Programmierzertifikat Objekt-Orientierte Programmierung mit Java

Vorlesung 10: Netzwerk

Peter Thiemann

Universität Freiburg, Germany

SS 2008

Inhalt

Netzwerkprogrammierung in Java

- In Package `java.net`

Internet-Adressen (IP-Adressen)

- Internet-Adresse = vier Oktette (je 8 Bit)
  Beispiele

132.230. 1. 5	WWW-Server der Uni Freiburg
132.230.150.17	WWW-Server der Informatik
127. 0. 0. 1	localhost (eigener Rechner, für Experimente)

- jedes direkt mit dem Internet verbundene Endgerät besitzt eindeutige Internet-Adresse
- maximal $2^{32} = 4.294.967.296$ Endgeräte
  (überhöht, da Adressraum strukturiert und teilweise reserviert)

Zukünftige IP-Adressen: IPv6 [RFC 2060]

- Befürchtung: IPv4 Adressraum bald erschöpft
- daher: 128bit IP-Adressen [RFC 2373]
- viele Konzepte eingebaut bzw. vorgesehen
  - selbständige Adresskonfiguration (mobiler Zugang)
  - *quality of service* Garantieen möglich
  - Authentisierung, Datenintegrität, Vertraulichkeit
- Schreibweise: 4er Gruppen von Hexziffern

```
1080:0:0:0:8:800:200C:417A  a unicast address
1080::8:800:200C:417A       ... compressed
```

Die Klasse `java.net.InetAddress`

- Objekte repräsentieren IP-Adressen
- Subklassen für IPv4 und IPv6
- kein öffentlicher Konstruktor, stattdessen

```
// sämtliche IP-Adressen von host
public static InetAddress[] getAllByName(String host)
                    throws UnknownHostException

// IP-Adresse des lokalen Rechners
public static InetAddress getLocalHost()
                    throws UnknownHostException

// liefert die IP-Adresse als Text
public String getHostAddress()
```

Die Klasse java.net.InetAddress

Beispiel

```
import java.net.*;

public class DomainName2IPNumbers {
  public static void main(String[] args) {
    try {
      InetAddress[] a = InetAddress.getAllByName(args[0]);
      for (InetAddress ia : a)
        System.out.println(ia.getHostAddress());
    } catch (UnknownHostException e) {
      System.out.println("Unknown host!");
    }
  }
}
```

```
...> java DomainName2IPNumbers www.google.com
216.239.59.147 216.239.59.104 216.239.59.99
```

Die Klasse java.net.InetAddress

2. Beispiel

```
import java.net.*;

public class MyAddress {
  public static void main(String[] args) {
    try {
      InetAddress a = InetAddress.getLocalHost();
      System.out.println("domain name: "+a.getHostName());
      System.out.println("IP address: "+a.getHostAddress());
    } catch (UnknownHostException e) {
      System.out.println("Help! I don't know who I am!");
    }
  }
}
```

```
...> java MyAddress
domain name:  abacus.informatik.uni-freiburg.de
IP address:   132.230.166.150
```

Sockets

Ein Socket (Steckdose) ist eine Datenstruktur zur Administration von (Netzwerk-) Verbindungen. An jedem Ende einer Verbindung ist ein Socket erforderlich. Es wird unterschieden zwischen:

