {
            this.quit();
            return;
        }

        // Сохраняем полученные данные
        this.username = loginPopup.getUsername();
        this.port = loginPopup.getPort();
        this.address = loginPopup.getAddress();
        setTitle(String.format("Chat | Logged in as [%s]",this.username));

        // Инициализация компонентов окна
        chatBox = new ChatBox();
        sendBox = new SendBox(this.username);



        add(chatBox, BorderLayout.CENTER);
        add(sendBox, BorderLayout.PAGE_END);
        pack();

        try {
            // Установка подключения
            this.socket = new ChatClientSocket(this.address, this.port,
this.username);

            Runtime.getRuntime().addShutdownHook(new Thread(){
                public void run(){
                    try {
                        socket.close();
                    } catch (IOException e) {}
                }
            });
```

```
            this.socket.addListener(chatBox.getMessageListener());
            sendBox.addListener(this.socket.getMessageListener());
            sendBox.addListener(chatBox.getMessageListener());
            JOptionPane.showMessageDialog(this, "Connected successfully");
            this.setVisible(true);
        } catch (IOException e) {
            JOptionPane.showMessageDialog(this, e,"Error",
JOptionPane.ERROR_MESSAGE);
            System.out.println(e);
            this.quit();
        }

    }

    /**
     * Завершает работу
     */
    public void quit() {
        this.dispose();
    }
}
```

Листинг 4. MessageEvent.java

```
package chat;

import java.util.EventObject;

/**
 * Событие, обозначающее отправку сообщения
 * @see MessageListener
 */
public class MessageEvent extends EventObject {

    /**
     * Отправитель сообщения
     */
    private String content;
    /**
     * Содержание сообщения
     */
    private String sender;

    /**
     * Возвращает содержание сообщения
     * @return содержание сообщения
     */
    public String getContent(){
        return content;
    }
```

```
    /**
     * Возвращает отправителя сообщения
     * @return отправитель сообщения
     */
    public String getSender(){
        return sender;
    }



    /**
     * @param source источник сообщения. Наследовано от {@link EventObject}
     * @param sender отправитель сообщения
     * @param content содержание сообщения
     */
    public MessageEvent(Object source, String sender, String content) {
        super(source);
        this.content = content;
        this.sender = sender;
    }
}
```

Листинг 5. MessageListener.java

```
package chat;

import java.util.EventListener;

/**
 * <code>MessageListener</code> может быть зарегистрирован у объекта
 * для получения сообщений
 */
public interface MessageListener extends EventListener {
    /**
     * Вызывается объектом для отправки сообщения
     * @see MessageEvent
     * @param event
     */
    void newMessage(MessageEvent event);
}
```

Листинг 6. ChatClientSocket.java

```
package chat;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.net.InetAddress;
import java.net.Socket;
import java.util.ArrayList;
import java.util.List;

/**
 * Компонент, осуществляющий взаимодействие с сервером
 *
 */
public class ChatClientSocket extends Socket {

    private final PrintWriter out;
    private final BufferedReader in;
    private List<MessageListener> listeners;

    private MessageListener sendMsgListener;

    /**
     * Устанавливает подключение с сервером
     * @param address адрес, к которому нужно подключится
     * @param port порт, на котором открыт сервер
     * @param username юзернейм
     * @throws IOException ошибка при подключении
     */
    public ChatClientSocket(InetAddress address, int port, String username)
    throws IOException {
        // Установка подключения
        super(address, port);

        this.out = new PrintWriter(getOutputStream(), true);
        this.in = new BufferedReader(new
InputStreamReader(getInputStream()));

        this.out.println(username);
        this.out.flush();

        this.listeners = new ArrayList<>();

        this.sendMsgListener = new MessageListener() {
            public void newMessage(MessageEvent evt) {
                String text = evt.getContent();
                out.println(text);
                out.flush();
            }
        };
        InputHandler h = new InputHandler(this.in);
        Thread t = new Thread(h);
        t.start();
    }
```

