In diesem Kapitel:
- *Was ist ein Socket?*
- *Die Socket Klasse*
- *Socket Ausnahmen / Exceptions*
- *Beispielprogramme*

7 *Sockets für Clients*

7.1. Was ist ein Socket?

In den zwei folgenden Kapitel werden wir uns mit Socket Client und Server beschäftigen. Anschliessend werden wir uns mit Sockets für UDP untersuchen.

7.2. Lernziele

Sie können einfache Socket Programme realisieren und mit Ihren Kollegen mit Hilfe einer Socket Client / Server Verbindung kommunizieren.

7.3. Aufbau

Der Aufbau ist wie in den vorangehenden Kapiteln:
- was ist ein Socket:
  kurze Erklärung des Begriffes
- die Socket Klasse
  Konstruktor
  Methoden
  Datenelemente
- Socket Exceptions
  Ausnahmen, die geworfen werden können und abgefangen werden müssen.
- einführende Beispiele (im obigen Text eingestreut)
- einige brauchbare, komplexere Beispiele

7.4. Was ist ein Socket?

Daten werden in Paketen über das Internet übermittelt. Diese Pakete bezeichnet man als *Datagramme*. Jedes Datagramm besitzt einen *Header* und einen Datenteil, im Englischen als *payload* (bezahlte Last) bezeichnet.

Im Header steht die Adresse und der Port, an die / den das Paket gesandt wird, sowie Port und Adresse der Datenquelle (Absender). Zudem enthält der Header weitere "Status" Informationen, mit deren Hilfe eine sichere Verbindung hergestellt und aufrecht erhalten werden kann.

Die Daten müssen in der Regel auf mehrere Datagramme aufgeteilt werden, da sie in einem Paket nicht Platz haben. Die Daten müssen also zerlegt, versandt und wieder zusammen gesetzt werden. Falls ein oder mehrere Pakete verloren gehen, müssen einzelne Pakete erneut angefordert werden. Es kann auch passieren, dass die Pakete in einer falschen Reihenfolge ankommen, da sie eventuell über unterschiedliche Transportwege vom Sender zum Empfänger gelangen. Dies ist die Aufgabe der Protokolle (TCP sorgt zum Beispiel für die richtige Reihenfolge).

Bei der Entwicklung von Berkeley Unix war unter anderem der Sun Microsystems Technische Direktor Bill Joy beteiligt war (er ist auch einer der Initiatoren von Jini und vielen andern Innovationen; Solaris entstand als direkter Abkömmling von Berkeley Unix).

Im Rahmen des Berkeley Projektes versuchte man, alle Eingaben und Ausgaben im Unix zu vereinheitlichen. Alle Eingaben und Ausgaben sollten wie einfache Datei Operationen aussehen. Der Benutzer braucht sich also nicht um die technischen Details zu kümmern. Die Netzwerk Version dieses Konzeptes bezeichnet am als Sockets. Der Benutzer soll also mit andern Benutzern des Systems kommunizieren können, ohne sich um die Details des Protokolls zu kümmern.

Diese Konzept hat sich als sehr brauchbar erwiesen: Sockets wurden für fast alle Betriebssysteme und entwickelt.

Was kann nun ein Socket?
1) Verbindung zu einem entfernten Rechner aufbauen
2) Daten senden
3) Daten empfangen
4) eine Verbindung beenden
5) einen Port binden
6) auf ankommende Daten hören
7) Verbindungsaufforderungen von andern Rechnern verarbeiten

Die Java Socket Klasse wird von Client und Server eingesetzt. Sie besitzt Methoden, welche die obigen ersten drei Operationen unterstützen.
Die nächsten drei Operationen braucht man nur auf der Server Seite. Sie werden in der Java ServerSocket implementiert.

Wie sieht nun eine Socket Verbindung aus Client Seite aus?

1. ein Socket Objekt wird erstellt
2. das Socket Objekt versucht mit einem entfernten Host Verbindung aufzunehmen.
3. nachdem die Verbindung aufgebaut ist, senden und empfangen der lokale und der remote Host Daten aus einem Input beziehungsweise Output Stream.
   Die Verbindung ist *voll-duplex* , Host und Client können Daten senden und empfangen.
   Die Bedeutung der Daten selber ergibt sich aus der Anwendung. Eine FTP oder ein HTTP sehen unterschiedlich aus.
4. nachdem die Datenübermittlung abgeschlossen ist, schliesst der Host oder der Client die Verbindung. Einige Protokolle, wie HTTP verlangen, dass nach jeder Datenübermittlung die Verbindung abgebaut wird (HTTP : GET Befehl holt sich Daten; der HTTP Server merkt sich aber nicht wer was angefordert hat.).

7.5. Untersuchen von Protokollen mit Hilfe von Telnet

Damit wir etwas besser verstehen, wie die einzelnen Protokolle funktionieren, wollen wir uns einzelne Protokolle, speziell HTTP, echo, ... mit Hilfe von Telnet ansehen. Sockets selber sind recht einfach; die Protokolle sind das Komplexe an der Kommunikation.

Wir können verschiedene Protokolle mit Hilfe von Telnet untersuchen, indem wir die Protokolle mit Hilfe spezieller Eingaben simulieren. Telnet versucht per Default über den Port 23 zu kommunizieren. Wir können aber durch Angabe einer Port Nummer auch andere Ports ansteuern.

telnet <host> <port>

7.5.1. SMTP (Port 25)

Schauen wir mal, wie sich ein SMTP Server verhält:
SMTP ist das Protokoll, mit dessen Hilfe Meldungen (emails) zwischen dem lokalen Rechner und dem Mailserver ausgetauscht werden.

Um heraus zu finden, wie sich SMTP verhält, können wir, falls wir in einem lokalen Netzwerk sind, ein eMail über Telnet versenden.

Hier der Ablauf:

telnet <host> <port>
... der Host meldet sich
<Eingabe>
... der Host antwortet

Meines Wissens hat Herr Müller eine Anleitung geschrieben, wie ein email über Telnet verschickt werden kann. Falls nicht, dann würde ich die Details noch nach liefern.

7.6. Die Socket Klasse

7.6.1. Lernziele

- Sie kennen die Konstruktoren der Socket Klasse und können diese anwenden.
- Sie kennen die Methoden der Socket Klasse und diese in einfachen Programmen einsetzen
- Sie die Parameter der Socket Konstruktoren und wissen, wie diese mit Hilfe der Methoden abgefragt werden können

7.6.2. Socket Klasse Übersicht

7.6.2.1. public class Socket extends Object

Diese Klasse implementiert Client Sockets (kurz "Sockets" genannt). Ein Socket ist ein Endpunkt einer Kommunikation zwischen zwei Maschinen. Die aktuelle Arbeit des Sockets wird durch eine Instanz der `SocketImpl` Klasse bewerkstelligt. Eine Applikation kann, indem sie eine eigene Socket Factory implementiert, eine Factory, die Sockets erzeugen kann, die der Umgebung am Besten angepasst sind, um zum Beispiel optimal mit einer Firewall zu kommunizieren.

