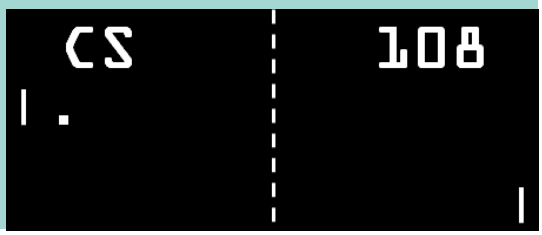
University of Basel

Kapitel 5 – Netzwerkprogrammierung in Java

cs108 Programmierprojekt

H. Schuldt, FS 2020



Netzwerkprogrammierung

Charakterisierung von Netzwerken: Übertragungstechnik ...

- Punkt-zu-Punkt-Netze
  - Bestehen aus vielen Verbindungen von Paaren von Rechnern
  - Um von der Quelle zum Ziel zu gelangen, muss ein Paket eventuell mehrere „Zwischenrechner“ durchlaufen
  - In der Regel sind von der Quelle zum Ziel mehrere Routen unterschiedlicher Länge möglich

... Charakterisierung von Netzwerken: Übertragungstechnik ...

- Broadcast-Netze
  - Es gibt einen Übertragungskanal, der von allen am Netz angeschlossenen Maschinen gemeinsam benutzt wird.
  - Alle Rechner, die am Netz angeschlossen sind, erhalten eine gesendete Nachricht (Paket)
  - Sämtliche Pakete besitzen ein Adressfeld, in dem der eigentliche Empfänger angegeben ist
  - Jeder Empfänger eines Pakets testet, ob das Paket für ihn bestimmt ist
    - Falls ja wird es verarbeitet
    - Falls nein wird es ignoriert (bzw. neu ins Netz gegeben)
  - Beispiel: Durchsage im Kaufhaus / am Flughafen

… Charakterisierung von Netzwerken: Übertragungstechnik

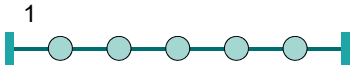
- Multicasting
  - Variante des Broadcasting
  - Nachricht (Paket) wird an eine Teilmenge der angeschlossenen Rechner geschickt
  - Multicasting benötigt die Möglichkeit, eine Gruppe von Rechnern gemeinsam zu adressieren, z.B. über eine Gruppennummer bzw. Gruppenkennung
  - Beispiel: Verschicken von Emails an Mailing-Listen

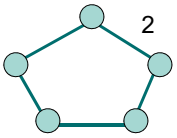
Netz-Topologien

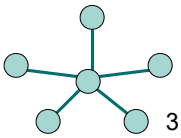
- Die topologische Struktur eines Daten- oder Rechnernetzes gibt an, wie die Teilnehmer, Vermittlungseinrichtungen und Leitungen wechselseitig zugeordnet sind.

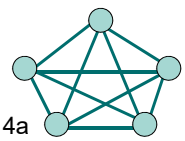
Topologien

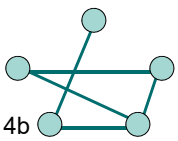
1. Busnetze
2. Ringnetze
3. Sternnetze
4. Vermaschte Netze: beliebige Verbindungen möglich
   a) Voll-Vermaschung: jeder Teilnehmer ist mit jedem anderen verbunden
   b) Teil-Vermaschung











Klassifikation: räumliche Ausdehnung …

- Local Area Networks (LAN)
  - Hochleistungsdatentransfer (mehrere 100 Mbit/s bis hin zu mehreren Gbit/s) auf räumlich begrenztem Gebiet
  - in der Regel private Netze (corporate networks)
  - vorwiegend Bus-, Stern- oder Ringtopologie
  - zumeist mit Broadcast-Übertragungstechniken
- Wide Area Networks (WAN)
  - Datenfernübertragung (Kabel, Satellit, Richtfunkstrecken)
  - vorwiegend vermaschte Topologien
  - Zumeist Punkt-zu-Punkt-Übertragungstechniken
  - Zusammenschluss mehrerer eigenständiger Netze (Netzwerkverbund); benötigt spezielle Vermittlungselemente (Router)

