MYCOMPANY

Dokumentation A11

Gradonski Janusz

DOKUEMNTATION VON DER AUFGABE A11

AUFGABENSTELLUNG

Erstellen Sie einen Server-Client-Chat!

- Im ersten Schritt (Grundanforderungen) sollen simple Strings zwischen Server und Client übertragen werden können
- Der Client
  - Übernimmt IP-Adresse und Port des Ziel-Hosts als Kommandozeilenparameter
  - Verbindet sich mithilfe der angegebenen Parameter mit dem Server
  - Liest über die Konsole zeilenweise Benutzereingaben als Strings ein
  - Diese Strings werden an den Server übertragen, welcher sie an alle anderen verbundenen Clients weiterleitet
- Der Server
  - Ist ebenfalls eine Konsolenanwendung und übernimmt den Port als Kommandozeilenparameter
  - Horcht auf eingehende Verbindungen von Clients
  - Startet für neue Clients einen neuen Thread
  - Verwaltet die verbundenen Threads in einer threadsicheren Collection
  - Verteilt empfangene Strings an alle anderen Clients
- Überlegen Sie sich eine geeignete Architektur, sodass Sie die kommenden Erweiterungen problemlos einbauen können
  - Nachrichten können in weiterer Folge nicht nur simple Strings sein, sondern auch serialisierte Objekte
  - Bauen Sie Design Patterns ein: insbesondere eignet sich der Decorator für das Dekorieren von Nachrichten über Streams
- Denken Sie an ein sauberes Exception Handling!
- Dokumentieren Sie Ihren Code!

Viel Spaß!

AUFWAND

Geschätzt	Gebraucht
3h	6h

CODE

SERVER.JAVA

```
import javax.swing.*;
import java.awt.*;
import java.io.*;
import java.net.Socket;
import java.net.ServerSocket;
import java.net.UnknownHostException;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.concurrent.CopyOnWriteArrayList;
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;
/*
 * Gradonski Janusz Aufgabe Server-Client Chat
 *
 *
 */
public class Server extends JFrame {

    JTextArea textArea;
    private static ServerSocket serverSocket = null;
    private static Socket clientSocket = null;
    private CopyOnWriteArrayList<ClientThread> threads = new
CopyOnWriteArrayList<>();
    private JList list;

    /**
     * Die Main-methode.. wenn ein Port angegeben wird , so wird dieser auch
ubernommen , wenn nicht dann defaultport 7171
     * @param args
     */
    public static void main(String args[]) {
        int portNumber = 7171;
        if (args.length > 1) {
            portNumber = Integer.valueOf(args[0]);
        }

        new Server(portNumber);
    }

    /**
     * Server-Konstruktor mit einem Layout
     * @param portNumber
     */
    public Server(int portNumber) {
      //GUI
        getContentPane().setLayout(new BorderLayout());
        setVisible(true);
        setLocation(750, 0);
        JPanel centerPanel = new JPanel();

        setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);

        centerPanel.setLayout(new FlowLayout(FlowLayout.CENTER, 5, 5));
        centerPanel.add(new JScrollPane());
        getContentPane().add(centerPanel, BorderLayout.CENTER);
        Button button_1 = new Button("Clear");
```

```
        button_1.addActionListener(new ActionListener() {
             public void actionPerformed(ActionEvent e) {
                   textArea.setText("");
             }
        });

        list = new JList();
        centerPanel.add(list);
        textArea = new JTextArea(15, 40);
        centerPanel.add(textArea);
        textArea.setEditable(false);
        centerPanel.add(button_1);

        Button button = new Button("Stop Server");
        button.addActionListener(new ActionListener() {
             public void actionPerformed(ActionEvent e) {
                    try {
                     serverSocket.close();
                } catch (Exception e1) {
                     System.out.println(e);
                }
                   System.exit(1);
             }
        });
        centerPanel.add(button);
        pack();

        // Es wird ein neuer Server fur die Verbindungen geoffnet
        try {
            serverSocket = new ServerSocket(portNumber);
            changeTitle();
            textArea.append("Server Started... \nWaiting for Connections...\n");
        } catch (IOException e) {
            System.out.println(e);
        }

        // Wenn ein Client sich verbindet wird er somit in ein threads Array
hinzugeugt
        while (true) {
            try {
                clientSocket = serverSocket.accept();
                ClientThread temp = new ClientThread(clientSocket, threads, this);
                temp.start();
                threads.add(temp);
            } catch (IOException e) {
                System.out.println(e);
            }
        }
    }
    /**
     * Methode um in die TextArea was reinzuschreiben
     * @param message
     */
    public void writeInTextArea(String message) {
        textArea.append(message);
    }
```

```
    /**
     * Der Titel von dem Fenster wird da geandert wenn eine Verbindung hinzugefugt
wird usw.
     */
    public void changeTitle() {
        String host = "";
        int portNumber = 0;
        try {
            host = serverSocket.getInetAddress().getLocalHost().getHostAddress();
            portNumber = serverSocket.getLocalPort();
        } catch (UnknownHostException e) {
            System.out.println("Falsch Host");
        }
        setTitle("Gradonski Chat | HOST: " + host + " | PORT: " + portNumber + " |
NUMBER OF CLIENTS: " + threads.size());
    }
}
```