Seit:

JDK1.0

Siehe auch:

`setSocketImplFactory(java.net.SocketImplFactory)`, `SocketImpl`

7.6.2.1.1. Konstruktor Übersicht

protected	**Socket**() Kreiert einen unabhängigen Socket, mit Hilfe des System Standard Types SocketImpl.
	Socket(InetAddress address, int port) Kreiert einen Stream Socket und verbindet ihn mit dem spezifizierten Port zur IP Adresse address.
	Socket(InetAddress host, int port, boolean stream) **Veraltet.** *Verwenden Sie statt dessen DatagramSocket an Stelle des UDP Transportes.*
	Socket(InetAddress address, int port, InetAddress localAddr, int localPort) kreiert einen Socket und verbindet ihn mit der spezifizierten remote Adresse an einem spezifizierten remote Port.
protected	**Socket**(SocketImpl impl) kreiert einen unverbundenen Socket mit Hilfe einer eigenen SocketImpl Klasse.
	Socket(String host, int port) kreiert einen Stream Socket und verbindet diesen mit dem spezifizierten Port Nummer des angegebenen Hosts.
	Socket(String host, int port, boolean stream) **Veraltet.** *Verwenden Sie DatagramSocket an Stelle eines UDP Transportes* .

	Socket(String host, int port, InetAddress localAddr, int localPort) kreiert ein Socket und verbindet diesen mit dem remote Host.

7.6.2.1.2. Methoden Übersicht

void	**close**() Schliesst diesen Socket.
InetAddress	**getInetAddress**() liefert die Adresse mit welcher der Socket verbunden ist.
InputStream	**getInputStream**() liefert einen Input Stream für diesen Socket.
InetAddress	**getLocalAddress**() liefert die lokale Adresse an die der Socket gebunden ist.
int	**getLocalPort**() liefert den lokalen Port, an den der Socket gebunden ist.
OutputStream	**getOutputStream**() liefert den Output Stream des Sockets.
int	**getPort**() liefert den remote Port dieses Sockets, mit dem dieser Socket verbunden ist.
int	**getReceiveBufferSize**() liefert den Wert der SO_RCVBUF Option dieses Sockets, der Buffer Grösse, welche auf dieser Plattform für diesen Socket eingesetzt wird.
int	**getSendBufferSize**() liefert den Wert der SO_SNDBUF Option für diesen Socket, also der Buffer Grösse, welche auf dieser Plattform für diesen Socket eingesetzt wird.
int	**getSoLinger**() liefert den Wert der Variable SO_LINGER.
int	**getSoTimeout**() liefert den Wert der Variable SO_TIMEOUT.
boolean	**getTcpNoDelay**() testet ob TCP_NODELAY gesetzt ist.
void	**setReceiveBufferSize**(int size) setzt die SO_RCVBUF Option für diesen DatagramSocket.
void	**setSendBufferSize**(int size) setzt die SO_SNDBUF Option für diesen DatagramSocket.
static void	**setSocketImplFactory**(SocketImplFactory fac) setzt die Client Socket Implementation Factory für diese Applikation.
void	**setSoLinger**(boolean on, int linger) Enable/disable SO_LINGER mit der spezifizierten linger Zeit in Sekunden.
void	**setSoTimeout**(int timeout) Enable/disable SO_TIMEOUT mit dem spezifizierten Timeout, in Millisekunden.
void	**setTcpNoDelay**(boolean on) Enable/disable TCP_NODELAY (disable/enable Nagle's Algorithmus).
String	**toString**() konvertiert den Socket in eine Zeichenkette.

7.6.2.2.1. Methoden, die von der Klasse java.lang.**Object** geerbt wurden

`clone`, `equals`, `finalize`, `getClass`, `hashCode`, `notify`, `notifyAll`, `wait`, `wait`, `wait`

Schauen wir jetzt die einzelnen Grössen im Detail an.

7.6.3. Die Konstruktoren

Die Konstruktoren für die Sockets sind recht einfach. Wir haben die Möglichkeit, Port und Host zu spezifizieren. Hosts können wir auch mit Hilfe der `InetAddress` oder einer Zeichenkette spezifizieren.

Ports nehmen einen Wert zwischen 0 und 65'535 an.

Einige Konstruktoren gestatten auch die Spezifikation von Port und Host der lokalen Datenquelle. Dies ist insbesondere dann nötig, wenn wir lokal einen Multihoming Host haben (siehe die Beschreibung eines Multihoming Hosts im vorigen Kapitel: auf eine IP Adresse werden mehrere symbolische Adressen gelegt).

Die Konstruktoren, welche geschützt sind (`protected`), sind wichtig, falls Sie Sockets mit spezieller Authentisierung, Datenkompression oder Verschlüsselung benötigen.

Schauen wir uns nun einige der Konstruktoren genauer an, in Form einfacher Beispiele.

7.6.3.1. public Socket(String host, int port) throws UnknownHostException, IOException

Dieser Konstruktor baut einen TCP Socket zu einem spezifizierten Port eines spezifizierten Hosts und versucht die Verbindung zum remote Host aufzubauen.

7.6.3.1.1. Programm Fragment

```
try (
     Socket sokLocal = new Socket("localhost", 80);
} catch(UnknownHostException uhe) {
     System.err.println(uhe);
} catch(IOException ioe) {
     System.err.println(ioe);
}
```

Mit diesem Konstruktor wird der Hostname als Zeichenkette ("localhost"), nicht in Form einer URL (http://localhost) oder einer `InetAddress` angegeben. Falls dieser Host nicht existiert oder nicht vorhanden ist, was auf das Gleiche herausläuft, wird eine Exception geworfen (`UnknownHostException`), falls die Verbindung nicht aufgebaut werden kann, dann wird eine `IOException` geworfen.

7.6.3.1.2. Instanzierung der Socket Klasse mit Port-Prüfung

Dieses Beispiel benutzt den Konstruktor und überprüft gleichzeitig, ob bestimmte Ports noch frei sind.