… Klassifikation: räumliche Ausdehnung

- Metropolitan Area Networks (MAN)
  - decken Kommunikationsbedarf in Ballungszentren ab
  - Übertragungsgeschwindigkeiten: einige hundert Mbit/s
  - Verwenden WAN-ähnliche Technologien, jedoch mit höheren Geschwindigkeiten

Art der Verbindung …

- Verbindungslose Kommunikation / Kommunikationsdienste
  - Zu übertragende Daten werden in kleine Einheiten (Rahmen) aufgeteilt
  - Jeder dieser Rahmen wird einzeln und unabhängig von den anderen Rahmen der Nachricht verschickt
  - Es existiert keine ständige logische Verbindung
  - Reihenfolgetreue ist nicht garantiert
  - Beispiel: Absenden eines Briefes

… Art der Verbindung

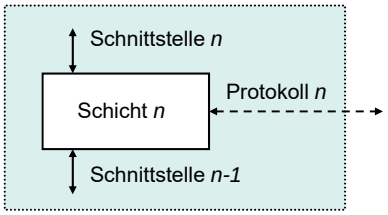
- Verbindungsorientierte Kommunikation / Kommunikationsdienste
  - Vor dem Verschicken von Rahmen bauen Quelle und Senke eine Kommunikationsverbindung auf
  - Dies beinhaltet die Initialisierung von Variablen, die für die Sicherung der Kommunikation (des Austauschs von Rahmen) erforderlich sind
    1. Aufbau der Verbindung
    2. Kommunikation: Übertragung von Rahmen
    3. Trennen der Verbindung
  - Reihenfolge der Übertragung von Rahmen wird eingehalten
  - Beispiel: Telefonanruf

Art der Bestätigung

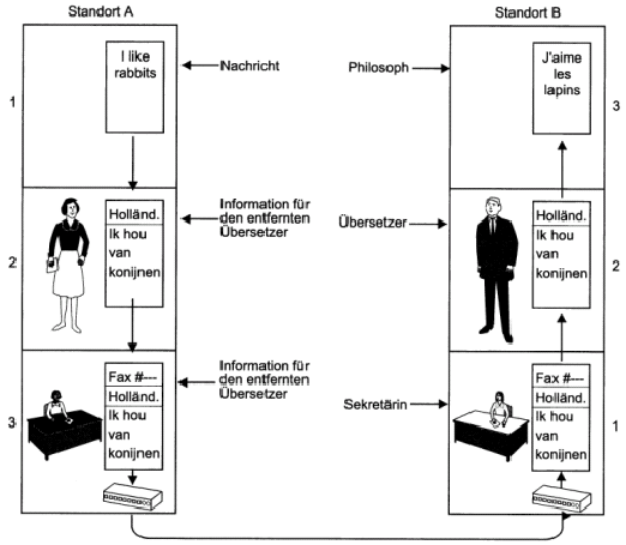
- Unbestätigte Kommunikation / Kommunikationsdienste
  - Der Empfänger gibt keine Rückmeldung, ob ein Rahmen empfangen wurde oder nicht
  - Der Empfang von Daten kann also nicht garantiert werden, der Verlust eines Rahmens fällt nicht auf
  - Beispiel: Versenden eines Briefes
- Bestätigte Kommunikation / Kommunikationsdienste
  - Der Empfang jedes Rahmens wird einzeln bestätigt
  - Bei Ausbleiben der Bestätigung (Überschreiten eines time-out) kann der Sender den Rahmen nochmals verschicken
  - Beispiel: Versenden eines Einschreibe-Briefes mit Rückschein

Protokolle und Protokollhierarchien

- Rechnerkommunikation basiert auf Kommunikationsprotokollen, die das Format, den Zeitpunkt, die Art, etc. der Datenübertragung festlegen
- Zur Reduktion der Komplexität werden unterschiedliche Abstraktionsstufen betrachtet.
- Jede Abstraktionsebene bildet eine eigene Schicht mit eigenem Protokoll ...
  - ... und stützt sich dabei auf die nächsttiefere Schicht ab (verwendet die Dienste der nächsttieferen Schicht über deren Schnittstelle)
  - Konzeptionell kommuniziert Schicht i des Quellsystems mit Schicht i des Empfängers. Jedoch erfolgt diese Kommunikation sukzessive über die tieferen Schichten auf beiden Seiten.
  - Sowohl beim Sender als auch beim Empfänger existieren also mehrere, aufeinander aufbauende Protokollschichten: eine Protokollhierarchie