CLIENT.JAVA

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.io.*;
import java.net.Socket;
import java.awt.event.ActionListener;

public class Client extends JFrame {

    private Socket socket;
    private BufferedReader in;
    private PrintWriter out;

    private JTextField inputTextField = new JTextField(48);
    private JTextArea textArea = new JTextArea(15, 60);

    // Background Thread um aus den Reader zu lesen
    private Thread backgroundThread = new Thread(new Runnable() {
        @Override
        public void run() {
            String line;
            try {
                in              =              new              BufferedReader(new
InputStreamReader(socket.getInputStream()));
                while (true) {
                    line = in.readLine();
                    textArea.append(line + "\n");
                }
            } catch (IOException e) {
             JOptionPane.showMessageDialog(null, "Verbindung wurde unterbrochen \n
Kein Input.",null, JOptionPane.ERROR_MESSAGE);
             System.exit(1);
            }
        }

    });

    /**
```

```
     * Methode um nach dem drucken von Enter die Nachricht geschickt wird
     */
    private AbstractAction onEnterPressAction = new AbstractAction() {
        @Override
        public void actionPerformed(ActionEvent e) {
            String textToSend = inputTextField.getText();
            inputTextField.setText("");
            try {
                out = new PrintWriter(new
OutputStreamWriter(socket.getOutputStream(), "ISO-8859-1"), true);
                out.write(textToSend + "\n");
                out.flush();
                if (textToSend.startsWith("/quit")) {
                    if (out != null) out.close();
                    if (in != null) in.close();
                    if (socket != null) socket.close();
                    System.exit(0);
                }
            } catch (IOException e1) {
                e1.printStackTrace();
            }
        }
    };
    /**
     * Konstruktor der Klasse
     * @param host
     * @param port
     */
    public Client(String host, int port) {
      // Socket wird initialisiert
        try {
            socket = new Socket(host, port);
        } catch (IOException e) {
            //Wenn keine Verbindung
            JOptionPane.showMessageDialog(null, "Could not connect to server",
"Connection Error", JOptionPane.ERROR_MESSAGE);
            System.exit(0);
        }

        // Wenn verbunden dann wird der Host und die Port in der TitelZeile
angezeigt
        setTitle("CONNECTED TO SERVER: " + host + " IN PORT: " + port);

        // ein Thread wird im hintergrund gestartet um die Nachrichten vom Server
zu empfangen
        backgroundThread.start();

        // Die GUI wird initialisiert.
        getContentPane().setLayout(new BorderLayout());
        setVisible(true);
        setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
        textArea.setEditable(false);

        JPanel southPanel = new JPanel();
        southPanel.add(inputTextField);
        inputTextField.addActionListener(onEnterPressAction);

        JPanel centerPanel = new JPanel();
        JScrollPane scrollPane = new JScrollPane(textArea);
        centerPanel.add(scrollPane);
```