7.6.3.1.3. Programm Code

```
//Titel:       public Socket(String host, int port) throws
//             UnknownHostEXception, IOException
//Version:
//Copyright:    Copyright (c) 1999
//Autor:       J.M.Joller
//Firma:
//Beschreibung:       Suchen von einem Ports (0...1024) ,
/                          der TCP versteht

package Beispiel7_2;
import java.net.*;
import java.io.*;
class KlassenBeispiel7_2 {
  public static void main(String[] args) {
    System.out.println("Kapitel 7 : Sockets Beispiel 7.2");
    Socket theSocket;
    String host = "localhost";
    if (args.length > 0) {
      host = args[0];
    }
    for (int i = 0; i < 1024; i++) {
      try {
        theSocket = new Socket(host, i);
        System.out.println("Port "+i+" des Hosts "+host+" kennt TCP.");
      }
      catch (UnknownHostException uhe) {
        System.err.println("Host "+host+" Port "+i+" UnknownHostException : "+uhe);
        break;
      }
      catch (IOException ioe) {
        System.err.println("Host "+host+" Port "+i+" IOException : "+ioe);
        // kein Server an diesem Port
      }
    } // end for
    System.out.println("Ende des Beispiels");

  }  // end main

}  // end Klassen_Beispiel7_2
```

7.6.3.1.4. Ausgabe

Es bietet sich an, nicht alle `IOExceptions` und `UnknownHostExceptions` aus zu drucken! Falls Sie das Programm testen wollen, dann kommentieren Sie die `IOException` Ausgabe aus.

```
Kapitel 7 : Sockets Beispiel 7.2
Port 80 des Hosts localhost kennt TCP.
Port 135 des Hosts localhost kennt TCP.
Ende des Beispiels
```

7.6.3.2. public Socket(InetAddress host, int port) throws IOException

Dieser Konstruktor baut einen TCP Socket zu einem spezifizierten Port eines spezifizierten Hosts , der als InetAddress spezifiziert wird, und versucht die Verbindung zum remote Host aufzubauen.

7.6.3.2.1. Programm Fragment

```
InetAddress iaLocal = null;
Socket sokLocal;
try (
     iaLocal = new InetAddress("www.hta.fhz.ch");
} catch(UnknownHostException uhe) {
     System.err.println(uhe);
}
try {
     sokLocal = new Socket(iaLocal, 80);
} catch(IOException ioe) {
     System.err.println(ioe);
}
```

Mit diesem Konstruktor wird der Hostname als InetAddress also als Internet Adresse angegeben. Falls dieser Host nicht existiert oder nicht vorhanden ist, was auf das Gleiche herausläuft, wird eine Exception geworfen (`UnknownHostException`), falls die Verbindung nicht aufgebaut werden kann, dann wird eine `IOException` geworfen.

Diese Methode ist schneller als die Methode, die wir eben besprochen haben. Betrachten wir zur Illustration das gleiche Beispiel wie eben, aber diesmal mit der `InetAddress` Adresse des Hosts. Damit einige weitere Ports getestet werden, wurde das Programm leicht verändert.

7.6.3.2.2. Selbsttest Aufgabe

Versuchen mit System Routinen die Zeiten der beiden Beispiele zu messen und zu vergleichen. Vergessen Sie nicht, gleiche Port Bereiche zu vergleichen.

Hinweis : wir haben im letzten Jahr im Beispiel BufferedIO eine Zeitmessung durch geführt.Aus diesem Beispiel stammt das folgende Programm Fragment :

```
.
long lZeit=0; // ausserhalb der try ... catch Blöcke definieren
..
Date start = new Date();// Startzeit

...
Date end = new  Date(); // Endzeit

....
lZeit = end.getTime() - start.getTime(); // delta T
```

7.6.3.2.3. Instanzierung der Socket Klasse mit Port-Prüfung

Dieses Beispiel benutzt den Konstruktor und überprüft gleichzeitig, ob bestimmte Ports noch frei sind.

7.6.3.2.4. Programm Code

```
//Titel:        public Socket(InetAddress host, int port) throws
//                                                        IOException
//Version:
//Copyright:    Copyright (c) 1999
//Autor:       J.M.Joller
//Firma:
//Beschreibung:  Abfrage der Ports des Hosts (InetAddress), welche TCP
verstehen
package Beispiel7_3;

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

public class KlassenBeispiel7_3 {
  public static void main(String[] args) {

    System.out.println("Kapitel 7 Beispiel 7_3");
    Socket theSocket;
    String host = "localhost";

    if (args.length > 0) {
      host = args[0];
    }

    try {
      InetAddress theAddress = InetAddress.getByName(host);
      for (int i = 1024; i < 65536; i++) {
        try {
          theSocket = new Socket(theAddress, i);
          System.out.println("TCP Port " + i + " des Hosts " + host);
        }  // end try
        catch (IOException e) {
          // kein Server
        } // end catch
      } // end for
    } // end try
    catch (UnknownHostException e) {
      System.err.println(e);
    } // end catch
    System.out.println("Ende des Beispiel 7_3");
  }  // end main
}  // end Beispiel
```

7.6.3.2.5. Ausgabe

Es bietet sich an, nicht alle `IOExceptions` und `UnknownHostExceptions` aus zu drucken!

```
Kapitel 7 Beispiel 7_3
TCP Port 1026 des Hosts localhost
TCP Port 1027 des Hosts localhost
...
```

7.6.4. Methoden : Informationen über Sockets

Die Methodenübersicht lässt einem vermuten, dass ein Socket mehrere Datenfelder hat, auf die man mit `get...()` Methoden zugreifen kann.

Das ist aber nicht der Fall! Sockets besitzen eigentlich nur ein Feld `SocketImpl` Die Datenfelder, die man scheinbar abfragt, gehören in Wahrheit zu `SocketImpl`. Falls man die Socket Implementation ändert (indem man eine eigene `SocketImpl` konstruiert), braucht man an den Programmen nichts zu ändern.

7.6.4.1. public InetAddress getInetAddress()

Diese Methode liefert, bei gegebenem Socket Objekt, die Adresse des Hosts, mit dem eine Verbindung aufgebaut werden soll, beziehungsweise. ist.

7.6.4.1.1. Programm Fragment

```
try {
     Socket sokSocket = new Socket(" www.hta.fhz.ch ", 80);
     InetAddress iaHost = sokSocket.getInetAdress();
     ...
} catch (UnknownHostException uhe) {
     // Fehlerbehandlung
     ...
} catch (IOException ioe) {
     //   Fehlerbehandlung
}
```

7.6.4.2. public int getPort()

Diese Methode liefert, bei gegebenem Socket Objekt, den Port des Hosts, mit dem eine Verbindung aufgebaut werden soll, beziehungsweise. ist.

7.6.4.2.1. Programm Fragment

```
try {
     Socket sokSocket = new Socket(" www.hta.fhz.ch ", 80);
     int iPort = sokSocket.getPort();
     ...
} catch (UnknownHostException uhe) {
     // Fehlerbehandlung
     ...
} catch (IOException ioe) {
     //   Fehlerbehandlung
}
```

7.6.4.3. public int getLocalPort()

Diese Methode liefert, bei gegebenem Socket Objekt, den lokalen Port des Hosts, der mit einem remote Host eine Verbindung aufgebaut hat, beziehungsweise. aufbauen soll.

