In diesem Kapitel:

- *DNS, IP Adressen und all das Zeugs*
- *Die InetAddress Klasse*
- *Umwandlung von IP Adressen in Namen*
- *Beispielprogramme*

4

Verarbeiten von Internet Adressen

4.1. DNS, IP Adressen und all das Zeugs

Rechner, die mit dem Internet verbunden sind, werden als *host* bezeichnet. Jeder Host wird (heute : 1999) mit Hilfe einer eindeutigen 32-Bit Zahl identifiziert, der Internet Adresse, der IP Adresse, der Host Adresse (alles das Gleiche). Eine IP Adresse wird normalerweise, wegen der besseren Lesbarkeit, in Form von vier Zahlen geschrieben, die je zwischen 0 und 255 liegen und durch einen Punkt getrennt werden:

xxxx.yyyy.zzzz.hhhh
zum Beispiel : **192.65.123.12**

IP Adressen sind gut geeignet für eine automatische Verarbeitung mit Hilfe von Rechner - Programmen. Lesbar sind sie nicht unbedingt! Um 1950 fanden Psychologen heraus, dass der Mensch sich fünf plus minus zwei Dinge merken kann. Sie kennen das vom Telefon und den Telefon Nummern. Diese werden in der Regel in Gruppen angeordnet, zum Beispiel 01 940 60 70 (meine Privatnummer, besonders einfach zu merken). Jede IP Nummer, die einfach aus 12 Zahlen bestehen würde, könnte man sich sicher nur dann merken, wenn man täglich damit arbeitet.
Um die IP Adressen benutzerfreundlicher zu machen, fügten die Erfinder des Internets eine weitere Ebene, eine Art Wörterbuch, das Domain Name System (DNS) hinzu. DNS erlaubt es, jeder IP Adresse einen Namen zu geben:

<xxx.yyyy.zzzz.hhhh, domain_name>
zum Beispiel 127.0.0.1 localhost

Wir können nun den Rechner "localhost" entweder mit Hilfe der IP Adresse 127.0.0.1 oder mit Hilfe der "symbolischen" Adresse "localhost" ansprechen.
Die meisten Hosts haben mindestens einen Namen, mit Ausnahme der Hosts, die keine fixe IP Adresse haben, also eine IP Adresse dynamisch zugeteilt erhalten. Da diese Rechner nicht als Server einsetzbar sind, erübrigt sich die Vergabe eines Namens. Ein Name ist dann fakultativ.
Einige Maschinen haben mehr als eine IP Adresse. Dies trifft in der Regel auf Firewalls, auf Internet Server und generell Maschinen zu, auf denen mehrere Dienste installiert sind (FTP, Mail, Gopher,...). Falls ein Name mehreren IP Adressen gemeinsam zugeteilt wurde, dann liegt es am DNS Server bei konkreten Anfragen zu bestimmen, welcher Rechner zugeteilt wird.
Jeder Rechner, der am IP Netz angeschlossen ist, sollte Zugang zu einem DNS Server haben. Ein Domain Name Server kann die IP Adressen in Namen und die Namen in IP Adressen umwandeln. Die meisten Domain Name Server kennen nur die Auflösung der lokalen Umgebung. Falls weitere Namen oder IP Adressen benötigt werden, muss der Domain Name Server Kontakt mit einem weiteren Domain Name Server aufnehmen, der seinerseits mit weiteren Domain Name Servern verbunden ist ....

4.1.1. Selbsttestaufgaben

- Warum unterscheidet man eigentlich überhaupt zischen der IP Adresse und dem Domain Namen?
- Definieren Sie für Ihr Notebook mehrere IP Adressen.Wie kann man das machen? Wie und unter welcher Adresse ist Ihr Notebook über das IP Netz ansprechbar?

4.2. Übersicht über java.net

Bevor wir uns den konkreten Klassen des Packages Java.net zuwenden ist es sinnvoll, sich eine Gesamtübersicht zu verschaffen:

- welche Klassen gibt es?
- welche Exceptions gibt es?
- wie sieht der Klassenbaum aus?

4.2.1. Package java.net

Liefert die Klassen, die man zur Entwicklung von Netzwerk Applikationen benötigt.

4.2.2. Interface Zusammenfassung

ContentHandlerFactory	Dieses Interface definiert eine Faktory für Content Handler
FileNameMap	Ein einfaches Interface, mit dessen Hilfe ein Zusammenhang zwischen Filenamen und MIME Typen hergestellt werden kann.
SocketImplFactory	Dieses Interface definiert eine Factory (ein Design Pattern) mit dessen Hilfe Sockets implementiert werden können.
SocketOptions	Interface für Methoden zum Setzen und Lesen von Socket Optionen.
URLStreamHandlerFactory	Dieses Interface definiert eine Factory für URL Stream Protokoll Handler.

4.2.3. Zusammenfassung

Authenticator	Diese Klasse stellt Objekte dar, die wissen, wie man die Authentifizierung für ein Netzwerk erhält.
ContentHandler	Die abstrakte Klasse `ContentHandler` ist die Superklasse aller Klassen, die ein `Object` von einer `URLConnection lesen`.
DatagramPacket	Wie der Name sagt : die Klasse stellt ein Datagramm Paket dar.
DatagramSocket	Diese Klasse repräsentiert Socket(s) zum Senden und Empfangen von Datagramm Paketen.
DatagramSocketImpl	Abstrakte Datagram und Multicast Socket Implementation Basis Klasse.
HttpURLConnection	Eine URLConnection mit Support für HTTP- spezifische Eigenschaften.
InetAddress	Diese Klasse repräsentiert eine Internet Protocol (IP) Adresse.
JarURLConnection	Eine URL Verbindung zu einer Java ARchive (JAR) Datei oder einem Eintrag in einer JAR Datei.

