

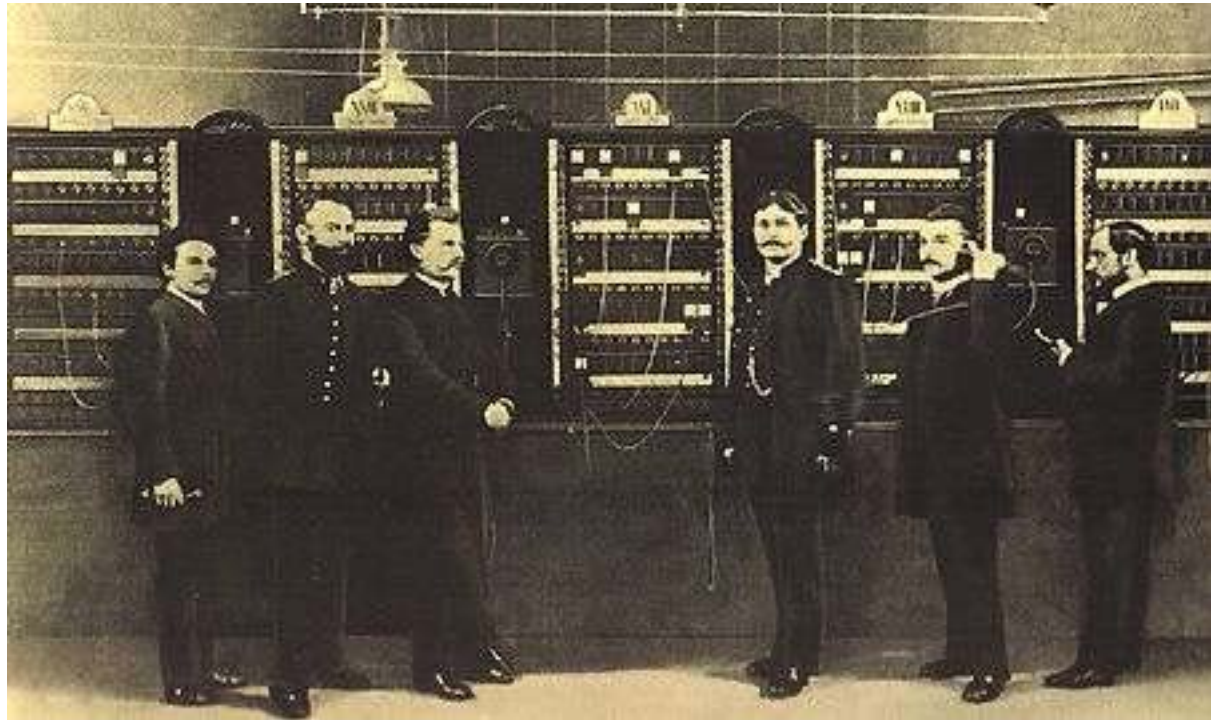
Automatisierung von Java Anwendungen (9)

Bean Scripting Framework (BSF), 3

Sockets (Paket "java.net")

Prof. Dr. Rony G. Flatscher

Telefon-Vermittlung, 1



Telefon-Vermittlungs-Stelle in Berlin um 1881

Quelle: <http://www.sultan-zonk.de/history/telefon/vermittlung.htm> (Abruf: 2003-06-06)

Sockets, 1

- "Socket" (deutsch: "Steck-Öffnung")
 - Konzept/Vorstellung kommt ursprünglich aus den Anfangszeiten der Telefonie
 - Vermittlungs-Schrank (englisch: "Switchboard"), der über eine entsprechende Anzahl an Steck-Öffnungen aufweist
 - Für jede "Amtsleitung" eine Steck-Öffnung
 - Für jeden Telefonanschluss innerhalb des Unternehmens eine Steck-Öffnung
 - Steck-Verbinder (Verbindungskabel), um zwei Steck-Öffnungen miteinander zu verbinden ("stöpseln", "verstöpseln")
 - Über diese (händisch hergestellte) Verbindung wird kommuniziert (telefoniert)