7.6.4.3.1. Programm Fragment

```
try {
     Socket sokSocket = new Socket(" www.hta.fhz.ch ", 80);
     int iPort = sokSocket.getLocalPort();
     ...
} catch (UnknownHostException uhe) {
     // Fehlerbehandlung
     ...
} catch (IOException ioe) {
     //   Fehlerbehandlung
}
```

Dieser lokale Port hat es in sich! Während der Port des entfernten Hosts vorgegeben wird, wird der lokale Port zur Laufzeit zugeteilt! Das System prüft zuerst, welche lokale Ports noch frei sind und ordnet der Kommunikation dann einen dieser Ports zu. Da die Port Nummer mit den Datagrammen an den entfernten Host übermittelt wird, kann dieser jederzeit seine Meldungen an den korrekten Port senden.

7.6.4.4. public InetAddress getLocalAddress()

Die Methode liefert, bei gegebenem Socket Objekt, die lokale Adresse des Hosts, der mit einem remote Host eine Verbindung aufgebaut hat, beziehungsweise. aufbauen soll.

7.6.4.4.1. Programm Fragment

```
try {
     Socket sokSocket = new Socket(hostname, 80);
     int iPort = sokSocket.getLocalAddress();
     ...
} catch (UnknownHostException uhe) {
     // Fehlerbehandlung
     ...
} catch (IOException ioe) {
     //   Fehlerbehandlung
}
```

Schauen wir uns ein umfangreicheres Beispiel an

7.6.4.4.2. Programm Beispiel

```
//Titel:       public InetAddress getLocalAddress()
//Version:
//Copyright:   Copyright (c) 1999
//Autor:       J.M.Joller
//Firma:
//Beschreibung:  Abfrage der Socket Informationen
package Beispiel7_4;
import java.net.*;
import java.io.*;

public class KlasseBeispiel7_4 {

  public static void main(String[] args) {
    System.out.println("Kapitel 7 Beispiel 4");
    for (int i = 0; i < args.length; i++) {
      try {
        Socket theSocket = new Socket(args[i], 80);
        System.out.println("Der lokale Host ist verbunden mit " + theSocket.getInetAddress()
         + " an Port "  + theSocket.getPort() + " vom lokalen Port "
         + theSocket.getLocalPort() + " der lokalen Adresse " + theSocket.getLocalAddress());
      }  // end try
      catch (UnknownHostException e) {
        System.err.println("Host " + args[i]+"  kann nicht gefunden werden");
      }
      catch (SocketException e) {
        System.err.println("Verbindung zum  Host " + args[i]+" kann nicht aufgebaut werden");
      }
      catch (IOException e) {
        System.err.println(e);
      }

    } // end for
    System.out.println("Ende Beispiel 7.4");
  }  // end main

}  // end
```

7.6.4.4.3. Ausgabe

Kapitel 7 Beispiel 4
Der lokale Host ist verbunden mit localhost/127.0.0.1 an Port 80 vom lokalen Port **4602** der lokalen Adresse localhost/127.0.0.1
Der lokale Host ist verbunden mit localhost/127.0.0.1 an Port 80 vom lokalen Port **4603** der lokalen Adresse localhost/127.0.0.1
Verbindung zum Host ztnw293 kann nicht aufgebaut werden
Ende Beispiel 7.4

Und hier die Ausgabe eines weiteren Programm Laufes. Wie Sie sehen, wird der lokale Port jedesmal neu gewählt!

Kapitel 7 Beispiel 4
Der lokale Host ist verbunden mit localhost/127.0.0.1 an Port 80 vom lokalen Port **4683** der lokalen Adresse localhost/127.0.0.1
Der lokale Host ist verbunden mit localhost/127.0.0.1 an Port 80 vom lokalen Port **4684** der lokalen Adresse localhost/127.0.0.1
Verbindung zum Host ztnw293 kann nicht aufgebaut werden
Ende Beispiel 7.4

7.6.4.5. public InputStream getInputStream() throws IOException

Die Methode liefert einen Eingabe Strom, aus dem Daten, die an den Client übermittelt werden, gelesen werden können

Schauen wir als erstes das *daytime* Protokoll an, welches im RFC867 spezifiziert wurde. Dieser Service steht normalerweise auf Port 13 zur Verfügung.

Sie können das Protokoll mit telnet testen:

telnet <host> 13
...
<Datum>
...

Der Daytime Server sendet eine Zeichenkette an den Client, der die Uhrzeit und das Datum festhält.

Schauen wir uns einmal an, wie wir diese Zeichenkette mit Hilfe eines Sockets erhalten und über einen InputStream an das Programm weiter leiten können.

7.6.4.5.1. Programm Fragment

```
DataInputStream dis;
try {
     Socket sokSocket = new Socket(hostname, 13);
     dis = new DataInputStream(sokSocket.getInputStream() );
     String strZeit = dis.readLine();
```

```
      ...
} catch (UnknownHostException uhe){...}catch (IOException ioe)
```

Schauen wir uns ein umfangreicheres Beispiel an

7.6.4.5.2. Programm Beispiel

```
//Titel:       public InputStream getInputStream() throws IOException
//Version:
//Copyright:   Copyright (c) 1999
//Autor:       J.M.Joller
//Firma:
//Beschreibung:  ein einfacher Daytime Client
//liest den Datumsstring über einen
//DataInput Stream
package Beispiel7_5;
import java.net.*;
import java.io.*;

public class KlasseBeispiel7_5 {

  public static void main(String[] args) {

    Socket sokSocket;
    String hostname;
    DataInputStream dis;

    if (args.length > 0) {
      hostname = args[0];
    }
    else {
      hostname = "localhost";
    }

    try {
      sokSocket = new Socket(hostname, 13);
      dis = new DataInputStream(sokSocket.getInputStream());
      String strDaytime = dis.readLine();
      System.out.println("Es ist " + strDaytime + " am Host " + hostname);
    }  // end try
    catch (UnknownHostException e) {
      System.err.println(e);
    }
    catch (IOException e) {
      System.err.println(e);
    }

  }  // end main

} // end daytimeClient
```

7.6.4.5.3. Ausgabe

Die Ausgabe können Sie sich ja vorstellen. Lassen Sie das Programm in einem Netzwerk kaufen, und fragen Sie einen Host ab, der den Daytime Service implementiert.

Typischerweise erhalten Sie eine Ausgabe wie:
It is<Wochentag><Datum> <Uhrzeit> at <host>

7.6.4.6. public OutputStream getOutputStream() throws IOException

Die Methode liefert einen Ausgabe Strom, mit dessen Hilfe Daten an den Socket Server übermittelt werden können

Schauen wir das *echo* Protokoll an, welches im RFC862 spezifiziert wurde. Dieser Service steht normalerweise auf Port 7 zur Verfügung.

Sie können das Protokoll mit telnet testen:

telnet <host> 7
...
<Zeichenkette>
<*Zeichenkette*> (echo vom Echo Server)
...

Schauen wir uns einmal an, wie wir eine Zeichenkette mit Hilfe eines Sockets senden und von einem Echo Server erhalten. Die Kommunikation erfolgt über einen InputStream an das Programm und einen OutputStream an den Server.