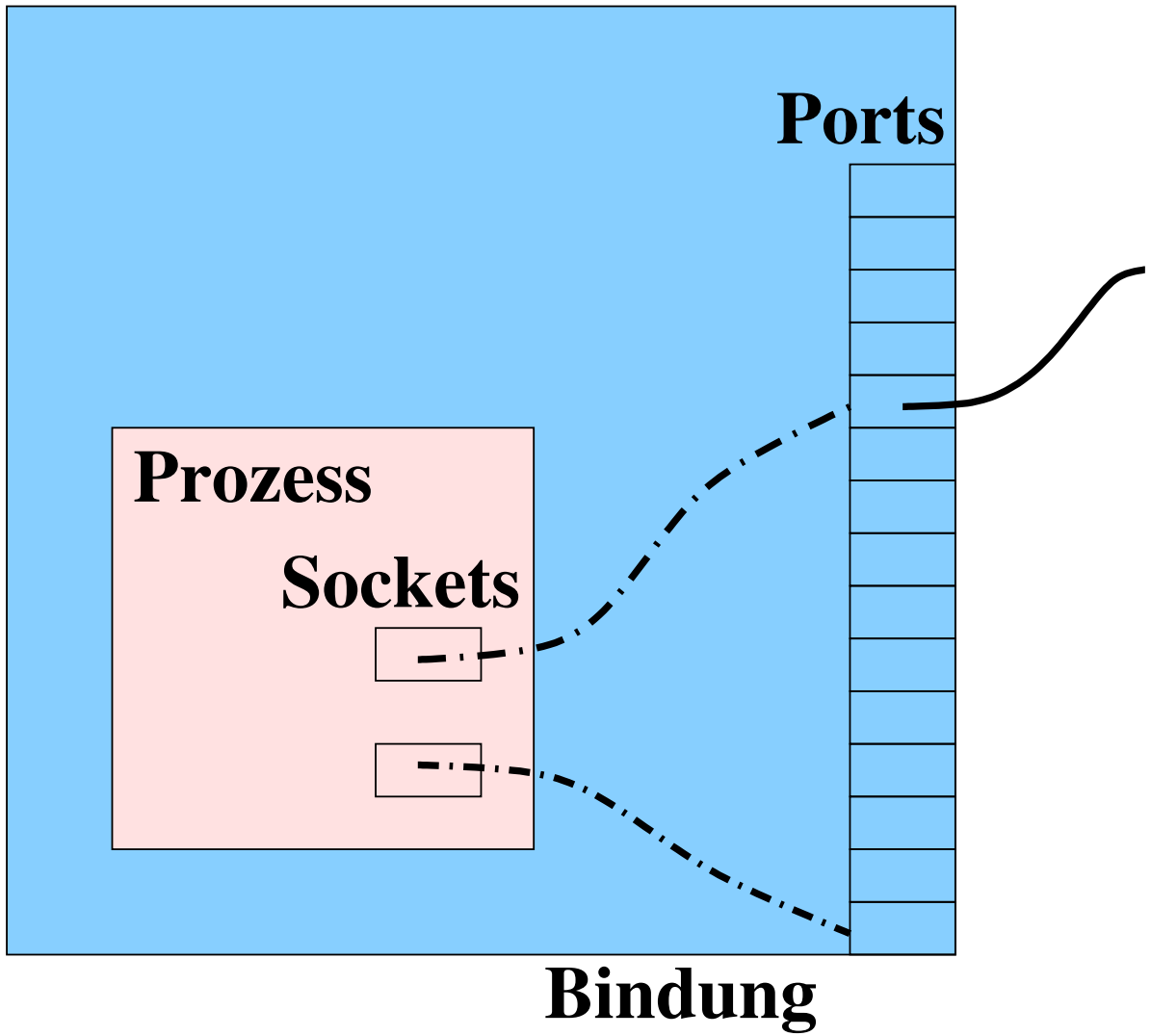
Aktivität

- Client Socket:
  Verbindung mit existierendem Dienst
- Server Socket:
  Stellt Dienst zur Verfügung

Verbindungsart

- UDP (Datagram, unidirektional)
- TCP (Stream, bidirektional)

Sockets und Ports



Anfrage einer Webseite aus Sicht des Clients

Ablauf

1. Server öffnet den Serverport 80, und wartet auf Anfragen, ...
2. Client verbindet sich mit dem Server, indem er URL und Port des Servers angibt, z.B.:
   `http://www.google.de:80`
   Das Socket auf Clientseite besitzt auch einen Port. Dieser wird aber vom Betriebssystem vergeben.
   $\Rightarrow$ wir müssen uns darum nicht selber kümmern.
3. Der Client schickt dem Server Informationen, welche Webseite er gerne sehen möchte
4. Der Server schickt die angefragte Seite, falls diese vorhanden ist, oder antwortet mit einer Fehlermeldung.
5. Das Socket des Clients, d.h. die Verbindung zwischen den beiden Rechnern, wird beendet

Bemerkungen zum Ablauf und zu Netzwerken

- Der Server kann gleichzeitig mit vielen Clients reden
- Es kann jederzeit ein Netzwerkproblem auftreten
- Meist haben wir die Verbindung und den anderen Computer nicht unter unserer Kontrolle.

