

## 2 Kommunikation und Synchronisation von Prozessen (ca. 20 Std.)

Lp: Beim weltweiten Austausch von Information spielt die Kommunikation zwischen vernetzten Rechnern eine entscheidende Rolle. Die Schüler lernen, dass es hierzu fester Regeln für das Format der auszutauschenden Daten sowie für den Ablauf des Kommunikationsvorgangs bedarf. Der gemeinsame Zugriff auf Ressourcen führt sie zum Problem der Kommunikation und Synchronisation parallel ablaufender Prozesse, bei dessen Lösung die Jugendlichen erneut den Anwendungsbereich ihrer Modellierungstechniken erweitern.

Bei jedem der nachfolgenden Kapitel wird eine kurze inhaltliche Zusammenfassung der Grundlagen gegeben. Diese inhaltlichen Grundlagen sind unabhängig von der Umsetzungsreihenfolge im Unterricht und gehen an manchen Stellen etwas tiefer als im Unterricht benötigt, wenn vermutet werden kann, dass hier Schülerfragen auftreten.

In den Abschnitten „Umsetzungshinweise“ werden einerseits zur Unterrichtsunterstützung geeignete Programme vorgestellt, andererseits javaspezifische Hinweise zu den auf der CD zur Verfügung gestellten Projekte gegeben.

Darüber hinaus werden im Anhang dieses Kapitels hilfreiche Praxistipps angeführt.

### 2.1 Topologie von Rechnernetzen; Internet als Kombination von Rechnernetzen

Lp: Topologie von Rechnernetzen (Bus, Stern, Ring); Internet als Kombination von Rechnernetzen

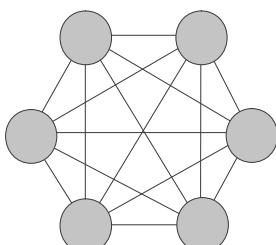
#### 2.1.1 Inhaltliche Grundlagen

Die Verbindung mehrerer Rechner zu einem Netzwerk kann verschiedene Strukturen (Topologien) aufweisen.

Die wichtigsten Grundstrukturen (Bus-, Stern- oder Ringtopologie) sollen in diesem Kapitel besprochen werden. Für ein möglichst gut funktionierendes Netzwerk können dabei folgende Bewertungsaspekte für die verschiedenen Topologien dienen (siehe auch Übersichtstabelle):

- **Störanfälligkeit:** Wie störanfällig ist das Netzwerk beim Ausfall eines Verbindungspartners (Rechners) bzw. bei einer Trennung einer Verbindungsleitung?
- **Systempflege:** Wie groß ist der Aufwand bei der Fehlersuche? Ist eine einfache Erweiterbarkeit des Netzes gegeben?
- **Kosten:** Wie viele Leitungen werden benötigt? Wie hoch sind die Kosten spezieller Komponenten?
- **Übertragungsrate:** Wie gut ist die Übertragungsrate bzw. wie verändert sich diese, wenn sich die Anzahl der Verbindungsteilnehmer erhöht? Ist gleichzeitiges Senden und Empfangen möglich oder kommt es möglicherweise zu sogenannten „Kollisionen“?  
Anmerkung: Die genaue Analyse der Übertragungsrate erfordert sehr viel Spezialwissen und physikalische Kenntnis!

#### Direkte Verbindung aller beteiligter Rechner (Vermaschtes Netz)



Der naive Versuch, ausgehend von zwei Rechnern, alle Rechner eines Netzwerks direkt zu verbinden, zeigt sehr schnell, dass eine viel zu große Anzahl von Leitungen erforderlich wäre.

Bei sechs Rechnern benötigt man z. B. schon  $5 + 4 + 3 + 2 + 1 = 15$  Verbindungsleitungen. Für  $n$  Rechner würde man  $(n-1) + (n-2) + \dots + 1 = \frac{n \cdot (n-1)}{2}$  Verbindungsleitungen benötigen, da bei jedem zusätzlichen  $i$ -ten Rechner ( $i - 1$ ) Leitungen hinzukommen.

Eine solche Netzwerkstruktur ist zwar sehr sicher, da ein Ausfall eines Verbindungspartners andere Verbindungen nicht beeinflusst, zur Verbindung einer großen Anzahl von Rechnern innerhalb eines Netzwerks aufgrund der vielen benötigten Verbindungsleitungen allerdings nicht geeignet.

### Adressierung von Rechnern

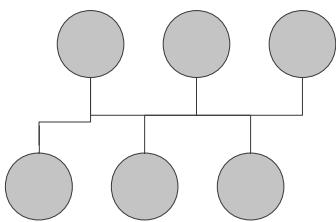
Wenn Rechner nicht mehr direkt miteinander verbunden werden, sondern sich wie in den nachfolgenden Netzwerktopologien das Verbindungsmedium teilen, muss jeder Rechner im Netzwerk einen eindeutigen Bezeichner besitzen.

Im Kontext des Internets werden als Bezeichner sogenannte IP-Adressen verwendet. Diese bestehen aus einer 32-Bit-Zahl, die aus Lesbarkeitsgründen byteweise (Zahlen zwischen 0 und 255 und durch Punkt getrennt) dargestellt wird:

So ergibt sich aus der binär dargestellten Adresse  $01111110000000000000000000000001_2$  bzw. 2130706433 in dezimaler Schreibweise die Darstellung 127.0.0.1 der IP-Adresse für den lokalen Rechner.

Weil IP-Adressen für Menschen schlecht lesbar sind, können diesen IP-Adressen über sogenannte DNS-Server lesbare Bezeichnungen zugeordnet werden und umgekehrt. Wird eine Anfrage (beispielsweise durch einen Webseitenaufruf) an die Adresse „alp.dillingen.de“ gestartet, so erfolgt zunächst eine Namensauflösung durch einen DNS-Server, danach werden die Datenpakete an die richtige IP-Adresse (194.95.207.10) über verschiedene Stationen gesandt (siehe auch Abschnitt 2.1.2).

### Bustopologie



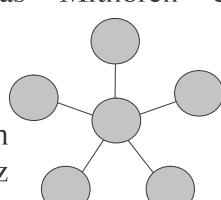
Sollen Rechner möglichst einfach und kostengünstig miteinander verbunden werden, so kann die Bustopologie verwendet werden. Weil dabei alle Netzwerkgeräte an ein gemeinsames Übertragungsmedium angeschlossen sind, muss ein gleichzeitiges Senden von Daten (Kollision) vermieden werden. Jedes angeschlossene Gerät prüft, ob die Leitung zur Übertragung frei ist, und beginnt zu senden. Falls ein weiteres Gerät gleichzeitig den Sendevorgang beginnt, kommt es zur Kollision, da alle Geräte ein gemeinsames Kabel benutzen. Diese Kollision muss im Netzwerk erkannt und der Sendevorgang wiederholt werden. Bei hohem Netzwerkverkehr steigt die Anzahl der Kollisionen überproportional, was zu einer starken Verminderung der tatsächlich übertragenen Daten führt.

Die Nachricht wird an alle Teilnehmer übertragen und muss von allen angeschlossenen Teilnehmern gelesen werden können, damit diese entscheiden können, ob die Nachricht für sie selbst bestimmt ist.

Vertiefende Information: Dadurch ist sogenanntes Sniffing, also das Mithören des Netzwerkverkehrs, möglich!

### Sterntopologie

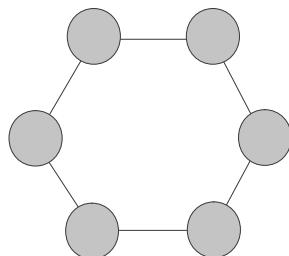
Jeder Rechner (jedes Endgerät) wird direkt mit einem zentralen Knoten verbunden. Selbst wenn ein Endgerät ausfällt oder nur zeitweise mit dem Netz



verbunden ist, hat dies keine Auswirkungen auf das gesamte Netzwerk. Fällt allerdings der Knoten aus, so ist das gesamte Netzwerk gestört.

Vertiefende Information: In einem in Schichten aufgebauten Netzwerk muss klar werden, auf welcher Schicht die Topologie gesehen wird. In der Literatur wird hier häufig zwischen logischer und physischer Ebene unterschieden. Bei der twisted-pair Etherneverbekabelung über einen Hub oder Switch entsteht physikalisch ein Stern, auf logischer Ebene jedoch ein Bus oder Punkt-zu-Punkt-Verbindungen.

### Token-Ring



Im Token-Ring sind Rechner ringförmig miteinander verbunden. Es treten keine Kollisionen auf, da ein Rechner nur dann senden darf, wenn er ein spezielles Datenpaket (das Token) besitzt, das von der Kontrolleinheit des Rings zugeteilt wird. Deshalb bleibt auch bei hohen Lasten die Übertragungsrate konstant. Da eine Nachrichtenübertragung allerdings höchstens so lange dauern kann, wie das Signal zum Umlauf um den Ring benötigt, liegt die Übertragungsrate unterhalb der eines Bussystems.

Vertiefende Information: Wird das Netzwerk unterbrochen, so kann der Token nicht mehr rundum laufen. Die Netzwerkübertragung erfolgt dann über die verbleibende Verbindung mit hin- und herlaufendem (pendelndem) Token.