- "Socket" (deutsch: "Steck-Öffnung")
 - Ablaufbeispiel
 - (1) Die Telefonvermittlung nimmt einen Anruf entgegen und fragt nach dem gewünschten Teilnehmer
 - (2) Die Telefonvermittlung "verstöpselt" mit einem Kabel die Öffnung des eingehenden Anrufs mit der Öffnung des gewünschten Teilnehmers
 - (3) Anrufer und gewünschter Teilnehmer können nun miteinander kommunizieren

Sockets, 3

- TCP/IP "Sockets"
 - Kommunikationspartner sind zwei Programme
 - 65.536 Steck-Öffnungen stehen prinzipiell zur Verfügung
 - Nummeriert von 0 bis 65.535
 - "Portnumber" ("Öffnungsnummer", "Port"-Nummer)
 - Port-Nummern 0 bis 1024 besonders geschützt
 - Z.B. in Unix-Systemen nicht für normale Benutzer verfügbar
 - "Well-Known-Ports"
 - Z.B. Port-Nummer "80": für WWW-Anfragen
 - Programm, das über das Protokoll "http" angesprochen werden muss
 - Z.B. Apache Server, Microsoft's IIS (Internet Information Server)

Sockets, 4

- TCP/IP "Sockets"
 - Zuordnung von "Port"-Nummern
 - Vordefinierte "well-known ports": http, ftp, telnet, gopher, ...
 - Normalerweise aus dem reservierten Nummernkreis: von 1 bis 1024
 - Datei "services"
 - Z.B. auch Öffnungsnummern für Datenbanken wie DB2, Oracle, ...
 - Ab Port-Nummer 1025 frei verfügbar
 - Kollisionen dann, wenn eine Port-Nummer bereits von einem anderen Programm benutzt wird
 - Aber auch hier Konventionen, z.B. Portöffnung "8080", für WWW-Serversoftware von normalen (nichtprivilegierten) Benutzern
 - Wird häufig für "http"-Server benutzt, die von Programmierern vorübergehend (z.B. für Testzwecke) benötigt werden

Telefon-Vermittlung, 2

Auf Ihrem Rechner:
65.535 Anschlüsse (Ports)!

Vermittlungs-Schrank ("Switchboard") mit 100 Anschlüssen
Quelle: <http://www.sultan-zonk.de/history/telefon/vermittlung.htm>
(Abruf: 2003-06-06)



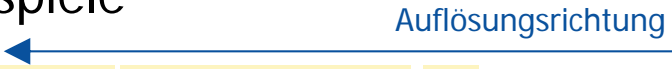
IP-Adressen und Rechnernamen, 1

- Jeder Rechner besitzt eine **weltweit (!) eindeutige** IP-Adresse
 - Internet-"Telefonnummer" eines Rechners
 - Zwei Versionen von Adressen im Einsatz
 - **IPv4-Adressen: 32-Bit** Werte (4 Bytes), Notation "A.B.C.D"
 - Viele Rechner verfügen auch über eine "loopback"-Adresse
 - 127.0.0.1
 - "localhost"
 - Für die Entwicklung und das Testen von Socket-Programmen
 - **IPv6-Adressen: 128-Bit** Werte (16 Bytes), verschiedene Notationen
 - Adressraum für die Welt viel zu klein geworden
 - Neuere Version, soll langfristig IPv4-Adressen ablösen
 - Adressraum praktisch unzählbar groß
 - Java-Klasse "**InetAddress**"

IP-Adressen und Rechnernamen, 2

- Rechnername
 - Für Menschen leichter merkbar als "nackte" Zahlen
 - "Namensdienste"
 - Verwalten die Abbildung "Rechnername → IP-Adresse"
 - Erlauben es, dass Rechner über beliebig viele verschiedene Rechnernamen verfügen können
 - Besitzt ein physischer Rechner mehrere Rechnernamen, werden alle Rechnernamen mit derselben IP-Adresse "aufgelöst"
 - Programm "**nslookup**"
 - Erlaubt u.a. das interaktive Auflösen von Rechnernamen zu IP-Adressen über die Kommandozeile

