



TECHNISCHE HOCHSCHULE MITTELHESSEN

THM

**CAMPUS
GIESSEN**

MNI

Mathematik, Naturwissenschaften
und Informatik

Netzwerkcommunication

CS1016 Programmierung interaktiver Systeme

von Prof. Dr. Weigel

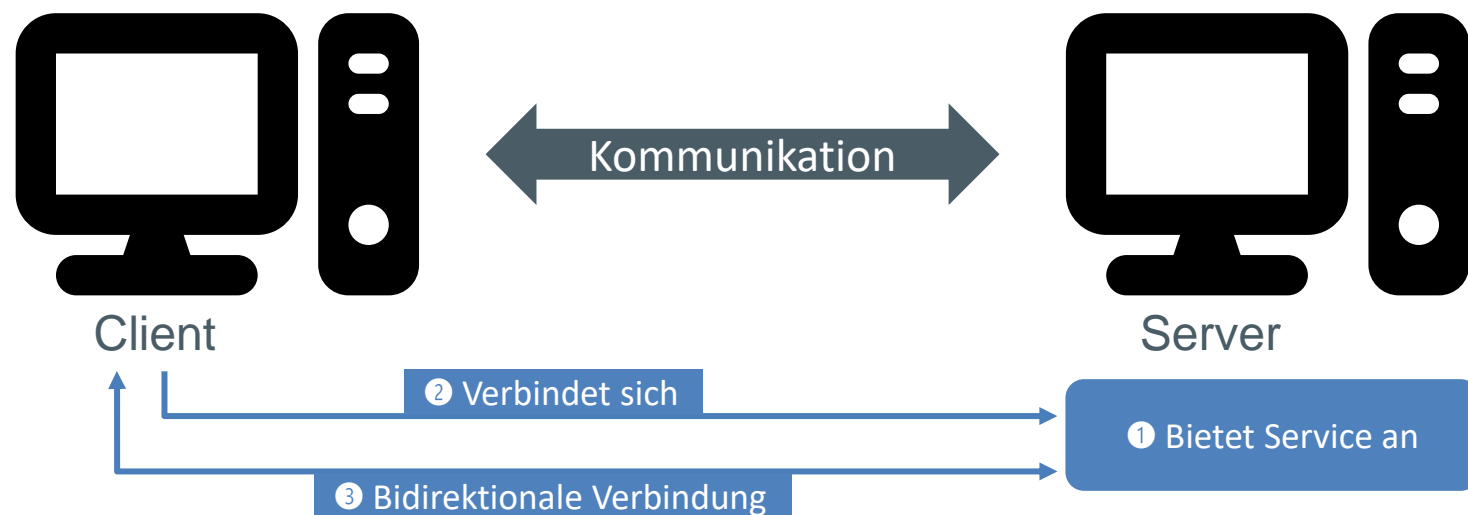


Netzwerkcommunication

1. Client-Server Kommunikation
2. TCP und UDP
3. Sockets in Java
4. Serialisierung von Daten

Client-Server Kommunikation

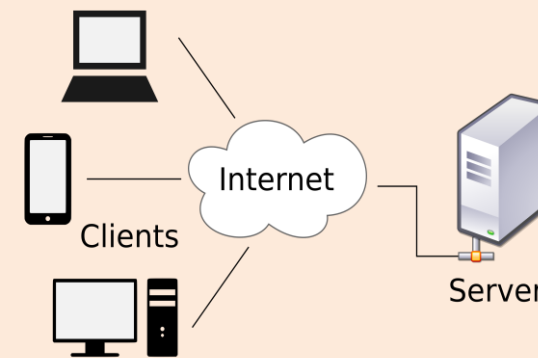
Kommunikation zwischen **zwei Computern** (ein Server und ein Client)