7.6.4.6.1. Programm Fragment

```
DataInputStream dis;
DataOutputStream dos;
try {
     Socket sokSocket = new Socket(hostname, 7);
     dis = new DataInputStream(sokSocket.getInputStream() );
     dos = new DataOutputStream(sokSocket.getOutputStream() );
     sin = new DataInputStream(System.in); // Konsole

     String strEingabe = sin.readLine();

     dos.println(strEingabe); // sendetdieEingabeanEchoServer
     System.out.println(dis.readLine() ); // Echo
     ...
} catch (UnknownHostException uhe){...}catch (IOException ioe)
```

Schauen wir uns ein umfangreicheres Beispiel an

7.6.4.6.2. Programm Beispiel

```
//Titel:       public OutputStream getOutputStream() throws IOException
//Version:
//Copyright:   Copyright (c) 1999
//Autor:       J.M.Joller
//Firma:
//Beschreibung:  Beispiel eines einfachen Echo Client
package Beispiel7_6;
import java.net.*;
import java.io.*;

public class KlasseBeispiel7_6 {
  public static void main(String args[]) {

    Socket theSocket;
    String hostname;
    DataInputStream theInputStream;
    DataInputStream userInput;
    PrintStream theOutputStream;
    String theLine;

    if (args.length > 0) {
      hostname = args[0];
    }
    else {
      hostname = "localhost";
    }

    try {
      theSocket = new Socket(hostname, 7);
      theInputStream = new DataInputStream( theSocket.getInputStream());
      theOutputStream = new PrintStream(theSocket.getOutputStream());
      userInput = new DataInputStream(System.in);
      while (true) {
        theLine = userInput.readLine();
        if (theLine.equals(".")) break;
        theOutputStream.println(theLine);
        System.out.println(theInputStream.readLine());
      }
    }  // end try
    catch (UnknownHostException e) {
      System.err.println(e);
    }
    catch (IOException e) {
      System.err.println(e);
    }
```

```
  }  // end main
}  // end echoClient
```

7.6.4.6.3. Ausgabe

Die Ausgabe können Sie sich ja vorstellen. Lassen Sie das Programm in einem Netzwerk kaufen, und fragen Sie einen Host ab, der den Echo Service implementiert.

Typischerweise erhalten Sie eine Ausgabe wie:
<EingabeString>
<Echostring>
...
...

Die Eingabe endet, wenn Sie einen Punkt eingeben : ".".

```
if (theLine.equals(".")) break;
```

7.6.5. Schliessen eines Sockets : Verbindungsabbau

Bisher haben wir uns keine Gedanken darüber gemacht, was mit einer Socket Verbindung geschieht, wenn die Kommunikation abgeschlossen ist.

In den bisherigen Beispielen hat der Garbage Collector jeweils dafür gesorgt, dass wieder aufgeräumt wird.

7.6.5.1. public synchronized void close() throws IOException

Die Methode schliesst einen Socket. Insbesondere beim local Port Tester bietet es sich an, den Socket zu schliessen, da im Extremfall sonst recht lange gewartet werden muss, bis der Socket endlich geschlossen wir. Sie erkennen dies auch daran, dass die Beispielprogramme (im JBuilder zu mindest) den gesamten verfügbaren Speicher aufbraucht, bis der Garbage Collector mal Zeit zum Aufräumen hat.

7.6.5.1.1. Programm Fragment

```
try {
     Socket sokSocket = new Socket(hostname, 13);
     sokSocket.close();
     ...
} catch (UnknownHostException uhe){...}catch (IOException ioe)
```

Schauen wir uns ein umfangreicheres Beispiel an

7.6.5.1.2. Programm Beispiel

```
//Titel:       public synchronized volid close() throws IOException
//Version:
//Copyright:   Copyright (c) 1999
//Autor:       J.M.Joller
//Firma:
//Beschreibung:  Port Tester, der den Socket nach Gebrauch schliesst
package Beispiel7_7;
import java.net.*;
import java.io.*;

public class KlasseBeispiel7_7 {

  public static void main(String[] args) {

    Socket theSocket;
    String host = "localhost";

    if (args.length > 0) {
      host = args[0];
    }

    try {
      InetAddress theAddress = InetAddress.getByName(host);
      for (int i = 1; i <= 65535; i++) {
       try {
         // falls der Host die Verbindung zum Port nicht zulaesst, wird
         // eine Exception ausgeloest
         theSocket = new Socket(theAddress, i);
         System.out.println("Es gibt einen Server am Port " + i + " des Hosts " + host);
         theSocket.close();          // schliessen des Sockets
       }
       catch (IOException e) {
         // kein Server an diesem Port
       }
      } // end for
    } // end try
    catch (UnknownHostException e) {
      System.err.println(e);
    }
  }
}
```

7.6.5.1.3. Ausgabe

Die Ausgabe ist identisch mit jener der andern Port Scanner Beispiele.
Der Unterschied besteht darin, dass jetzt die Sockets, welche laufend instanziert werden, auch wieder explizit vernichtet werden.

```
Es gibt einen Server am Port 80 des Hosts localhost
Es gibt einen Server am Port 135 des Hosts localhost
Es gibt einen Server am Port 1026 des Hosts localhost
Es gibt einen Server am Port 1027 des Hosts localhost
```

7.7. Socket Implementation SocketImpl

Die Klasse `SocketImpl` gestattet es dem Programmierer, eine eigene Socket Factories zu definieren.

7.7.1. java.net Class SocketImpl

```
java.lang.Object
  |
  +--java.net.SocketImpl
```

---

public abstract class **SocketImpl**
extends Object
implements SocketOptions
Die abstrakte Klasse `SocketImpl` ist die gemeinsme Superklasse aller Klassen, die Sockets implementieren. Die Klasse wird eingesetzt, um Client und Server Sockets zu implementieren.
Ein "einfacher" Socket implementiert diese Methoden genau wie beschrieben, ohne zu versuchen durch Firewall oder Proxy Server zu kommunizieren.

Seit:

JDK1.0

7.7.1.1. Übersicht über die Datenfelder

protected InetAddress	**address** Die IP Adresse des remote Endes dieses Sockets.
protected FileDescriptor	**fd** Die Dateibeschreibung (Objekt) dieses Sockets.
protected int	**localport** Die lokale Port Nummer mit dem dieser Socket verbunden ist.
protected int	**port** Die Port Nummer des remote Hosts mit dem dieser Socket verbunden ist.