IP-Adressen und Rechnernamen, 3

- Rechnername und Domain ("Domäne")
 - Jeder Rechner gehört zu einer Gruppe ("Domain")
 - Gruppennamen werden weltweit verwaltet
 - Sollen sicherstellen, dass die Namensauflösung weltweit funktioniert
 - Beispiele
 -  `www.uni-augsburg.de`
 - `www.wiwi.uni-augsburg.de`
 - `www.wu-wien.ac.at`
 - `teletext.orf.at`
 - `www.sony.com`
 - `www.netscape.net`

IP-Adressen und Rechnernamen, 4

- IP-Adress-Objekte in Java

- Klasse "**InetAddress**"

- *Achtung! Keine Konstruktoren öffentlich verfügbar!*

- Benutzen Sie daher die folgenden statischen Methoden, die Instanzen dieser Klasse zurückliefern:

- InetAddress InetAddress.getName(String hostname)**

- Z.B.

- `InetAddress ia=InetAddress.getName("www.uni-augsburg.de");`

- InetAddress InetAddress.getByAddress(byte[] addr)**

- Z.B.

- `byte addr[]={ (byte)137, (byte)250, (byte)121, (byte)221 };`
 - `InetAddress ia=InetAddress.getByAddress(addr);`

- InetAddress InetAddress.getLocalHost()**

- Adresse *Ihres* eigenen Rechners, auf dem das Programm läuft

- `InetAddress ia=InetAddress.getLocalHost();`

IP-Adressen und Rechnernamen, 5

```
import java.io.*;
import java.net.*;
import java.util.*;
class getHostInfo
{
    public static void main (String args[])
    {
        InetAddress ia=null;
        if (args.length==0) // give localhost-infos
        {
            // attempt to get InetAddress object of localhost
            try { ia=InetAddress.getLocalHost(); }
            catch (UnknownHostException uhe)
            {
                System.err.println("No localhost defined for your computer!");
                System.exit(-1);
            }
        }
        else
        {
            // attempt to get InetAddress object by hostname
            try { ia=InetAddress.getByHost(args[0]); }
            catch (UnknownHostException uhe)
            {
                try // attempt to get InetAddress object by IP address
                {
                    StringTokenizer st = new StringTokenizer(args[0], ".");
                    byte ab[] = new byte [st.countTokens()]; // create appropriate byte array
                    for (int i=0; st.hasMoreTokens();i++) // loop over tokens
                        ab[i]=Byte.parseByte(st.nextToken()); // get value and create byte of it

                    ia=InetAddress.getByAddress(ab);
                }
                catch (UnknownHostException uhe2)
                {
                    System.err.println("[ "+args[0]+" ]: not a valid hostname nor IP-address!");
                    System.exit(-1);
                }
            }
        }
        System.out.println("          HostName: ["+ia.getHostHost()+"]\n"+
            "          IP address: ["+ia.getHostAddress()+"]\n\n"+
            "          canonicalHostName: ["+ia.getCanonicalHostName()+"]\n"+
            "          toString(): ["+ia+"]");
    }
}
```

Java-Klasse "Socket"

- Java-Klasse "Socket"
 - Baut eine Verbindung zu einem Rechner und einem Ziel- (Server)-programm auf, das auf diesem Rechner läuft
 - Rechner wird mit einer IP-Adresse oder einem Namen angegeben
 - Port-Nummer des Zielprogrammes ist dabei eine ganze Zahl ("int")
 - Ein- und Ausgabeoperationen sind über "Stream" realisiert
 - "InputStream"-Objekt zum Lesen von erhaltenen Daten
 - "getInputStream()"
 - "OutputStream"-Objekt zum Senden von Daten
 - "getOutputStream()"
 - Sockets werden mit der Methode "close()" geschlossen