```
        getContentPane().add(centerPanel, BorderLayout.CENTER);
        getContentPane().add(southPanel, BorderLayout.SOUTH);

        /* Button Senden von der Nachricht.. Die Nachricht wird aus
         * dem InputTextField rausgelesen und dann uber den PrintWriter an den
Socket weitergeleitet.
         */
        JButton btnNewButton = new JButton("Senden");
        btnNewButton.addActionListener(new ActionListener() {
             public void actionPerformed(ActionEvent arg0) {
                String textToSend = inputTextField.getText();
                inputTextField.setText("");
                try {
                    out             =             new             PrintWriter(new
OutputStreamWriter(socket.getOutputStream(), "ISO-8859-1"), true);
                    out.write(textToSend + "\n");
                    out.flush();
                    if (textToSend.startsWith("/quit")) {
                        if (out != null) out.close();
                        if (in != null) in.close();
                        if (socket != null) socket.close();
                        System.exit(0);
                    }
                } catch (IOException e1) {
                    e1.printStackTrace();
                }

             }
        });
        southPanel.add(btnNewButton);
        pack();
        setLocationRelativeTo(null);

    }
/*
 * Die Main-methode , wenn keine
 * angaben zum Server uber Kommandozeile angegeben werden , dann uber Fensetr manu
 */
    public static void main(String[] args) {
      try{
        if (args.length == 0) {

            new                        Client(JOptionPane.showInputDialog("Server"),
Integer.parseInt(JOptionPane.showInputDialog("Port")));
        } else if (args.length == 1) {
            new Client(JOptionPane.showInputDialog("Server"), 7171);
        } else if (args.length == 2) {
            new Client(args[0], Integer.parseInt(args[1]));
        }
       }catch(Exception e){
             JOptionPane.showMessageDialog(null,                        "Falsche
eingaben",null,JOptionPane.ERROR_MESSAGE);
       }
    }

}
```

CLIENTTHREAD.JAVA

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.net.Socket;
import java.util.ArrayList;
import java.util.Collections;
import java.util.Iterator;
import java.util.concurrent.CopyOnWriteArrayList;

public class ClientThread extends Thread {

    private BufferedReader in = null;
    private PrintWriter out = null;
    private Socket clientSocket = null;
    private CopyOnWriteArrayList<ClientThread> threads;
    private Server frame;
    private String name;
    /**
     * Konstruktor der Klasse ClientThread , initialisiert alles.
     * @param clientSocket
     * @param threads
     * @param frame
     */
    public  ClientThread(Socket  clientSocket,  CopyOnWriteArrayList<ClientThread>
threads, Server frame) {
        this.clientSocket = clientSocket;
        this.threads = threads;
        this.frame = frame;
    }
    /**
     * Thread-Run
     */
    public void run() {
        try {
             // Der Input und Output werden hier initialisiert.
            in               =               new               BufferedReader(new
InputStreamReader(clientSocket.getInputStream()));
            out = new PrintWriter(clientSocket.getOutputStream(), true);
            out.write("<<< Enter your name >>>\r");
            out.flush();
            name = in.readLine().trim();
            out.write("<<< Hi " + name + ". Welcome to the chat room >>>\n<<<
Enter /quit in a new line to exit >>>\r");
            out.flush();
            frame.changeTitle();
            synchronized (this) {
                for (ClientThread c : threads) {
                    if (c != this) {
                        c.out.write("<<< User " + name + " has connected >>>\r");
                        c.out.flush();
                    }
                }
                frame.writeInTextArea("<<<  User  "  +  name  +  "  has  connected
>>>\n");
            }
            while (true) {
                String line = in.readLine();
                if (line.startsWith("/quit")) {
```

```
                    // Wenn der user /Quit in seinem Fesnter eingibt wird das
ganze Prozess abgebrochen
                        break;
                }
                for (ClientThread c : threads) {
                    c.out.write("<" + name + "> " + line + "\r");
                    c.out.flush();
                }
                frame.writeInTextArea("<" + name + "> " + line + "\n");
            }

            synchronized (this) {
                for (ClientThread c : threads) {
                    if (c != null && c != this) {
                        c.out.write("<<< User " + name + " has disconnected
>>>\r");
                        c.out.flush();
                    }
                }
                frame.writeInTextArea("<<< User " + name + " has disconnected
>>>\n");
                out.write("<<< Goodbye " + name + " >>>");
                out.flush();

                Iterator i = threads.iterator();
                while (i.hasNext()) {
                    ClientThread c = (ClientThread) i.next();
                    if (c == this) {
                        c.interrupt();
                        threads.remove(c);
                        frame.changeTitle();
                    }
                }

                // Schliessen von den Verbindungen
                out.close();
                in.close();
                clientSocket.close();
            }
        } catch (IOException e) {
            synchronized (this) {
                // Alle Clients werden Informiert, wenn der Juser ausgeloggt wird.
                for (ClientThread c : threads) {
                    if (c != null) {
                        c.out.write("<<< User " + name + " has disconnected
>>>\r");
                        c.out.flush();
                    }
                }
                frame.writeInTextArea("<<< User " + name + " has disconnected
>>>\n");

                Iterator i = threads.listIterator();
                while (i.hasNext()) {
                    ClientThread c = (ClientThread) i.next();
                    if (c == this) {
                        c.interrupt();
                        i.remove();
                        frame.changeTitle();
                    }
```