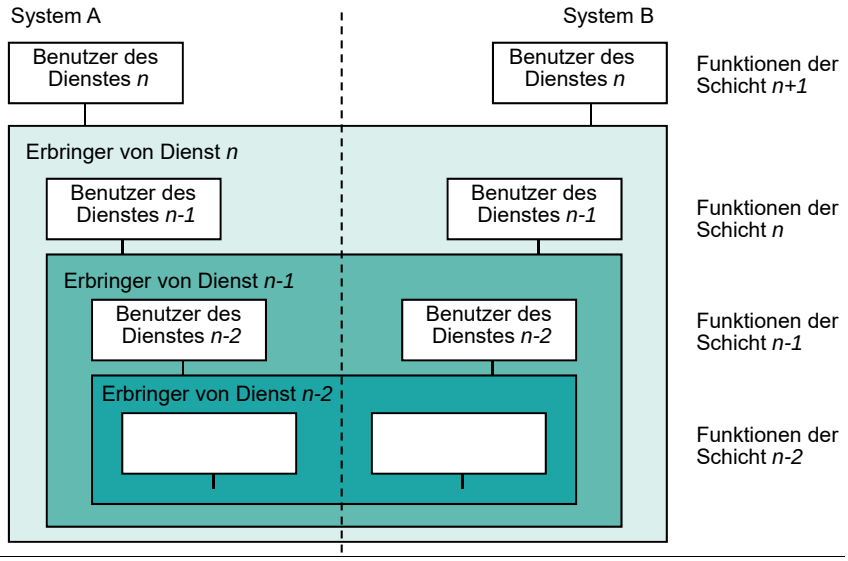
Protokollhierarchien: Beispiel



Beispiel: kommunizierende Philosophen

Quelle: [Tanenbaum 00]

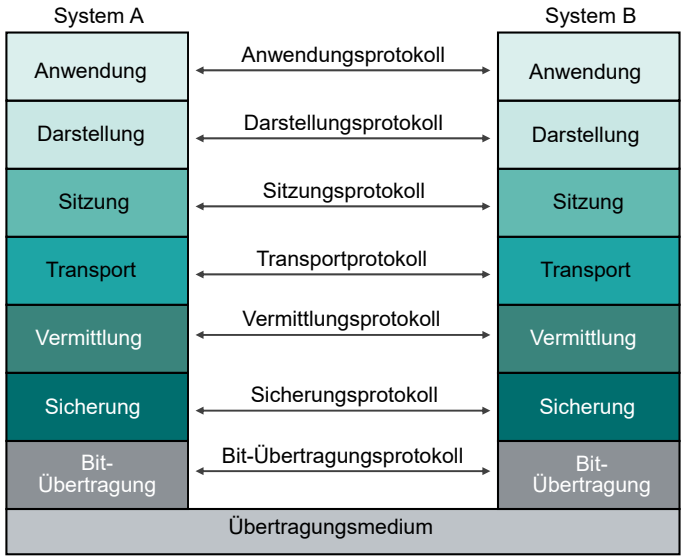
Protokollhierarchien: Dienststruktur



OSI-Referenzmodell

- Das OSI-Referenzmodell ist ein abstraktes, logisch-funktionelles Architekturmodell der ISO (International Standards Organization) für die Datenkommunikation in offenen Systemen
  - OSI: Open Systems Interconnection
  - Standard: OSI 7498, ab 1977 entworfen
- Besteht aus sieben Schichten
  - Eine Schicht erbringt für die jeweils darüber liegende Schicht bestimmte Dienste

Schichten im OSI-Modell ...



... Schichten im OSI-Modell ...