Server-Programm

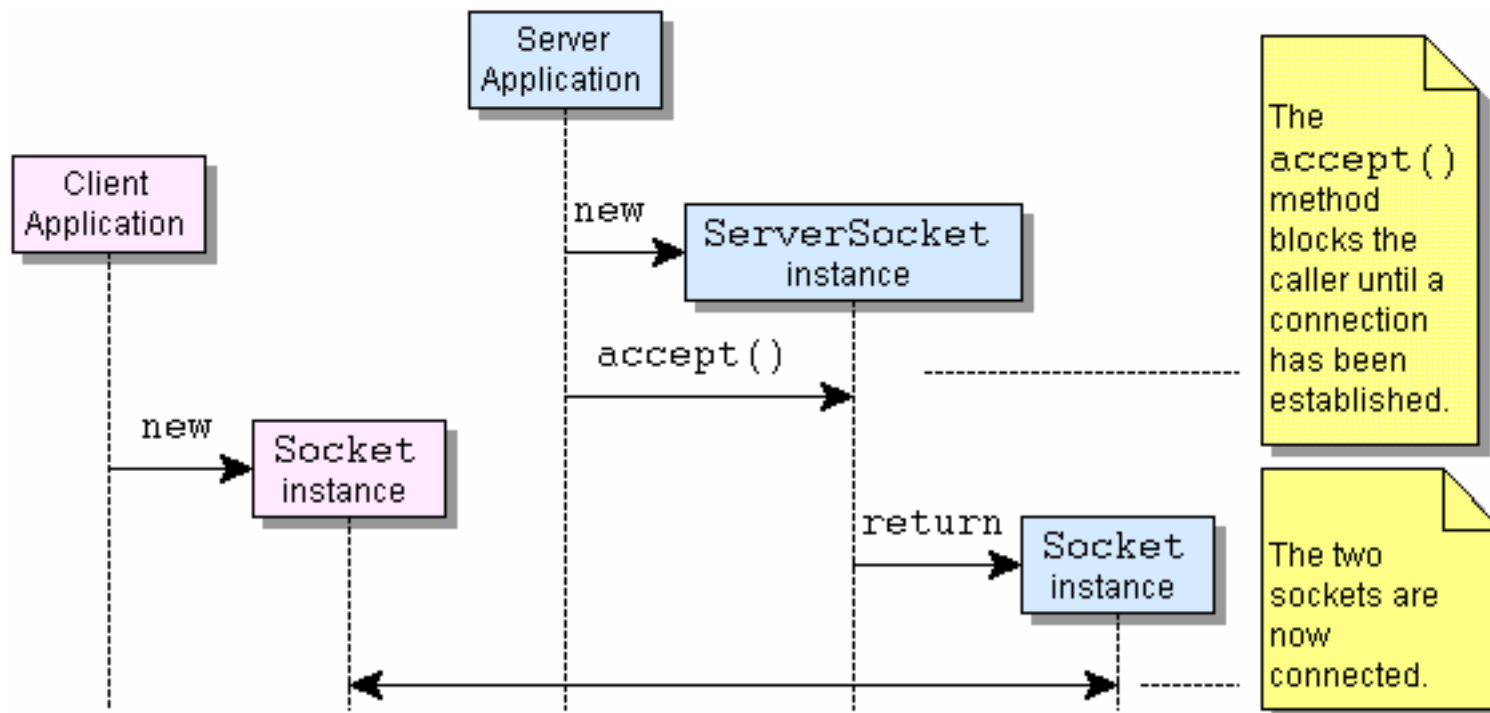
- "Server" benutzt Klasse "**ServerSocket**"
 - Programm, das auf Anfragen wartet und antwortet
 - Muss einem Port ("Stecköffnung") zugeordnet sein
 - Programm "hört" (engl. "**listen**") auf der festgelegten Öffnung
 - Programme, die als Server eingesetzt werden sollen, müssen
 - Eine Instanz der Klasse "**ServerSocket**" bilden
 - Die "abzuhörende" Port-Nummer kann beim Instanzieren oder über die Methode "**bind(...)**" festgelegt werden
 - Auf eine Anfrage warten/hören
 - Methode "**accept()**"
 - Auf eine Anfrage antworten
 - Kommunikation erfolgt über ein **Socket**-Objekt, das von "**accept()**" erzeugt und zurückgegeben wird