Im tabellarischen Überblick sind exemplarisch vorteilhafte Aspekte mit [+] und nachteilige mit [-] gekennzeichnet:

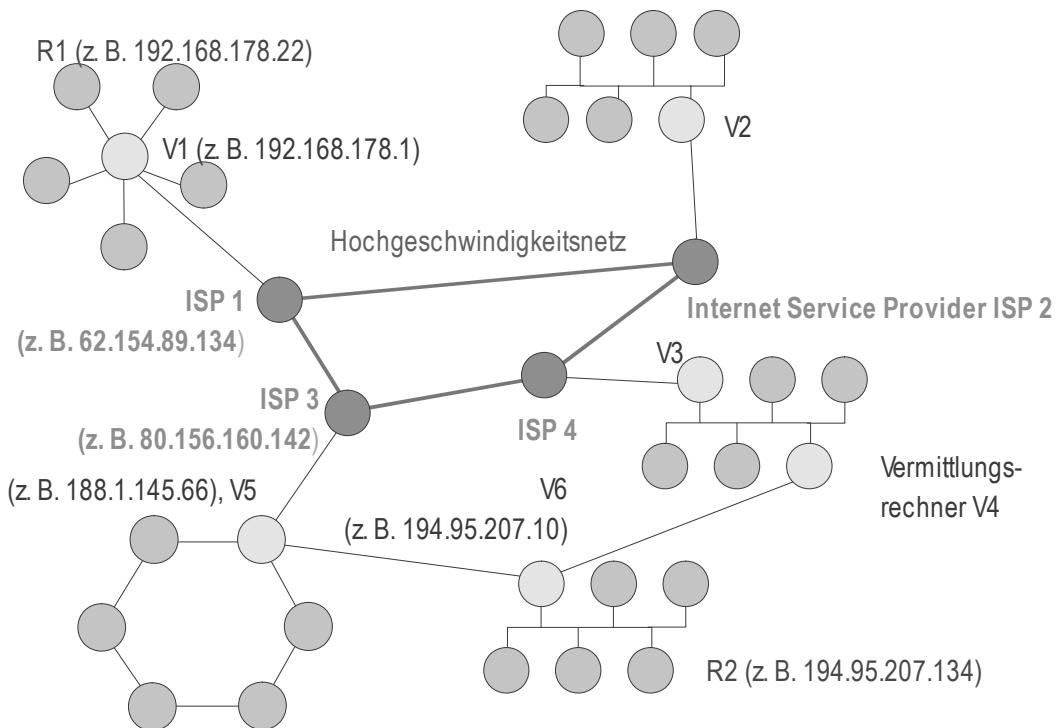
	Bus	Stern	Token-Ring
Ausfallsicherheit	[+] Ausfall eines Gerätes hat für die Funktionalität des Netzwerkes keine Konsequenzen [-] Störung des Übertragungsmediums an einer einzigen Stelle im Bus (defektes Kabel) blockiert den gesamten Netzstrang	[+] Der Ausfall eines Endgerätes hat keine Auswirkung auf den Rest des Netzes [-] Durch Ausfall des zentralen Knotens wird Netzverkehr unmöglich	[+] Pendelbetrieb bei Unterbrechung einer Leitungsvverbindung ist möglich
Kosten, Erweiterbarkeit	[+] geringe Kosten, da nur geringe Kabelmengen und keine aktiven Netzwerkkomponenten nötig; leicht erweiterbar	[+] geringe Kosten, da nur geringe Kabelmengen und keine aktiven Netzwerkkomponenten nötig; leicht erweiterbar	[-] teure Komponenten
Sicherheit	[-] Datenübertragungen können leicht abgehört werden (Sniffing)		[-] Datenübertragungen können leicht abgehört werden (Sniffing)
Übertragungsrate	[-] Reduktion der Übertragungsrate bei hohem Netzwerkerkehr durch Kollisionen	[-] niedrige Übertragungsrate bei vielen Hosts, wenn ein Hub benutzt wird	[+] gleichzeitiges Senden wird durch Token vermieden, dadurch garantierte Übertragungsbandbreite
Anwendungsbereiche	kleine Netzwerke	Anschluss von Haushalten an Verbindungsrechner des Providers, Heimnetzwerke, Computerräume	Rechnernetze in Universitäten, werden aus v.a. Kostengründen nach und nach durch andere Strukturen abgelöst.

## Internet als Kombination von Rechnernetzen

Da jede der oben beschriebenen Strukturen Vor- und Nachteile hat, verwendet man eine Kombination der verschiedenen Topologien, um weltweit Millionen von Rechnern zum Internet zu verbinden. Dabei werden die Rechnernetze (Teilnetze) über Verbindungsrechner (sogenannte Router) miteinander verbunden. Das dadurch entstehende globale Netz ist besonders ausfallsicher, da durch geschickte Verbindung der Teilnetze alternative Datenwege entstehen.

Verbindungsrechner sind für die Vermittlung und Weiterleitung (Routing) der Datenpakete zuständig und können mit Postämtern verschiedener Zustellbezirke (eigenes Teil- bzw. Subnetz) verglichen werden:

Ein Datenpaket startet zunächst im eigenen Subnetz. Anhand der IP-Adresse kann der Vermittlungsrechner erkennen, ob das Datenpaket für ein fremdes Subnetz bestimmt ist, und gegebenenfalls weiterleiten (Vergleich: Das Postamt eines Zustellbezirks leitet einen Brief an eine Adresse eines anderen Zustellbezirks weiter an das zuständige Postamt). Gibt es alternative Weg durch das Netz, so versucht man, Wege mit starker Netzlast zu vermeiden. Auch bei Störungen reagieren die Router und schicken die Datenpakete über alternative Wege im Netz. So können zum Beispiel, wie in der nachfolgenden Graphik dargestellt, Daten vom Rechner R1 über V1, ISP 1 (Internet Service Provider), ISP 3, V5, V6 zu R2 oder von R1 über V1, ISP1, ISP2, ISP4, V3, V4, V6 zu R2 geschickt werden.



Vertiefende Information: Das Hochgeschwindigkeitsnetz, das die Internetserviceprovider verbindet (in der Graphik rot gezeichnet), wird oft auch als „Backbone“ (= Rückgrat) bezeichnet. Auf die Einführung dieses zusätzlichen Begriffs wird hier bewusst verzichtet.

### 2.1.2 Umsetzungshinweise

Um die Netzwerkinformationen bei einem Rechner mit dem Betriebssystem Windows zu ermitteln, startet man das Programm „cmd.exe“ und gibt dort den Befehl „ipconfig“ ein. Man erhält man eine Übersicht über alle Netzwerkadapter des Computers:

```
Drahtlos-LAN-Adapter Drahtlosnetzwerkverbindung:
  Medienstatus . . . . . : Medium getrennt
  Verbindungsspezifisches DNS-Suffix:

Ethernet-Adapter LAN-Verbindung:
  Verbindungsspezifisches DNS-Suffix:
  Verbindungslokale IPv6-Adresse . . . : fe80::4c49:c13:ac0c:43e1%12
  IPv4-Adresse . . . . . : 192.168.178.35
  Subnetzmaske . . . . . : 255.255.255.0
  Standardgateway . . . . . : 192.168.178.1
```

Im Beispiel ist der Adapter für die Drahtlosnetzwerkverbindung nicht verbunden. Der Adapter für die LAN-Verbindung ist mit dem Router (hier als Standardgateway bezeichnet) 192.168.178.1 verbunden. Die eigene Adresse des Rechners im Netz ist 192.168.178.35. Nachrichten, die an diesen Rechner geschickt werden, müssen als Empfänger die IP-Adresse 192.168.178.35 enthalten.

Hinweis: Die Adressen beginnend mit 192.168... sind typisch für einfache Heimnetzwerke, die über einen Router mit dem Internet verbunden werden.

Mit dem Befehl „tracert“ kann man den Ablauf einer Sendung von Daten (hier nach alp.dillingen.de) verfolgen.

```
C:\Users\Admin>tracert alp.dillingen.de
Routenverfolgung zu alp.dillingen.de [194.95.207.10] über maximal 30 Abschnitte:

 1   1 ms    1 ms    1 ms  fritz.fonwlan.box [192.168.178.1]
 2   45 ms   44 ms   45 ms  217.0.116.92
 3   44 ms   44 ms   45 ms  217.0.70.122
 4   54 ms   55 ms   54 ms  l-eb2-i.L.DE.NET.DTAG.DE [62.154.89.134]
 5   55 ms   60 ms   59 ms  80.156.160.142
 6   74 ms   97 ms   96 ms  xr-aug1-ge6-1.x-win.dfn.de [188.1.145.66]
 7   70 ms   68 ms   71 ms  kr-afldil2.x-win.dfn.de [188.1.232.50]
 8   70 ms   68 ms   69 ms  bndlq-fw01-transfer.bndlq.de [194.95.207.108]
 9   70 ms   66 ms   65 ms  alp.dillingen.de [194.95.207.10]
```

Die wichtigsten Stationen im obigen Beispiel sind:

Der DSL-Router im Heimnetzwerk (192.168.178.1), der mit dem Verbindungsrechner des Providers (217.0.116.92) verbunden ist, die Verbindung über das DFN (deutsches Forschungsnetz nach Augsburg und Dillingen), zum dortigen Provider (Bürgernetz Dillingen) und von dort in die Akademie.

Hinweise:

- Zur beispielhaften Visualisierung kann das in 2.1.1 skizzierte Netz verwendet werden, dort sind die gleichen IP-Adressen eingetragen.
- Unter Linux können die Befehle „ifconfig“ und „traceroute“ verwendet werden.

### 2.1.3 Möglicher Unterrichtablauf

#### Topologie von Rechnernetzen, IP-Adresse, Internet

Ausgehend vom Aufruf einer Webseite wird die Frage untersucht, auf welchem Weg die Information zum Rechner des „Internetnutzers“ gelangt. Dass dazu eine Anforderung vom Clientrechner an den Webserver gesendet werden muss, wird zunächst nicht weiter vertieft. Interessant ist, von wo und über welchen Weg die Daten kommen, die der Browser zur Darstellung einer Webseite benötigt. Im einfachsten Fall handelt es sich um eine Webseitendatei, die auf dem

Webserver eines Providers liegt. Zur Erarbeitung der verschiedenen Stationen auf dem Weg der Webseitendatei wird der Datenweg quasi „rückwärts“ vom Clientrechner aus betrachtet.

Im Unterrichtsgespräch werden mögliche Stationen identifiziert. Befindet sich z. B. der Clientrechner im Computerraum der Schule, so geht der Weg zumindest über den Router der Schule zum Internet Service Provider der Schule und von dort weiter zum Provider, der den Webserver betreibt.

Zur vorläufigen Bestätigung kann nach der einführenden Diskussion über den Befehl „traceroute“ (alternativ „traceroute“ unter Linux) der tatsächliche Weg gezeigt und anhand der dort sichtbaren IP-Adressen die notwendige Adressierung (auch des eigenen Rechners) festgehalten werden. Auch die erfolgte Namensauflösung über eine (nicht sichtbare) Anfrage an einen DNS-Server wird hier thematisiert. Da das Konsolenfenster die IP-Adresse des eigenen Rechners nicht zeigt, wird diese anschließend ermittelt und der Befehl „ipconfig“ besprochen.

