

UNIVERSITÄT LEIPZIG
LPZ E-BUSINESS
applied telematics

Übungen zum Java-Praxiskurs für Fortgeschrittene

Matthias Book
Malte Hülder

Lehrstuhl für Angewandte Telematik / e-Business WS 2005/2006

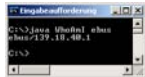
UNIVERSITÄT LEIPZIG
LPZ E-BUSINESS
applied telematics

Wdh.: Verwendung von InetAddress

- Beispiel für die Verwendung von `InetAddress`:

```
import java.net.*;

public class WhoAmI {
    public static void main(String[] args) throws Exception {
        if (args.length != 1) {
            System.err.println("Usage: WhoAmI MachineName");
            System.exit(1);
        }
        InetAddress a = InetAddress.getByName(args[0]);
        System.out.println(a);
    }
}
```



Der Java-Praxiskurs II 2

UNIVERSITÄT LEIPZIG
LPZ E-BUSINESS
applied telematics

Ü4.1 Bestimmung eigener IP-Adresse

- Ändern Sie das `WhoAmI`-Programm aus der Vorlesung so ab, dass die IP-Adresse des Rechners angezeigt wird, auf dem das Programm läuft.

```
import java.net.*;

public class WhoAmI {
    public static void main(String[] args)
        throws Exception {

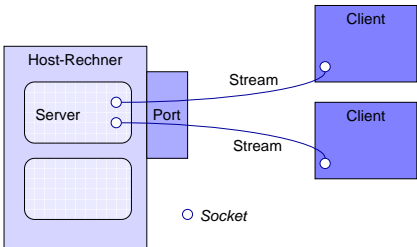
        InetAddress a = InetAddress.getLocalHost();
        System.out.println(a);
    }
}
```

Der Java-Praxiskurs II 3

UNIVERSITÄT LEIPZIG
LPZ E-BUSINESS
applied telematics

Wdh.: Ports und Sockets

- mehrere Clients können zur selben Zeit mit einem **Port** eines Servers verbunden sein
- Verbindungen werden über **Sockets** identifiziert



Der Java-Praxiskurs II 4

UNIVERSITÄT LEIPZIG
LPZ E-BUSINESS
applied telematics

Wdh.: Grundfunktion eines Clients

- Szenario: Auf Port 5155 unseres Rechners läuft ein Server, der bei Verbindungsaufbau eine Textzeile mit der aktuellen Uhrzeit überträgt.*
- Was muss ein Client tun, um auf diesen Dienst zuzugreifen?

- Verbindung zu Port 5155 auf localhost aufbauen
 - wir erhalten ein `Socket` für diese Verbindung
- Stream zum Lesen von Daten aus dem `Socket` besorgen
 - wir erhalten einen `InputStream`, der mit dem `Socket` verbunden ist
- einen `BufferedReader` an den `InputStream` hängen
 - damit können wir nicht nur Bytes, sondern ganze Textzeilen lesen
- eine Zeile aus dem `BufferedReader` lesen
 - Zeile ausgeben - fertig!

generischer Teil

Der Java-Praxiskurs II 5

UNIVERSITÄT LEIPZIG
LPZ E-BUSINESS
applied telematics

Wdh.: Grundfunktion eines Servers

- Szenario: Auf Port 5155 unseres Rechners läuft ein Server, der bei Verbindungsaufbau eine Textzeile mit der aktuellen Uhrzeit überträgt.*
- Was muss der Server tun, um diesen Dienst anzubieten?

- auf Port 5155 ein `Socket` für Verbindungsanfragen öffnen
 - wir erhalten einmalig ein sog. `Server-Socket`, das immer offen bleibt
- am `Server-Socket` nach Verbindungsanfragen lauschen
 - für jede Verbindung erhalten wir ein individuelles `Verbindungssocket`
- Stream zum Senden von Daten an das `V.Socket` besorgen
 - wir erhalten einen `OutputStream`, der mit dem `V.Socket` verbunden ist
- einen `PrintWriter` an den `OutputStream` hängen
 - damit können wir nicht nur Bytes, sondern auch Textzeilen schreiben
- eine Zeile in den `PrintWriter` schreiben
 - die das Datum enthält - fertig!

generischer Teil

Der Java-Praxiskurs II 6

Wdh.: Multi-threaded Server

- Aufteilung des Programms auf zwei Klassen:
- **Server** lauscht, nimmt Anfragen an und startet Threads
 - auf Server-Socket lauschen
 - Anfragen annehmen und Verbindungs-Socket besorgen
 - für jede neue Anfrage einen **Handler**-Thread instanziiieren
 - und `start()`-Methode des Threads aufrufen
 - Programmfortsetzung unmittelbar mit Lauschen
- **Handler** sind Threads, die Anfragen bearbeiten
 - Bearbeitung in `run()`-Methode parallel zu Server/anderen Threads
 - `run()` nimmt aber keine Parameter an und gibt nichts (`void`) zurück
 - benötigte Werte und Kanäle für Datenrückgabe müssen schon bei der Instanziierung des Handlers im Konstruktor übergeben werden!