Erlaubt auch eine Kommunikation zwischen zwei Programmen auf dem gleichen PC

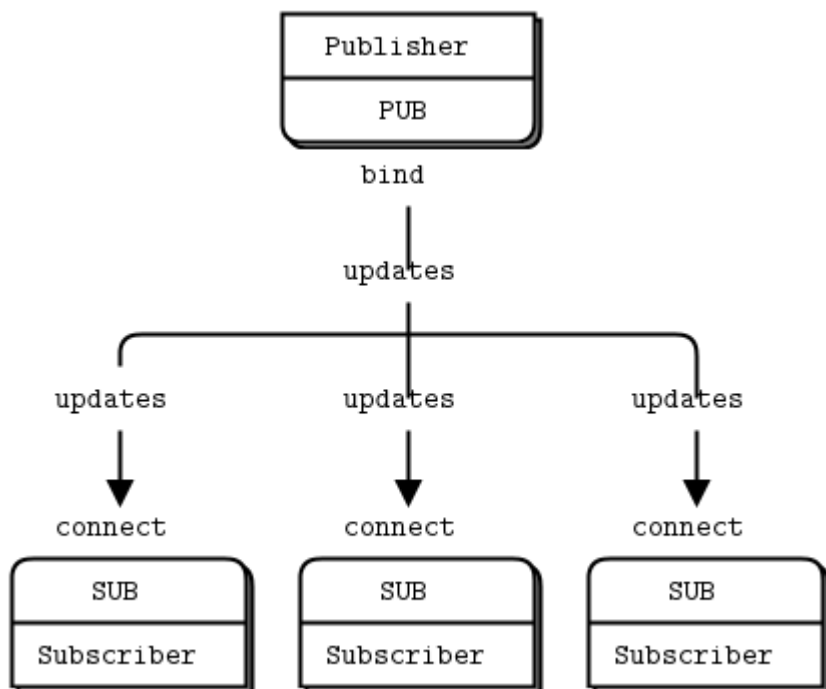
→ Gut zum Testen 😊

Client-Server Kommunikation im Internet:



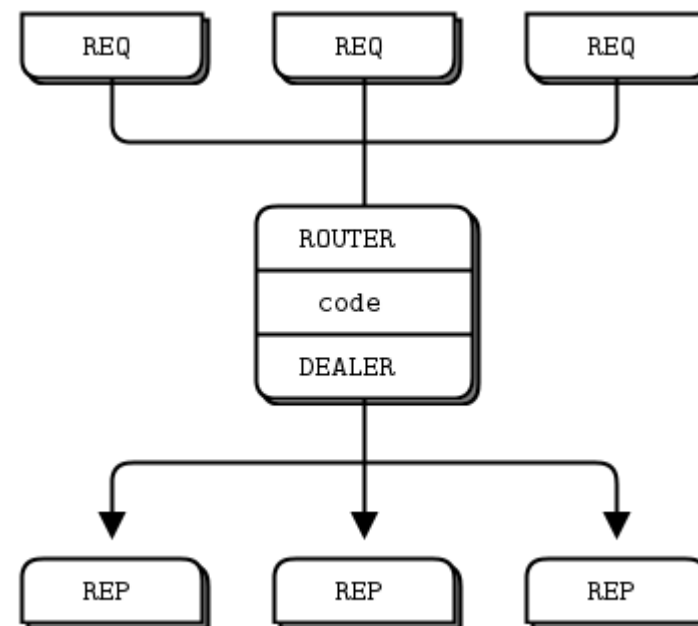
Andere Kommunikationsarten (nicht Teil dieser Vorlesung)

Publish-Subscribe (z.B. MQTT)



Asynchronous Request – Response

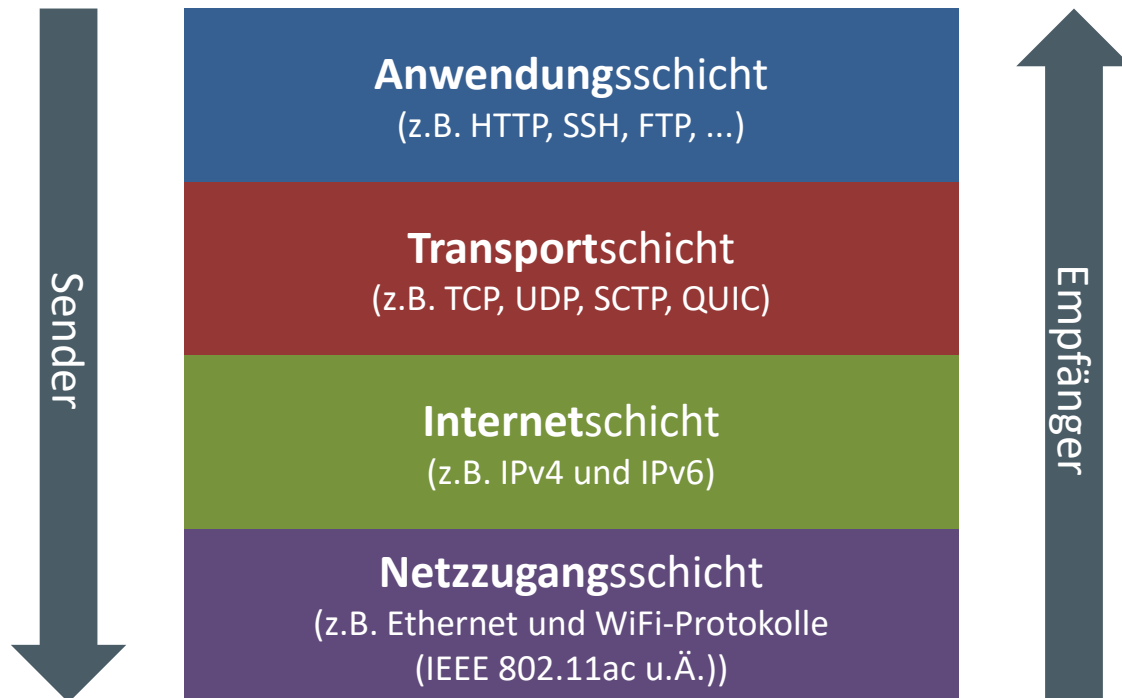
(uvm.)