MulticastSocket	Die Multicast Datagram Socket Klasse ist nützlich für das Senden und Empfangen von IP Multicast Paketen.
NetPermission	Diese Klasse wird benötigt für verschiedene Netzwerk Zugriffsrechte.
PasswordAuthentication	Die Klasse PasswordAuthentication ist ein Daten Halter und wird vom Authenticator benötigt.
ServerSocket	Diese Klasse implementiert Server Sockets.
Socket	Diese Klasse implementiert Client Sockets (auch einfach als "Sockets" bezeichnet).
SocketImpl	Die abstrakte Klasse `SocketImpl` ist eine gemeinsame Superklasse aller Klassen, die Sockets konkret implementieren.
SocketPermission	Diese Klasse repräsentiert den Zugriff zu einem Netzwerk mit Hilfe von Sockets.
URL	Die Klasse `URL` repräsentiert einen Uniform Resource Locator, einen Pointer zu einer "Resource" auf dem World Wide Web.
URLClassLoader	Dieser Klasse Lader wird benutzt, um Klassen und Resourcen von einem Suchpfad von URLs zu laden. Der Suchpfad kann auf JAR Dateien oder Verzeichnisse verweisen.
URLConnection	Die abstrakte Klasse `URLConnection` ist die Superklasse aller Klassen die eine Kommunicationsverbindung zwischen der Applikation und einer URL herstellen.
URLDecoder	Die Klasse enthält eine Hilfsmethode für die Konversion vom MIME Format "`x-www-form-urlencoded`" in einen `String. Um einen String zu konvertieren, wird jedes Zeichen einzeln untersucht und geprüft:` die ASCII Zeichen '`a`' bis '`z`', '`A`' bis '`Z`', und '`0`' bis '`9`' bleiben unverändert.
URLEncoder	Diese Klasse enthält eine Hilfsmethode, um eine Zeichenkette in ein MIME Format"`x-www-form-urlencoded`" umzuwandeln.
URLStreamHandler	Die abstrakte Klasse `URLStreamHandler` ist die gemeinsame Superklasse für alle Stream Protokoll Handler.

4.2.4. Exception Zusammenfassung

BindException	Signalisiert, dass ein Fehler auftrat, während versucht wurde, einen Socket an einen lokalen Port und seine Adresse zu binden.
ConnectException	Signalisiert, dass ein Fehler auftrat, während versucht wurde einen Socket an eine entfernte (remote) Adresse und Port anzubinden.
MalformedURLException	Signalisiert, dass eine URL nicht korrekt spezifiziert wurde.
NoRouteToHostException	Signalisiert, dass ein Fehler auftrat beim Versuch einen Socket an eine entfernte Adresse und Port zu binden.
ProtocolException	Signalisiert, dass ein Fehler in einem eingesetzten Protokoll auftrat, typischerweise ein TCP Fehler.
SocketException	Signalisiert, dass ein Fehler in einem eingesetzten Protokoll auftrat, typischerweise ein TCP Fehler.
UnknownHostException	Signalisiert, dass die IP Adresse eines Hosts nicht bestimmt

	werden konnte.
UnknownServiceException	Signalisiert, dass ein unbekannter Service Ausnahmezustand auftrat.

4.2.5. Package java.net Kurzbeschreibung

Diese Klasse liefert die Klassen, die man zum Implementieren von Netzwerk Applikationen benötigt. Socket Klassen kann man einsetzen, um mit irgend einem Server auf dem Internet eine Verbindung aufzubauen, oder selber eigene Server zu entwickeln. Einige Klassen kann man einsetzen, um URL's zu verwenden und Daten von diesen Resourcen zu lesen.

4.3. Klassen-Hierarchie des Packages java.net

Paket Hierarchie:

4.3.1. Klassenhierarchie

- class java.lang.**Object**
  - class java.net.**Authenticator**
  - class java.lang.**ClassLoader**
    - class java.security.**SecureClassLoader**
      - class java.net.**URLClassLoader**
  - class java.net.**ContentHandler**
  - class java.net.**DatagramPacket**
  - class java.net.**DatagramSocket**
    - class java.net.**MulticastSocket**
  - class java.net.**DatagramSocketImpl** (implements java.net.SocketOptions)
  - class java.net.**InetAddress** (implements java.io.Serializable)
  - class java.net.**PasswordAuthentication**
  - class java.security.**Permission** (implements java.security.Guard, java.io.Serializable)
    - class java.security.**BasicPermission** (implements java.io.Serializable)
      - class java.net.**NetPermission**
    - class java.net.**SocketPermission** (implements java.io.Serializable)
  - class java.net.**ServerSocket**
  - class java.net.**Socket**
  - class java.net.**SocketImpl** (implements java.net.SocketOptions)
  - class java.lang.**Throwable** (implements java.io.Serializable)
    - class java.lang.**Exception**
      - class java.io.**IOException**
        - class java.net.**MalformedURLException**
        - class java.net.**ProtocolException**
        - class java.net.**SocketException**
          - class java.net.**BindException**
          - class java.net.**ConnectException**
          - class java.net.**NoRouteToHostException**
        - class java.net.**UnknownHostException**
        - class java.net.**UnknownServiceException**

- class java.net.**URL** (implements java.io.Serializable)
- class java.net.**URLConnection**
  - class java.net.**HttpURLConnection**
  - class java.net.**JarURLConnection**
- class java.net.**URLDecoder**
- class java.net.**URLEncoder**
- class java.net.**URLStreamHandler**

4.3.2. Schnittstellenhierarchie

- interface java.net.**ContentHandlerFactory**
- interface java.net.**FileNameMap**
- interface java.net.**SocketImplFactory**
- interface java.net.**SocketOptions**
- interface java.net.**URLStreamHandlerFactory**

Jetzt wenden wir uns diesen Klassen im Einzelnen zu. Die Reihenfolge in der wir die Klassen besprechen, ist nicht ganz willkürlich, aber auch nicht optimal. Sie ist vorgegeben durch das Buch "Java Networking" von Rusty Harold.

4.4. Die InetAddress Klasse

Als erstes schauen wir uns die Klasse als Ganzes an. Hier die vollständige Beschreibung der Klasse:

4.4.1. java.net Class InetAddress

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

public final class **InetAddress**
extends Object
implements Serializable

Diese Klasse stellt eine Internet Protokoll (IP) Adresse dar. Applikationen sollten die Methoden `getLocalHost`, `getByName`, oder `getAllByName` um eine neue `InetAddress` Instanz zu bilden.