- Verarbeitung (application layer)
  - Anwendungsspezifische Protokolle, z.B. für Dateitransfer
- Darstellung (presentation layer)
  - Codierung von Daten, Abbildung zwischen Datencodes, evtl. Standardcodierung für Übertragung
- Kommunikationssteuerung (Sitzung) (session layer)
  - Dialogsteuerung (wer darf jeweils senden)
- Transport (transport layer)
  - Zerlegung von Daten in kleine Einheiten (Pakete)
  - Evtl. Aufbau mehrerer Verbindungen (Kanäle), Festlegen der Art der Verbindung (Punkt-zu-Punkt, Broadcast), Angabe des Service Access Point (SAP) des Empfänger-Hosts

... Schichten im OSI-Modell

- Vermittlung (network layer)
  - Auswahl von Paketrouten (evtl. Abrechnung)
- Sicherung (data link layer)
  - Übertragung frei von Übertragungsfehlern machen
  - Aufteilung von Daten in Datenrahmen (Frames). Kennzeichnen des Beginns (Endes) von Frames
- Bitübertragung (physical layer)
  - Physikalische, mechanische, elektrische Aspekte der Übertragung

TCP/IP-Referenzmodell

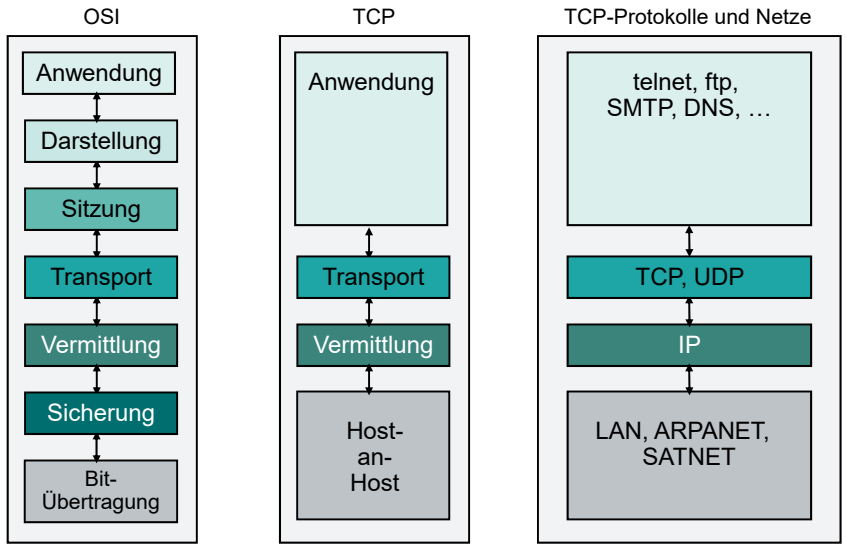
Besteht aus 4 Schichten

- Oberste Schicht: Anwendungen, z.B.email, ftp, telnet
- Transport: Ende-zu-Ende-Protokolle
  - TCP (transmission control protocol): verbindungsorientiert (und zuverlässig)
  - UDP (user datagram protocol): verbindungslos (und unzuverlässig)
- Verbindung/Vermittlung: Internet
  - IP-Protokoll für Routing von Paketen
- Host-an-Host (nicht näher spezifiziert, Verwendung bestehender Netze)
  - Paketvermittelndes Netz. Referenzmodell macht keine Aussagen zu verwendeter Netzwerktechnologie (→ IP auf der Basis verschiedener Netze möglich, z.B. WAN, Ethernet-LAN, etc.)

TCP/IP

- Die in der Vorlesung notwendigen Netzwerkfähigkeiten in Java basieren auf dem Internet-Protokoll TCP/IP
  - TCP ist ein verbindungsorientiertes Protokoll, das auf der Basis von IP (eindeutige Nummer im Internet) eine sichere und fehlerfreie Punkt-zu-Punkt-Verbindung realisiert
  - Es gibt in Java auch eine TCP/UDP Verbindung (User Datagram Protocol), ein verbindungsloses Protokoll