→ Falls benötigt, empfehle ich eine Bibliothek wie z.B. ZeroMQ

[Bildquelle: <https://blog.scottlogic.com/2015/03/20/ZeroMQ-Quick-Intro.html>]

TCP/IP-Referenzmodell



Anwendungsschicht

Heutige Vorlesung

- Protokolle für Anwendungsprogrammen zum Austausch anwendungsspezifischer Daten

Transportschicht

- Ermöglicht eine Ende-zu-Ende-Kommunikation
- Erstellt Pakete aus den Anwendungsdaten

Internetschicht

- Vermittelt Pakete mittels Internet Protocol (IP)
- Heute meist IPv4 oder IPv6

Netzzugangsschicht

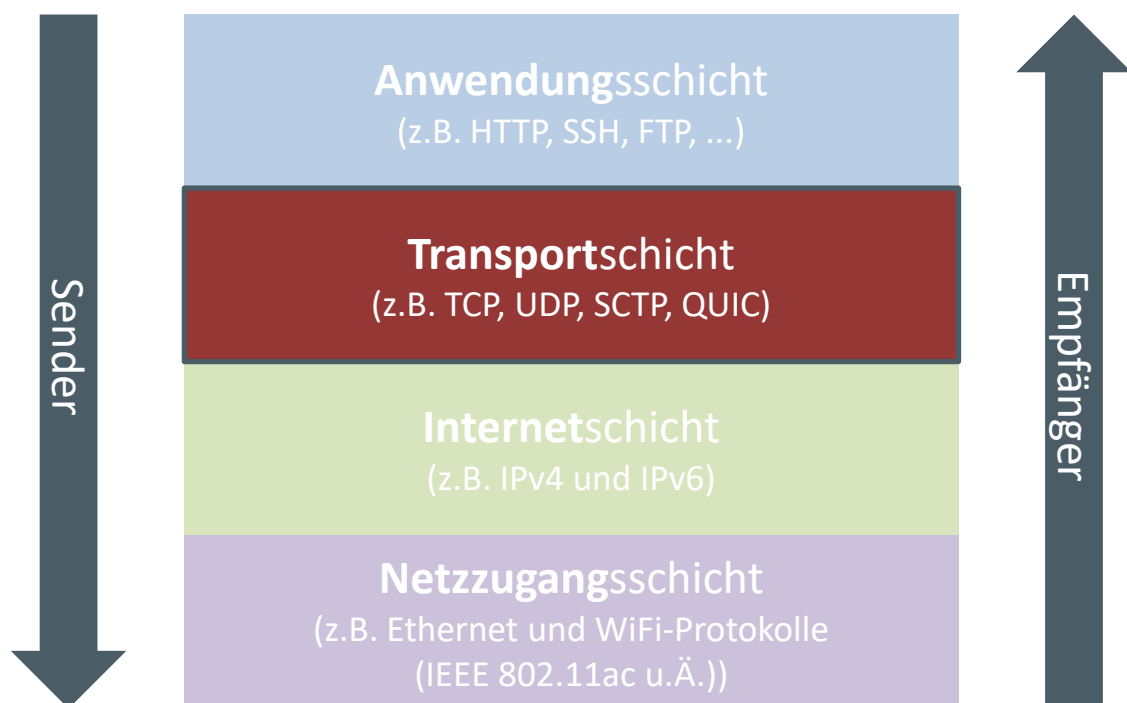
- Techniken zur Datenübertragung
- Sicherungs- und Bitübertragungsschicht im ISO/OSI-Referenzmodells



