CAMPUS 02 GRAZ
FACHHOCHSCHULE DER WIRTSCHAFT

NETWORK IN JAVA

Programmieren 3

Bernhard Fuchs

LERNZIELE

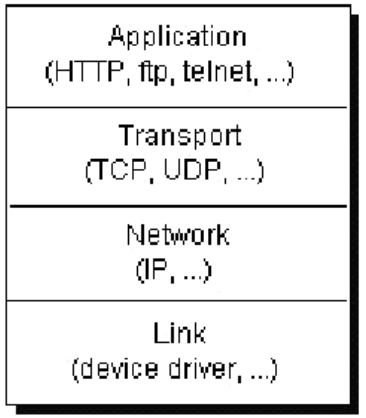
- Die Verwendung der Klasse URL kennen
- Netzwerkkommunikation mittels Sockets und Java IO verwenden können
- Serverapplikationen mit Hilfe von Server Sockets implementieren können
- Verteilte Applikationen mit parallelisierter Verarbeitung erstellen können

LERNHILFEN

- https://www.javatpoint.com/java-networking

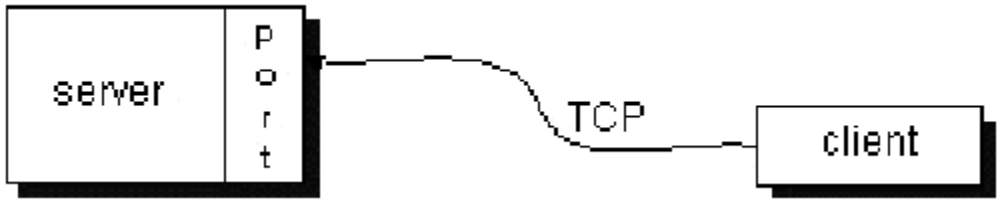
NETZWERK ALLGEMEIN

- Java Programme befinden sich im Application Layer
- Abhängig vom verwendeten Protokoll (TCP oder UDP) werden unterschiedliche Klassen verwendet



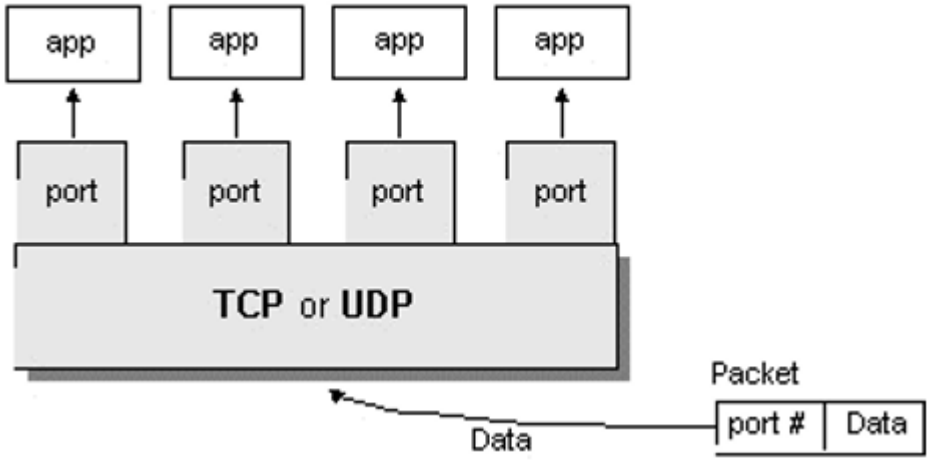
PORTS

- Auf einem Rechner, bzw. einer IP-Adresse, werden unterschiedliche Ports verwendet, um die Zuordnung von Datenpaketen zu Applikationen zu steuern
- Eine Serverapplikation hört auf einem Port auf eingehende Verbindungen eines Clients



PORTS

- 65535 Ports (16 bit) sind verfügbar
- Die ersten 1024 sind reserviert (HTTP, FTP, …)



PORTS

- HTTP – 80
- HTTPS – 443
- SSH – 22
- FTP - 21

PACKAGE JAVA.NET

- Die wichtigsten Klassen in diesem Package:
  - `URL`
  - `URLConnection`
  - `Socket`
  - `ServerSocket`
  - `DatagramPacket`
  - `DatagramSocket`
  - `MulticastSocket`

URL

URL

- **U**niform **R**esource **L**ocator
- Repräsentieren Ressourcen im Netzwerk (z.B. Webseiten oder Datenbankabfragen)
- Zwei wesentliche Komponenten:
  - Protocol identifier: http
  - Resource name – www.wetter.at
    - Host Name: www.wetter.at
    - Filename /wetter/…/graz/index.html
    - Port Number (optional): 80
    - Reference (optional) # Prognose

URL - KONSTRUKTOREN

- **URL myURL =**
  - **new URL (“http://www.wetter.at/.../index.html“)**
  - **new URL („http“, „www.wetter.com“, „/wetter/oesterreich/steiermark/graz/index.html“)**