OSI und TCP/IP



Entstehung des Internet

- Ursprung: ARPANET in den USA
  (Advanced Research Projects Agency NETwork)
  - Mitte der 60er Jahre
  - Zunächst für militärische Nutzung
  - Verbindung von Mini-Rechnern (Interface Messaging Processors, IMPs). Ausfallsicherheit: jeder IMP an mindestens zwei weitere IMPs angeschlossen
  - Nachrichtengrösse 8063 Bit (!), aufgeteilt in Pakete von maximal 1008 Bit (!)
  - Zunächst: vier Netzwerkknoten
  - Später kamen immer weitere lokale Netze hinzu
  - Die ursprüngliche ARPANET-Protokolle konnten diese jedoch nicht integrieren
  - Entwicklung der TCP/IP-Protokolle

Internet Protocol Version 6 (IPv6)

- Protokoll seit 1998 standardisiert
  - Hat aber IPv4 noch immer nicht komplett abgelöst (Beispiel: Uni Basel!)
  - Verbreitung in der Schweiz: 35.78 % (Quelle: https://www.google.de/ipv6/statistics.html#tab=per-country-ipv6-adoption)
- Adressen der Länge 128 (2^{128} mögliche Adressen)
  - Darstellung: hexadezimal, in acht Blöcken zu jeweils 16 Bit (= 4 Hexadezimalstellen)
  - Beispiel: FABC:A5C4:382B:23C1:AA49:45092:4EFE:9987
  - In einer URL: http://[FABC:A5C4:382B:23C1:AA49:45092:4EFE:9987]

IPv4: Adressierung - IP-Adressen ...

- Die Adressierung sollte unabhängig von den Hardware-Adressen der im Netzwerk beteiligten Rechnern geschehen
  - Ziel: Kommunikation von Anwendungsprogrammen, ohne die Hardware-Adressen der Kommunikationspartner zu kennen
- In IPv4 besteht die IP-Adresse aus einer Binärzahl der Länge 32 Bit (4 Bytes), aufgeteilt in
  - Präfix (Netzwerknummer); gibt das Netzwerk an, in dem sich der Empfänger befindet. Diese Nummer wird global vergeben
  - Suffix, gibt die lokale Adresse des Empfängers in seinem Netzwerk an. Diese Nummer kann lokal vom Administrator vergeben werden
- Problem: Wie soll die Aufteilung zwischen Präfix und Suffix erfolgen?
  - Langer Präfix: viele Netze (aber mit jeweils wenigen Rechnern, da dann kurzer Suffix), oder
  - Langer Suffix (dann viele Rechner pro Netzwerk, aber wenige Netzwerke, da kurzer Präfix)

… IPv4: Adressierung - IP-Adressen …

- Unterteilung des Adressraums in drei Klassen.
  Trennung zwischen Präfix und Suffix jeweils an Bytegrenzen.
  - Klasse A: Erstes Bit ist 0_2
    - Präfix: folgende 7 Bit
    - Suffix: letzte 24 Bit
  - Klasse B: Erste beiden Bit: 10_2
    - Präfix: folgende 14 Bit
    - Suffix: letzte 16 Bit
  - Klasse C: Erste drei Bit: 110_2
    - Präfix: folgende 21 Bit
    - Suffix: letzte 8 Bit
  - Klasse D: Erste vier Bit: 1110_2. Spezielle Klasse für Multicast
    - Bereich danach frei wählbar (muss für alle Rechner einer Gruppe gleich sein; diese sind dann alle Empfänger von Paketen, die via IP-Multicast verschickt wurden)
  - Klasse E: Erste vier Bit 1111_2. Freigelassen für zukünftige Nutzung

… IPv4: Adressierung - IP-Adressen

- Notation: „Dotted Decimal Notation“
  - Jeder 8 Bit-Block wird in eine Dezimalzahl umgewandelt. Diese Dezimalzahlen werden dann, durch Punkte getrennt, aneinandergefügt
  - Adressbereich von 0.0.0.0 bis 255.255.255.255
- Vergabe der Netzwerkkennungen in CH durch Switch (`www.switch.ch`)
- Problem: Was macht man bei vielen kleinen Netzen mit nur wenigen Hosts (z.B. 9 Stück). Soll hierfür ein komplettes Netz der Klasse C verwendet werden (obwohl insgesamt 16 solcher Mini-Netze im Adressraum eines C-Netzes Platz hätten). Gehen nicht irgendwann die Netzwerkadressen aus?
- Mit zusätzlicher Information kann nur ein Teil einer Klasse einem Netzwerk zugeordnet werden.
  - Adressmaske (Subnetzmaske). Gesetzte Bits zeigen an, welcher Teil der Adresse zum Präfix gehört, welcher zur Hostadresse.
    Die Trennung muss jetzt nicht mehr an der Byte-Grenze geschehen.