Netzwerkcommunication

1. Client-Server Kommunikation
2. TCP und UDP
3. Sockets in Java
4. Serialisierung von Daten

Die Transportschicht



Beispiele:

Transmission Control Protocol

- Zuverlässige, verbindungsorientierte Datenübertragung

User Datagram Protocol

- Verbindungslose Datenübertragung

Stream Control Transmission Protocol

- Verbindungsorientiert, aber mit Datagramm support
- Komplex und geringe Router-Unterstützung

QUIC

- Beinhaltet TLS Schlüsselaustausch
- UDP-basierend mit mehreren "Streams"

...

Transmission Control Protocol

TCP ist ein **zuverlässiges**

= Sender kann sich sicher sein, dass die Pakete angekommen sind

verbindungsorientiertes

= es wird (durch einen sogenannten „Handshake“) eine dauerhafte Verbindung hergestellt

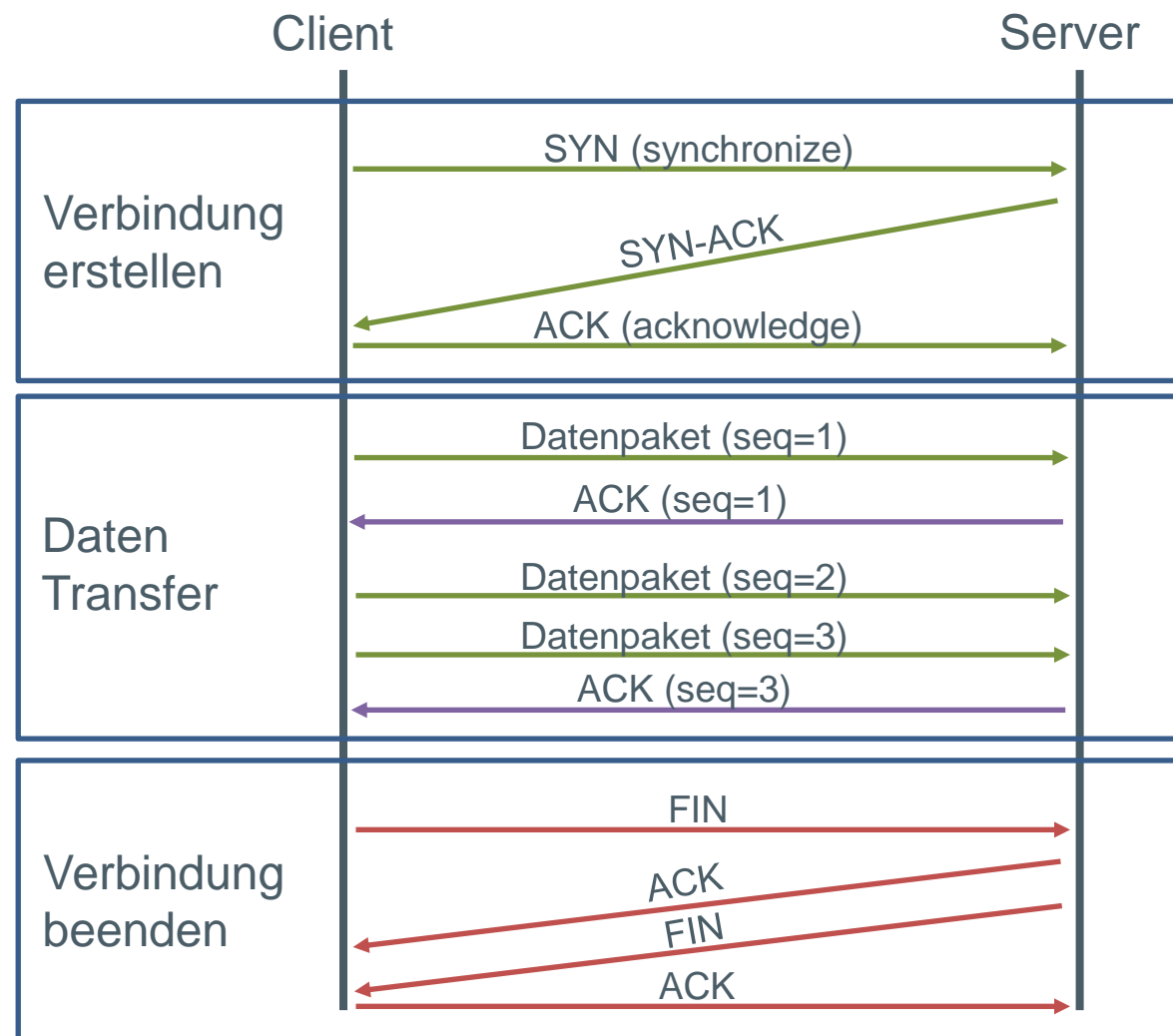
Protokoll zur Datenübertragung

Benutzt von:

- Webseiten
- SSH / FTP / Telnet
- SMTP / IMAP / POP3 (d.h. E-Mail)

Aber:

- Komplex (z.B. nicht immer ein FIN)
- Denial-of-Service Attacken (z.B. SYN Flood)



User Datagram Protocol

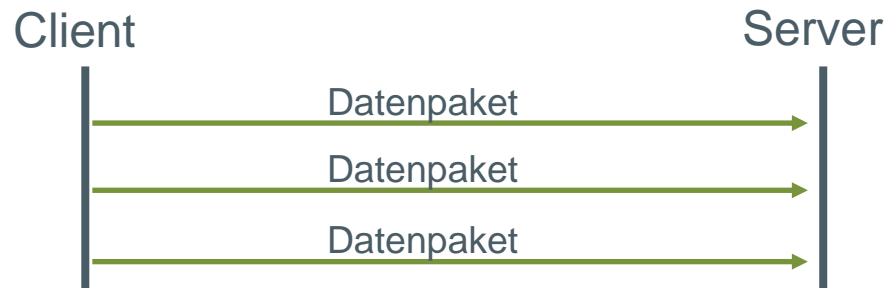
UDP ist ein verbindungsloses

= es können sofort Daten verschickt werden

Protokoll zur Datenübertragung

Benutzt von:

- Real-Time Anwendungen bei denen Paket-Verluste akzeptabel sind
z.B. *Media-Streams* oder *Videospiele*
- Anwendungen mit vielen Verbindungen
z.B. *NTP* oder (teilweise) *DNS*



[Bildquelle: <https://medium.com/javarevisited/fundamentals-of-udp-socket-programming-in-java-4a6972370592>]



Netzwerkcommunication

1. Client-Server Kommunikation
2. TCP und UDP
3. Sockets in Java
4. Serialisierung von Daten

Was ist ein Socket?

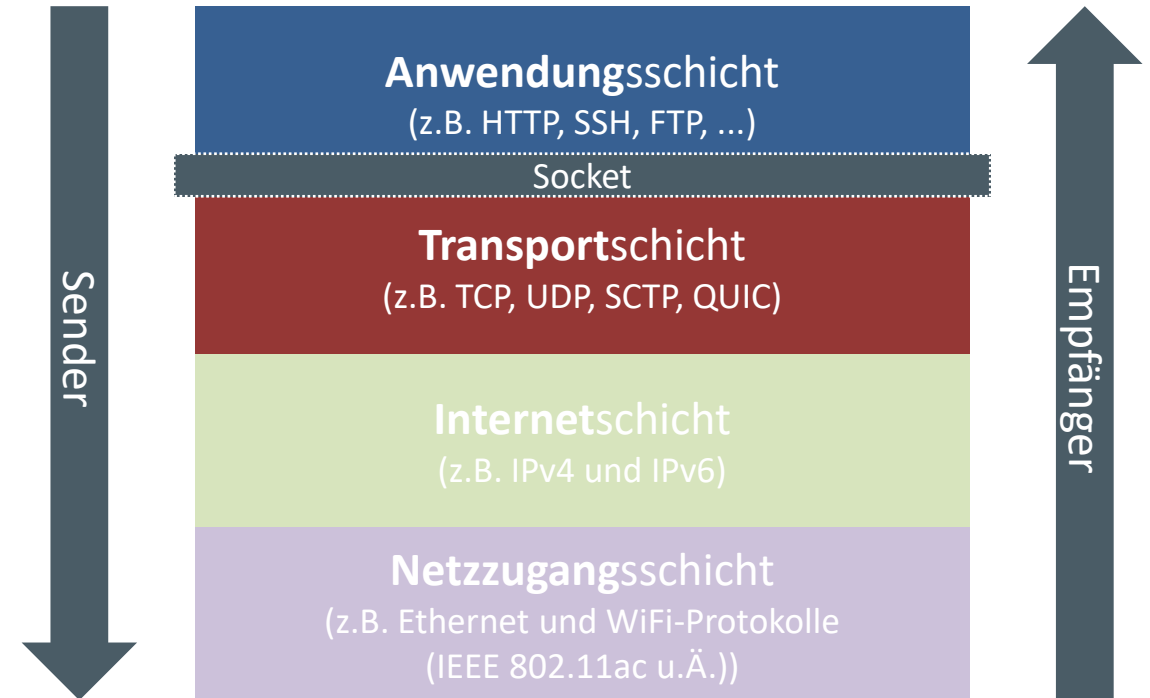
“Plattformunabhängige standardisierte Schnittstelle (API) zwischen der Netzwerkprotokoll-Implementierung des Betriebssystems und der eigentlichen Anwendungssoftware.” – Wikipedia