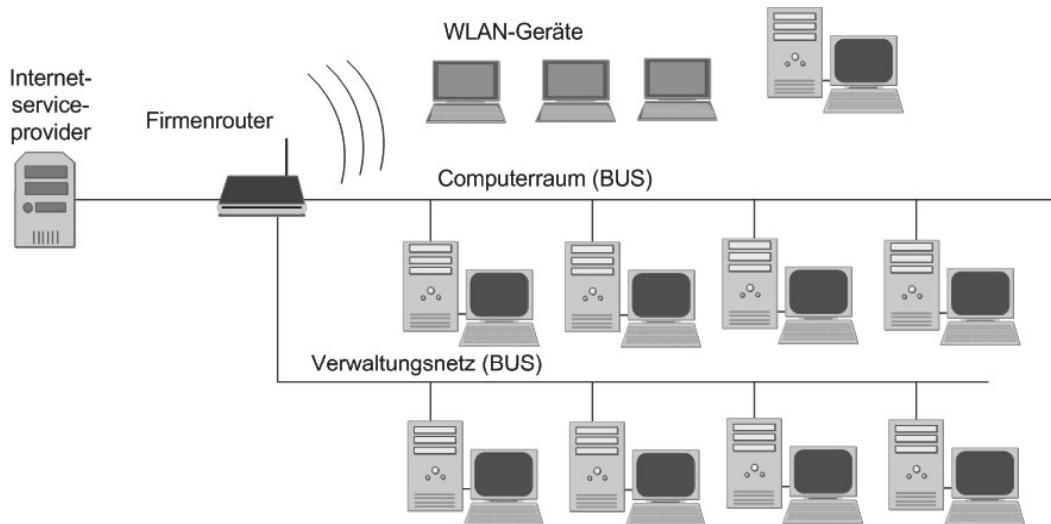
Die Frage nach Performanz (Kollisionsvermeidung und Übertragungsrate) und Ausfallsicherheit (Ausgangspunkt der Entwicklung des Internets war der „Kalte Krieg“) motiviert die Auseinandersetzung mit der netzartigen Struktur des Internets. Ausgehend von einer „naiven“ Vorstellung einer linearen Verbindung bis zum Webserver können Nachteile der dabei entstehenden Strukturen erkannt und diskutiert werden. Die dabei entstehende Skizze enthält bereits die Topologien „Stern“ (Anbindung an der Internet Service Provider) und „Bus“ (typisches Beispiel Computerraum). Die wenig häufige Ringtopologie muss möglicherweise extra motiviert werden.

Folgende Aufgaben können schließlich im Unterricht besprochen werden:

### Aufgabe 1: Entwurf eines Firmennetzwerks

Als Systembetreuer einer Firma sollen Sie für den künftigen Neubau die Vernetzung der Firmenrechner planen. Aus Sicherheitsgründen sollen dort die Teilnetze der Verwaltung und die existierenden Computerräume nicht direkt verbunden sein. Zusätzlich gibt es in den Gängen der Firma Infoterminale, die per WLAN mit dem Firmenrouter verbunden sind. Skizzieren Sie ein mögliches Firmennetzwerk.

### Lösungsvorschlag zu Aufgabe 1:



### Aufgabe 2: Traditionelle Kommunikationswege

In einer Firma gibt es üblicherweise zwei Wege zur Information der Mitarbeiter, die alle ein persönliches Postfach besitzen.

Erster Weg: Beim sogenannten Rundlauf eines Dokumentes wird eine Mitteilung mit einem Deckblatt versehen, auf dem in einer Liste alle Adressaten verzeichnet sind, und in das Postfach des ersten Adressaten gegeben. Dieser bestätigt durch Unterschrift seine Kenntnisnahme und gibt die Mitteilung in das Postfach des nächsten Adressaten.

Zweiter Weg: Den betroffenen Mitarbeitern wird jeweils eine Kopie in ihr Postfach gelegt.

- Diskutieren Sie Vor- und Nachteile beider Vorgehensweisen.
- Vergleichen Sie die Kommunikation von Rechnern, die über einen gemeinsamen Bus verbunden sind, mit dem Rundlauf und zeigen Sie Analogien und Unterschiede auf.

### Lösungsvorschlag:

zu a)

Rundlauf	Kopien
<ul style="list-style-type: none"> <li>- geringer Materialaufwand</li> <li>- Weiterleitung ist mit Aufwand für den Mitarbeiter verbunden</li> <li>- mögliche Unterbrechung, wenn ein Mitarbeiter nicht weiterleitet</li> <li>- höherer Aufwand beim Erstellen der Adressatenliste</li> <li>- höherer Zeitbedarf, bis alle Mitarbeiter informiert sind.</li> <li>- Bestätigung der Kenntnisnahme liegt an Ende vor.</li> </ul>	<ul style="list-style-type: none"> <li>- hoher Materialaufwand</li> <li>- die Information wird sicher und quasi gleichzeitig zugestellt, aber es erfolgt keine Kontrolle der Kenntnisnahme</li> <li>- schnelle Information der Mitarbeiter, falls diese regelmäßig ihr Postfach kontrollieren.</li> </ul>

zu b)

- Auch bei der Bustopologie führt (wie beim Rundlauf) eine Unterbrechung im Übertragungsmedium zum Verlust der Information. Die Rechner hinter der Unterbrechungsstelle sind nicht mehr erreichbar.
- Die Nachricht enthält normalerweise nur einen Empfänger, kann aber von allen Rechnern gelesen werden. Nachrichten an mehrere Rechner müssen mehrfach verschickt werden.

Vertiefender Hinweis: Spezielle Broadcastingverfahren machen es möglich, dass die Nachricht an alle angeschlossenen Rechner adressiert werden kann, ohne die Nachricht in Kopie mehrfach zu verschicken. Dies wird jedoch hier nicht weiter problematisiert.

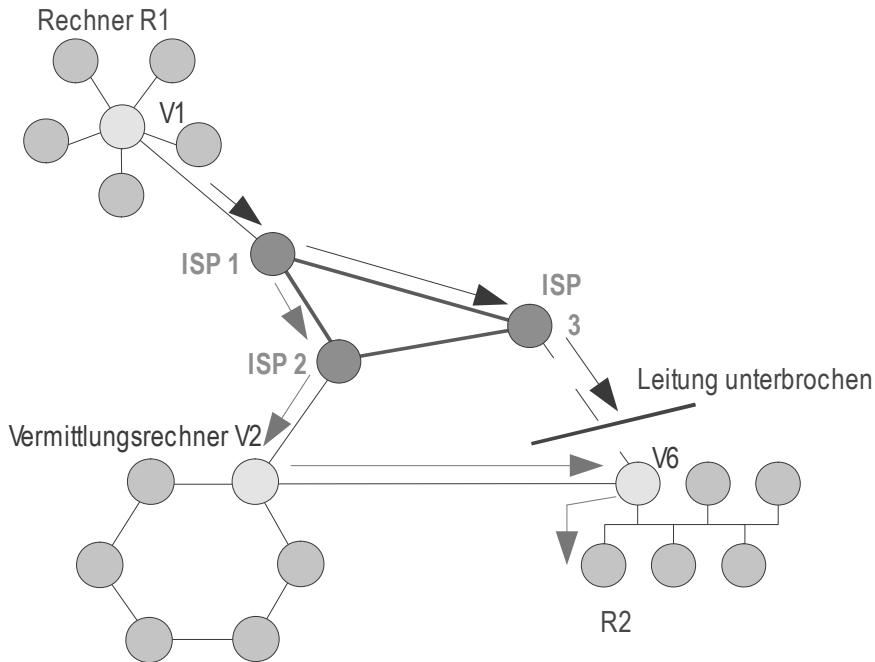
- Damit der sendende Rechner überprüfen kann, ob die Nachricht korrekt zugestellt wurde, bekommt er vom Empfänger eine Bestätigung.
- Mehrere Rundläufe können gleichzeitig stattfinden, die Größe eines Postfachs ist jedoch genauso beschränkt wie die Bandbreite beim Bus.

### Aufgabe 3: Verbindungsaufall

Erläutern Sie anhand einer geeigneten Skizze eines Ausschnitts aus dem Internet, wie Vermittlungsrechner trotz des Ausfalls einer Leitung die Verbindung zweier Rechner ermöglichen.

### Lösungsvorschlag:

In der nachfolgenden Abbildung kann eine Nachricht von Rechner R1 über den Internetserviceprovider 1 (ISP 1) über den Internetserviceprovider ISP 2 und die Vermittlungsrechner V2 und V6 zum Rechner R2 gelangen, falls die Verbindung von Internetserviceprovider ISP 3 zum V6 unterbrochen ist.



## 2.2 Kommunikation zwischen Prozessen

Lp: Aus vielen Bereichen der Computernutzung wie beispielsweise dem Homebanking oder einem Buchungssystem für Reisen ist den Schülern die Notwendigkeit zur Zusammenarbeit mehrerer Computer bekannt. Sie sehen ein, dass zur korrekten Verständigung von Computern spezielle formale Regeln (Protokolle) existieren müssen. Anhand der Kommunikation in Rechnernetzen erfahren die Jugendlichen, dass es sinnvoll ist, Kommunikationsvorgänge in verschiedene, aufeinander aufbauende Schichten aufzuteilen.

Als grundlegendes Beispiel wird ein Chatserver betrachtet. Ausgehend von der Anwendungserfahrung einer Chatsitzung werden die für die Kommunikation zwischen Prozessen notwendigen Bedingungen studiert.

Die Aufgaben im Verlauf des Kapitels können dabei schrittweise zur Implementierung von Chatserver und Chatclient-Programm führen. Eine vollständige Implementierung kann jedoch nur für besonders leistungsstarke Klassen bzw. zur Binnendifferenzierung empfohlen werden. Die Verwendung der beiliegenden Programme und deren Diskussion ist jedoch auch ohne Implementierung möglich.

### Begriffsverwendungen

**Prozess:** Die in der Informatik gebräuchliche Definition eines Prozesses als eines laufenden Programms wird in diesem Kapitel etwas weiter gefasst. Ein Prozess wird allgemein als Abfolge verschiedener Aktivitäten eines Objekts (Person, Webserver usw.) verstanden.

**Server und Client:** Die in diesem Kapitel verwendeten Begriffe „Server“ und „Client“ meinen die Programme, die auf den jeweiligen Rechnern (oder auch auf ein und demselben Rechner) laufen.

Beteiligte Personen werden mit „User“ bezeichnet. Die Computer, auf denen die Programme laufen, werden mit „Server-Rechner“ und „Client-Rechner“ bezeichnet.

## 2.2.1 Dienste, Ports und Protokolle

### 2.2.1.1 Inhaltliche Grundlagen

In diesem Kapitel wird die Kommunikation von beteiligten Prozessen bei den verschiedenen Diensten des Internets untersucht. Die für die einzelnen Dienste typisch genutzten Ports und Protokolle stehen dabei im Vordergrund.

#### Dienste des Internets, Ports

Die weltweite Vernetzung von vielen Computern ist die Grundlage für viele verschiedene Dienste, wie z. B. E-Mail, Dateübertragung über FTP (File Transfer Protokoll), Telefonieren über VOIP (Voice over IP). Auch das World Wide Web, kurz WWW, ist ein Dienst des Internets, obwohl man umgangssprachlich diesen Dienst oft als „das Internet“ bezeichnet. Dieser Internetdienst erlaubt die Kommunikation eines Clientrechners unter Verwendung eines Browsers mit einem Webserver, der Webseiten ausliefert.