„Sprechende“ IP-Adressen

- IP-Adressen sind nicht sprechend, haben wenig Aussagekraft (für den Menschen)
- Daher werden den IP-Adressen symbolische Rechnernamen zugewiesen
- Diese Namen werden in einer verteilten Namensdatenbank verwaltet. Dieses System bezeichnet man als Domain Name System (DNS)
- Das DNS stellt Dienste bereit, mit denen symbolische Namen aufgelöst und in die eigentlichen IP-Adressen umgewandelt werden können.
  - Ein Client stellt eine Anfrage an einen DNS-Server. Falls dieser den symbolischen Namen auflösen und die IP-Adresse bestimmen kann, so gibt er das Resultat zurück. Ansonsten wird er selbst Client eines anderen DNS-Servers
- Beispiel:
  - www.unibas.ch besitzt die IP-Adresse 131.152.228.33

Ports und Applikationen

- Die Kommunikation zwischen zwei Rechnern läuft in der Regel auf der Basis von Client/Server-Interaktionen ab, d.h., die beteiligten Rechner übernehmen bestimmte Rollen
- Auf einem Host laufen meist unterschiedliche Serveranwendungen, die von mehreren Clients benutzt werden können. Um die Server voneinander unterscheiden zu können, werden Server-Prozesse (Threads) mit Portnummern assoziiert
  - Portnummern werden oberhalb von IP auf der Transportschicht definiert
  - Bereich zwischen 0 und 65‘535 (= 2^{16}-1)
- TCP-Socket = 4-Tupel, bestehend aus:
  ⟨`src-IP, src-Port, dest-IP, dest-Port`⟩

Netzwerkprogrammierung

Netzwerkprogrammierung in Java

- Wie kann man eine **`java`**-Verbindung auf einen entfernten Rechner herstellen?
- Klasse **`InetAddress`** des Pakets **`java.net`** für die Adressierung
- **`localhost`** ist eine Pseudo-Adresse für den eigenen Host: 127.0.0.1

```
import java.net.*;
public class IpAddress {
   // Usage: java IpAddress <host>
   public static void main(String[] args) {
      try { // Get requested address
         InetAddress addr =
            InetAddress.getByName(args[0]);
         System.out.println(addr.getHostName());
         System.out.println(addr.getHostAddress());
      } catch (UnknownHostException e) {
         System.err.println(e.toString());
         System.exit(1);
      }
   }
}
```

`getByName()` erwartet IP-Adresse oder Hostname

`getHostName()` liefert symbolischen Namen

`getHostAddress()` liefert IP-Adresse

`getLocalHost()`liefert **`InetAddress`** für den eigenen Rechner

Aufbau einer Socket-Verbindung

Wie können Server und Client miteinander kommunizieren?

- Als Socket bezeichnet man eine streambasierte Schnittstelle zur TCP/IP-Kommunikation zweier Rechner
- Übertragen von Daten ähnelt dem Zugriff auf eine Datei:
  - Verbindungsaufbau, Daten lesen/schreiben, Verbindung schliessen
- Die Klassen **`Socket`** (Client) und **`ServerSocket`** (Server) repräsentieren Sockets aus der Sicht einer Client-Server Anwendung
- **`Socket`** besitzt zwei Konstruktoren
  - **`public Socket(String host, int port);`**
  - **`public Socket(InetAddress address, int port);`**
- Nachdem die Socket-Verbindung erfolgreich aufgebaut wurde, kann mit den beiden Methoden **`getInputStream()`**, **`getOutputStream()`** je ein Stream zum Empfangen und Versenden verfügbar gemacht werden