URL – LESEN VON URL

- **InputStream openStream() throws IOException**
- Liefert einen InputStream zum Auslesen des Dokuments. Dasselbe geht mit: **myURL.openConnection().getInputStream();**
- Die weitere Verarbeitung funktioniert analog wie das Auslesen einer Datei:
- **BufferedReader br = new BufferedReader(new InputStreamReader(myURL.openStream()));**

URL – LIVE-DEMO

- Live-Demo Beispiel wird im nachhinein auf Moodle zu finden sein
  - Demo:
    - Network → beispiel1/ReadFromUrl.java
  - Weitere Beispiele in Package: network

URL – UE 0.9

- Erstellen Sie ein Programm, das den Inhalt einer URL in eine Datei “content.html” schreibt.
- Verwenden Sie die Klasse „URL“

URL – UE 1

- Erstellen Sie ein Programm, das eine Internetadresse **aus einer Textdatei** liest, und den Inhalt dieser URL in eine Datei “content.html” schreibt.

URLConnection

URLCONNECTION

- **URLConnection myConn =**
  - **myURL.openConnection();**
  - **myConn.setDoOutput (true); //um schreiben zu können**
  - **myConn.getOutputStream ();**
  - **myConn.getInputStream ();**

UE-2

- Analysieren Sie das Beispiel im package *network/beispiel2*
- Kommentieren Sie die Zeile
  - “conn.setDoOutput(true);”
- aus und führen Sie das Programm aus. Was passiert?

Sockets

SOCKETS

- Sind die Endpunkte einer TCP Verbindung
- Sockets werden an Ports gebunden
- Ein Server hört an einem bekannten Port auf eingehende Verbindungen
- Der Client öffnet ein Socket und baut eine Verbindung zum Server auf

SOCKETS

- Der Server akzeptiert die Verbindung
- Am Server wird ein neues Socket für die bestätigte Verbindung erzeugt
- Am bekannten Port wird weiterhin auf eingehende Verbindungen gewartet

SOCKETS

- **Socket mySocket =**
  - **new Socket(); //unconnected**
  - **new Socket (“www.wetter.at“, 80);**
  - **byte[] remAdr = {173,194,35,152}; new Socket (InetAdress.getByAdress(remAdr), 80);**
  - **byte[] locAdr = {10,124,79,0}; new Socket (InetAdress.getByAdress(remAdr), 80; InetAdress.getByAdress(locAdr), 8000);**

SOCKETS – LESEN

- **InputStream getInputStream() throws IOException**
- Liefert einen InputStream zum Auslesen des Dokuments. Wird der InputStream geschlossen, so wird auch der Socket geschlossen.
- Die weitere Verarbeitung funktioniert analog wie das Auslesen einer Datei:
  - **BufferedReader br = new BufferedReader(new InputStreamReader(mySocket.getInputStream()))**

SOCKETS – SCHREIBEN

- **OutputStream getOutputStream() throws IOException**
  Liefert einen InputStream zum Auslesen des Dokuments. Wird der OutputStream geschlossen, so wird auch der Socket geschlossen.

- Die weitere Verarbeitung funktioniert analog wie das Schreiben in eine Datei:
  - **BufferedWriter br = new BufferedWriter(new OutputStreamWriter(mySocket.getOutputStream()));**

SOCKETS – UE 3

- Analysieren Sie das Beispiel im package network/beispiel3
- Schreiben Sie ein Programm, das sich mit der Internetadresse time-a.timefreq.bldrdoc.gov auf Port 13 oder 37 verbindet und dann die aktuelle Zeit liest

ServerSocket

SERVERSOCKET

- **ServerSocket mySocket =**
  - **new ServerSocket(); //unbound**
  - **new ServerSocket (9090)**

SERVERSOCKET

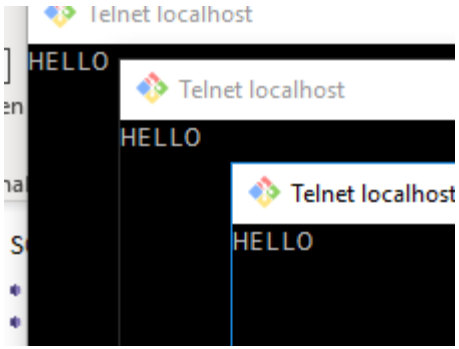
- Akzeptieren eingehender Verbindungen
- **Socket accept() throws IOException**
- Wartet auf eingehende Verbindungsanfragen und stellt die Verbindung her. Der serverseitige Endpunkt der Verbindung ist eine neue Socket Instanz (diese wird zurückgegeben).

SERVERSOCKET – UE-4

- Analysieren Sie das Beispiel im package network/beispiel4.
- Adaptieren Sie das Server-Programm so, dass mehr als eine Client-Verbindung aufgenommen werden kann. Testen Sie Ihre Implementierung mit Hilfe von telnet
  - https://windowsreport.com/telnet-windows-10/
  - Via CommandLine (cmd): `>telnet localhost 9090`
- Lösung: network/beispiel4/loesung

SERVERSOCKET – LÖSUNG 4

- Erweitern Sie Ihren Server so, dass mehrere Clientverbindungen parallel behandelt werden.
  - Test mittels mehreren Telnet Instanzen (telnet localhost 9090)



- Lösung: network/beispiel4/loesung