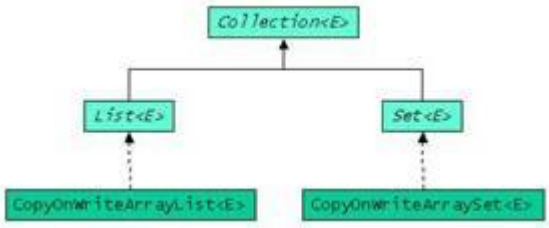
```
                }

                try {
                    out.close();
                    in.close();
                    clientSocket.close();
                } catch (IOException e1) {
                    e1.printStackTrace();
                }
            }
        }
    }
}
```

COLLECTION AUSWAHL

Für die Aufgabe habe ich eine Copy-On-Write Collection verwendet, welche auch zu den threadsicheren Collections wie Concurrent Collection zählt.

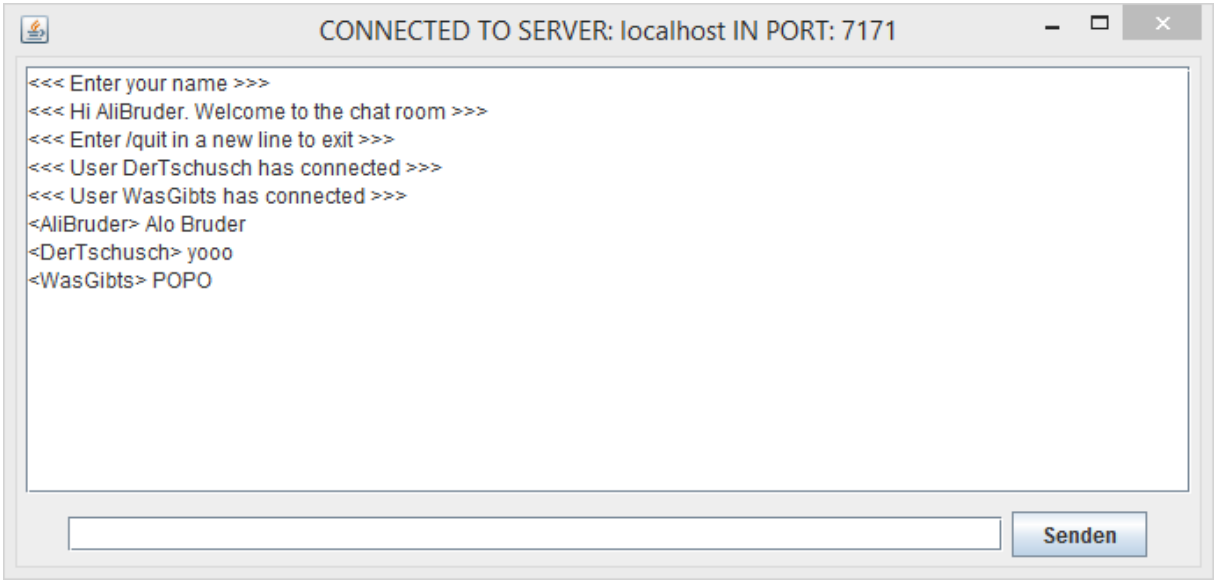
Dazu gehören die Copy-On-Write Collections: CopyOnWriteArrayList und CopyOnWriteArraySet. Das sind threadsichere Collections, die ebenfalls ohne Sperren auskommen, aber nicht auf den atomaren Variablen beruhen, sondern Veränderungen auf Snapshots durchführen.