Seit:

JDK1.0

Siehe auch:

`getAllByName(java.lang.String)`, `getByName(java.lang.String)`, `getLocalHost()`, Serialized Form

4.4.2. Methoden Kurzbeschreibung

`boolean`	**`equals`**`(Object obj)` Vergleichen des Objektes mit dem spezifizierten Objekt.
`byte[]`	**`getAddress`**`()`

	Liefert die IP Adresse dieses `InetAddress` Objektes.
`static InetAddress[]`	**`getAllByName`**`(String host)` Bestimt alle IP Adressen eines Hosts bei gegebenem Host Namen.
`static InetAddress`	**`getByName`**`(String host)` Bestimmt die IP Adresse eines Hosts bei gegebenem Host Namen.
`String`	**`getHostAddress`**`()` Liefert die IP Adresse als String "%d.%d.%d.%d".
`String`	**`getHostName`**`()` Liefert den Hostnamen dieser Adresse.
`static InetAddress`	**`getLocalHost`**`()` Liefert den local host.
`int`	**`hashCode`**`()` Liefert einen Hashcode für diese IP Adresse.
`boolean`	**`isMulticastAddress`**`()` Hilfsroutine zum Prüfen, ob die InetAddresse eine IP Multicast Adresse ist.
`String`	**`toString`**`()` Konvertiert diese IP Adresse in einen `String`.

Methoden, die von **class java.lang.Object geerbt wurden**
`clone, finalize, getClass, notify, notifyAll, wait, wait, wait`

4.4.3. Methoden Details

isMulticastAddress

`public boolean` **`isMulticastAddress`**`()`

Hilfsroutine, zum Prüfen, ob die InetAddress eine IP Multicast Adresse ist. IP Multicast Adresse ist eine Class D Adresse dh. die ersten vier Bits der Adresse sind 1110.

Seit:

JDK1.1

getHostName

`public String` **`getHostName`**`()`

Liefert den Hostnamen für diese Adresse. Falls der Host null ist, dann zeigt die Adresse auf eine beliebige Netzwerk Adresse einer verfügbaren lokale Maschine. Falls ein Security Manager implementiert wurde, dann wird zuerst dessen `checkConnect` Methode aufgerufen, mit dem Hostnamen und `-1` als Argumente, um zu prüfen, ob die Operation erlaubt ist:

Returns:

die Hostadresse dieser IP Adresse.

Throws:

SecurityException - falls ein Securiy Manager existiert und dessen `checkConnect` Methode den Zugriff nicht erlaubt.

Siehe auch:

`SecurityManager.checkConnect(java.lang.String, int)`

getAddress

`public byte[] **getAddress**()`

Liefert die rohe IP Adresse dieses `InetAddress` Objektes. Das Resultat ist in der Netzwerk Byte Reihenfolge: das Byte höchster Ordnung der Adresse steht in `getAddress()[0]`.

Returns:

die rohe IP Adresse dieses Objektes.

getHostAddress

`public String **getHostAddress**()`

Liefert die IP Adresse als String "%d.%d.%d.%d".

Returns:

die rohe IP Adresse in einem String Format.

Seit:

JDK1.0.2

hashCode

`public int **hashCode**()`

Liefert einen Hashcode für diese IP Adresse.

Returns:

ein Hash Code Wert für diese IP Adresse.

Overrides:

hashCode in der Klasse Object

equals

`public boolean **equals**(Object obj)`

Vergleicht dieses Objekt mit dem spezifizierten Objekt. Das Resultat ist `true` falls und nur falls das Argument nicht `null` ist und es die selbe IP Adresse wie dieses Objekt darstellt.

Zwei Instanzen von `InetAddress` stellen die selbe IP Adresse dar, falls die Länge des Byte Arrays, welcher von `getAddress` geliefert wird, für beide gleich sind, und jede der Array Komponenten des Byte Arrays ebenfalls.

Parameters:

`obj` - das Objekt mit dem verglichen wird.

Returns:

`true` falls die Objekte die selben sind; `false` sonst.

Overrides:

equals in der Klasse Object

Siehe auch:

`getAddress()`

toString

`public String **toString**()`

Konvertiert diese IP Adresse in einen `String`.

Returns:

eine String Darstellung dieser IP Adresse.

Overrides:

toString in der Klasse Object

getByName

```
public static InetAddress getByName(String host)
        throws UnknownHostException
```

Bestimmt die IP Adresse eines Hosts, bei gegebenem Host Namen. Der Host Name kann entweder ein Maschinenamen, wie "`java.sun.com`", oder eine String Darstellung der IP Adresse sein, wie etwa "`206.26.48.100`".

Parameters:

`host` - der spezifizierte Host, oder `null` für den local host.

Returns:

eine IP Adresse für den vorgegebenen Host Namen.

UnknownHostException - falls keine IP Adresse für den `Host` gefunden werden konnte.

getAllByName

```
public static InetAddress[] getAllByName(String host)
          throws UnknownHostException
```

Bestimmt alle IP Adressen eines Hosts, bei gegebenem Host Namen. Der Host Namen kann entweder ein Maschinennamen sein wie "`java.sun.com`", oder eine Zeichenkette, die eine IP Adresse darstellt, wie etwa "`206.26.48.100`".
Falls ein Security Manager installiert wurde und der Host nicht null ist und `host.length()` nicht Null ist, dann wird die Methode `checkConnect`des Security Managers aufgerufen, mit dem Hostnamen und `-1` als Argumente, um festzustellen, ob dieser Zugriff erlaubt ist.

Parameters:

`host` - der Name des Hosts.

ein Array aller IP Adressen für einen gegebenen Host Namen.

UnknownHostException - falls keine IP Adresse für den `Host` gefunden werden konnte.
SecurityException - falls ein Security Manager existiert und seine `checkConnect` Methode diese Operation nicht erlaubt.

Siehe auch:

`SecurityManager.checkConnect(java.lang.String, int)`

getLocalHost

```
public static InetAddress getLocalHost()
     throws UnknownHostException
```

Liefert den local host. Falls ein Security Manager installiert wurde, dann wird mit Hilfe dessen `checkConnect` Methode und dem local host Namen und `-1` als Argumente bestimmt, ob diese Operation erlaubt ist.

Returns:

die IP Adresse des local host.

Throws:

UnknownHostException - falls keine IP Adresse für den `Host` gefunden werden konnte.
SecurityException - falls ein Security Manager existiert und dessen `checkConnect` Methode diese Operation nicht erlaubt.