Um zwischen Rechnern gleichzeitig mehrere Verbindungen aufbauen zu können, müssen die einzelnen Verbindungen vom Kommunikationspartner identifiziert werden können. Dies geschieht über sogenannte Ports. Dabei handelt es sich um Nummern, die zusätzlich zur Adresse angegeben werden. Die Portnummer kann mit der Angabe einer Zimmernummer oder einer Abteilung einer Firma zusätzlich zur Briefadresse verglichen werden. So findet beispielsweise durch Angabe „Abteilung Kundenbetreuung Zi. 08.15“ der Brief den Weg zum richtigen Sachbearbeiter und wird dem Kundenbetreuungsdienst der Firma zugeordnet.

Für die verschiedenen Dienste des Internets werden üblicherweise festgelegte Ports verwendet. Typische Beispiele zeigt folgende Tabelle:

Port	Dienst
20,21	Dateitransfer (Datentransfer vom Server zum Client und umgekehrt)
25	E-Mail-Versand
80	Webserver
110	E-Mail-Abruf
443	Webserver mit verschlüsseltem Zugang
3306	Zugriff auf MySQL-Datenbanken

#### Protokolle

Zur Kommunikation sind zusätzlich zur Verbindung der Rechner via Internet Protokolle nötig, die das Datenformat und den Ablauf einer Kommunikation zwischen zwei Rechnern festlegen. Eine gängige Definition ist beispielsweise:

In der Informatik und in der Telekommunikation ist ein **Protokoll** eine Vereinbarung, nach der die Kommunikation zwischen zwei Parteien abläuft. Ein Protokoll ist eine formale Sprache, bei der die Syntax auch den Ablauf der Kommunikation definiert.

Hinweis: Eine formale Sprache umfasst sowohl die Syntax als auch die Semantik der Kommunikation (siehe Kapitel 1.2.7).

Die Beschreibung des Datenformats eines Protokolls kann mit der erweiterten Backus-Naur-Form (EBNF, siehe Kapitel 1.3.2), die Beschreibung eines beispielhaften Ablaufs mit einem Sequenzdiagramm erfolgen.

Hinweis: Im Zusammenhang mit dieser grundlegenden Einführung des Begriffs wird bewusst noch nicht auf das Schichtenmodell eingegangen. Die Thematik wird in Kapitel 2.2.4 ausführlich behandelt, dabei werden auch schichtenspezifische Protokolle diskutiert.

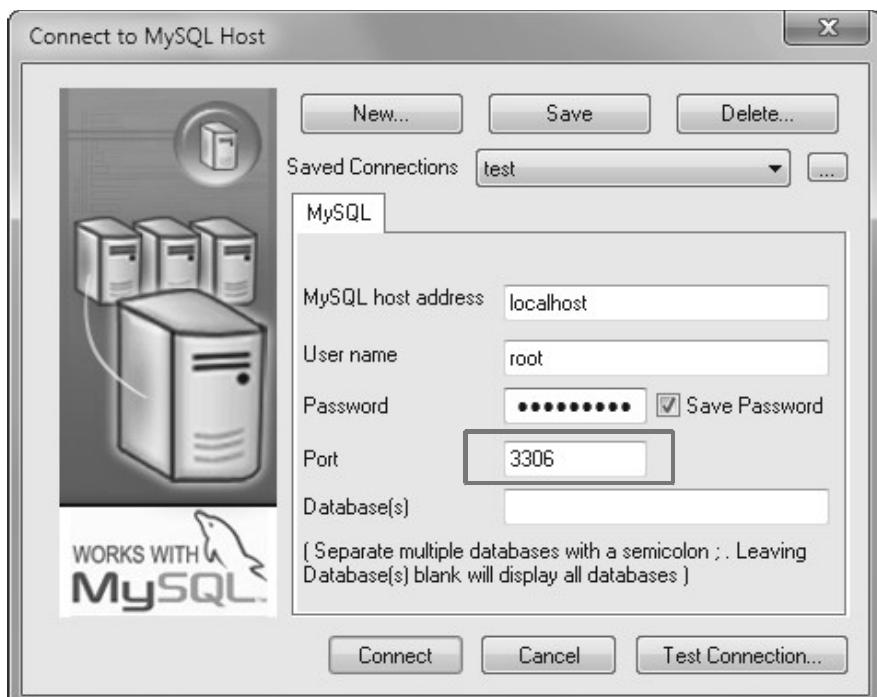
Der bereits angesprochene Vergleich mit der Kundenbetreuungsabteilung einer Firma kann an dieser Stelle erweitert werden:

Dienst	World-Wide-Web	Kundenbetreuung
Adressierung	IP-Adresse	Briefadresse
Port	80	Adresszusatz, z. B. Zimmernummer
Protokoll	HTTP	Die benötigten Daten werden durch ein Formular erfasst. Das Verfahren wird durch die Allgemeinen Geschäftsbedingungen der Firma geregelt (Nachbesserungsfristen, Garantiezeit), sodass der Dienst von jedem Sachbearbeiter nach bestimmten Regeln ausgeführt wird.

### 2.2.1.2 Umsetzungshinweise

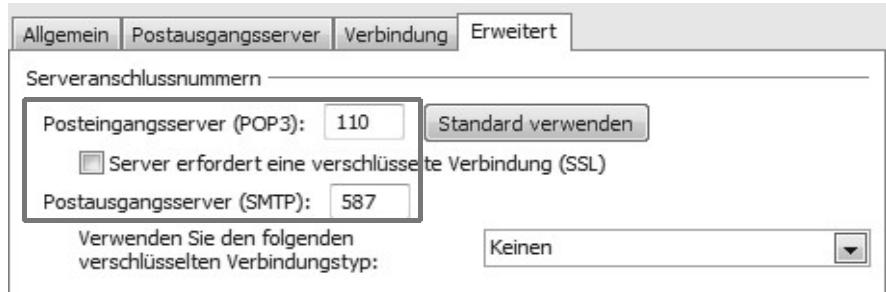
Die Kenntnis der typisch verwendeten Ports der verschiedenen Dienste des Internets ist beim Konfigurieren der entsprechenden Programme hilfreich. Beispiele entsprechender Einstellungen könnten sein:

- Ein SQL-Client, wie er möglicherweise auch schon in der Jahrgangsstufe 9 beim Thema Datenbanken verwendet wurde:

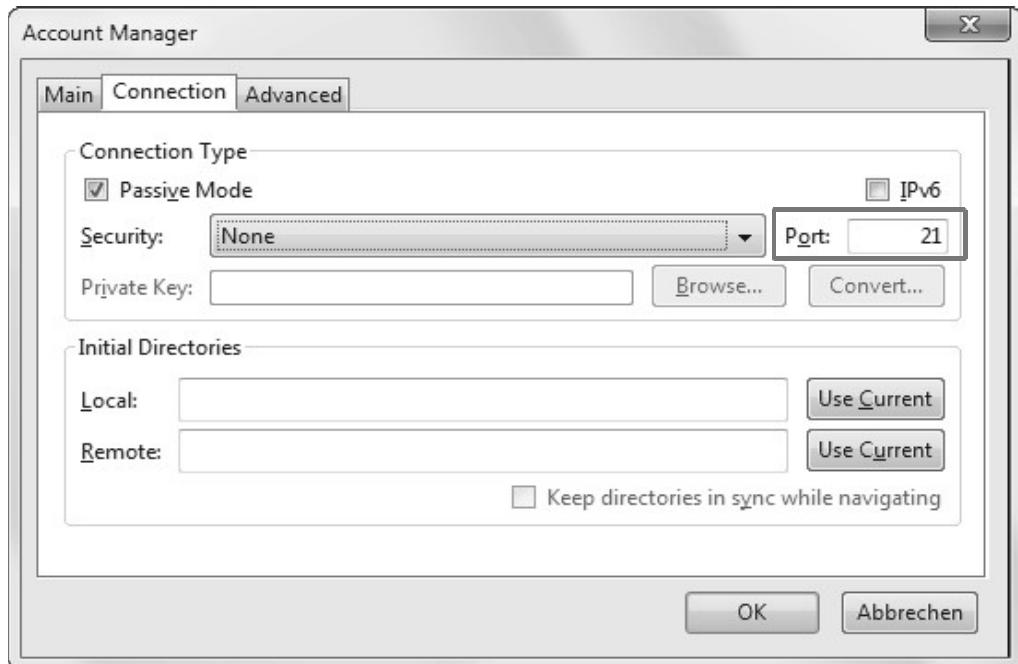


- Ein E-Mail Programm:

Hinweis: In diesem Beispiel ist abweichend vom typischen Port 25 für die Verbindung zum Postausgangsserver der Port 587 eingetragen, da der Port 25 im Firmennetzwerk durch eine Firewall gesperrt ist.



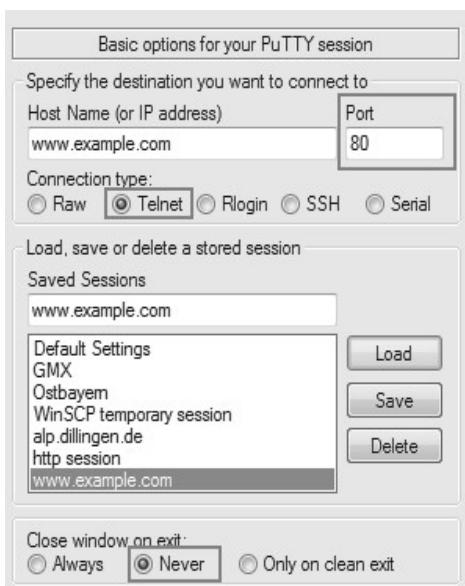
- Ein Programm zum Übertragen von Dateien mittels FTP:



### Telnet-Client Putty als Werkzeug zur Untersuchung eines Webservers

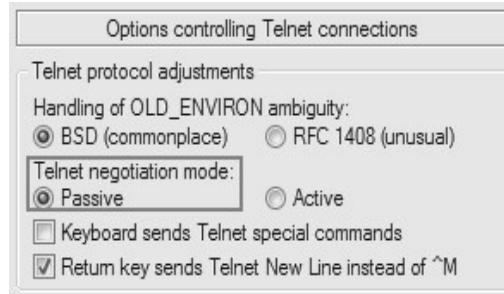
Zur Untersuchung des HTTP-Protokolls ist das Programm telnet.exe, das unter Windows standardmäßig zur Verfügung steht, nur bedingt geeignet. Bei Verwendung von telnet.exe sind die eigenen Eingaben per Voreinstellung im Konsolenfenster nicht sichtbar. Es wird deshalb empfohlen, als Telnet-Client das Programm „Putty“ zu verwenden, das kostenlos für Windows und Linux verfügbar ist (Download unter: <http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html>). „Putty“ kann zudem die Verbindungsdaten abspeichern, was das erneute Verbinden zum WWW-Server erleichtert.