→ Wir müssen den Daten misstrauisch gegenüber stehen, d.h. seltsame Daten erkennen und passend reagieren . . .

Klasse `java.net.Socket` für Clients

Socket Konstruktoren für den zweiten Schritt

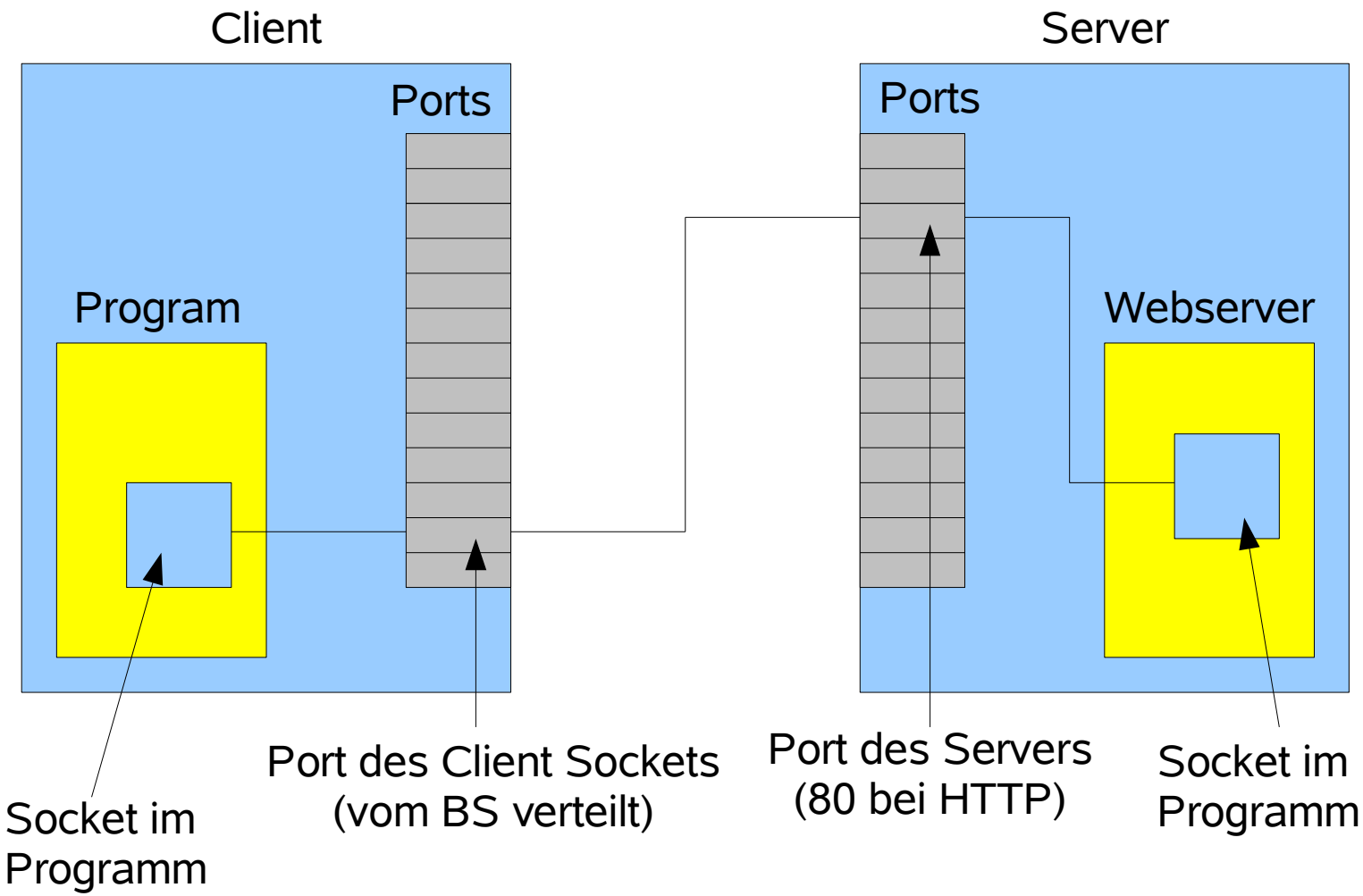
```
// Verbindung zum Server auf "address" und "port"
Socket (InetAddress address, int port)

// Verbindung zum Server "host" und "port"
Socket (String host, int port) {
  Socket (InetAddress.getByName (host), port);
}
```

→ auch ein Client Socket ist auf dem lokalen Rechner an einen (meist beliebigen) Port gebunden.