Siehe auch:

`SecurityManager.checkConnect(java.lang.String, int)`

4.5. Kreieren eines neues InetAddress Objektes

Die Klasse java.net.InetAddress repräsentiert eine Internet Adresse. Die Klasse enthält zwei Felder :

- **hostname** (String)
- **address** (int)

Das Feld hostname enthält den Namen eines Hosts.
Das Feld address enthält die 32 Bit IP Adresse.
Beide Felder sind nicht public. Man kann also nur indirekt, mit Hilfe von Zugriffsmethoden, darauf zugreifen.

Die Klasse verfügt gemäss der obigen Zusammenstellung über die drei Methoden:

- **public static InetAddress InetAddress.getByName(String hostname);**
- **public static InetAddress[] InetAddress.getAllByName(String hostname);**
- **public static InetAddress InetAddress.getLocalHost();**

4.5.1. public static InetAddress InetAddress.getByName(String hostname)

Die Methode, die am meisten eingesetzt wird, ist InetAddress.getByName Diese statische Methode liefert die IP Adresse des Hosts, der als Parameter eingegeben wurde. Java verwendet für die Auflösung DNS.

Der voll qualifizierte Aufruf der Methode wäre:

```
java.net.InetAddress address = java.net.InetAddress.getByName("www.fhz.ch");
```

Falls das Package java.net importiert wurde, dann reicht auch die Kurzform:

```
import java.net.*;
...
InetAddress adresse = InetAddress.getByName("www.fhz.ch");
```

Es empfiehlt sich also, die Klassen zu importieren! In Zukunft setzen wir dies voraus. Sie sehen dies auch an den Beispielprogrammen.

Da die Methode einen Ausnahmezustand auslösen kann, müssen wir einen
try {...} catch (...){...}
Block definieren, und den Methodenaufruf kapseln.

```
try {
        InetAddress adresse = InetAdress.getByName("www.fhz.ch");
        System.out.println(addresse);
} catch (UnknownHostException e) {
        System.out.println("Konnte den Host www.fhz.ch nicht finden!");
}
```

Schauen wir uns ein Beispiel an:

```
package Beispiel41;

import java.net.*;
class Beispiel41 {

  public static void main (String args[]) {

    try {
      InetAddress address = InetAddress.getByName("www.oreilly.com");
      System.out.println(address);
    }
    catch (UnknownHostException e) {
      System.out.println("Der Host www.oreilly.com konnte nicht gefunden werden.");
    }
    try {
      InetAddress address = InetAddress.getByName("localhost");
      System.out.println(address);
    }
    catch (UnknownHostException e) {
      System.out.println("Der Host 'localhost' konnte nicht gefunden werden!");
    }
  }
}
```

Da wir zwei Hosts abfragen und das Notebook nicht am Internet angeschlossen war, benötigen wir zur Verdeutlichung der Ausnahmebehandlung zwei Fehlerbehandlungsblöcke.

Und hier die Ausgabe:

```
Der Host www.oreilly.com konnte nicht gefunden werden.
localhost/127.0.0.1
```

Falls der Rechner, mit dem Sie eine Verbindung aufbauen möchten, keinen Namen hat, dann können Sie die IP Adresse als String an Stelle des Hostnamens eingeben:

```
package Beispiel42;
import java.net.*;
class Beispiel42 {
  public static void main (String args[]) {
```

```
    try {
      InetAddress address = InetAddress.getByName("204.29.207.217");
      System.out.println(address);
    }
    catch (UnknownHostException e) {
      System.out.println("Konnte die Adresse 204.29.207.217 nicht finden!");
    }
    try {
      InetAddress address = InetAddress.getByName("127.0.0.0");
      System.out.println(address);
    }
    catch (UnknownHostException e) {
      System.out.println("Konnte die Adresse 127.0.0.0 nicht finden!");
    }

    try {
      InetAddress address = InetAddress.getByName("127.0.0.1");
      System.out.println(address);
    }
    catch (UnknownHostException e) {
      System.out.println("Konnte die Adresse 127.0.0.1 nicht finden!");
    }
  }

}
```

Und hier die Ausgabe:
204.29.207.217/204.29.207.217
127.0.0.0/127.0.0.0
localhost/127.0.0.1

Und siehe da, localhost wird gefunden!
Dies steht im Gegensatz zur Version in JDK1, welche für das Buch verwendet wurde (Seite 60). Dort wurde der Hostname nicht rückwärts aufgelöst!

Bekanntlich sind Rechnernamen stabiler als die IP Adressen. Ein Beispiel sind die MIT FAQ's welche im Laufe der Zeit von einem Rechner auf den andern verschoben wurden. Die symbolische Adresse (zum Beispiel www.fhz.ch) bleibt erhalten, selbst wenn aus unterschiedlichen Gründen auf einen andern Rechner gewechselt werden musste.

Daher : *IP Adressen sind nur dann sinnvoll, falls der Name nicht verfügbar ist*

Es gibt auch Fälle, wo ein Zugriff auf einen Rechner nur mit Hilfe der IP Adresse sinnvoll ist.

Beispiel:
Sie wollen den Web Auftritt der Firma XYZ vom Rechner A auf den Rechner B bei einem globalen Internet Provider verschieben.
Während der Transfer Periode und der intensiven Test Phase muss ein Zugriff auf beide Rechner möglich sein. Der Zugriff mit Hilfe des Namens "WWW.XYZ.COM" ist nur für einen der Rechner möglich.

4.5.2. public static InetAddress[] InetAddress.getAllByName(String hostname)

Wenden wir uns jetzt jenen Fällen zu, bei denen der Rechner mehrere IP Nummern hat.
Also:

- ein Rechner XYZ besitzt
- mehrere IP Adressen a.b.c.d e.f.g.h, ...

Die passende Methode ist sicher InetAddress.getAllByName welche als Ergebnis ein Array liefert.