```
    /**
     * Добавляет <code>MessageListener</code>,
     * получающий оповещение при приеме нового сообщения от сервера
     * @param l
     */
    public void addListener(MessageListener l) {
        this.listeners.add(l);
    }

    /**
     * Возвращает <code>MessageListener</code>, отправляющий сообщение на
сервер
     * @return
     */
    public MessageListener getMessageListener() {
        return sendMsgListener;
    }

    /**
     * Обработчик входящих сообщений, запускаемый в отдельном потоке
     */
    private class InputHandler implements Runnable {

        private final BufferedReader in;
        String serverInput;

        public InputHandler(BufferedReader in){
            this.in = in;
        }

        public void run() {
            try {
                // Цикл, обрабатывающий входящие сообщения
                // .readLine() блокирует поток до тех пор, пока от сервера не
придет сообщение
                while((this.serverInput = this.in.readLine()) != null) {

                    // Формирование MessageEvent
                    String sender = serverInput.substring( 1,
serverInput.indexOf("]"));
                    String msg = serverInput.substring(
serverInput.indexOf("]")+1, serverInput.length());
                    MessageEvent evt = new MessageEvent(this, sender, msg);
                    // Отправка MessageEvent получателям
                    for (MessageListener l : listeners){

                        l.newMessage(evt);
                    }
                }
            } catch (IOException e) {
                System.out.println(e);
            }
        }
    }
}
```

Листинг 7. ChatBox.java

```
package chat;

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

/**
 * Компонент, отображающий сообщения
 */
public class ChatBox extends JPanel {

    private final JTextPane textBox;
    private final MessageListener appendNewMsg;

    /**
     * Возвращает <code>MessageListener</code>, печатающий сообщение в окно
     * @return обработчик событий
     */
    public MessageListener getMessageListener() {
        return appendNewMsg;
    }

    public ChatBox(){
        // Настройка текстового окна
        this.setBorder(BorderFactory.createEmptyBorder(5, 5, 5, 5));
        this.setLayout(new BorderLayout());
        this.textBox = new JTextPane();
        this.textBox.setEditable(false);
        this.textBox.setMargin(new Insets(5,5,5,5));
        this.textBox.setPreferredSize(new Dimension(700,500));
        JScrollPane scroll = new JScrollPane (this.textBox);
    scroll.setVerticalScrollBarPolicy(JScrollPane.VERTICAL_SCROLLBAR_ALWAYS);
 scroll.setHorizontalScrollBarPolicy(JScrollPane.HORIZONTAL_SCROLLBAR_NEVER);
        add(scroll);
        // Стиль для отображения жирного текста
        SimpleAttributeSet boldStyle = new SimpleAttributeSet();
        StyleConstants.setBold(boldStyle, true);

        // Обработчик событий MessageEvent
        appendNewMsg = (new MessageListener() {
            @Override
            public void newMessage(MessageEvent evt) {
                String header = String.format("[%s]: ",evt.getSender());
                String msg = evt.getContent()+"\n";
                StyledDocument doc = textBox.getStyledDocument();
                try {
                    doc.insertString(doc.getLength(), header, boldStyle);
                    doc.insertString(doc.getLength(), msg, null);
                } catch (BadLocationException e) {
                    e.printStackTrace();
                }
            }
        });
    }
}
```

Листинг 8. SendBox.java

```
package chat;

import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.ArrayList;
import java.util.List;

/**
 * Компонент, предоставляющий пользователю интерфейс для отправки сообщений
 */
public class SendBox extends JPanel {
    //private final JLabel usernameLabel;
    private final JButton sendBtn;
    private final JTextField msgField;
    private List<MessageListener> listeners;
    /**
     *
     * @param username юзернейм
     */
    public SendBox(String username) {
        // Настройка интерфейса
        this.setBorder(BorderFactory.createEmptyBorder(5, 5, 5, 5));
        this.setLayout(new BorderLayout());
        //this.usernameLabel = new JLabel(username);
        this.msgField = new JTextField(1);
        this.listeners = new ArrayList<>();
        this.sendBtn = new JButton("Send");
        //add(this.usernameLabel, BorderLayout.WEST);
        add(this.msgField, BorderLayout.CENTER);
        add(this.sendBtn, BorderLayout.EAST);