Erlaubt das...

- Senden von Daten aus Programmen
- Entwickeln einer eigenen Anwendungsschicht

Der Server bindet den Socket an einen **Port**

- Nummer von 0 – 65.535
- Erlaubt unterscheiden von Verbindungen
- Belegung prüfen:
<https://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.txt>
- HTTP: 80; HTTPS: 443; SMTP: 25; ...



Ein TCP Server in Java

```
Socket socket = null;
```

```
ServerSocket server = null;
```

```
try {
```

```
    server = new ServerSocket(port);
```

Öffnet einen Socket auf dem gegebenen Port

```
    socket = server.accept();
```

Blockiert die Funktion und wartet auf einen Client

```
    String line = "";
```

```
    var reader = new BufferedReader(  
        new InputStreamReader(socket.getInputStream()));
```

BufferedReader liest mehrere Zeichen
InputStreamReader liest ein Zeichen

```
    while (!line.equals("GAMEOVER")) {
```

```
        line = reader.readLine();
```

```
        System.out.println("RECEIVED:" + line);
```

```
    }
```

```
} finally {
```

```
    socket.close();
```

```
    server.close();
```

Sockets müssen immer geschlossen werden,
sonst könnte der Port blockiert bleiben.

```
}
```

Ein TCP Client in Java

```
Socket socket = null;
```

```
PrintWriter writer = null;
```

```
try {
```

```
    socket = new Socket(ip, port);
```

← Verbindet sich mit dem Server (IP , PORT)

```
    writer = new PrintWriter(socket.getOutputStream(), true);
```

```
    for (int i = 0; i < 10; i++) {
```

```
        writer.println("Hello World " + i);
```

```
        TimeUnit.SECONDS.sleep(1);
```

```
    }
```

```
    writer.println("GAMEOVER");
```

```
} finally {
```

```
    socket.close();
```

```
}
```

Sofortiges senden über den Socket bei *println()*
Ansonsten manuell mit *flush()*