Die wesentliche Programmzeile sieht wie folgt aus:

```
InetAddress[] adressen = InetAddress.getAllByName(" www.hta.fhz.ch ");
```

Hier ein vollständiges Beispiel

```
//Titel:        InetAddress.getAllByName
//Version:      1
//Copyright:    Copyright (c) 1999
//Autor:        J.M.Joller
//Beschreibung:  Abfrage eines Hosts mit Hilfe des Namens und Bestimmung aller IP
//                        Adressen dieses Hosts
package Beispiel43;
import java.net.*;
class Beispiel43 {
  public static void main (String args[]) {
    try {
      InetAddress[] addresses = InetAddress.getAllByName("www.apple.com");
      for (int i = 0; i < addresses.length; i++) {
        System.out.println(addresses[i]);
      }
    }
    catch (UnknownHostException e) {
      System.out.println("Konnte www.apple.com nicht finden!");
    }
    try {
      InetAddress[] addresses = InetAddress.getAllByName("localhost");
      for (int i = 0; i < addresses.length; i++) {
        System.out.println(addresses[i]);
      }
    }
    catch (UnknownHostException e) {
      System.out.println("Konnte localhost nicht finden!");
    }
  }
}
```

Und hier die Ausgabe (kein Glück gehabt!):

Konnte www.apple.com nicht finden!
localhost/127.0.0.1

Bei Eingabe von **ztnw293** erhalte ich (hoffentlich nur ich!) als Ergebnis **: ztnw293/147.88.74.117 ztnw293/147.88.74.153 localhost/127.0.0.1** (PCMCIA, Docking Station, localhost)

Es gibt nicht sehr viele Rechner mit mehreren Adressen; ich bin sicher Sie kennen einige!

4.5.3. public static InetAddress InetAddress.getLocalHost()

Diese Methode liefert Informationen über den Rechner, auf dem das Programm gestartet wird.

Wir haben in den obigen Beispielen den localhost, *aber nicht im Sinne des lokalen Hosts* immer als Spezialfall bereits mitgenommen. Von daher bietet das folgende Beispiel keine Überraschungen.

```
//Titel:        InetAddress.getLocalHost()
//Version:
//Copyright:    Copyright (c) 1999
//Autor:        J.M.Joller
//Firma:
//Beschreibung:  Abfrage des localhosts
package Beispiel44;
import java.net.*;
class Beispiel44 {

  public static void main (String args[]) {

    try {
      InetAddress address = InetAddress.getLocalHost();
      System.out.println(address);
    }
    catch (UnknownHostException e) {
      System.out.println("Ich konnte die Adresse dieses Rechners nicht bestimmen.");
    }

  }

}
```

Und hier die Ausgabe:
ztnw293/147.88.74.117

Es geht also nicht um "localhost / 127.0.0.1"!

4.6. Bestimmen der Werte der Datenfelder

Jetzt wollen wir die restlichen Methoden kennen lernen, mit deren Hilfe die Datenfelder (IP Adresse und Hostname) abgefragt werden können.
Wir können diese Datenfelder lediglich abfragen, *nicht* setzen!

4.6.1. public String getHostName()

Das Ergebnis der Methode ist der Name des Rechners dessen InetAddress Objekt wir abfragen.

Wir müssen also zuerst ein InetAddress Objekt kreieren:

```
InetAddress adresse = InetAddress.getLocalHost(); // Beispiel : localhost existiert immer
```

Danach können wir das Datenfeld abfragen:

```
String strRechnername = adresse.getHostName();
```

Und hier ein vollständiges Beispiel:

```
//Titel:        public String getHostName();
//Version:
//Copyright:    Copyright (c) 1999
//Autor:        J.M.Joller
//Firma:
//Beschreibung:  eine Abfrage des Host Namens
package Beispiel45;
import java.net.*;
class Beispiel45 {

  public static void main (String args[]) {

    try {
      InetAddress address = InetAddress.getLocalHost();
      System.out.println("Hallo. Mein Name ist " +  address.getHostName());
    }
    catch (UnknownHostException e) {
      System.out.println("Keine Ahnung wie ich heisse.");
    }
  }
}
```

Und hier die Ausgabe:
Hallo. Mein Name ist ztnw293

4.6.2. Selbsttestaufgabe

Wie sieht die Ausgabe aus, wenn ein InetAddress Objekt kreiert wird für einen Rechner, den es nicht gibt?
Antwort : der Text steht in der Behandlung des Ausnahmezustandes.

4.6.3. public byte[] getAddress()

Das Ergebnis der Methode ist die Adresse des Rechners, dessen InetAddress Objekt wir abfragen.

Wir müssen also zuerst ein InetAddress Objekt kreieren:

```
InetAddress adresse = InetAddress.getLocalHost(); // Beispiel : localhost existiert immer
```

Danach können wir das Datenfeld abfragen:

```
byte[] bRechneradresse = adresse.getAddress();
```

Bevor wir ein vollständiges Beispiel anschauen, müssen wir noch einige Spezialitäten von Java kennen lernen. Bytes mit einem Wert grösser als 125 werden negativ dargestellt. man muss deswegen im Falle eines negativen Wertes 255 addieren, um die IP Adresse zu erhalten.

Dies geschieht mit einer Abfrage:

```
if (myByte < 0) { myByte = myByte + 256; }
// oder anders ausgedrückt , wie ein C / C++ Programmierer
int unsignedByte = myByte < 0 ? myByte + 256 : myByte;
```

Und hier ein vollständiges Beispiel:

```
//Titel:        public byte[] getAddress()
//Version:
//Copyright:    Copyright (c) 1999
//Autor:       J.M.Joller
//Firma:
//Beschreibung:  Beispielabfrage der IP Adresse eines InetAddress Objektes
package Beispiel46;
import java.net.*;
class Beispiel46 {
  public static void main (String args[]) {
    try {
      InetAddress thisComputer = InetAddress.getLocalHost();
      byte[] address = thisComputer.getAddress();
      System.out.print("Meine Adresse ist :");
      for (int i = 0; i < address.length; i++) {
        // int unsignedByte = address[i] < 0 ? address[i] + 256 : address[i];
        int unsignedByte = address[i];
        if (unsignedByte < 0) unsignedByte = address[i] + 256;
        System.out.print(unsignedByte + ".");
      }
      System.out.println();
    }
    catch (UnknownHostException e) {
      System.out.println("Pech gehabt. Konnte die Adresse nicht feststellen!");
    }
  }
}
```

Und hier die Ausgabe:
Meine Adresse ist :
147.88.74.117.
Neben den InetAddress spezifischen Methoden stehen auch noch alle vererbten Methoden zur Verfügung, speziell jene von Object.

4.7. Objekt Methoden

Da alle Java Klassen die Object Klasse als Wurzel haben, erben alle Klassen die Eigenschaften und Methoden dieser Klasse.
Jede Klasse hat ihrerseits die Möglichkeit, diese Methoden zu überschreiben.