Für den Webseitenauftruf von „www.example.com“ unter Verwendung von Putty sind folgende Schritte nötig:

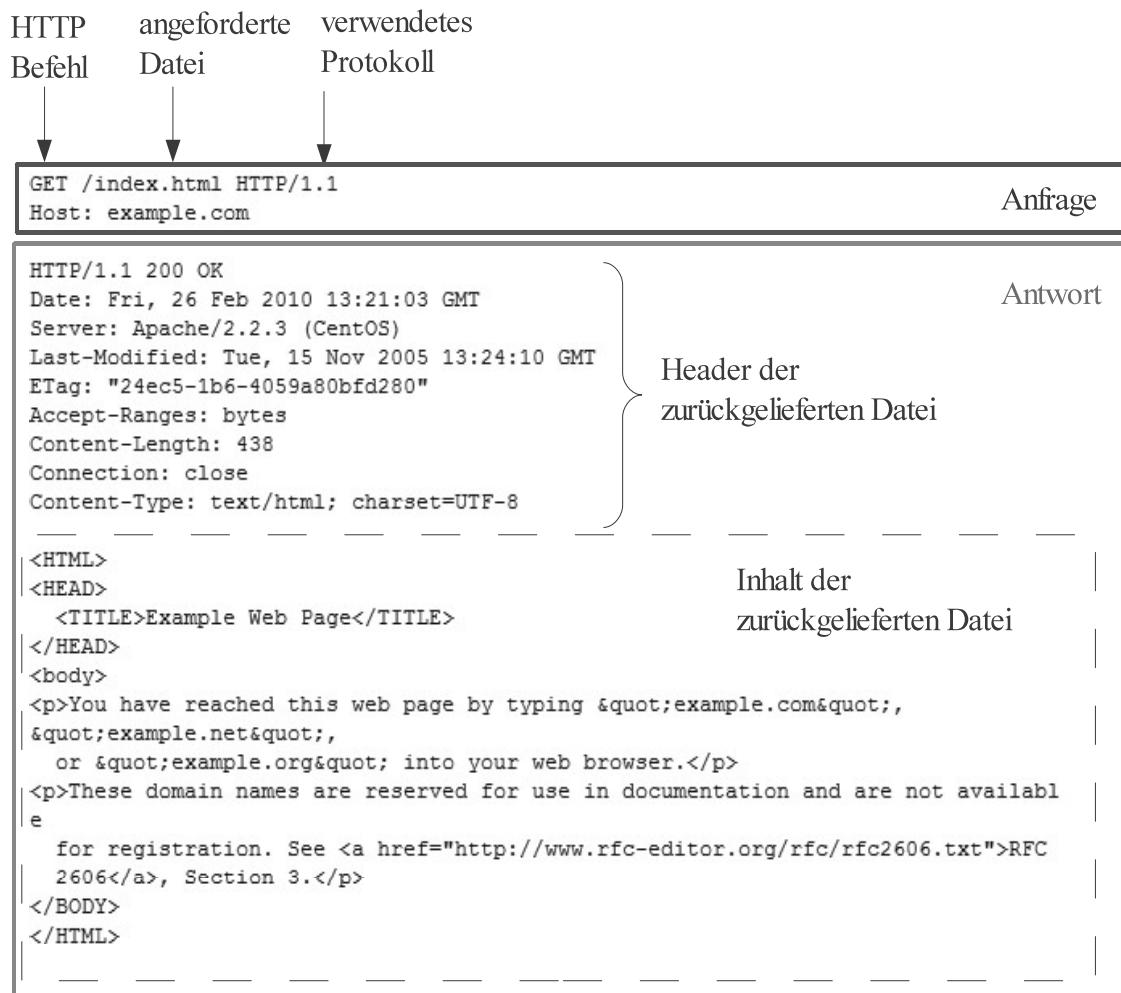


Nachdem das Programm gestartet ist, kann man die Verbindungsdaten eingeben.

Hinweis: Je nach verwendeten Router kann es nötig sein, dass in der Abteilung „Telnet“ noch der **Passive Modus** eingeschalten werden muss, damit die Authentifizierung korrekt erfolgen kann.



Nach dem Öffnen der Verbindung fordert man eine Webseite mit dem GET-Befehl an:



Die Antwort des Webservers besteht aus einem Header mit Informationen über den Sendezzeitpunkt, Webserversoftware und über das nachfolgende Dokument (Länge, Codierung usw.) sowie dem eigentlichen Inhalt der Webseite.

### Kommunikation mit einem Mailserver via Telnet

Zur Untersuchung des Protokolls bei E-Mail kann das Programm „putty“ oder das Programm „telnet“ verwendet werden.

1. Zur Herstellung einer Verbindung zum E-Mail-Server gibt man im Konsolenfenster (bei Windows cmd.exe aufrufen) den Befehl telnet <Adresse des Pop3-Servers> 110 ein.
2. Nun können verschiedene Anweisungen ausgeführt werden. Die folgende Tabelle zeigt einige Möglichkeiten:

POP3-Anweisung	Beschreibung
USER username	wählt den Benutzernamen „username“ bzw. das Benutzerkonto auf dem E-Mail-Server.
PASS passwort	übergibt das Passwort „passwort“ in Klartext.
QUIT	beendet die Verbindung zum POP3-Server . Alle gespeicherten Änderungen werden dann ausgeführt (Zum Beispiel der Befehl DELE).
LIST (n)	listet alle E-Mails auf. liefert die Anzahl und die Größe der (n-ten) E-Mail(s).
STAT	liefert den Status der Mailbox, u.a. die Anzahl aller E-Mails im Postfach und deren Gesamtgröße (in Byte).
RETR n	liefert die n-te E-Mail aus.
DELE n	löscht die n-te E-Mail.
NOOP	Dieser Befehl dient zur Aufrechterhaltung der Verbindung. Der Server würde sonst die Verbindung nach einem bestimmten Zeitintervall trennen.
RSET	setzt die aktive Verbindung zurück (setzt alle DELE Kommandos zurück).

### Beispiel: Abrufen einer E-Mail

```
+OK GMX POP3 StreamProxy ready
USER max.muster@gmx.de
+OK May I have your password, please?
PASS 74jfzsj389
+OK Mailbox locked and ready
LIST
+OK
1 1745
2 1016496
3 19406
.
RETR 1
+OK
Return-Path:
.....(technische Informationen zur E-mail.....)

From: "Andreas Wagner"
To: max.muster@gmx.de
Subject: test
Mime-Version: 1.0
Content-Type: text/plain; charset=UTF-8
Content-Transfer-Encoding: 8bit
Content-Disposition: inline
X-GMX-Antivirus: 0 <no virus found>
X-GMX-Antispam: 0 <Mail was not recognized as spam>;
Detail=5D7Q89H36p6i75npGen84eVAEFLK/syJmQWFUJhD4BPL42vD+2ZuqoFSikZQiMYiOsSe7p
SYKeehYeY/pC/61vSJiYuLlK/gn7NmaJC9RKxsrIbx8cvvG23XQhj1AgqqgYmh0p58QrY0lhI1457
m3/Gw==U1;
X-GMX-UID: d3KUeHxxPTRtAtoUMzIwIQYxc2tpZAvB

Hallo Andi
.
QUIT
+OK GMX POP3 server signing off
```

### 2.2.1.3 Möglicher Unterrichtsablauf

#### Dienste des Internets, Ports

Zunächst werden im Unterrichtsgespräch die verschiedenen Nutzungsmöglichkeiten des Internets gesammelt und anschließend den jeweiligen Diensten des Internets zugeordnet. Aus eigener Erfahrung wissen die Schülerinnen und Schüler, dass man z. B. gleichzeitig im World Wide Web surfen, E-Mails empfangen und eine VOIP-Schaltung nutzen kann. Damit dies möglich ist, haben die Datenpakete, die den Clientrechner erreichen, eine zusätzliche Information zur Unterscheidung des Kommunikationskanals – die Portnummer, die in den Einstellungen der jeweiligen Clientprogramme entsprechend konfiguriert werden kann. In der Grundeinstellung sind dort meist die typischen Portnummern (siehe Tabelle in 2.2.1.1) voreingestellt. Für einige ausgewählte Programme werden diese Einstellungen gesucht, notiert und den jeweiligen Diensten zugeordnet.

#### Protokolle

Am Beispiel eines Chatserver (p10\_chatserver) wird die Kommunikation zwischen mehreren Rechnern demonstriert und die Notwendigkeit von Protokollen für diese Kommunikation erarbeitet.

Nach einer Chatsitzung mit der gesamten Klasse werden folgende Fragen thematisiert und damit die Begriffe IP-Adresse, Port, Server und Client vertieft:

- Welche Rechner sind beteiligt?  
Die Rechner der Chatteilnehmer und der Rechner, auf dem der Chatserver läuft.
- Wie unterscheiden sich die Rollen, die die Rechner innerhalb der Chatsitzung spielen?  
Aufgabe des Chatprogramms am sogenannten Clientrechner (kurz Chatclient) ist das Senden der vom Chatteilnehmer eingegebenen Botschaft zum Serverrechner und das Empfangen von Botschaften vom Serverrechner.

Das Chatprogramm am Serverrechner (kurz Chatserver) hält pro Chatclient je eine Verbindung, empfängt Botschaften von einem Chatclient und schickt diese an einen oder mehrere Clients weiter.

- Auf welchem Weg gelangt die Botschaft eines Chatteilnehmers zum Ziel?  
Es besteht keine direkte Verbindung zwischen den Clients. Botschaften werden über den Chatserver zum jeweiligen Adressaten gesandt.
- Welche Informationen sind dazu nötig?  
Im vorliegenden Beispiel ist vereinbart, dass eine Botschaft an alle Chatteilnehmer verteilt wird, in diesem Fall genügt der Text der Botschaft.
- Wieso müssen zu Beginn der Sitzung Verbindungsdaten eingegeben werden und wozu dienen diese Informationen?  
Diese Informationen dienen dazu, die Verbindung zum Chatserver aufzubauen. Die IP-Adresse legt den Rechner fest, der als Chatserver fungiert. Die Vereinbarung eines freien Ports ermöglicht eine Verbindung, auch wenn noch weitere Dienste des Internets am Server- oder Clientrechner genutzt werden.