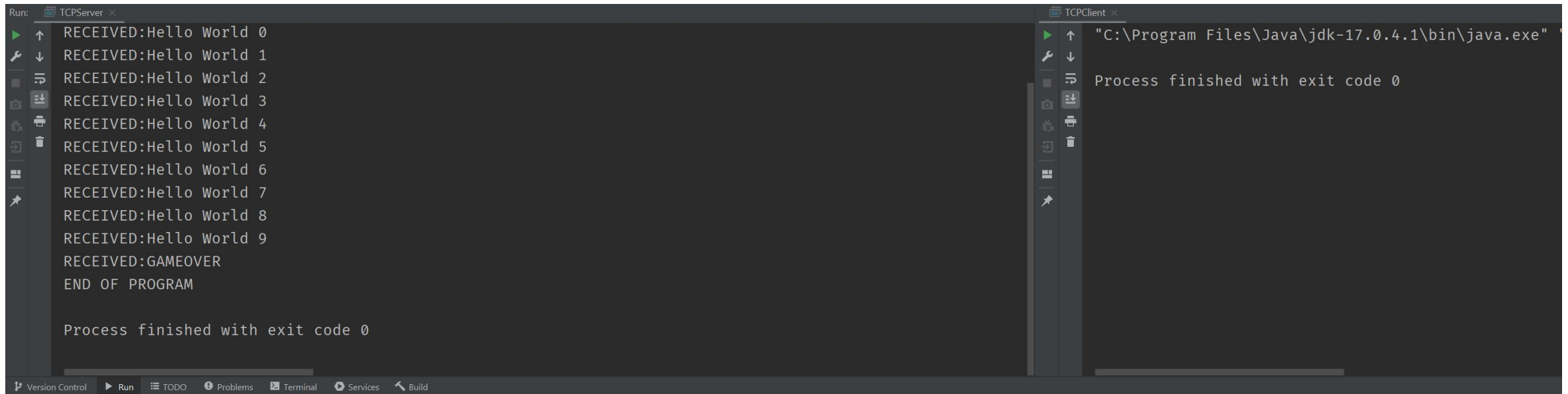
Schreibt einen Text über den Socket

Sockets müssen immer geschlossen werden
Terminiert die Verbindung

Live Demo

Siehe Übung von VW07 (Paket: vw07.lecture)

1. Starten von TCPServer
2. Starten von TCPClient



The screenshot shows two terminal windows side-by-side. The left window, titled 'Run: TCPServer', displays the following output:

```
RECEIVED:Hello World 0
RECEIVED:Hello World 1
RECEIVED:Hello World 2
RECEIVED:Hello World 3
RECEIVED:Hello World 4
RECEIVED:Hello World 5
RECEIVED:Hello World 6
RECEIVED:Hello World 7
RECEIVED:Hello World 8
RECEIVED:Hello World 9
RECEIVED:GAMEOVER
END OF PROGRAM

Process finished with exit code 0
```

The right window, titled 'TCPClient', displays the following output:

```
"C:\Program Files\Java\jdk-17.0.4.1\bin\java.exe"
Process finished with exit code 0
```

The bottom of the image shows the IDE's status bar with icons for Version Control, Run, TODO, Problems, Terminal, Services, and Build.

Senden und Empfangen mit UDP in Java

```
DatagramChannel channel = null;
```

```
try {
```

```
    channel = DatagramChannel.open();
```

Öffnet einen UDP Channel

```
    SocketAddress server = new InetSocketAddress("time.nist.gov", 37);
```

NTP Zeitserver

```
    for (int i = 0; i < 3; i++) {
```

```
        ByteBuffer buffer = ByteBuffer.allocate(8);
        buffer.put("test".getBytes());
        channel.send(buffer, server);
```

Sendet ein Datenpaket an den Server

```
        buffer.clear();
        buffer.putInt(0);
```

```
        channel.receive(buffer);
        buffer.flip(); // write to read flip
        System.out.println(buffer.getLong());
        TimeUnit.SECONDS.sleep(10);
```

Empfängt ein Datenpaket vom Server
Speichert empfangene Daten in buffer

```
    }
```

```
} finally { channel.close(); }
```



Netzwerkcommunication

1. Client-Server Kommunikation
2. TCP und UDP
3. Sockets in Java
4. **Serialisierung von Daten**

Problem: Wie können wir strukturelle Daten übertragen?

Ein Socket überträgt einen String (d.h. Charakter-Array)
oder einen Byte-Array → Serielle Datenstruktur

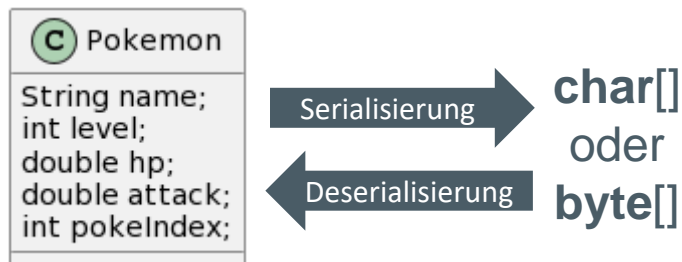
Serialisierung:

Übersetzung von strukturellen Daten (z.B. eines Objektes) in eine
sequentielle Darstellungsform, z.B. für:

- Senden von Daten (Netzwerk, serielle Schnittstelle)
- Speichern von Daten in Dateien

Deserialisierung:

- Umkehrfunktion der Serialisierung
- Bekommt die sequentielle Darstellungsform,
erzeugt daraus Datenobjekte