        // Обработчик события ActionEvent,
        // возникающего когда пользователь хочет отправить сообщение
        ActionListener onSend = new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                msgField.requestFocus();
                String msg = msgField.getText();
                msgField.setText("");
                // Если пользователь ничего не ввел, то сообщение не
отправляется
                if(msg.length() < 1) return;
                // Формирование MessageEvent
                MessageEvent evt = new MessageEvent(this, username, msg);
                // Отправка сообщения слушателям
                for (MessageListener l : listeners){
                    l.newMessage(evt);
                }
            }
        };
        msgField.addActionListener(onSend);
        sendBtn.addActionListener(onSend);
    }
```

```
    /**
     * Добавляет <code>MessageListener</code> получающий
     * оповещение при отправке сообщения пользователем
     * @param l
     */
    public void addListener(MessageListener l) {
        this.listeners.add(l);
    }
}
```

Листинг 9. LoginPopup.java

```
package chat;

import javax.swing.*;
import java.net.InetAddress;
import java.net.UnknownHostException;

/**
 * Компонент, управляющий окном получения данных для подключения
 */
public class LoginPopup {
    // Поля окна
    private final JTextField usernameField = new JTextField();
    private final JTextField addressField = new JTextField();
    private final JTextField portField = new JTextField();
    private final JLabel errorLabel = new JLabel();

    private final Object[] message = {
            "Username:", usernameField,
            "Address:", addressField,
            "Port:", portField,
            errorLabel
    };

    private String username = null;
    private InetAddress address = null;
    private int port;

    /**
     *
     * @param defaultPort Устанавливает в поле порта значение по умолчанию
     */
    public LoginPopup(int defaultPort) {
        portField.setText(Integer.toString(defaultPort));
        this.port = defaultPort;
    }

    /**
     * Возвращает адрес, указанный пользователем
     * @return адрес, указанный пользователем
     */
    public InetAddress getAddress(){
        return this.address;
    }
```

```
    /**
     * Возвращает порт, указанный пользоваелем
     * @return порт, указанный пользоваелем
     */
    public int getPort(){
        return this.port;
    }

    /**
     * Возвращает юзернейм, указанный пользоваелем
     * @return юзернейм, указанный пользователем
     */
    public String getUsername() {
        return this.username;
    }

    /**
     * Отображает окно
     * @param error сообщение об ошибке, которое нужно отобразить. null если
сообщение отображать не нужно
     * @return результат взаимодействия с пользователем. Подробнее в {@link
javax.swing.JOptionPane}
     */
    private int show(String error) {
        errorLabel.setText(error);
        return JOptionPane.showConfirmDialog(
                null,
                this.message,
                "Login",
                JOptionPane.OK_CANCEL_OPTION
        );
    }

    /**
     * Показывает окно до тех пор, пока пользователь не введет все поля
корректно, или не закроет окно
     * @return результат взаимодействия с пользователем. Подробнее в {@link
javax.swing.JOptionPane}
     */
    public int run() {
        int option;
        String errorMsg = "";
        while (true) {
            option = show(errorMsg);

            // Если пользователь нажал что-то кроме OK,
            // то он не хочет дальше взаимодействовать с окном.
            // Завершаем выполнение
            if (option != JOptionPane.OK_OPTION)
                break;

            // Валидация адреса
            try {
                this.address = InetAddress.getByName(addressField.getText());
            } catch (UnknownHostException e) {
                errorMsg = "Invalid address";
```

```
                continue;
            }

            // Валидация порта
            try {
                this.port = Integer.parseInt(portField.getText());
            } catch (NumberFormatException e) {
                errorMsg = "Invalid port number";
                continue;
            }

            // Валидация юзернейма
            this.username = usernameField.getText();
            if(username.length() < 3) {
                errorMsg = "Username should be at least 3 characters long";
                continue;
            }

            // Если выполнение дошло до сюда,
            // значит пользователь ввел все значения корректно,
            // и выполнение можно завершить
            break;

        }
        return option;
    }
}
```