JUST DO IT
HÖHERE ABTEILUNG FÜR
INFORMATIONSTECHNOLOGIE

tgm
Die Schule der Technik

Laborprotokoll

CORBA Overview

SEW
4DHIT 2017/18

Peter HANSLIK

Version 0.1
Note:
Betreuer: Borko
Begonnen am 16. März 2018
Beendet am 16. März 2018

Inhaltsverzeichnis

1 Einführung

Dieses Protokoll soll meine Vorgehensweise

1.1 Ziele

Ziel ist es die Nebenläufigkeit in Java zu lernen und zu üben.

1.2 Voraussetzungen

Vorraussetzung sind nur Java Kentnisse und verwendung der API. Es können jedoch auch andere Hilfsmittel wie Youtube-Tutorials und Stackoverflow zur Hilfe genommen haben.

1.3 Aufgabenstellung

Es ist ein Chat in Java zu programmieren, in dem mehrere Clients miteinander 'chatten' können. Für jeden verbundenen Client muss ein neuer 'Handler'-Thread gestertet verden, der sich um den Client kümmert. Dieser 'Handler' muss auch wieder sauber geschlossen werden.

2 Ergebnisse

Mein Ergebnis ist unter https://github.com/phanslik-tgm/JavaChat zu finden.

2.1 Vorgehensweise

Als erstes habe ich mit einem IOStream und den dazugehörigen reader und writer einen Chat mit nur einem Client und einem Server zum laufen gebracht. Da der Server nicht nebenläufig ist, kann sich nur ein Client verbinden.
Damit sich also mehrere Clients auf den Server verbinden können, braucht der server Handler-Threads. Also wenn sich ein Client auf den Server verbindet, startet der Server einen Handler-Thread, der die Verbindung 'verwaltet' (sich um den Client kümmert). Um diesen Handler-Thread erstellen zu können, habe ich eine Klasse namens Handler gemacht, die Runnable implementiert und daher eine Methode namens run() besitzt. der Handler hat als Parameter einen Socket.
Wenn man den Client schließt, muss der Thread wieder geschlossen werden.

2.2 Code erklärung

2.2.1 Server

Der Server hat 2 wichtige Variablen.

```
1 ServerSocket server;
  static ArrayList<PrintWriter> list_clientWriter;
```

Wenn sich ein neuer Client verbindet, erstellt listenToClients() einen neuen Handler-Thread und startet ihn. Da while(true) eine Endlos-Schleife ist, wird immer wieder geschaut ob es einen neuen Client gibt.

```
  public void listenToClients()
2     {
          while (true)
4         {
              try
6             {
                  Socket client = server.accept();
8
                  Thread clientThread = new Thread(new Handler(client));
10                clientThread.start();
              }
12            catch (IOException e)
              {
14                e.printStackTrace();
              }
16        }
      }
```

In dieser Methode wird der Serversocket gestartet und eine ArrayList erstellt. Er liefert ausserhalb solange true zurück, bis ein Fehler auftritt.

```
1  public boolean runServer()
   {
3      try
       {
5          server = new ServerSocket(5050);
           appendTextToConsole("Server wurde gestartet!", LEVEL_ERROR);
7
           list_clientWriter = new ArrayList<PrintWriter>();
9          return true;
       }
11     catch (IOException e)
       {
13         appendTextToConsole("Server konnte nicht gestartet werden!", LEVEL_ERROR);
           e.printStackTrace();
15         return false;
       }
17 }
```

Die Methode sendToAllClients, wie der Name schon sagt, sendet alle Nachrichten, die er von einem Client bekommt an alle verbundenen Clients zurück.

```
1  public static void sendToAllClients(String message)
   {
3      Iterator it = list_clientWriter.iterator();

5      while (it.hasNext())
       {
7          PrintWriter writer = (PrintWriter) it.next();
           writer.println(message);
9          writer.flush();
       }
11 }
```

In der main() wird ein Objekt von der Klasse Server erstellt. Die Methode runServer() startet den Server Socket. Und listenToClient() erstellt für jeden neuen Client einen Handler-Thread.

```
1  public static void main(String[] args)
   {
3      Server s = new Server();
       if (s.runServer())
5      {
           s.listenToClients();
7      }
       else
9      {
           // Do nothing
11     }
   }
```