Die unterschiedlichen Rollen (Client- und Serverrechner) werden bereits beim Programmstart deutlich. Um eine Verbindung zum Server aufzubauen, muss die IP-Adresse des Servers bekannt sein und man muss sich auf einen (freien) Port einigen, über den die Kommunikation stattfinden soll. Diese Informationen müssen nach Programmstart (p10\_starteChatclient.bat) eingegeben werden (☒ p10\_chatserver).

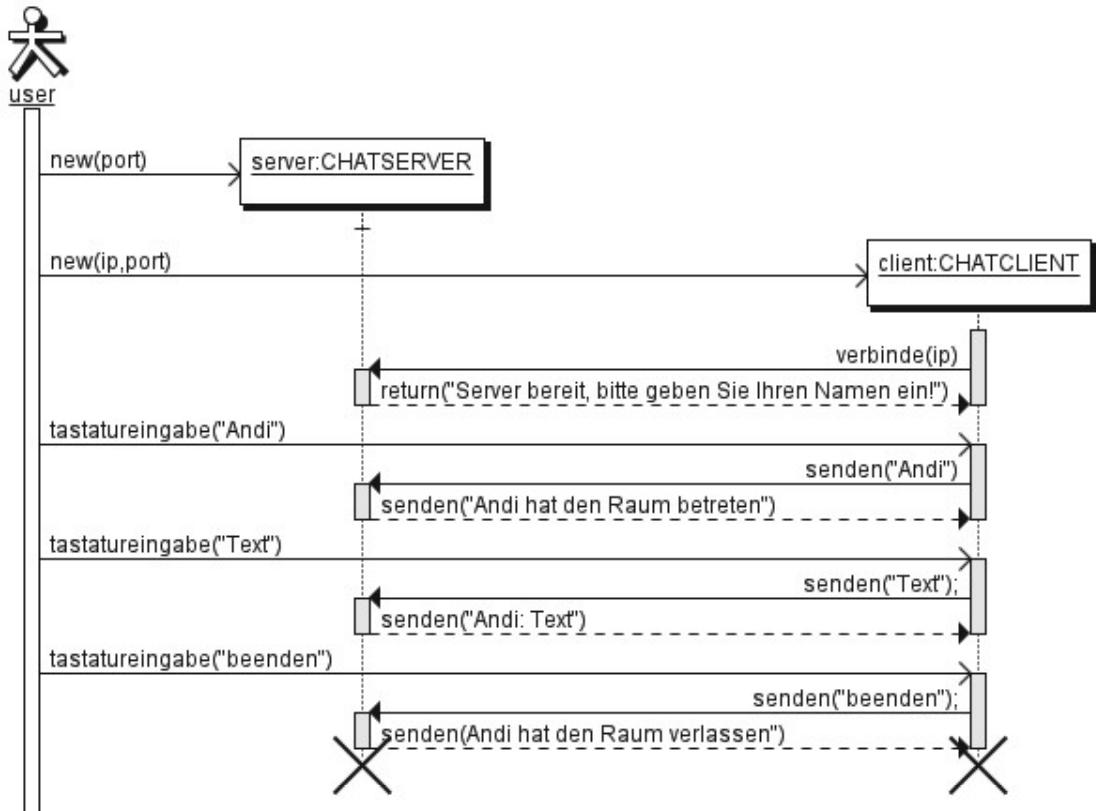
Die Definition des Begriffs Protokoll wird durch die Bearbeitung der nachfolgenden Aufgaben erarbeitet.

### Aufgabe 1: Beschreiben Sie den Ablauf einer Chatsitzung

- a) in Worten;
- b) mithilfe eines Sequenzdiagramms.

### Lösungsvorschläge zu Aufgabe 1:

- a) Beschreibung des Ablaufs einer Chatsitzung
  - Starten des Clients (Eingabe von IP-Adresse des Servers und Portnummer)
  - Anmelden am Server mit Namen
  - Bestätigung des Servers
  - Senden einer Mitteilung, Empfangen von Mitteilungen
- b) Darstellung einer möglichen Kommunikation mithilfe eines Sequenzdiagramms:



## Aufgabe 2:

Beschreiben Sie die verwendeten Daten und deren Formate

- in Worten
- unter Verwendung der EBNF

für den vorliegenden Chatserver

### Lösungsvorschlag zu b)

Anmeldung:

```

Ziffer = '0' | '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9';
Portnummer = Ziffer [ Ziffer [ Ziffer [ Ziffer [ Ziffer ] ] ] ];
Zahl = Ziffer [ Ziffer [ Ziffer ] ];
IP-Adresse = Zahl '.' Zahl '.' Zahl '.' Zahl;
Zeilenende = '<CR>';

```

Mitteilung:

```

Zeichen = 'A' .. 'Z' | 'a' .. 'z';
Zeichenkette = Zeichen { Zeichen };
Clientname = Zeichenkette;
Botschaft = Zeichenkette;
Mitteilung = Botschaft Zeilenende;
Befehl = 'beenden';

```

Hinweis: Nach dieser EBNF wäre auch eine Adresse wie 456.999.999.111 gültig. Bei solchen Adressen kommt eine Verbindung nicht zustande. Eine Einschränkung auf gültige IP-Adressen würde eine umfangreichere EBNF erfordern.

Die Aufgaben verdeutlichen, dass zur Beschreibung der Kommunikation zwischen zwei Prozessen (insbesondere zweier Programmabläufe) die Beschreibung der verwendeten Datenformate und des

Kommunikationsablaufs gehört. In einem Protokoll werden entsprechende Regeln zur Kommunikation festgelegt. Ausgehend von einer Besprechung der Aufgaben und gegebenenfalls der Untersuchung eines Seitenaufrufs von „www.example.com“ oder der Lösung der Aufgaben zur Chatsitzung wird die Definition eines Protokolls formuliert.

Am Beispiel der Kommunikation mit einem Mailserver via Telnet werden die gefundene Definition und die Begriffe „Datenformat“ und „Kommunikationsablauf“ vertieft.

### Aufgabe 3: Datenformate und Ablauf bei E-Mail

Beschreiben Sie die verwendeten Datenformate und den Ablauf beim Abrufen einer E-Mail.

#### Lösungsvorschlag:

##### Datenformate:

```

Zeichen = 'A' .. 'Z';
Zeichenkette = Zeichen { Zeichen };
Ziffer = '0' | '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9';
Nummer = Ziffer { Ziffer };
Username = Zeichenkette;
Passwort = Zeichenkette;
Emailadresse = Zeichenkette '@' Zeichenkette '.' Zeichenkette;
Befehle = 'USER' Username | 'PASS' Passwort | 'QUIT' |
          'LIST' [Nummer] | 'STAT' | 'RETR' Nummer | 'DELE' Nummer |
          'NOOP' | 'RSET';
  
```

#### Ablauf:

User	Mailserver
Verbindungsdaten zum Mailserver in Clientprogramm eingeben	Mailserver bereit
Username eingeben	Username akzeptiert
Passwort eingeben	Passwort ist akzeptiert, Mailbox ist bereit
Auflisten der vorhandenen E-Mails mit „LIST“	gibt Liste der E-Mails aus
Die erste E-Mail in der Liste mit „RETR 1“ anfordern	Mailserver liefert E-Mail aus
Die Verbindung durch Eingabe von „QUIT“ beenden.	Bestätigung, dass Verbindung beendet wird und Verbindung beenden.

#### Mögliche weitere Aufgaben:

1. Welche Protokolle nutzen die Ihnen bekannten Dienste des Internets?

#### Lösungsvorschlag:

Dienst	Protokoll
Dateitransfer (Datentransfer vom Server zum Client und umgekehrt)	FTP
E-Mail-Versand	SMTP
Webserver	HTTP
E-Mail-Abruf	POP3
Webserver mit verschlüsseltem Zugang	HTTPS

2. Im Bereich der Diplomatie wird oft auch der Begriff Protokoll verwendet. Welche Unterschiede und Gemeinsamkeiten lassen sich im Vergleich zur Definition eines Protokoll in der Informatik finden?

Lösungsvorschlag:

Das Protokoll im Bereich der Diplomatie umfasst auch Vereinbarungen, die nicht unmittelbar der Kommunikation der beteiligten Parteien dienen. So sind z. B. über die Regeln der verbalen Kommunikation hinaus Kleiderordnung, Tischordnung und Rangordnungen genau festgelegt.

## 2.2.2 Umsetzung einer Client-Server-Kommunikation

In diesem Kapitel wird eine erste Client-Server-Kommunikation betrachtet. Dabei soll durch die Untersuchung (bzw. wahlweise auch Implementierung) von Protokollen bisher Gelerntes vertieft werden. Die Analyse bzw. der Test des ersten naiven Implementierungsversuchs eines Servers und dessen Verbesserung zeigt die Notwendigkeit einer Parallelisierung auf und motiviert so das nächste Kapitel. Die Diskussion kann unter Verwendung der beiliegenden Javaprogramme in Abhängigkeit vom Leistungsstand der Klasse auch völlig ohne Implementierungen durch Schülerinnen und Schüler erfolgen (Hinweise an jeweiliger Stelle).

### 2.2.2.1 Umsetzungshinweise

Die vorliegende JAVA-Implementierung ( p01\_wiegehts) der ersten Client-Server-Kommunikation verwendet die drei Klassen SERVER, CLIENT und SERVERVERHALTEN, um ein einfaches Gesprächsprotokoll zu ermöglichen.

#### Server- und Clientprozess im Pseudocode für das vorliegende Beispiel

Unabhängig von der verwendeten Programmiersprache gliedert sich eine erste naive Implementierung des Serverprozesses und des Clientprozesses im vorliegenden Beispiel in wenige Teile, die im Pseudocode formuliert werden können:

##### Serverprozess

Starten des Servers

Warten auf eine Clientverbindung und bei Verbindungsaufnahme Herstellen der Verbindung  
tue

    Client-Botschaft lesen;  
    Antwort ermitteln und senden;  
    solange Antwort nicht gleich „Auf Wiedersehen!“

Beenden der Clientverbindung

Stoppen des Servers

##### Clientprozess

Verbindung zum Server herstellen

wiederhole solange gelesene Server-Botschaft nicht gleich „Auf Wiedersehen“

    neue Botschaft von Tastatur lesen und an Server senden

Endewiederhole

Beenden der Verbindung zum Server

## Zustandsdiagramm des Servers

Die Antwort des Servers auf die Clientbotschaft wird im Objekt der Klasse SERVERVERHALTEN ermittelt, die das Zustandsdiagramm (siehe Kapitel 2.2.2.2) der Kommunikation abbildet. Dadurch bleibt der Code übersichtlich und spätere Änderungen am Serververhalten können leicht als Zusatzaufgabe implementiert werden.

