NORDAKADEMIE
HOCHSCHULE DER WIRTSCHAFT

I104 Technische Grundlagen der Informatik

Dozenten: André Kowalk und Sebastian Harnau

Exkurs: Socket-Programmierung mit Java

Einführung

Wir wollen nun selber einen einfachen Echo-Server und einen einfachen Client schreiben. Beide Programme erstellen wir mit Java.

Der Echo-Server basiert auf TCP und gibt die erhaltene Nachricht unverändert an den Client zurück. Er dient dem Test der Basisfunktionalität. Es ist nicht Ziel, ein echtes standardisiertes Schicht 7-Protokoll zu implementieren.

Zunächst setzen wir den Server auf. Wenn dieser läuft, implementieren wir den Client und versuchen, mit dem Server zu kommunizieren.

Fragen, die Sie dabei beantworten können:

1. Was passiert, wenn Sie Ihren Server auf Port 77777 starten?

2. Was passiert, wenn Sie mehrere Server auf dem gleichen Port laufen lassen?

3. Was passiert, wenn Sie Ihren Client senden lassen, ohne, dass ein Server läuft?

Sie erhalten sowohl für den Server als auch den Client jeweils ein Template mit Imports, Objektdefinitionen und Argumenteinlesung. Machen Sie eine lauffähige Anwendung gemäß den Prozessbeschreibungen auf den nächsten Seiten daraus.

Was wir nicht tun

Wir bauen lediglich eine unverschlüsselte TCP-Verbindung auf. Hinweise für UDP- und SSL-Modifikationen finden sich am Ende dieses Dokuments.

Erklärungen

Um das geplante Projekt umsetzen zu können, benötigen wir zunächst einige Java-Klassen, die wir importieren müssen:

```
//OutputStream, inputStream, IOException
import java.io.*;

//Basisklasse für Client-Socket-Programmierung
import java.net.*;
```

Die Angabe von Portnummern, IP-Adresse und Nachricht wollen wir per Übergabeparameter ermöglichen. Dazu gibt es das String-Array args:

```
if (args.length>0) {
    serverPort = args[0];
} else {
    serverPort = "14770";
}
```

Ist kein Wert angegeben, setzen wir einen Default-Wert.

Die benötigten Objekte sind:

```
// genutzte Portnummer, IP und Nachricht
String           serverPort;
String           serverIP;
String           message;

// Sockets und Puffer
ServerSocket     socketServer;
Socket           socketClient;
BufferedReader   bufferRead;
DataOutputStream bufferWrite;
```

Beim Server brauchen wir die IP nicht und beim Client das Server-Socket nicht, aber ansonsten benutzen wir in beiden Programmen die gleichen Objekte und Objektnamen. Die Namen sollten eigentlich Sinn und Zweck bereits erklären. Falls nicht, bitte Fragen.

Die gesendeten Nachrichten müssen immer mit einem Zeilenumbruch `+"\n"` enden. Dadurch wird die Nachricht abgeschlossen und die Weiterverarbeitung angestoßen.

Zu guter Letzt natürlich der Hinweis auf ein Error-Handling. Wir umklammern unsere Socket-Aufbau-Versuche mit einem try-Catch-Konstrukt, um so einen unkontrollierte Absturz zu verhindern und wenigstens noch ordentliche Fehlermeldungen zu bekommen. Dies ist in den Prozessbeschreibungen NICHT enthalten und muss eigenständig hinzugefügt werden:

```
try {

    // ...und hier geschieht ein Wunder...

} catch (IOException e) {
    System.out.println("Error connecting to server!");
    System.out.println(e);
}
```