Wdh.: Server-Klasse ChatServer

- Konstruktor und lauschende Endlosschleife

```
public ChatServer(int port) throws IOException {
    ServerSocket listener = new ServerSocket(port);
    Socket socket;
    while(true) {
        socket = listener.accept();
        new ChatHandler(socket).start();
        clientList.add(socket);
        id++;
    }
}
```

Wdh.: Server-Thread ChatHandler

- Für jeden verbundenen Client wird ein **ChatHandler**-Thread erzeugt, der Eingaben von ihm entgegennimmt.

```
import java.io.*;
import java.net.*;

public class ChatHandler extends Thread {
    private Socket socket;

    public ChatHandler(Socket socket) {
        this.socket = socket;
    }

    public void run() {...} // siehe nächste Folie
}
```

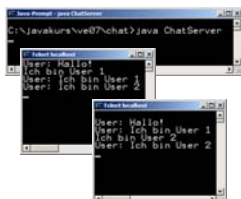
Wdh.: Server-Thread ChatHandler

- In Endlosschleife Textzeilen vom betreffenden Client lesen und diese zum Broadcast an den Server übergeben

```
public void run() {
    BufferedReader br; String s;
    try {
        br = new BufferedReader(
            new InputStreamReader(
                socket.getInputStream()));
        while (true) {
            s = br.readLine().trim();
            ChatServer.broadcast(s, "User");
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

Ü4.2.1 Aufgebotter Chat-Server

- ❖ 1. Geben Sie die beiden Klassen **ChatServer** und **ChatHandler** (aus der Vorlesung) ein. Prüfen Sie mittels des Telnet-Clients, ob das Chatten mit diesen beiden Klassen funktioniert. Beachten Sie dazu die Hinweise aus der Vorlesung.



Ü4.2.2 Chat-Server mit Namensabfrage

- ❖ 2. Zu Beginn einer jeden Chat-Session soll ein neuer Benutzer nach seinem Namen gefragt werden. Dieser soll immer vor dem Text ausgegeben werden, der von diesem User gebroadcastet wird.
- Ein **ChatHandler** kümmert sich ja immer um die Kommunikation mit genau einem Client.
- **Ansatz:** Wenn ein neuer **ChatHandler** gestartet wird, fragt er zuerst nach dem Benutzernamen des betreffenden Clients.
- Dann erst beginnt er, in einer Endlosschleife Eingaben zu empfangen und zum Broadcast an den **ChatServer** weiterzugeben.

Ü4.2.2 Chat-Server mit Namensabfrage

- Ergänzung der run()-Methode der Klasse ChatHandler:

```
public void run() {
    String s;
    try {
        BufferedReader br = new BufferedReader(
            new InputStreamReader(socket.getInputStream()));

        ... // Abfrage von name einfügen (siehe nächste Folie)

        while (true) {
            s = br.readLine().trim();
            ChatServer.broadcast(s, name);
        }
    } catch (Exception e) { e.printStackTrace(); }
}
```

Ü4.2.2 Chat-Server mit Namensabfrage

- Ergänzung der run()-Methode der Klasse ChatHandler:

```
public void run() {
    ...
    try {
        ...
        PrintWriter pw = new PrintWriter(
            socket.getOutputStream(), true);
        pw.println("****Welcome to LPZ-Chatter****");
        pw.print("What is your name? "); pw.flush();

        String name = br.readLine();
        ChatServer.broadcast(name +
            " has joined the chat.", "LPZ-Chatter");
        ...
    } catch (Exception e) { e.printStackTrace(); }
}
```

Ü4.2.3 Beenden der Chat-Sitzung

- 3. Wenn ein User beim Chatten BYE eingibt, so soll er den Chat auch verlassen können. Sprich, die while-Schleife im ChatHandler muss terminieren und das entsprechende Socket muss aus der Broadcast-Liste entfernt werden.

- Ansatz:** Bei jeder vom Client empfangenen Zeile überprüft der ChatHandler, ob sie das Kommando BYE enthält
- Ist das der Fall, so wird die „Endlosschleife“ beendet, der Client aus der Broadcast-Liste entfernt und die Verbindung zum Client geschlossen.
- Wie zuvor sind nur Änderungen am ChatHandler notwendig - der ChatServer bleibt unverändert
 - ChatServer implementiert ja auch nur die Verbindungsverwaltung, während die eigentliche Interaktion zwischen dem Server und diversen Clients in ihren jeweiligen ChatHandler-Threads passiert

Ü4.2.3 Beenden der Chat-Sitzung

```
public void run() {
    String s;
    try {
        ... // BufferedReader br & PrintWriter pw holen, name abfragen
        pw.println("Enter BYE to leave the chat.");

        boolean inLoop = true;
        while (inLoop) {
            s = br.readLine().trim();
            if (s.equals("BYE")) {
                ChatServer.broadcast(name +
                    " has left the chat.", "LPZ-Chatter");
                inLoop = false;
            }
            else
                ChatServer.broadcast(s, name);
        }
        ChatServer.remove(socket);
        socket.close();
    } catch (Exception e) { e.printStackTrace(); }
}
```

Demonstration des Chat-Servers

- Server-Start mit `java ChatServer`
- Client-Verbindung mit `telnet localhost 8190`

```

Der Java-Praxiskurs II
17
```