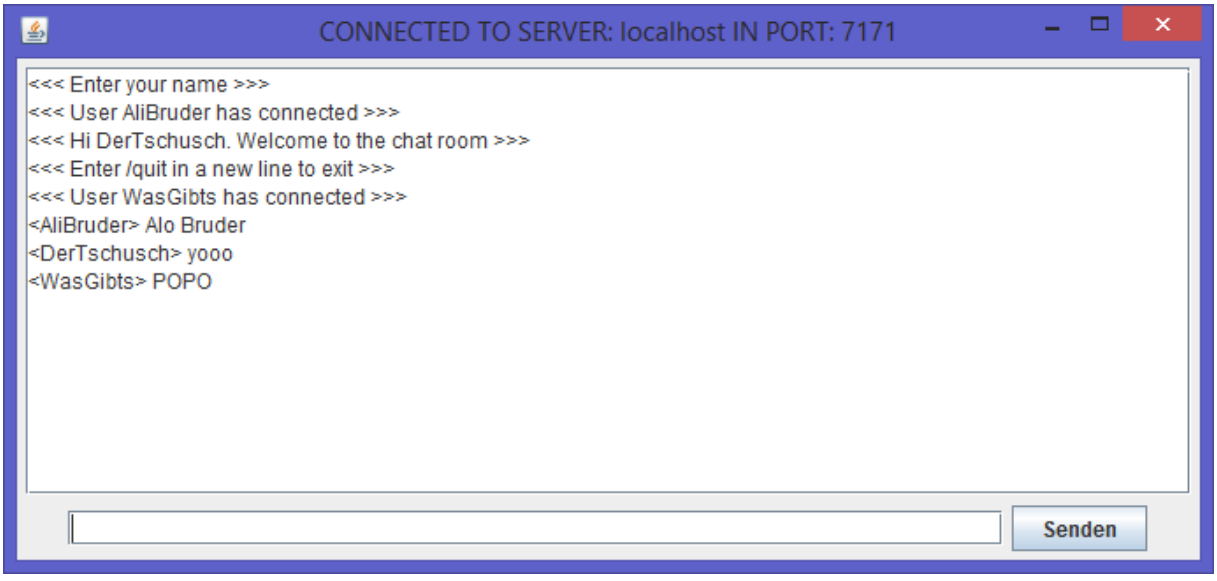


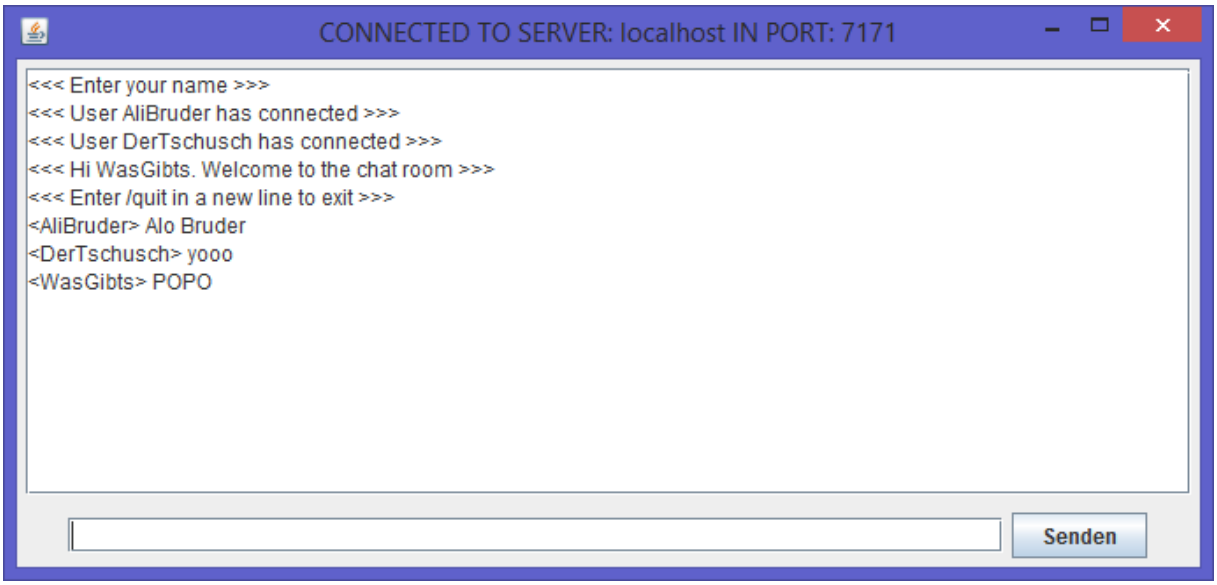
Vereinfacht kann man sich das so vorstellen: die Copy-On-Write Collection hält intern ein Array mit allen Elementen der Collection. Wenn nun eine Veränderung gemacht werden soll (über Methoden wie add, set, usw.), dann wird zunächst einmal eine Kopie des aktuellen Arrays gemacht. Dabei wird das Array nur gelesen, nicht verändert. Es gibt also keine Kollisionen mit anderen lesenden Threads. Auf dieser Kopie wird dann die Veränderung vorgenommen. Die Kopie kennt nur der verändernde Thread. Es kann also auch hier keine Kollisionen mit anderen Threads geben. Danach wird die intern gehaltene Referenz auf das Array so umgebogen, dass sie auf das veränderte Array zeigt. Das Umbiegen der Referenz ist eine Adreßzuweisung und ist atomar, kann also nicht unterbrochen werden. Nach der Referenzzuweisung steht die Veränderung den konkurrierenden Threads zur Verfügung.

RESULT


Gradonski Chat | HOST: 192.168.56.1 | PORT: 7171 | NUMBER OF CLIEN...
Server Started...
Waiting for Connections...