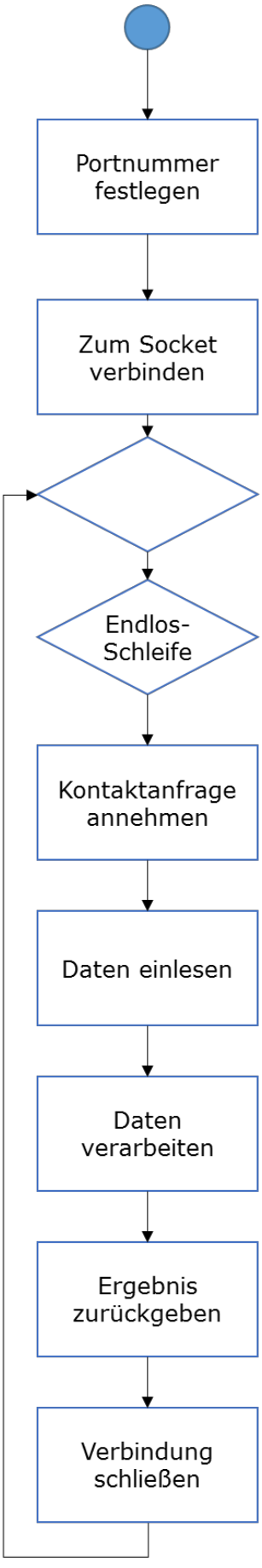
Sollten beide Programme laufen, Sie die Fragen von der vorgehenden Seite beantwortet haben und Sie noch etwas Zeit übrig haben, können Sie den Server dahingehen manipulieren, dass er nicht den Originaltext zurückgibt, sondern die Nachricht manipuliert. Anbei einige Vorschläge dazu:

```
// Für Server, der immer die Antwort der Vorsendung zurückgibt
String messageAlt = "";
String messageHelp;

// **********************
// Verarbeiten der Eingabe

// Immer die Antwort der Vorsendung zurückgeben
messageHelp = message;
message = messageAlt;
messageAlt = messageHelp;

// Großbuchstaben
message = message.toUpperCase();

// Nachricht rückwärts
message = new StringBuffer(message).reverse().toString();
```

Vielleicht haben Sie ja auch eigene Ideen…

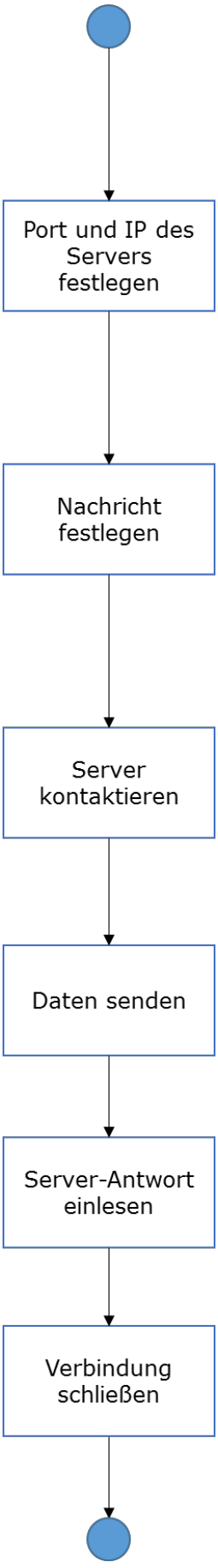
Wenn alles läuft, schicken Sie auch eine Nachricht an den Dozentenserver. Dadurch werden Ihre Verbindungsdaten für alle sichtbar und Sie können auch Nachrichten von Dritten empfangen. (und natürlich auch selber an Dritte senden!)

Server-Prozess




```
// Festlegen der Portnummer
if (args.length>0) {
    serverPort = args[0];
} else {
    serverPort = "57913";
}

System.out.println("Choosen Port: " + serverPort);

// Abhören des Ports starten
socketServer =
     new ServerSocket(Integer.parseInt(serverPort));



while(true) {

System.out.println("");
System.out.println("Listening...");

// Verbindung zum Client wird hergestellt
socketClient = socketServer.accept();
System.out.println("Connected client from " +
          socketClient.getRemoteSocketAddress());

// Einlesen der Daten
bufferRead = new BufferedReader(new
  InputStreamReader(socketClient.getInputStream()));
message=bufferRead.readLine();
System.out.println("Received Message: " + message);

// **********************
// Verarbeiten der Eingabe
// -> Leer=Echo-Server



// Ausgabe des Ergebnisses
bufferWrite = new
   DataOutputStream(socketClient.getOutputStream());
bufferWrite.writeBytes(message + "\n");
System.out.println("Answered Message: " + message);

// Verbindungsabbau
socketClient.close();

}
```