4.7.1. public boolean equals(Object einObjekt)

Ein InetAddress Objekt ist dann und nur dann gleich einem anderen Objekt, falls dieses Objekt ebenfalls eine Instanz der InetKlasse ist und die selbe IP Adresse hat. Der Rechnername braucht *nicht* gleich zu sein. Die IP Adresse ist also das Gleicheitskriterium.

Schauen wir uns ein Beispiel Programm an:

```
//Titel:       equals(Object InetAddressObjekt)
//Version:
//Copyright:    Copyright (c) 1999
//Autor:       J.M.Joller
//Firma:
//Beschreibung:  Vergleich zweier Objekte, die beide Instanzen der Klasse InetAdress sind.
package Beispiel47;
import java.net.*;
class Beispiel47 {
  public static void main (String args[]) {
    try {
      InetAddress notebook = InetAddress.getByName("ztnw293");
      InetAddress local = InetAddress.getLocalHost();
      if (notebook.equals(local)) {
        System.out.println
          ("ztnw293 und localhost sind ein und der selbe Rechner");
      }
      else {
        System.out.println
          ("ztnw293 und localhost sind zwei verschieden Rechner");
      }
    }
    catch (UnknownHostException e) {
      System.out.println("Programm konnte den Rechner nicht finden.");
    }
  }
}
```

mit der Ausgabe : **ztnw293 und localhost sind ein und der selbe Rechner**

4.7.2. public int hashCode()

Falls die InetAddress Objekte als Index einer Hash- Tabelle verwendet werden sollen, dann muss es eine Funktion geben, die pro InetAddress Objekt einen eindeutigen Index generiert.

Schauen wir uns an einem Beispiel an, wie dies aussieht:

```
//Titel:        public int hashCode()
//Version:
//Copyright:    Copyright (c) 1999
//Autor:        J.M.Joller
//Firma:
//Beschreibung:  Bestimmen eines eindeutigen Indices für eine Hashtabelle pro InetAddress
package BeispielHashTable;

import java.net.*;

class BeispielHashTabelle {
  public static void main(String args[]) {
  try {
    InetAddress notebook = InetAddress.getByName("ztnw293");
    InetAddress local    = InetAddress.getLocalHost();
    InetAddress localh   = InetAddress.getByName("localhost");
    System.out.println("Hashcode von ZTNW293 : "+notebook.hashCode());
    System.out.println("Hashcode von 'local host' : "+local.hashCode());
    System.out.println("Hashcode von localhost : "+localh.hashCode());
  } catch (UnknownHostException uh) {
    System.out.println("Konnte den Host nicht nachsehen");
  }
  }
}
```

Die Ausgabe zeigt ein Problem:

Hashcode von ZTNW293 : -1822930315
Hashcode von 'local host' : -1822930315
Hashcode von localhost : 2130706433

bei gleichem InetAddress ist der resultierende Hashcode identisch. Stellt sich die Frage, was man konkret mit dem Hashcode erreichen kann.

4.7.3. public String toString()

Die Umwandlung eines InetAddress Objektes in eine Zeichenkette haben wir bereits in verschiedenen Beispielen gesehen.

Hier der Vollständigkeit halber noch ein weiteres Beispiel:

```
//Titel:       public String toString()
//Version:
//Copyright:    Copyright (c) 1999
//Autor:       J.M.Joller
//Firma:
//Beschreibung:  Beispiel für die Umwandlung eines InetAddress Objektes in eine
Zeichenkette
package InetAddress_toString;
import java.net.*;
class InetAddress_toString {
  public static void main(String args[]) {
  try {
    InetAddress notebook = InetAddress.getByName("ztnw293");
    System.out.println("InetAddress notebook als Zeichenkette : "+notebook.toString());
  } catch (UnknownHostException ue) {
    System.out.println("Der Hostname 'ztnw293' konnte nicht aufgelöst werden");
  }
  }
}
```

mit der Ausgabe:

InetAddress notebook als Zeichenkette : ztnw293/147.88.74.117

4.8. Konversion von IP Adressen

Im Gegensatz zur Version Java 1.0, die dem Buch von Rusty Harold zugrunde lag, kann die neuste Version von Java die Auflösungen problemlos und korrekt durchführen.
Wir haben bereits verschiedene Beispiele gesehen.

Beispiel 4-9 im Buch zeigt, wie mit Hilfe des Factory Design Pattern eine alternative Lösung realisiert werden kann.

Die Klasse InetAddressFactory ist recht interessant, wie Sie im Beispiel 4-7 erkennen können:

```
...
byte[] xcluster = ( 18, 81, 0, 21);       // IP Adresse
...
        InetAddress ia = InetAddressFactory.newInetAddress(xcluster);
..
```

und umgekehrt:

...
String meinNotebook = "192.12.56.90";
...
InetAddress = InetAddressFactory.newInetAddress(meinNotebook);
...
Hier der Quellcode der InetAddressFactory Klasse

```
public class InetAddressFactory {
  // Use a byte array like {(byte) 199, (byte) 1, (byte) 32, (byte) 90}
  // to build an InetAddressObject
  public static InetAddress newInetAddress(byte addr[])
    throws UnknownHostException  {
     try {
       InetAddress ia = new InetAddress();
       ia.address  = addr[3] & 0xFF;
       ia.address |= ((addr[2] << 8) & 0xFF00);
       ia.address |= ((addr[1] << 16) & 0xFF0000);
       ia.address |= ((addr[0] << 24) & 0xFF000000);
       return ia;
     }
     catch (Exception e) {  // primarily ArrayIndexOutOfBoundsExceptions
       throw new UnknownHostException(e.toString());
     }
  } // end newInetAddress
  // Use a String like 199.1.32.90 to build
  // an InetAddressObject
  public static InetAddress newInetAddress(String s)
   throws UnknownHostException {
     // be ready for IPv6
     int num_bytes_in_an_IP_address = 4;
     byte addr[] = new byte[num_bytes_in_an_IP_address];
     StringTokenizer st = new StringTokenizer(s, ".");// make sure the format is correct
     if (st.countTokens() != addr.length) {
       throw new UnknownHostException(s
        + " is not a valid numeric IP address");
     }
     for (int i = 0; i < addr.length; i++) {
       int thisByte = Integer.parseInt(st.nextToken());
       if (thisByte < 0 || thisByte > 255) {
             throw new UnknownHostException(s
        + " is not a valid numeric IP address");
       }       // check this
       if (thisByte > 127) thisByte -= 256;
       addr[i] = (byte) thisByte;
     }  // end for
    return newInetAddress(addr);
  } // end newInetAddress
  } // end InetAddressFactory
```

4.9. Einige nützliche Programme

Die Java Netzwerkklassen, die wir bisher kennen gelernt haben, erlauben bereits erste nützliche Programme.
Als erstes schauen wir uns an, wie ein Java Programm aussieht, welches wie NSLOOKUP funktioniert, allerdings nur ungefähr!