- Der Clientport wird nicht mit angegeben, da er vom Betriebssystem bestimmt wird.
- Das bedeutet port ist Teil des Ziels.

Sockets mit Client und Server



Socket Methoden

Informationen senden und empfangen

- Ausgabe auf diesem Strom wird zum Server gesendet
  OutputStream getOutputStream() **throws** IOException
- Eingaben von diesem Stream stammen vom Server
  InputStream getInputStream() **throws** IOException
- Schließen der Verbindung **void** close() **throws** IOException

Socket Methoden

Beispiel eines Webclient (Browser)

```
public static void main (String[] args) throws Exception {
    String hostname = "www.google.de";
    String path = "index.html";
    Socket s = new Socket (hostname, 80);
    PrintWriter out = new PrintWriter (s.getOutputStream ());
    // send request
    out.print ("GET "+path+" HTTP/1.1\r\n");
    out.print ("Host: "+hostname+"\r\n\r\n"); out.flush();
    // read & echo response
    System.out.println ("-----------------------------");
    in = new BufferedReader (new InputStreamReader (s.getInputStream ()));
    String line = in.readLine ();
    while (line != null) {
      System.out.println (line);
      line = in.readLine ();
    }
    // may hang for a while
    System.out.println ("-----------------------------");
}
```

Sever Sockets

Konstruktoren

ServerSocket (int port) **throws** IOException

Erzeugt einen Socket für Verbindungen über port. Dient nur zum Verbindungsaufbau.

Wichtige Methoden

- Socket accept() **throws** IOException
  Wartet am `port` des `ServerSocket` auf eine (externe) Verbindung. Liefert einen gewöhnlichen `Socket` für die Abwicklung der Kommunikation.
- **void** close() **throws** IOException
  Schließt den `ServerSocket`

Beispielserver

- Das Interface `DialogHandler` trennt die Handhabung der Verbindung von der Abwicklung der Kommunikation.

```
import java.io.*;

public interface DialogHandler {
  /* @param br receive information from client
   * @param pw sends information to client
   * @return false to exit the server loop
   */
  boolean talk (BufferedReader br, PrintWriter pw);
}
```

- Dieses Interface kann man so implenentieren, dass der DialogHandler in einem anderen Thread läuft, und somit den Server nicht blockiert.

$\Rightarrow$ wichtig für Tetris, da es nicht blockieren soll

Beispiel – BackTalk

Socket

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

public class TCPServer {
    private ServerSocket s_s;
    public TCPServer (int port) throws IOException {
        s_s = new ServerSocket (port);
    }
    public void run (DialogHandler dh) throws IOException {
        boolean acceptingConnections = true;
        while (acceptingConnections) {
            Socket s = s_s.accept ();
            BufferedReader br = new BufferedReader
                (new InputStreamReader (s.getInputStream ()));
            PrintWriter pw = new PrintWriter (s.getOutputStream (), true);
            acceptingConnections = dh.talk (br, pw);
            s.close ();
        }
    }
}
```

Beispiel – BackTalk

DialogHandler