<<< User AliBruder has connected >>>
<<< User DerTschusch has connected >>>
<<< User WasGibts has connected >>>
<AliBruder> Alo Bruder
<DerTschusch> yooo
<WasGibts> POPO
Clear
Stop Server


CONNECTED TO SERVER: localhost IN PORT: 7171
<<< Enter your name >>>
<<< Hi AliBruder. Welcome to the chat room >>>
<<< Enter /quit in a new line to exit >>>
<<< User DerTschusch has connected >>>
<<< User WasGibts has connected >>>
<AliBruder> Alo Bruder
<DerTschusch> yooo
<WasGibts> POPO
Senden


CONNECTED TO SERVER: localhost IN PORT: 7171
<<< Enter your name >>>
<<< User AliBruder has connected >>>
<<< Hi DerTschusch. Welcome to the chat room >>>
<<< Enter /quit in a new line to exit >>>
<<< User WasGibts has connected >>>
<AliBruder> Alo Bruder
<DerTschusch> yooo
<WasGibts> POPO
Senden



QUELLEN

JAVADOC und

http://www.javamex.com/tutorials/synchronization_concurrency_8_hashmap2.shtml

http://markusjais.com/java-concurrency-understanding-copyonwritearraylist-and-copyonwritearrayset/

http://crunchify.com/hashmap-vs-concurrenthashmap-vs-synchronizedmap-how-a-hashmap-can-be-synchronized-in-java/