4.9.1. java lookup

Das Programm *nslookup* unter Unix ist eines der auch unter Windows NT oft verwendete Programm. Das Programm liefert zu einem Rechnernamen dessen IP Adresse, oder liefert bei gegebenem Rechnernamen die dazugehörige(n) IP Adresse(n).

Das Java Programm funktioniert so, dass bei Eingabe von "exit" die Programmschleife abgebrochen wird.

Schauen wir uns das Programm erst mal an, modifiziert, d a die InetAddressFactory nicht benötigt wird:

```
//Titel:       lookup
//Version:
//Copyright:    Copyright (c) 1999
//Autor:       J.M.Joller
//Firma:
//Beschreibung:  eine Java Version (Kommando Zeile) des UNIX nslookup Utilities
package lookup;
import java.net.*;
import java.io.*;
class lookup {

  public static void main (String args[]) {

  if (args.length > 0) { // Eingabe auf der Kommandozeile
    for (int i = 0; i < args.length; i++) {
      lookup(args[i]);
    }
  }
  else {
    DataInputStream myInputStream = new DataInputStream( System.in);
    System.out.println
      ("Bitte den Rechnernamen oder die IP Adresse eingeben. Mit \"exit\" wird das Programm
beendet.");
    while (true) {
      String s;
      try {
        s = myInputStream.readLine();
      }
      catch (IOException e) {
```

```
        break;
       }
       if (s.equals("exit")) break;
       lookup(s);
     }

   }

  } /* end main */


  private static void lookup(String s) {

    InetAddress thisComputer;
    byte[] address;

    // bestimmen der IP Adresse
    try {
      thisComputer = InetAddress.getByName(s);
      address = thisComputer.getAddress();
    }
    catch (UnknownHostException ue) {
       System.out.println("Rechner wurde nicht gefunden " + s);
       return;
    }

    if (isHostname(s)) {
      // Ausgabe der IP Adresse
      for (int i = 0; i < address.length; i++) {
        int unsignedByte = address[i] < 0 ? address[i] + 256 : address[i];
        System.out.print(unsignedByte + ".");
      }
      System.out.println();
    }
    else {  // dies ist eine IP Adresse
      try {
        InetAddress ina = InetAddress.getByName(s);
        System.out.println(ina);
      }
      catch (UnknownHostException e) {
        System.out.println("Adresse wurde nicht gefunden" + s);
      }
    }

  }  // end lookup

   private static boolean isHostname(String s) {

     char[] ca = s.toCharArray();
     // falls ein Zeichen weder eine Zahl noch ein Punkt ist
```

```
    // dann ist s wahrscheinlich ein Rechnernamen (Zeichenkette)
    for (int i = 0; i < ca.length; i++) {
      if (!Character.isDigit(ca[i])) {
        if (ca[i] != '.') {
          return true;
        }
      }
    }

    // es wurde eine Zahl oder ein Punkt gefunden
    // es sieht so aus, als ob s eine IP Adresse in der üblichen Schreibweise sei
    return false;

  }  // end isHostName

} // end javalookup
```

Die Ausgabe hängt von Ihrer Eingabe ab.
Beispiel Dialog:
Bitte den Rechnernamen oder die IP Adresse eingeben. Mit "exit" wird das Programm beendet.
127.0.0.1
localhost/127.0.0.1
ztnw293
147.88.74.117.
exit

Der Aufbau des Programmes ist recht transparent:
das Programm besteht aus main, lookup und isHostname.
Die leere Eingabe wir als "localhost" interpretiert.

Schauen wir uns das Programm genauer an.

Zu diesem Zwecke lösen Sie als Vorbereitung die folgende Selbsttestaufgabe.

4.9.2. Selbsttestaufgabe : lookup

- warum wird bei einer leeren Eingabe der localhost als Rechner genommen?
- modifizieren Sie das Programm so, dass bei Eingabe des Rechnernamens nicht nur die IP Adresse, sondern auch der Rechnername ausgegeben wird.

4.10. Aufgabe : Analyse eines Logfiles eines Webservers

Im Gegensatz zum Beispiel mit Hilfe der InetAddressFactory soll ein Java Programm geschrieben werden, welches ein Logfile (eines Netscape Enterprise Servers, oder ein frei von Ihnen gewähltes Logfile [Zugriffslog]) analysiert werden.

Die Aufgabe besteht aus mehreren Teilen:

a) das bestehende Programm soll ausschliesslich mit Hilfe der Java.net Klassen realisiert werden

b) zusätzlich soll ein Programm geschrieben werden, welches in einem ersten Schritt das Logfile, welches oft recht gross ist, nach Zugriffsverzeichnissen "zerlegt" werden:
aus dem Logfile geht hervor, worauf zugegriffen wird, auf welche Dateien und Verzeichnisse.
Wir gehen davon aus, dass auf einem Webserver verschiedene Personen ihre Websites gespeichert haben.
[root]
|
[User1] [User2] [User3] ...

[User1]
|
[User1 Subverzeichnis]
|
...
In solchen Fällen ist die "Kunden gerechte" Auswertung aussagekräftiger, als eine pauschale Auswertung.
Zudem sollen alle IP Adressen (Client Adressen) durch den Hostnamen ersetzt werden.

c) Sie müssen dieses Programm in einer Woche abgegeben haben. Abgabe erfolgt auf dem Klassenlaufwerk, unter Ihrem Namen, in einem "PVS" Verzeichnis.
Die Abgabe muss enthalten:
1) Java Source
2) Eingabe Logfile
3) Ausgabe Logfiles
Bedingung:
das Logfile Root enthält mindestens drei Unterverzeichnisse, nach denen aufgeteilt werden muss.
diese Verzeichnisse werden als Kommandozeilenparameter übergeben, also micht automatisch (in einem ersten Durchlauf) bestimmt.

d) Freiwillig: wie müssten Sie das Programm modifizieren, so dass es selbständig die Level1 Unterverzeichnisse findet.
Warum ist eine automatische Suche dieser Verzeichnisse wenig sinnvoll?