```
public class BackTalkDialog implements DialogHandler {
    public boolean talk (BufferedReader br, PrintWriter pw) {
        String line = null;
        BufferedReader terminal = new BufferedReader
                (new InputStreamReader (System.in));
        while (true) { try {
                if (br.ready ()) {
                    line = br.readLine ();
                    System.out.println (line);
                } else if (terminal.ready ()) {
                    line = terminal.readLine ();
                    if (line.equals ("STOP!")) break;
                    pw.println (line);
                }
            } catch (IOException ioe) { return false; }
        }
        return false; // stop the server
    }
}
```

Beispiel – BackTalk

Main Methode

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

public class BackTalk {
    public static void main (String[] arg) throws Exception {
        if (arg.length != 1) {
            System.out.println ("Usage: BackTalk port");
        } else {
            try {
                int port = new Integer (arg[0]).intValue ();
                TCPServer server = new TCPServer (port);
                server.run (new BackTalkDialog ());
            } catch (RuntimeException e) {
                System.out.println ("Argument not an integer");
            }
        }
    }
}
```

Verbindungen über URLs

- URL (Uniform Resource Locator) RFC 1738, RFC 1808, RFC 2368
- Symbolische Adresse für ein Dokument
- Format: ⟨*Schema*⟩:⟨*schemaspezifische Information*⟩
- Beispiele:

mailto: Internet-Mailadresse
Beispiel: `mailto:admin@test.de`

http: //⟨*User*⟩:⟨*Password*⟩@⟨*Host*⟩:⟨*Port*⟩/⟨*URL-Path*⟩
Dabei sind optional:

- ⟨*User*⟩:⟨*Password*⟩@
- :⟨*Port*⟩

Beispiel: `http://www.google.com`

ftp: //⟨*User*⟩:⟨*Password*⟩@⟨*Host*⟩:⟨*Port*⟩/⟨*Path*⟩
Optional: ⟨*User*⟩:⟨*Password*⟩@, :⟨*Port*⟩
Beispiel: `ftp://ftp.informatik.uni-freiburg.de/iif`

URLs in Java – Die Klasse URL

Wichtiger Konstruktor

URL(String spec) **throws** MalformedURLException

analysiert den String spec und –falls erfolgreich– erstellt ein URL Objekt.

Wichte Methode

URLConnection openConnection() **throws** IOException

liefert ein Objekt, über das

1. die Parameter der Verbindung gesetzt werden
2. die Verbindung hergestellt wird
3. die Verbindung abgewickelt wird

URLs in Java – Die Klasse `URLConnection`

Die Klasse ist abstrakt, daher keine Konstruktoren

- Herstellen der Verbindung:
  void connect()
- zum Lesen der Verbindung:
  InputStream getInputStream()
- zum Parsen von der Verbindung in ein passendes Objekt, dass selber bestimmt werden kann (setContentHandlerFactory):
  Object getContent ()
- Methoden zum Setzen von Anfrageparametern:
  setUseCaches, setIfModifiedSince, setRequestProperty

Beispiel – Inhalt eines Dokuments als byte[]

```
public class RawURLContent {
    private URLConnection uc;

    public RawURLContent (URL u) throws IOException {
        uc = u.openConnection ();
    }

    public byte[] getContent () throws IOException {
        int len = uc.getContentLength ();
        if (len <= 0) {
            System.err.println ("Length cannot be determined");
            return new byte[0];
        } else {
            byte[] rawContent = new byte [len];
            uc.getInputStream ().read (rawContent);
            return rawContent;
        }
    }
}
```

Beispiel – I'm Feeling Lucky

```
import java.net.*; import java.io.*;
public class ImFeelingLucky {
    public static void main(String[] args) {
        String req = "http://www.google.com/search?"+
          "q="+URLEncoder.encode(args[0], "UTF8")+"&"+
          "btnI="+URLEncoder.encode("I\u2019m Feeling Lucky", "UTF8");

        URLConnection urlc = new URL(req).openConnection();
        HttpURLConnection con = (HttpURLConnection) urlc;
        con.setRequestProperty("User-Agent", "IXWT");
        con.setInstanceFollowRedirects(false);

        String loc = con.getHeaderField("Location");
        if (loc != null)
            System.out.println("Direct your browser to "+loc);
        else
            System.out.println("I am sorry - my crystal ball is blank.");
    }
}
```

Hypertext Transfer Protocol (HTTP)

- Definition in RFC 2616 (HTTP 1.1)
- Request/Response Protokoll, d.h.
  - Client öffnet Verbindung
  - Der Client fragt nach Daten
  - Der Server antwortet
    - mit den angefragten Daten
    - oder einer Fehlermeldung
  - Client schließt Verbindung (ab HTTP 1.1 kann Client vor dem Schließen nochmals nach Daten fragen)
- Protokoll ist ohne Zustand