## Verwendung der JAVA-Klassen ServerSocket und Socket

Um mit dem Server kommunizieren zu können, verwendet das Clientprogramm die Klassen BufferedReader zum Lesen des eingehenden Datenstroms und PrintWriter zur Ausgabe der Daten zum Server. Zur Erstellung dieser Objekte wird im Konstruktor als Parameter der Output- bzw. Inputstream eines Objekts der Klasse Socket benötigt, die in Java eine bidirektionale Schnittstelle für die Netzwerkkommunikation bereitstellt.

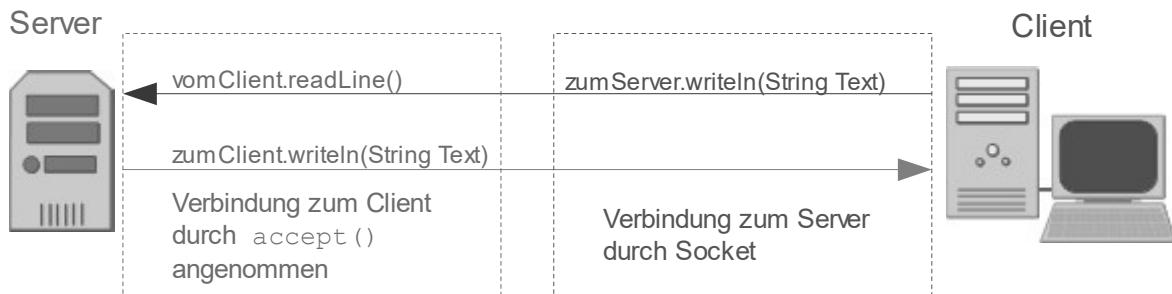
### Verbindung zum Server im Clientprogramm:

```
//Verbindung über Port 4444 zur IP-Adresse „192.168.178.22“
clientSocket = new Socket(„192.168.178.22“, 4444);
//Objekte zum Lesen und Schreiben über die Verbindung erzeugen
zumServer =new PrintWriter(clientSocket.getOutputStream(), true);
vomServer =new BufferedReader(new InputStreamReader(clientSocket.getInputStream()));
```

Die Verbindung zum Server wird durch den Aufruf des Socketkonstruktors hergestellt, falls im Netz auf dem Rechner mit der angegebenen IP-Adresse ein Programm auf eine eingehende Verbindung über den angegebenen Port wartet und dieser Port noch frei ist. Im Serverprogramm wird dazu die Klasse ServerSocket verwendet. Die Methode *accept()* eines Objekts der Klasse ServerSocket wartet auf die eingehenden Verbindungen unter der beim Aufruf des Konstruktors festgelegten Portnummer. Nimmt der Client nun eine Verbindung auf, so liefert *accept()* den Clientsocket zurück, mit dessen Hilfe die jeweiligen Datenströme für In- und Output genutzt werden können.

### Verbindung zum Client im Serverprogramm:

```
serverSocket = new ServerSocket(4444); //ServerSocket auf Port 4444 erzeugen
clientSocket = serverSocket.accept(); //auf die Verbindung warten
//Objekte zum Lesen und Schreiben über die Verbindung erzeugen
zumClient =new PrintWriter(clientSocket.getOutputStream(), true);
vomClient =new BufferedReader(new InputStreamReader(clientSocket.getInputStream()));
```



## 2.2.2.2 Praktische Umsetzung

Die Schülerinnen und Schüler untersuchen eine einfache Client-Server-Kommunikation. Dabei wird das zugrunde liegende Protokoll herausgearbeitet und anschließend untersucht, wie solche Protokolle implementiert werden können. Die strengen Regeln, nach denen das Gespräch zwischen Server und Client abläuft, werden zunächst in Form eines Zustandsdiagramms für den Server verdeutlicht und eine Serversitzung exemplarisch in einem Sequenzdiagramm festgehalten.

Beispielsweise in Zweierteams aufgeteilt erhalten die Schülerinnen und Schüler dazu folgende Aufgabe:

**Aufgabe 1: Das Projekt „Hallo, wie geht's?“** ( p01\_wiegehts)

Für diese Aufgabe sind zwei verschiedene Rechner nötig.

- a) Ermitteln Sie die IP-Adresse der beiden Rechner.
- b) Starten Sie auf einem der beiden Rechner das Programm p01\_starteServer.bat und wählen Sie einen Port, über den die Kommunikation stattfinden soll.
- c) Starten Sie auf dem anderen Rechner das Programm p01\_starteClient.bat und geben Sie die notwendigen Informationen zur Verbindung zum Server ein.
- d) Erkunden Sie die Arbeitsweise des Servers und stellen diese in einem Zustandsdiagramm dar.
- e) Zeichnen Sie ein beispielhaftes Sequenzdiagramm für eine Sitzung.
- f) Welche Vereinbarungen wurden im vorliegenden Beispiel getroffen bzw. welche Regeln müssen, angefangen vom Start der Programme p01\_starteServer.bat und p01\_starteClient.bat, eingehalten werden, damit ein Gespräch geführt werden kann?

Nach der Untersuchung der Kommunikation bei „Hallo, wie geht's?“ wird der beiliegende Implementierungsvorschlag ( p01\_wiegehts) mit den Jugendlichen unter Berücksichtigung der in obiger Aufgabe gestellten Fragen besprochen. Dabei kann eine Formalisierung des Ablaufs in Pseudocode zum Codeverständnis beitragen. Die in Java vorhandenen Socketklassen und die beteiligten Streams werden erläutert. Die beiden Input-Streams *tastatur* und *vomServer* in der Klasse CLIENT müssen dabei genau unterschieden werden.

Besondere Beachtung verdient die Implementierung des Serververhaltens (Gesprächsregeln) in der Klasse SERVERVERHALTEN, die die modellierten Zustände des Servers repräsentiert. In Hinblick auf eine flexible Erweiterung des Serververhaltens wird diese Kapselung empfohlen. Die Klassen SERVER und CLIENT bleiben dadurch in den nächsten Programmbeispielen im Wesentlichen unverändert.

Hinweis: Die Gesprächsregeln sind Teil des Protokolls auf der Anwenderschicht. Dass Protokolle schichtenspezifisch sind, wird jedoch erst im Kapitel 2.2.3 diskutiert.

**Lösungsvorschläge und Erläuterungen zu Aufgabe 1:**

Im Projekt „Wie geht's“ ( p01\_wiegehts) wird eine einfache Kommunikation zwischen einem Server und *einem* Client implementiert. Die Implementierung der Gesprächsregeln als Teil des Protokolls erfolgt in der Klasse SERVERVERHALTEN.

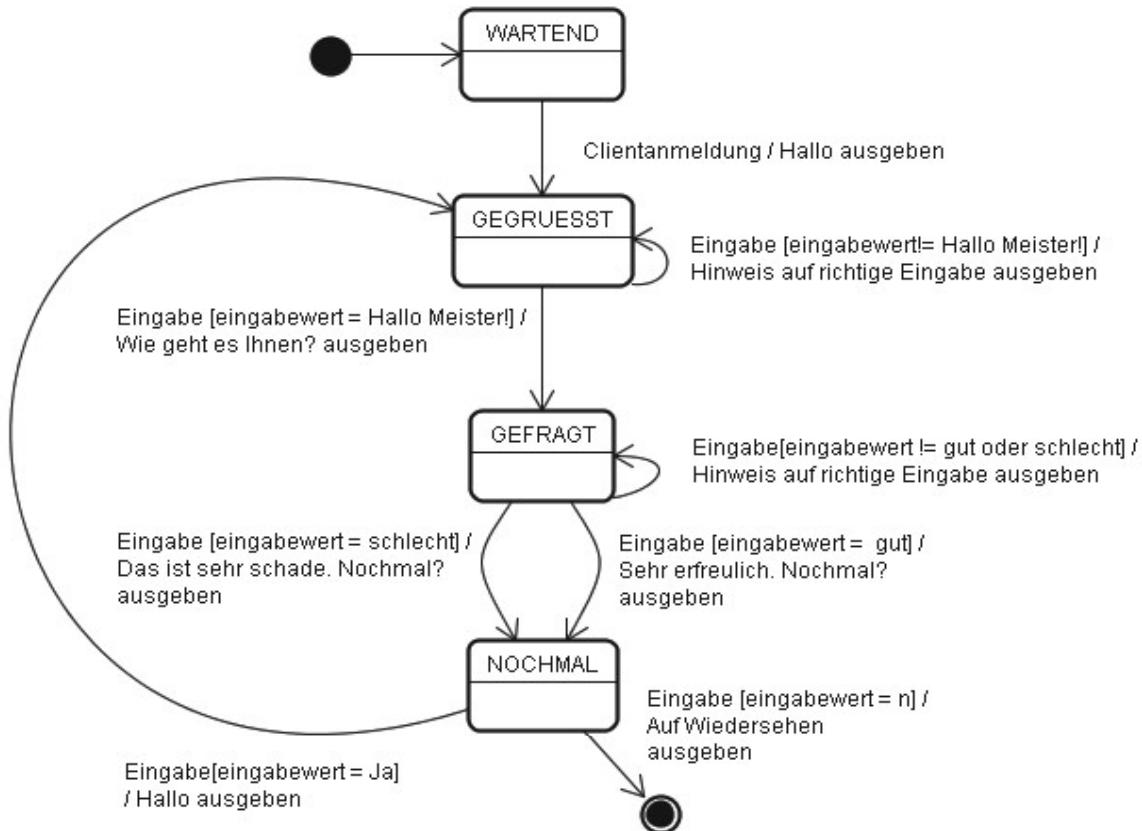
Folgende Klassen werden verwendet:

SERVER	SERVERVERHALTEN
private String clientBotschaft private String serverAntwort	private static int wartend private static int gegruesst private static int gefragt private static int nochmal private int zustand
void SERVER() void main()	public String HoleAntwort(textvomClient)
private void ServerStarten() private void ServerStoppen() private void ClientVerbindungStarten() private void ClientVerbindungBeenden()	

CLIENT
private String clientEingabe private String serverBotschaft
void CLIENT() void main()
private void VerbindungHerstellen() private void VerbindungBeenden()

Die Klasse SERVERVERHALTEN implementiert das folgende Zustandsdiagramm:



Das Protokoll umfasst im vorliegenden Beispiel mehr als die eigentliche Unterhaltung. Angefangen von der Verbindungsaufnahme (im vorliegenden Fall gehört bereits die Angabe von IP-Adresse und

Port-Nummer beim Start des Client-Programms bzw. des Serverprogramms dazu) über die geführte Unterhaltung bis hin zum Beenden des Programms sind Syntax, Semantik und Ablauf geregelt.