2.2.2 Client

In der main() wird die GUI gestartet. In der generateGUI() Methode wird jedoch auch ein MessagesFromServerListener-Thread erstellt. Ausserdem wird in connectToServer() ein Socket gestartet, der sich mittels IP und Port auf den Server verbindet.

```
public static void main(String[] args)
    {
        Client c = new Client();
        c.createGUI();
    }
```

Von dieser Klasse wird ein Thread gemacht, der mit reader.readLine() nachschaut ob Nachrichten im IOStream sind und gibt diese aus.

```
public class MessagesFromServerListener implements Runnable
    {

        @Override
        public void run()
        {
            String message;

            try
            {
                while ((message = reader.readLine()) != null)
                {
                    appendTextMessages(message);
                    textArea_Messages.setCaretPosition(textArea_Messages.getText().length());
                }
            }
            catch (IOException e)
            {
                appendTextMessages("Nachricht konnte nicht empfangen werden!");
                e.printStackTrace();
            }
        }
    }
```

Die Methode sendMessageToServer() schreibt mit writer.println() und writer.flush() in den IOStream und sendet somit die Nachricht an den Server.

```
public void sendMessageToServer()
    {
        writer.println(textField_Username.getText() + ": " + textField_ClientMessage.getText());
        writer.flush();

        textField_ClientMessage.setText("");
        textField_ClientMessage.requestFocus();
    }
```

Die Methode connectToServer() erstellt, einen neuen Socket, der sich über IP und Port auf den Server verbindet. Ausserdem wird ein Listener ausgeführt der, wenn das Fenster über das x geschlossen wird, 'exit' an den Server(um genauer zu sein Handler) sendet. Der server schließt dann den Handler der sich um den Client, der geschlossen wurde, gekümmert hat.

```
public boolean connectToServer()
    {
        try
        {
            client = new Socket("127.0.0.1", 5050);
            reader = new BufferedReader(new InputStreamReader(client.getInputStream()));
            writer = new PrintWriter(client.getOutputStream());
            appendTextMessages("Netzwerkverbindung hergestellt");

            clientFrame.addWindowListener(new WindowAdapter()
            {
                public void windowClosing(WindowEvent e)
                {
                    writer.println("exit");
                    writer.flush();
                }
            });

            return true;
        }
        catch (Exception e)
        {
            appendTextMessages("Netzwerkverbindung konnte nicht hergestellt werden");
            e.printStackTrace();

            return false;
        }
    }
```

2.2.3 Handler

Die Klasse Handler implementiert Runnable, damit man einen Thread davon machen kann.
Handler hat als Parameter einen Socket.
In der methode run() werden am server die Nachrichten vom zugewiesenen Client empfangen und ausgegeben. Falls die nachricht 'exit' empfangen wird, schließt sich die while im Handler-Thread, da das nur gesendet wird, wenn der Client geschlossen wurde. Un wenn der Thread ausläuft, wird er gelöscht.

```
public class Handler implements Runnable
    {

        Socket client;
        BufferedReader reader;

        public Handler(Socket client)
        {
            try
            {
                this.client = client;
                reader = new BufferedReader(new InputStreamReader(client.getInputStream()));
            }
            catch (IOException e)
            {
                e.printStackTrace();
            }
        }

        @Override
        public void run()
        {
            String nachricht;

            try
            {

                while ((nachricht = reader.readLine()) != null)
                {
                    Server.appendTextToConsole("Vom Client: \n" + nachricht, Server.LEVEL_NORMAL);
                    PrintWriter writer = new PrintWriter(client.getOutputStream());
                    Server.list_clientWriter.add(writer);
                    if(nachricht.equals("exit"))
                    {
                        client.close();
                        Server.list_clientWriter.remove(writer);
                        break;
                    }
                    else
                    {
                        Server.sendToAllClients(nachricht);
                    }
                }
            }
            catch (IOException e)
            {
                e.printStackTrace();
            }
        }
    }
```

2.3 GUI