Beispiel

Anfrage an google.de

Eingegebene URL in Browser:

http://www.google.de:80/

Dieser Request sagt dem Browser:

http:	HTTP Protokoll verwenden
//	trennt Schema von Rest
www.google.de	Hostname
:80	default Port von HTTP
/	der anzufragende Pfad ist /

Beispiel

Anfrage an google.de

Wir öffnen ein TCP/IP Socket nach `www` (unter Java mit Socket, oder auf der Konsole mit `telnet`) und senden den String über die Verbindung:

```
1 GET / HTTP/1.1
2 Host: www.informatik.uni-freiburg.de
3
```

GET	Wir wollen ein Dokument abrufen
/	Pfad zum Dokument auf dem Server
HTTP/1.1	wir wollen Protokoll Version 1.1 verwenden
Host: ...	unter HTTP 1.1 muss der Host angegeben werden

HTTP Nachrichten haben folgende Struktur

- Header (Zeile 1 und Zeile 2)
- einer leeren Zeile als Trennung von Header und Body (Zeile 3)
- Body (hier leer, deshalb keine Zeile)

Format einer Anfrage

```
1 GET / HTTP/1.1
2 Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, */*
3 Accept-Language: en-us
4 Accept-Encoding: gzip, deflate
5 User-Agent: Mozilla/4.0 (compatible; MSIE 5.01; Windows NT)
6 Host: www.hypo.com
7 Connection: Keep-Alive
8
```

Header

- erste Zeile ist die Request-Line (`GET ...`)
  - `HEADER`
  - `POST`
  - `PUT`
  - ...
- die folgenden Zeilen bestehen aus: ⟨*key-token*⟩:⟨*value*⟩
- Zeilen werden durch ⟨*CRLF*⟩ getrennt (CR ASCII-Kode 13, `\r`, LF ASCII-Kode 10, `\n`)

Beispiel

Antwort bei Anfrage an www

```
1 GET / HTTP/1.1
2 Host: www.google.de
3
```

$\longrightarrow$

```
1 HTTP/1.1 200 OK
2 Date: Thu, 19 Jun 2008 09:05:51 GMT
3 Server: Zope/(Zope 2.9.6-final, python 2.4.4, linux2) ZServer/1.1 Plone/2.5.2
4 Content-Length: 20310
5 Content-Language: de
6 Expires: Sat, 1 Jan 2000 00:00:00 GMT
7 Content-Type: text/html;charset=utf-8
8 Set-Cookie: I18N_LANGUAGE="de"; Path=/
9 Via: 1.1 www.informatik.uni-freiburg.de
10 X-Cache: MISS from www.informatik.uni-freiburg.de
11
12 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" ...
13 ...
```

Beispiel

Antwort

```
1 HTTP/1.1 200 OK
2 Content-Type: text/html;charset=utf-8
3 Content-Length: 20310
4 Content-Language: de
5 ** Rest des Headers **
6
7 <html> **Inhalt der Webseite** </html>
8
```

HTTP/1.1	Protokoll Version
200 OK	Ein Server Antwort Code
Content-Type	Information über den Type des Inhalts
Content-Length	Größe des Bodys der Nachricht
Content-Language	Sprache des Dokuments

Sever Antwort Code

Bereiche

Der erste Teil des Statuscodes ist eine Zahl. Diese Zahlen sind in die folgenden Bereiche eingeteilt:

100–199: Informative Meldung
200–299: Client-Request erfolgreich
300–399: Client-Request weitergeleitet, weitere Aktionen notwendig
400–499: Client-Request unvollständig
500–599: Server-Fehler