Client Socket: Lesen einer Socket-Verbindung

- Beispiel: Abfrage des DayTime-Services (Port 13) via Client-Socket

```
import java.net.*;
import java.io.*;
// Usage java SocketTest <host>
public class SocketTest {
   public static void main(String[] args) {
      try {
         Socket sock = new Socket(args[0], 13);
         InputStream in = sock.getInputStream();
         int len;
         byte[] b = new byte[100];
         while ((len = in.read(b)) != -1) {
            System.out.write(b, 0, len);
         }
         in.close();
         sock.close();
      } catch (IOException e) {
         System.err.println(e.toString());
         System.exit(1);
      }
   }
}
```

Das Programm gibt die vom Server gesendeten Daten aus, bis durch einen Rückgabewert -1 angezeigt wird, dass keine weiteren Daten gesendet werden.

Verwendung von **`getInputStream()`** um Serverdaten zu lesen

Start: **`java SocketTest time-c.nist.gov`**
Ausgabe: 58542 19-02-28 22:26:41 00 0 0 766.1 UTC (NIST) *

Client: Lesen & Schreiben von Socket-Verbindungen

```
import java.net.*; import java.io.*;
public class EchoClient {
  public static void main(String[] args) {
    try {
      Socket sock = new Socket(args[0],
Integer.parse(args[1]));
      InputStream in = sock.getInputStream();
      OutputStream out= sock.getOutputStream();
      // create server reading thread
      InThread th = new InThread(in);
      Thread iT = new Thread(th); iT.start();
      // stream input
      BufferedReader conin =
        new BufferedReader(
        new InputStreamReader(System.in));
      String line = " ";
      while (true) {
        // reading input stream
        line = conin.readLine();
        if (line.equalsIgnoreCase("QUIT")) {
          break;
        }
        // writing to ECHO server
        out.write(line.getBytes());
        out.write('\r\n');
      } // terminate program
      System.out.println("terminating ..");
      in.close(); out.close(); sock.close();
    }
  catch (IOException e) { ... }
  }
}
```

- Das Programm stellt eine Verbindung zum ECHO-Service her
- Client schickt Daten an Server
- Dieser liest die Daten und sendet sie unverändert zurück

```
import java.io.*;
class InThread implements Runnable {
  InputStream in;
  public InThread(InputStream in) {
    this.in = in;
  }
  public void run() {
    int len;
    byte[] b = new byte[100];
    try {
      while (true) {
        if ((len=in.read(b))==-1) {
          break;
        }
        System.out.write(b, 0, len);
      }
    }
    catch (IOException e) { ... }
  }
}
```


Server-seitige Sockets

- Details zu Server-Sockets
  - **`public ServerSocket(int port) // Konstruktor`**
  - **`public Socket accept() // Methode`**
- **`accept()`** blockiert solange, bis sich ein Client anmeldet.
  - Beispiel: Einfacher ECHO-Server, der auf einen Client auf Port 8090 wartet und alle Daten unverändert zurücksendet.
    Zur Kontrolle werden die Server-Daten auf die Konsole geschrieben

ECHO-Server

```
import java.net.*;
import java.io.*;
public class SimpleEchoServer {
  public static void main(String[] args) {
    try {
      System.out.println("Warte auf Port 8090...");
      ServerSocket echod = new ServerSocket(8090);
      Socket socket = echod.accept();
      System.out.println("Verbindung hergestellt");
      InputStream in = socket.getInputStream();
      OutputStream out = socket.getOutputStream();
      int c;
      while ((c = in.read()) != -1) {
        out.write((char)c);
        System.out.print((char)c);
      }
      System.out.println("Verbindung beendet");
      socket.close();
      echod.close();
    } catch (IOException e) {
      System.err.println(e.toString());
      System.exit(1);
    }
  }
}
```

Ports unter 1024 dürfen nur mit Root-Berechtigung gestartet werden. Hier: ECHO Server auf 8090

Wird der Server gestartet **`java SimpleEchoServer`** kann mit dem **`EchoClient`** und Port 8090 auf den Server zugegriffen werden

Verwendung Echo-Beispiel

- Terminal 1 (Echo-Server):
  ```
  $> javac SimpleEchoServer.java
  $> java SimpleEchoServer
  Warte auf Port 8090...
  ```
- Terminal 2 (Echo-Client):
  ```
  $> javac EchoClient.java
  $> java EchoClient localhost 8090
  ```
- Sämtliche Eingaben in Terminal 2, welche mit Enter gesendet wurden, werden umgehend vom SimpleEchoServer zurück geschickt.