Client-Prozess




```
// Festlegen der Ziel-IP
if (args.length>0) {
    serverIP = args[0];
} else {
    serverIP = "127.0.0.1";
}

// Festlegen der Ziel-Portnummer
if (args.length>1) {
    serverPort = args[1];
} else {
    serverPort = "57913";
}

// Festlegen der Nachricht
if (args.length>2) {
    message="";
    for (words=2;words<args.length;words++) {
       message = message + args[words]+ " ";
    }
    message = message.substring(
                            0, message.length() - 1);
} else {
    message = "Hello World!";
}
System.out.println("Choosen Server : " + serverIP +
 ":" + serverPort);

// Aufbau der Verbindung
socketClient = new Socket(serverIP,
Integer.parseInt(serverPort));

// Datenausgabe an Server
bufferWrite = new
    DataOutputStream(socketClient.getOutputStream());
bufferWrite.writeBytes(message + "\n");
System.out.println("Sending Message: " + message);

// Einlesen der Daten
bufferRead = new BufferedReader(new
   InputStreamReader(socketClient.getInputStream()));
message=bufferRead.readLine();
System.out.println("Receive Message: " + message);


// Verbindungsabbau
socketClient.close();
```

DOS-Shell für Einsteiger

Aufruf der DOS-Shell: Windows+R -> cmd -> OK

DOS-Befehle:

Leeren des DOS-Fensters: **cls**

Wechseln des Laufwerks: **<LW>:**

Wechseln des Pfads: **cd <pfad>**

Kompilieren des Servers:

```
javac socketServer.java
```

Starten des Servers:

```
java socketServer [serverPort]
```

Kompilieren des Clients:

```
javac socketClient.java
```

Starten des Clients:

```
java socketClient [serverIP] [serverPort] [message]
```

Alle Parameter sind optional:

- Wird keine IP angegeben, wird 127.0.0.1 benutzt.
- Wird kein Port angegeben, ist 57913 eingestellt.
- Wird keine Nachricht angegeben, wird "Hallo Welt!" gesendet.

Sollte der Aufruf von java.exe und javac.exe nicht funktionieren, fehlt eine Pfadangabe (z.B.: C:\Programme\Java\jdk1.8.0_66\bin):

```
set Path=%Path%;<Pfad zum Java bin-Ordner>
```

Weitergehende Verweise für Interessierte

Um eine **UDP**-Verbindung auszubauen, muss im Wesentlichen lediglich das Socket gegen ein Datagramm ausgetauscht werden:

https://systembash.com/a-simple-java-udp-server-and-udp-client/

http://www.heimetli.ch/udp/UDPServer.html

Der Abbau der Verbindung entfällt dann natürlich auch.

Ein einfaches Schicht 7-Protokoll ist SMTP. Es wird deshalb auch gerne in Beispielen für Socket-Server benutzt. Beispiele für den Aufbau von **SMTP-Clients** finden sich hier:

https://commons.apache.org/proper/commons-net/javadocs/api-3.4/org/apache/commons/net/smtp/SMTPClient.html

http://www.java2s.com/Code/Java/Network-Protocol/SendingMailUsingSockets.htm

Um eine **SSL**-Verschlüsselung aufzubauen, ist deutlich mehr Aufwand nötig (Zertifikate!), als in 3 UE leistbar. Hinweise dazu:

https://www.tutorials.de/threads/ssl-socketverbindung-mit-java.267445/

http://stackoverflow.com/questions/18787419/ssl-socket-connection

Quellen für diese Aufgabe

Es gibt massenweise Tutorien zur Erstellung einer Socket-Verbindung im Internet. Keines hat wirklich stabil funktioniert. Genutzt wurden u.a.:

http://www.javaworld.com/article/2077322/core-java/core-java-sockets-programming-in-java-a-tutorial.html

http://www.javatpoint.com/socket-programming

Dementsprechend sind große Teile der Musterlösung tatsächlich von den Dozenten selbst erstellt bzw. aus verschiedenen Tutorien adaptiert.