7.7.1.2. Datenfelder aus dem Interface java.net.SocketOptions

IP_MULTICAST_IF, SO_BINDADDR, SO_LINGER, SO_RCVBUF, SO_REUSEADDR, SO_SNDBUF, SO_TIMEOUT, TCP_NODELAY

7.7.1.3. Konstruktoren Übersicht

SocketImpl()	

7.7.1.4. Methoden Übersicht

protected abstract void	**accept**(SocketImpl s) Akzeptiert eine Verbindung.
protected abstract int	**available**() Liefert die Anzahl Bytes die von diesem Socket ohne Blackbildung gelesen werden können.
protected abstract void	**bind**(InetAddress host, int port) Bindet diesen Socket an eine spezifizierte Port Nummer des spezifizierten Host.
protected abstract void	**close**() Schliesse diesen Socket.
protected abstract void	**connect**(InetAddress address, int port) Verbindet diesen Socket andie spezifizierte Port Nummer des spezifizierten Hosts.
protected abstract void	**connect**(String host, int port) Bindet diesen Socket an den spezifizierten Port des benannten Hosts.
protected abstract void	**create**(boolean stream) kreiert entweder einen Stream oder einen Datagram Socket.
protected FileDescriptor	**getFileDescriptor**() Liefert den Wert des Socket `fd` Feldes.
protected InetAddress	**getInetAddress**() Liefert den Wert des Adressfeldes dieses Socket..
protected abstract InputStream	**getInputStream**() liefert einen Input Stream für diesen Socket.
protected int	**getLocalPort**() liefert den Wert des localport dieses Sockets.
protected abstract OutputStream	**getOutputStream**() Liefert einen Output Stream für diesen Socket.
protected int	**getPort**() Liefert den Wert des Port Feldes dieses Sockets.
protected abstract void	**listen**(int backlog) Setzt die maximale Länge der Warteschlange für ankommende Verbindungen.
String	**toString**() liefert Adresse und Port dieses Socket als ein `String`.

7.7.1.5. Methoden von class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait

Die Details des Sockets werden mit Hilfe des Interfaces SocketOptions spezifiziert

7.7.2. java.net Interface SocketOptions

wird implementiert von folgenden Klassen:
DatagramSocketImpl, SocketImpl

---

public interface **SocketOptions**
Interface für Methoden der get/set socket Optionen. Diese Interface wird implementiert durch: **SocketImpl** and **DatagramSocketImpl**. Subklassen dieser Klassen sollten die Methoden überschreiben, um eigene Optionen zu implementieren.
Die Methoden und Konstanten sind für die Socket Implementation gedacht. Daher sollten Subklassen von SocketImpl oder DatagramSocketImpl nicht verändert werde, ausser dies ist unbedingt nötig..
Ein Subset der Standard BSD (Berkeley) Socket Optionen werden in den JDK Basisklassen, **PlainSocketImpl** und **PlainDatagramSocketImpl** implementiert. Hier eine Kurzbeschreibung dieser Felder.

7.7.2.1. Datenfelder Übersicht

static int	**IP_MULTICAST_IF** Interface über das Multicast Pakete versandt werden.
static int	**SO_BINDADDR** Lies die lokale Adressbindung eines Socket.
static int	**SO_LINGER** Spezifikation eines linger-on-close Timeout Intervalls.
static int	**SO_RCVBUF** Buffergrösse für ankommenden Netzwerk IO
static int	**SO_REUSEADDR** Setzt SO_REUSEADDR für einen Socket.
static int	**SO_SNDBUF** Buffergrösse für zu senden Netzwerk IO.
static int	**SO_TIMEOUT** Setzt Timeout für Socket Operationen
static int	**TCP_NODELAY** Nagle's Algorithmus ausschalten.

7.7.2.2. Methoden Übersicht

Object	**getOption**(int optID) liest den Wert einer Option.
void	**setOption**(int optID, Object value) Enable/disable die Option *optID*.

Schauen wir uns einige Details genauer an:

7.7.2.3. public static synchronized void setSocketFactory(SocketImplFactory fac) throws SocketException

Diese Methode setzt die Socket Factory, mit der das *System* dann seine Sockets generieren wird.

7.7.2.4. Socket Optionen

Oben haben wir gesehen, dass man sehr viele zum Teil sicher unbekannte Optionen und Parameter setzen kann. Hier möchten wir einige dieser Parameter kurz besprechen.

7.7.2.4.1. TCP_NODELAY

Dieser Parameter sorgt dafür, dass die Datenpakete sofort, also ohne Verzögerung versendet werden. Normalerweise werden die Daten zu möglichst grossen Paketen zusammen gefasst. Zudem wartet das lokale System, bis der remote Host ein Paket (als Bestätigung oder aus Fairness) übermittelt hat: das bezeichnet man als *Nagel's Algorithmus*.
Das Problem bei diesem Algorithmus besteht darin, dass das lokale System unendlich lange warten muss, fass das remote System überhaupt nichts zurück sendet. Das System hängt sich quasi auf.
Falls TCP_DELAY auf `true` gesetzt wird, kann dieses Problem eliminiert werden.

`public void setTcpNoDelay(true) throws SocketException` schaltet das Buffern der Daten aus; `setTcpNoDelay(false)` schaltet das Buffern wieder ein.
Die Abfrage des Status geschieht mit Hilfe der Methode `public boolean getTcpNoDelay() throws SocketException.`

7.7.2.4.2. SO_LINGER

Mit Hilfe der Methoden:

```
public void setSoLinger(boolean on, int seconds) throws
SocketEXception;
public int getSoLinger() throws SocketEXception;
```

kann dieser Parameter gesetzt und abgefragt werden.
Was bedeutet der Parameter?

Falls ein Socket geschlossen wird, muss irgend wie definiert sein, was mit den Datagrammen geschieht, die noch nicht versandt wurden. Per Default wird das System versuchen noch alle ausstehenden Datenpakete zu versenden.
Falls die linger Zeit = 0 ist, dann werden alle noch ausstehenden Übertragungen abgebrochen, also alle noch nicht versandten Pakete weg geworfen.
Falls die linger Zeit positiv ist, dann wartet das System diese Zeit (in Sekunden). Es können also weiterhin Daten übermittelt werden. Danach werden alle noch nicht gesandten Datagramme vernichtet.
Falls die linger Zeit = -1 ist, dann stellt das System soviel Zeit wie nötig zur Verfügung, damit die Daten übertragen werden können.

Nach dem Programmcode ...setSoLinger(240) hat das System also 2 Minuten Aufräumzeit.

7.7.2.4.3. SO_TIMEOUT

Dieser Parameter kann mit Hilfe der folgenden Methoden gesetzt / abgefragt werden:

```
public synchronized void setSoTimeout(int ms) throws
SocketException;

public synchronized int getSoTimeout() throws SocketException;
```

Normalerweise wird beim Lesebefehl `read`() so lange gewartet, bis das System genügend Bytes gelesen hat. Falls die Timeout Zeit gesetzt wurde, wird die Leseoperation nach einer bestimmten Zeit abgebrochen, ohne aber den Socket zu schliessen!

Beispiel:

```
... sokSocket.setSoTimeout(180000);
// das System wartet max 3 Minuten
```

7.7.3. Socket Exceptions

Die Ausnahme SocketException ist in der Regel zu unpräzise.