Client-Programm, Client und Server

- "Client"
 - Programm, das Anfragen an einen Server sendet
 - Bildet eine Instanz der Klasse "**Socket**"
 - Mehrere Konstruktoren verfügbar
 - Möglichkeit, den gewünschten Rechner anzugeben (`String`, `InetAddress`)
 - Möglichkeit, die gewünschte Port-Nummer anzugeben
 - Möglichkeit mit Methode "`connect(...)`" Rechner und Port-Nummer anzugeben
- "Client" und "Server"
 - Vereinbarung von Kommunikationsregeln
 - "Protokoll", z.B.
 - `http`, `ftp`, `telnet`, `sendmail`, ...
 - Als RFCs ("request for comments") im Internet verfügbar

Zusammenfassung: Schematischer Überblick

- On-line-Dokumentation über Java
 - Aus Sun's Dokumentation j2sdk1.4.1_01/docs/guide/net/overview/overview.html



Beispiel, Server-Programm

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

class testSocketServer
{
    public static void main (String args[])
    {
        try
        {
            ServerSocket srvSock=new ServerSocket( 8888 ); // Port-Öffnung dieses Programmes

            Socket socket2client = null; // Socket für Client-Kommunikation
            System.out.println("Server: warte auf Client...");
            socket2client = srvSock.accept();
            System.out.println("Server: Anfrage von irgendeinem Client erhalten!");

            byte b[]=new byte [2048]; // maximal 2 KB Daten auf einmal lesen
            int anzahl=socket2client.getInputStream().read(b);
            System.out.println("Server: erhaltene Client-Daten: ["+new String(b, 0, anzahl)+"]");

            System.out.println("Server: sende Daten an Client...");
            String msg="Hallo vom Server!";
            socket2client.getOutputStream().write(msg.getBytes());
        }
        catch (Exception exc)
        {
            System.err.println("Exception: ["+exc+"]");
            exc.printStackTrace();
            System.exit(-1);
        }
    }
}
```

Beispiel, Client-Programm

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

class testSocketClient
{
    public static void main (String args[])
    {
        try
        {
            Socket socket2server= new Socket(InetAddress.getLocalHost(), 8888);
            System.out.println("CLIENT: Socket erhalten!");

            System.out.println("CLIENT: sende Daten an Server...");
            String msg="Hallo, von Deinem Klienten!";
            socket2server.getOutputStream().write(msg.getBytes());

            byte b[]=new byte [100];    // maximal 100 Bytes auf einmal lesen
            int anz=socket2server.getInputStream().read(b);
            System.out.println("CLIENT: vom Server erhalten: ["+new String(b,0,anz)+"]");
        }
        catch (Exception exc)
        {
            System.err.println("Exception: ["+exc+"]");
            exc.printStackTrace();
            System.exit(-1);
        }
    }
}
```

Server:

Client:

Server: warte auf Client...

CLIENT: Socket erhalten!

Server: Anfrage von irgendeinem Client erhalten!

CLIENT: sende Daten an Server...

Server: erhaltene Client-Daten: [Hallo, von Deinem Klienten!]

Server: sende Daten an Client...

CLIENT: vom Server erhalten: [Hallo vom Server!]

Weitere Informationen

- Dokumentation
 - "SDK"
 - Software Development Kit
 - Tutorials
 - Guides
- <http://java.sun.com/docs>
 - Online-Abruf
 - Download der Dokumentation als zip-Archiv
 - Lokal entpacken
 - Dokumentation immer verfügbar!
- Ad hoc Recherche mit Suchmaschinen, z.B. <http://www.google.de>