Server-Verbindungen zu mehreren Clients ...

- Der ECHO-Server soll wie folgt erweitert werden
  - Der Server soll mehr als einen Client gleichzeitig bedienen
  - Die Clients sollen durchnummeriert werden
  - Beim Verbindungsaufbau soll der Client eine Begrüssungsmeldung erhalten
  - Für jeden Client soll ein eigener Thread angelegt werden

... Server-Verbindungen zu mehreren Clients ...

```
public class EchoServer {
  public static void main(String[] args) {
    int cnt = 0;
    try {
      System.out.println(
      "Warte auf Verbindungen auf Port 8090...");
      ServerSocket echod = new ServerSocket(8090);
      while (true) {
        Socket socket = echod.accept();
        eC = new EchoClientThread(++cnt, socket);
        Thread eCT = new Thread(eC); eCT.start();
      }
    } catch (IOException e) {
      System.err.println(e.toString());
      System.exit(1);
    }
  }
}
```

… Server-Verbindungen zu mehreren Clients

```
class EchoClientThread implements Runnable {
  private int name;
  private Socket socket;
  // constructor
  public EchoClientThread(int name, Socket socket){
    this.name = name;
    this.socket = socket;
  }
  public void run() {
    String msg = "EchoServer: Verbindung " + name;
    System.out.println(msg + " hergestellt");
    try {
      InputStream in = socket.getInputStream();
      OutputStream out = socket.getOutputStream();
      out.write(("cs108:"+msg+"\r\n").getBytes());
      int c;
      while ((c = in.read()) != -1) {
        out.write((char) c);
        System.out.print((char)c);
      }
      System.out.println("Terminate "+name);
      socket.close();
    } catch (IOException e) {
      System.err.println(e.toString());
    }
  }
}
```


Netzwerkprogrammierung

Java-Projekt und Netzwerkprogrammierung …

- Der Spiel-Server und die verschiedenen Clients kommunizieren mit Hilfe eines textbasierten Netzwerk-Protokolls
  - Das Protokoll muss lesbar sein, um eine Fehlersuche zu vereinfachen
  - Idealerweise besitzen alle Anweisungen des Protokolls eine feste Länge (Nachrichten einfach zu parsen)
  - Einfache Reaktion auf eingehende Nachrichten. Beispiel:

```
enum Protocol {AAAA, BBBB, CCCC, ZZZZ};
...
Protocol msg;
...
switch (msg){
      case AAAA: ... ; break;
      case BBBB: ... ; break;
      ...
}
```


… Java-Projekt und Netzwerkprogrammierung …

- Beispiel POP3 (→ Kapitel 3 – Client/Server-Architektur)

```
enum Protocol {DELE, PASS, QUIT, RETR, STAT, USER};
...
Protocol msg;
...
switch (msg){
       case DELE: // delete chosen email
                  ... ; break;
       case PASS: // check password
                  ... ; break;
       case QUIT: // terminate connection
                  ... ; break;
       ...
}
```

… Java-Projekt und Netzwerkprogrammierung …

- Wichtig für die Implementierung des Protokolls insbesondere in der Gruppe ist es, die Semantik der einzelnen Befehle und speziell ihr Zusammenspiel festzulegen

- Empfohlene Vorgehensweise
  - Zuerst „Kommunikationsskelett“ für den Server implementieren
    - Wartet auf Verbindung, dann: akzeptiert Verbindung (baut diese auf)
    - Wartet auf Eingabe, dann: gibt Echo zurück
  - Danach, wenn Basisfunktionalität steht, sukzessive erweitern
  - Erst am Ende mit dem GUI verbinden

… Java-Projekt und Netzwerkprogrammierung

- Ports sorgfältig wählen
  - Nicht aus dem reservierten Bereich (unter 1024)
  - Vermeiden bereits bekannter Ports > 1024
    (z.B. Port 8080, Alternative für HTTP)