7.7.3.1. java.net Class SocketException

```
java.lang.Object
  |
  +--java.lang.Throwable
        |
        +--java.lang.Exception
              |
              +--java.io.IOException
                    |
                    +--java.net.SocketException
```

Direkt bekannte Subklassen:

BindException, ConnectException, NoRouteToHostException

public class **SocketException**
extends IOException
zeigt an, dass bei einem darunter liegenden Protokoll, wie TCP, ein Fehler auftrat..

Seit:

JDK1.0

Siehe Auch:

Serialized Form

7.7.3.1.1. Konstruktor Übersicht	
SocketException() Konstruiert eine Socket Exception, ohne speziellen Meldungstext.	
SocketException(String msg) Konstruiert eine Protokoll Exception mit der betreffenden Fehlermeldung	

7.7.3.1.2. Methoden aus class java.lang.Throwable
fillInStackTrace, getLocalizedMessage, getMessage, printStackTrace, printStackTrace, printStackTrace, toString

7.7.3.1.3. Methods aus class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait

Die NoRouteToHost Exception wird typischerweise dann gesetzt, wenn eine Firewall oder ein Proxy Server die Verbindung erschwert.

7.8. Einige komplexere Beispiele

Nach den grundlegenden Konstruktoren, die alle recht einfach sind, wollen wir nun einige Anwendungsbeispiele ansehen, die etwas komplexer sind:

7.8.1. Lernziele

Sie kennen einige der bekannteren TCP basierten Netzwerk Werkzeuge

- Finger
- Whois

und können deren Funktionsweise an Hand eines Java Programmes nach vollziehen.

7.8.2. Finger

Dieses TCP basierte Protokoll auf Port 79 liefert eine Kurzmeldung über das System oder einen Benutzer:

telnet <host> 79
...
Login Name TTY ...

Das System liefert bei Eingabe eines Benutzernamens dessen Anschluss, Namen und eventuell weitere Informationen.

Schauen wir uns das Ganze in Java implementiert an:

7.8.2.1. Programm Code

```
//Titel:       Finger
//Version:
//Copyright:   Copyright (c) 1999
//Autor:       J.M.Joller
//Firma:
//Beschreibung:  einfaches Finger Programm:
//liest vom Finger Server die Beschreibung
//und schreibt diesen String an System.out
package Beispiel7_8;
import java.net.*;
import java.io.*;

public class KlasseBeispiel7_8 {

  public final static int port = 79;

  public static void main(String[] args) {

    String hostname;
    Socket theSocket;
    DataInputStream theFingerStream;
    PrintStream ps;
```

```
    try {
      hostname = args[0];
    }
    catch (Exception e) {
      hostname = "localhost";
    }

    try {
      theSocket = new Socket(hostname, port, true);
      ps = new PrintStream(theSocket.getOutputStream());
      for (int i = 1; i < args.length; i++) ps.print(args[i] + " ");
      ps.print("\r\n");
      theFingerStream = new DataInputStream(theSocket.getInputStream());
      String s;
      while ((s = theFingerStream.readLine()) != null) {
        System.out.println(s);
      }
    }
    catch (IOException e) {
      System.err.println(e);
    }

  }

}
```

7.8.2.1.1. Programmaufruf

Das Finger Programm benötigt zwei Eingabeparameter (localhost als Defaultwert führt in der Regel höchstens zu einer Exception):
java KlasseBeispiel7_8.class <host> <Benutzer>

7.8.2.1.2. Ausgabe

```
java.net.ConnectException: Connection refused: no further information
java.net.NoRouteToHostException: Operation timed out: no further information
java.net.NoRouteToHostException: Operation timed out: no further information
```

Das passiert, wenn man das Ganze lokal testet!

7.8.3. Whois

Das Whois Protokoll ist eine Art Verzeichnisdienst, welches im RFC954 definiert wurde. In zwischen wurde eine neue Version definiert, in den RFC's RFC1834 und RFC1835.

Die grundlegende Struktur des WOIS Protokolls sieht wie folgt aus:

1. der Client öffnet einen TCP Socket auf Port 43, zum Beispiel am Server whois.internic.net oder Switch.ch
2. der Client sendet eine Suche (Zeichenkette), welche mit \r\n abgeschlossen wird. Die Zeichenkette enthält einen Namen, mehrere Namen oder Spezialbefehle.
3. der Server sendet eine unbekannte Anzahl Textzeichen an den Client zurück
4. der Client zeigt diese Informationen an

Das Whois Protokoll kennt mehrere Präfixes:

Präfix	**Bedeutung**
Domain	finde Domain Infos
Gateway	zeige nur Gateway Infos
Group	zeige nur Gruppen Datensätze
Host	Host Informationen
Network	Netzwerk Informationen
Organization	Organisationen
Person	Personendaten
Mailbox	email basierte Suche
Partial	alle Daten, die mit ... anfangen

7.8.3.1. Beispiel

whois Person Partial Jo

Liefere alle Informationen über die Person Jo oder Personen, die mit Jo anfangen.

7.8.3.2. Selbsttestaufgabe

Testen Sie den Whois Server von Internic.net mit Hilfe des Telnet Port 43.

7.8.3.3. Programm Code

```
//Titel:       Whois
//Version:
//Copyright:    Copyright (c) 1999
//Autor:       J.M.Joller
//Firma:
//Beschreibung:  einfache Whois Abfrage
package Beispiel7_11;
import java.net.*;
import java.io.*;

public class KlasseBeispiel7_11 {
```

```
 public final static int port = 43;
 public final static String hostname = "whois.internic.net";

 public static void main(String[] args) {

   Socket theSocket;
   DataInputStream theWhoisStream;
   PrintStream ps;

   try {
     theSocket = new Socket(hostname, port, true);
     ps = new PrintStream(theSocket.getOutputStream());
     for (int i = 0; i < args.length; i++) ps.print(args[i] + " ");
     ps.print("\r\n");
     theWhoisStream = new DataInputStream(theSocket.getInputStream());
     String s;
     while ((s = theWhoisStream.readLine()) != null) {
       System.out.println(s);
     }
   }
   catch (IOException e) {
     System.err.println(e);
   }

 }

}
```

7.8.3.4. Ausgabe

Die Ausgabe wurde in eine Datei umgeleitet. Sie ist recht umfangreich und wird daher hier nicht reproduziert. Hier ein Auszug:

```
The Data in Network Solutions' WHOIS database is provided by Network
Solutions for information purposes, and to assist persons in obtaining
information about or related to a domain name registration record.
Network Solutions does not guarantee its accuracy.  By submitting a
WHOIS query, you agree that you will use this Data only for lawful
purposes and that, under no circumstances will you use this Data to:
(1) allow, enable, or otherwise support the transmission of mass
unsolicited, commercial advertising or solicitations via e-mail
(spam); or  (2) enable high volume, automated, electronic processes
that apply to Network Solutions (or its systems).  Network Solutions
reserves the right to modify these terms at any time.  By submitting
this query, you agree to abide by this policy.
No match for "WWW.HTA.FHZ.CH".
```

7.8.3.5. Selbsttestaufgabe

Testen Sie das Programm, wobei Sie die Parameter nicht vergessen dürfen!