Serialisierung in Java

Interface implementieren

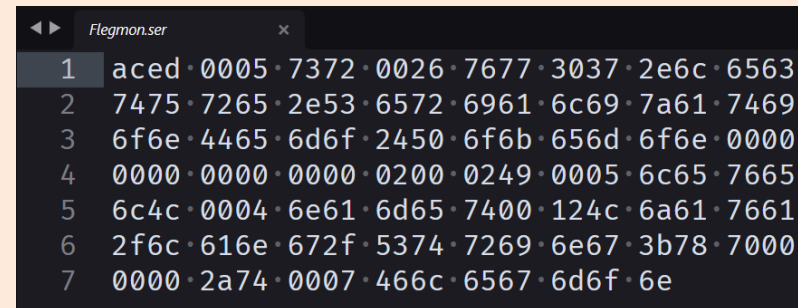
```
record Pokemon(String name, int level) implements Serializable {}
```

```
static void savePokemon(Pokemon p, String file) throws IOException {  
    ObjectOutputStream oos = null;  
    try {  
        Serialisiert das Objekt  
        oos = new ObjectOutputStream(new FileOutputStream(file));  
        oos.writeObject(p);  
    } finally { oos.close(); }  
}
```

```
static Pokemon loadPokemon(String file) throws IOException, ClassNotFoundException {  
    ObjectInputStream ois = null;  
    try {  
        Deserialisiert das Objekt  
        ois = new ObjectInputStream(new FileInputStream(file));  
        return (Pokemon) ois.readObject();  
    } finally { ois.close(); }  
}
```

Beispiel:

```
var p = new Pokemon("Flegmon", 42);  
savePokemon(p, "Flegmon.ser");
```



```
1 aced·0005·7372·0026·7677·3037·2e6c·6563  
2 7475·7265·2e53·6572·6961·6c69·7a61·7469  
3 6f6e·4465·6d6f·2450·6f6b·656d·6f6e·0000  
4 0000·0000·0000·0200·0249·0005·6c65·7665  
5 6c4c·0004·6e61·6d65·7400·124c·6a61·7661  
6 2f6c·616e·672f·5374·7269·6e67·3b78·7000  
7 0000·2a74·0007·466c·6567·6d6f·6e
```

```
System.out.println(loadPokemon("Flegmon.ser"));
```

Alternativen (insb. für Kompatibilität mit anderen Programmiersprachen)

JSON (= JavaScript Object Notation)

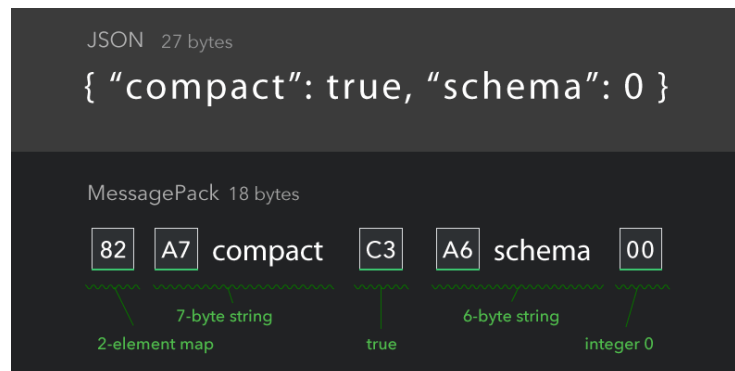
- 😊 Standardformat im Web
- 😊 Menschen-lesbar
- 😞 Größe
- 😞 Rundungen bei Float/Double

```
{
  "name": "Max",
  "age": 27,
  "drinking_age": true
}
```

z.B. mit der **gson** Bibliothek

MessagePack

- 😊 Ähnliche Struktur wie JSON
- 😊 Kompakter als JSON
- 😊 Genaue Float-Repräsentation
- 😞 Nicht Menschen-lesbar



XML (=Extensible Markup Language)

- 😊 Bewährtes Dateiformat
- 😊 Großes Ecosystem (Schema, Transformationen, ...)
- 😞 Viel Schreibarbeit </endtags>

```
<?xml version="1.0" encoding="UTF-8"?>
<data>
  <name>Max</name>
  <age>27</age>
  <drinking_age>true</drinking_age>
</data>
```

Fragen?



WRITTEN BY @ RAPHCOMICS

ART BY @PROLIFICPENCOMICS

[Bildquelle: <https://twitter.com/ProPenComics/status/1039886672765181953/photo/1>]