Server Antwort Code

Einige wichtige Codes

200 OK	Der Client-Request war erfolgreich, die Antwort erhält die angefragten Daten
300 Multiple Choices	Vom dem angefragten Dokument gibt es mehre, z.B. in unterschiedlichen Sprachen.
301 Moved Permanently	Das Dokument wurde von der angefragten Stelle entfernt. Die neue Position des Dokuments ist im `Location`-Header angegeben.
401 Unauthorized	HTTP kennt `Authenticate`-Headers, die z.B. per Usernamen und Passwort den Zugriff auf Dokumente steuern können
403 Forbidden	Dieser Request wurde abgelehnt, der Server will nicht mitteilen, wieso
404 Not Found	Das angefragte Dokument gibt es nicht
500 Internal Server Error	interner Server Fehler, z.B. bei einem `CGI` Programm

Mails

Format

- RFC 822 revised as RFC 2822
- original sehr restriktiv:
  - US-ASCII Zeichen 1-127
  - Nachricht besteht aus Zeilen
    "Each line of characters MUST be no more than 998 characters, and SHOULD be no more than 78 characters, excluding the CRLF."
- Teile:
  - Header ($\langle Feldname \rangle$:$\langle Wert \rangle$)
  - Leerzeile
  - Rumpf (einzige Einschränkung, die Zeilenlänge)

Mails

Beispiel

```
Message-Id: <5.0.0.25.0.20030521140008.00a44ca0@mailgw.ub.uni-freiburg.de>
Sender: maurer@mailgw.ub.uni-freiburg.de
Date: Wed, 21 May 2003 14:05:30 +0200
To: mitarbeiter@informatik.uni-freiburg.de
From: Beate Maurer <maurer@ub.uni-freiburg.de>
Subject: Neuerwerbungen
Mime-Version: 1.0
Content-Type: text/plain; charset="iso-8859-1"; format=flowed
Content-Transfer-Encoding: 8bit

Liebe Mitarbeiterinnen und Mitarbeiter,

in der Sitzung der Bibliothekskommission wurde vermutet, dass die
Neuerwerbungsliste zu wenig bekannt sein könnte.

Deswegen für alle die diesen Dienst noch nicht kennen, der Hinweis auf
die folgende URL.

http://www.ub.uni-freiburg.de/neuerwerb.html

Mit freundlichen Grüßen
  Beate Maurer
```

Headerfelder

- Die Reihenfolde der Headerfelder spielt keine Rolle
- Wichtige Headerfelder:

Feld	Req.	Beschreibung
orig-date	!	Datum der Mail
from	!	Sender
sender	*	Sender
to		Empfänger
cc		erhält Kopie
bcc		erhält Kopie, Mailadresse für andere unsichtbar
message-id	*	identifiziert die Mail
subject		Betreff Zeile

Req. Erklärung

!	Ist erforderlich
*	ist empfohlen

Mails

- Wir können die Felder mit falschen Daten füllen
- Keine Garantie für die Korrektheit der Headfelder

$\longrightarrow$ Deshalb ist Spam ist so einfach zu verschicken

Demo

```
$ HELO ???
> 250 atlas.informatik.uni-freiburg.de Hello ...
$ MAIL FROM:<???@informatik.uni-freiburg.de>
> 250 OK
$ RCPT TO:<????@informatik.uni-freiburg.de>
> 250 Accepted
$ DATA
> 354 Enter message, ending with "." on a line by itself
From:<???@informatik.uni-freiburg.de>
To:<????@informatik.uni-freiburg.de>
Subject: test

data
.
> 250 OK id=1KAI6G-0006Gx-PH
```

Zusammenfassung

- Internet Adressen identifizieren immer ein Gerät, in IP4 `XXX.XXX.XXX.XXX` mit $0 \leq$ `XXX` ≤ 255.
- Sockets dienen dem Verschicken und Empfangen von Daten über Netzwerke:
  - Jedes Socket besitzt einen Port
  - Die Java Klasse `Socket` wird zum erstellen von Sockets verwendet
  - Server öffnen ein Serversocket auf einem Port und warten auf Clients
  - Clients verbinden sich mit einem Serversocket durch die Angabe von Internet Adresse und Port des Servers
- Um sich nicht immer die Namen merken zu müssen, werden URLs verwendet
- Viele Protokolle im Internet sind standardisiert, und für diese gibt es in Java spezielle Klassen, z.B. `HTTPURLConnection`.
- HTTP ist ein Zustandsloses Protokoll, dass durch einfache Textnachrichten initiiert durch den Client vom Server Daten abfragt.