7.8.3.6. Programm Code als Applet mit GUI

```
//Titel:       Whois mit AWT Oberfläche
//Version:
//Copyright:   Copyright (c) 1999
//Autor:       J.M.Joller
//Firma:
//Beschreibung:  AWT + Whois
package Beispiel7_12;
import java.applet.Applet;
import java.awt.*;
import java.net.*;
import java.io.*;


public class whoisApplet extends Applet {

  public final static int port = 43;
  public final static String hostname = "whois.internic.net";

  TextField searchString;
  TextArea detailView;
  List names;
  Button findButton;
  CheckboxGroup searchIn;
  CheckboxGroup searchFor;
  Checkbox exactMatch;

  public static void main(String[] args) {

    whoisApplet a = new whoisApplet();
    a.init();
    a.start();

    Frame appletFrame = new Frame("whois");
    appletFrame.add("Center", a);
    appletFrame.resize(500,300);
    appletFrame.move(50,50);
    appletFrame.show();

  }

  public void init() {

    setLayout(new BorderLayout());
    // The TextArea in the center will expand to fill the
    // center of the BorderLayout
    detailView = new TextArea(12, 40);
    detailView.setEditable(false);
    names = new List(12, false);
```

```
  Panel CenterPanel = new Panel();
  CenterPanel.setLayout(new GridLayout(1, 2, 10, 10));
  CenterPanel.add("Center", names);
  CenterPanel.add(detailView);
  add("Center", CenterPanel);

  // You don't want the buttons in the south and north
  // to fill the entire sections so add Panels there
  // and use FlowLayouts in the Panel
  Panel NorthPanel = new Panel();
  Panel NorthPanelTop = new Panel();
  NorthPanelTop.setLayout(new FlowLayout(FlowLayout.LEFT));
  NorthPanelTop.add(new Label("Whois: "));
  searchString = new TextField(30);
  NorthPanelTop.add("North", searchString);
  exactMatch = new Checkbox("Exact Match", null, true);
  NorthPanelTop.add(exactMatch);
  NorthPanel.setLayout(new BorderLayout(2,1));
  NorthPanel.add("North", NorthPanelTop);
  Panel NorthPanelBottom = new Panel();
  NorthPanelBottom.setLayout(new GridLayout(1,2,5,5));
  NorthPanelBottom.add(initRecordType());
  NorthPanelBottom.add(initSearchFields());
  NorthPanel.add("Center", NorthPanelBottom);
  Panel SouthPanel = new Panel();
  findButton = new Button("Find");
  SouthPanel.add("South", findButton);
  add("South", SouthPanel);
  add("North", NorthPanel);

 }

 public Panel initRecordType() {

  Panel p = new Panel();
  p.setLayout(new GridLayout(6, 2, 5, 2));
  searchFor = new CheckboxGroup();
  p.add(new Label("Search for:"));
  p.add(new Label(""));
  p.add(new Checkbox("Any", searchFor, true));
  p.add(new Checkbox("Network", searchFor, false));
  p.add(new Checkbox("Person", searchFor, false));
  p.add(new Checkbox("Host", searchFor, false));
  p.add(new Checkbox("Domain", searchFor, false));
  p.add(new Checkbox("Organization", searchFor, false));
  p.add(new Checkbox("Group", searchFor, false));
  p.add(new Checkbox("Gateway", searchFor, false));
  p.add(new Checkbox("ASN", searchFor, false));
  return p;
```

```
}

public Panel initSearchFields() {

  Panel p = new Panel();
  p.setLayout(new GridLayout(6, 1, 5, 2));
  searchIn = new CheckboxGroup();
  p.add(new Label("Search In:"));
  p.add(new Checkbox("All", searchIn, true));
  p.add(new Checkbox("Name", searchIn, false));
  p.add(new Checkbox("Mailbox", searchIn, false));
  p.add(new Checkbox("Handle", searchIn, false));
  return p;

}

public boolean action(Event e, Object o) {

  if (e.target == searchString || e.target == findButton) {
    lookUpNames(searchString.getText());
    return true;
  }
  else if (e.target == names) {
    getFullRecord(names.getSelectedItem());
    return true;
  }
  else {
    return false;
  }

}

public String makeSuffix() {

  String suffix = "";
  if (!exactMatch.getState()) suffix = ".";
  return suffix;

}

public String makePrefix() {

  String searchForLabel = searchFor.getCurrent().getLabel() + " ";
  String searchInLabel = searchIn.getCurrent().getLabel() + " ";
  if (searchInLabel.startsWith("A")) searchInLabel = "";
  if (searchForLabel.startsWith("A")) searchForLabel = "";
  return searchForLabel + searchInLabel + "$";

}
```

```
  public void lookUpNames(String name) {

    Socket theSocket;
    DataInputStream theWhoisStream;
    PrintStream ps;
    String s;

    try {
      theSocket = new Socket(hostname, port, true);
      ps = new PrintStream(theSocket.getOutputStream());
      theWhoisStream = new DataInputStream(theSocket.getInputStream());
      /* A reader noticed that the folowing line had a redundant call to
         searchString.getText(). I've fixed it below */
//    ps.print(makePrefix() + searchString.getText() + makeSuffix() + "\r\n");
      ps.print(makePrefix() + name + makeSuffix() + "\r\n");
      names.clear();
      while ((s = theWhoisStream.readLine()) != null) {
        names.addItem(s);
      }
    }
    catch (IOException e) {
      System.err.println(e);
    }

  }

  public void getFullRecord (String summary) {

    Socket theSocket;
    DataInputStream theWhoisStream;
    PrintStream ps;
    String s;
    String handle = getHandle(summary);

    try {
      theSocket = new Socket(hostname, port, true);
      ps = new PrintStream(theSocket.getOutputStream());
      theWhoisStream = new DataInputStream(theSocket.getInputStream());
      ps.print("!" + handle + "\r\n");
      detailView.setText("");
      while ((s = theWhoisStream.readLine()) != null) {
        detailView.appendText(s + "\n");
      }
    }
    catch (IOException e) {
      System.err.println(e);
    }

  }
```

```
  String getHandle(String s) {

    int begin = s.indexOf("(") + 1;
    int end = s.indexOf(")", begin);

    return s.substring(begin,end);

  }

}
```

7.9. Schlussbemerkungen

Was sind Sockets und was kann man damit machen?
Diese Grundfrage lässt sich vereinfacht wie folgt beantworten:

- Sockets stellen eine Verbindung zwischen zwei Rechnern dar, analog zu einer Datei
- Mit Hilfe von Sockets können zwei Rechner leicht und effizient mit einander kommunizieren. Die Komplexität der Kommunikation ist dabei dem Benutzer nicht sichtbar; er kann einfach lesen und schreiben