Hinweise

das Logfile Format hängt vom Web Server ab.
Hier das Format eines Microsoft Internet Servers:

W3C Extended format

Beispiel:
#Microsoft Internet Information Server 4.0
#Version: 1.0
#Date: 1998-02-01 11:38:02
#Fields: date time c-ip cs-username s-ip cs-method cs-uri-stem cs-uri-query sc-status sc-bytes cs-bytes
time-taken cs-version cs(User-Agent) cs(Cookie) cs(Referer)

Details:
'date' = Date
'time' = Time
'c-ip' = Client IP
'cs-username' = Authenticated Username
's-ip' = Server IP
'cs-method' = Method
'cs-uri-stem' = Target Resource
'cs-uri-query' = Query String
'sc-status' = Status/Return Code
'sc-bytes' = Server-to-client Bytes
'cs-bytes' = Client-to-server Bytes
'time-taken' = Transfer Time
'cs-version' = Protocol version
'cs(User-Agent)' = User Agent
'cs(Cookie)' = Cookie
'cs(Referer)' = Referer

4.11. Zusammenfassung

Wir haben an einer ersten Klasse des Paketes java.net gesehen, wie einfach und doch mächtig java.net ist.

Bereits mit dieser Klasse lassen sich IP Adressen auflösen, in beiden Richtungen.
Die Methoden erlauben es auch, alle vorhandenen IP Adressen zu bestimmen.

Einige wichtige Punkte:

- auf die Datenfelder (IP Adresse und Host Name) kann nicht direkt zugegriffen werden
- unter getLocalHost() wird nicht localhost (oder loopback) sondern der lokale Host angezeigt
- beim Verwalten von InetAddress Objekten mit Hilfe einer HshTabelle darf nicht vergessen werden, dass zwei InetAddress Objekte, die zur gleichen IP Adresse gehören, den selben Hashcode liefern.

4.12. Anhang : Logfile Formate

Die Zusammenstellung steht auch auf dem Web unter
http://www.openwebscope.com/help/W3C_Logging.html

Logging Control In W3C httpd

W3C httpd can log all the incoming requests to an access log file. It also has an error log where internal server errors are logged. All log files are generated using the common log file format that several WWW servers use. This provides the possibility of using some of the generic statistics programs to analyze the log file contents.

- `AccessLog` - Set access log file name
- `ProxyAccessLog` - Log proxy accesses to a different log file
- `CacheAccessLog` - Log cache accesses to a different log file
- `ErrorLog` - Set error log file name
- `LogFileDateExt` - Common Time/Date extension to all log file names
- `LogFormat` - Set access log file format
- Common logfile format supported by all the main HTTP servers
- `LogTime` - Set time zone for log files
- `NoLog` - No log entries for listed hosts/domains

Access Log File

Access log file contains a log of all the requests. The name of the log file is specified either by `-l` *logfile* command line option, or with `AccessLog` directive; log file can be either an absolute path:

```
AccessLog /absolute/path/logfile
```

or relative to `ServerRoot`:

```
AccessLog logs/logfile
```

Proxy Access Log File

If you are running W3C httpd as a proxy server and you want to have a separate log of proxy transactions and normal HTTP server transactions, specify the proxy log file via `ProxyAccessLog` directive:

```
ProxyAccessLog  logfile
```

logfile can be either an absolute pathname, or relative to `ServerRoot`.
If `ProxyAccessLog` is not set all accesses will be logged to the normal `AccessLog` instead.

CacheAccessLog

Cache accesses can be logged to a different log file instead of the normal access log. The `CacheAccessLog` directive takes an absolute pathname of the cache access log file:

```
CacheAccessLog  logfile
```

logfile can be either an absolute pathname, or relative to `ServerRoot`.

Error Log File

Error log contains a log of errors that might prove useful when figuring out if something doesn't work. Error log file name is set by `ErrorLog` directive:

```
ErrorLog /absolute/path/errorlog
```

If error log file is not specified, it defaults to access log file name with `.error` extension. If the filename extension already exists, `.error` will replace it.

LogFileDateExt

The `LogfileDateExt` directive specifies a common extension to all the log files based on a time/date format. The value follows the LogTime directive. Any format can be specified using time/date directives as specified for *strftime()* function, e.g.

```
LogFileDateExt  %H:%M           =>      19:35
LogFileDateExt  %d-%m-%Y        =>      02-18-95
```

Spaces in the format are converted to '_'.

```
LogFileDateExt  log
```

Log File Format

Previously every server used to have its own logfile format which made it difficult to write general statistics collectors. Therefore there is now a *common logfile format* (which will eventually become the default). Currently it is enabled by

```
LogFormat  Common
```

The old W3C httpd format can be used by

```
LogFormat  Old
```

The Common Logfile Format

The common logfile format is as follows:

```
remotehost rfc931 authuser [date] "request" status bytes
```

remotehost
: Remote hostname (or IP number if DNS hostname is not available, or if DNSLookup is Off.

rfc931
: The remote logname of the user.

authuser
: The username as which the user has authenticated himself.

[*date*]
: Date and time of the request.

"*request*"
: The request line exactly as it came from the client.

status

The HTTP status code returned to the client.

bytes

The content-length of the document transferred.

Log Time Format

Times in the log file are by default local time. That can be changed to be GMT time by `LogTime` directive:

```
LogTime  GMT
```

Default is:

```
LogTime  LocalTime
```

Suppressing Log Entries For Certain Hosts/Domains

It's not always necessary to collect log information of accesses made by local hosts. The `NoLog` directive can be used to prevent log entry being made for hosts matching a given IP number or host name template:

```
NoLog  template
```

Examples

```
NoLog 128.141.*.*
NoLog *.cern.ch
NoLog *.ch  *.fr  *.it
```

---

httpd@w3.org, July 1995