## Datenformate

EBNF für das Beispiel „Wie geht's“

```

Ziffer = '0' | '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9';
Portnummer = Ziffer [ Ziffer [ Ziffer [ Ziffer [ Ziffer [ Ziffer ] ] ] ];
Zahl = Ziffer [ Ziffer [ Ziffer ] ];
IP-Adresse = Zahl '.' Zahl '.' Zahl '.' Zahl;
Zeilenende = '<CR>';
Mitteilung = Zeichenkette Zeilenende;
Servertext = 'Hallo!' | 'Wie geht es Ihnen?' | 'Sie muessen mich mit "Hallo
Meister!" begruessen! Probieren Sie es nochmal!' | 'Sehr erfreulich. Nochmal
(Ja)?' | 'Das ist sehr schade. Nochmal (Ja)?' | 'Sie koennen nur mit "gut"
oder "schlecht" antworten. Wie geht es Ihnen?' | 'Auf Wiedersehen!';
Servermitteilung = Servertext Zeilenende;
```

## Ablauf

Nach einer korrekten Anmeldung durch einen geregelten Start von Server und Client (auch die Reihenfolge ist vorgegeben und ist Teil des Protokolls!) kann das Gespräch zwischen User und Server stattfinden. Die Gesprächsregeln sind sehr genau festgelegt. Nachdem der Server vom Client mit „Hallo Meister!“ begrüßt wurde (auf andere Eingaben reagiert der Server mit einem Hinweis), stellt der Server die Frage „Wie geht es Ihnen?“ Die zugelassenen Antworten „gut“ oder „schlecht“ kommentiert der Server und fragt, ob die Kommunikation beendet werden soll. Auf alle andere Antworten reagiert der Server wiederum mit einem Hinweis.

## Implementierung der Gesprächsregeln

Die jeweilige Antwort des Servers wird in der Methode *HoleAntwort()* des SERVERVERHALTEN-Objekts des Servers ermittelt. Die Klasse SERVERVERHALTEN implementiert dazu einen Automaten mit den Zuständen *wartend*, *gegruesst*, *gefragt* und *nochmal*.

## Zustandsdiagramm des Servers

Implementierung der entsprechenden Zustände in der Methode *HoleAntwort* der Klasse SERVERVERHALTEN:

```

public String HoleAntwort(String textvomclient) {
String ausgabe = null;

switch (zustand) {

    case wartend:
        ausgabe = "Hallo!";
        zustand = gepruesst;
        break;

    case gepruesst:
        if (textvomclient.equalsIgnoreCase("Hallo Meister!")) {
            ausgabe = "Wie geht es Ihnen?";
            zustand = gefragt;
        } else {
            ausgabe = "Sie muessen mich mit \"Hallo Meister!\" begruessen!
                    + "Probieren Sie es nochmal!";
        }
        break;

    case gefragt:
        if (textvomclient.equalsIgnoreCase("gut")) {
```

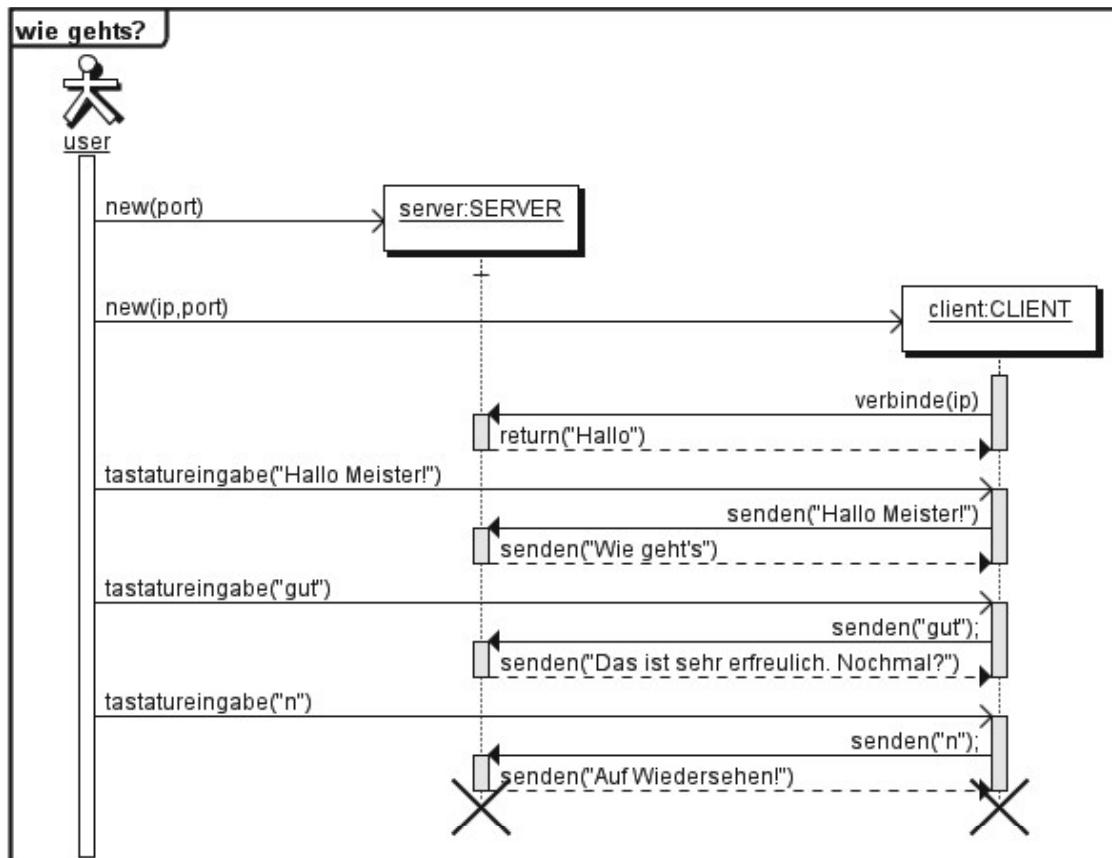
```

        ausgabe = "Sehr erfreulich. Nochmal (Ja)?" ;
        zustand = nochmal;
    } else if (textvomclient.equalsIgnoreCase("schlecht")) {
        ausgabe = "Das ist sehr schade. Nochmal (Ja)?" ;
        zustand = nochmal;
    } else {
        ausgabe = "Sie koennen nur mit \"gut\" oder \"schlecht\" antworten. "
            + "Wie geht es Ihnen?" ;
    }
    break;

case nochmal:
    if (textvomclient.equalsIgnoreCase("Ja")) {
        ausgabe = "Hallo!";
        zustand = gegruesst;
    } else {
        ausgabe = "Auf Wiedersehen!";
        zustand = wartend;
    }
    break;
}
return ausgabe;
}

```

### Beispielhafter Ablauf einer Sitzung:



Nach der Besprechung verwenden die Schülerinnen und Schüler den vorhandenen Quellcode als Grundlage für einige kleine Änderungen. Es bietet sich an, die veränderte Implementierung der Gesprächsregeln im Zustandsdiagramm zu dokumentieren.

Je nach Leistungsstand der Klasse können die notwendigen Quelltextänderungen entweder im Unterrichtsgespräch ausgehend von einem veränderten Zustandsdiagramm diskutiert oder von den Schülerinnen und Schülern selbst implementiert werden.

### Aufgabe 2: Veränderung des Protokolls

Verwenden Sie den Quellcode des Projekts p01\_wiegehts, um folgende Veränderungen im Programm zu implementieren:

- a) Das Verhalten des Servers soll so verändert werden, dass auf die Frage „Wie geht es Ihnen“ auch die Antwortoption „geht so“ zugelassen wird, auf die der Server mit „das hört sich nicht so toll an“ antwortet. Verändern Sie das Zustandsdiagramm entsprechend und passen Sie den Quellcode der Klasse SERVERVERHALTEN an.
- b) Nach der Antwort auf die Frage „Wie geht es Ihnen“ soll die Frage „Heute schon gelacht?“ mit den Antwortoptionen „ja“ und „nein“ eingefügt werden. Verändern Sie die Implementierung der Klasse SERVERVERHALTEN und diskutieren Sie die notwendigen Veränderungen im Zustandsdiagramm.

**Lösungshinweis:**  p02\_wiegehts2

Zur zusätzlichen Vertiefung eignet sich die Aufgabe der Implementierung einer Wetterauskunft. Auch hier bleiben die Implementierungen der Klassen SERVER und CLIENT im Wesentlichen noch unverändert. In der Klasse WETTERVERHALTEN werden jedoch zur Auffrischung der Java-Kenntnisse mehr Anforderungen gestellt. Sie muss komplett neu implementiert werden.

### Aufgabe: Implementierung einer Wetterauskunft ( p03\_wetterauskunft)

Ein Wetterauskunftsterminal meldet sich mit „Bereit, bitte geben Sie Ihren Namen ein“, falls ein Client die Verbindung aufgenommen hat. Nach Eingabe des Namens *name* wird der Client mit „Hallo *name*“ begrüßt und nach einer Stadt gefragt. Für die Städte „Mailand“, „Madrid“, „Berlin“, „London“ liegen Wetterdaten vor (in der Simulation wird eine Wetterlage zufällig ausgewählt). Für alle anderen Städte antwortet der Server „Für diese Stadt liegen leider keine Wetterdaten vor!“ Anschließend fragt der Server, ob die Verbindung geschlossen werden soll oder die Wetterdaten einer anderen Stadt ermittelt werden sollen.

Entwerfen Sie ein Zustandsdiagramm für den Server und setzen Sie das Serververhalten nach obiger Beschreibung in der Klasse WETTERVERHALTEN um.

**Lösungshinweis:**  p03\_wetterauskunft

## 2.2.3 Schichtenmodell

Lp: Anhand der Kommunikation in Rechnernetzen erfahren die Jugendlichen, dass es sinnvoll ist, Kommunikationsvorgänge in verschiedene, aufeinander aufbauende Schichten aufzuteilen.

### 2.2.3.1 Inhaltliche Grundlagen

#### Schichtenmodell

Mithilfe eines Schichtenmodells kann man komplexe und aufwendige Arbeitsabläufe strukturieren, die bei der Kommunikation zwischen Prozessen auftreten.

Die Idee eines Schichtenmodells besteht darin, dass jede Schicht ihre spezielle Aufgabe hat und die Schichten grundsätzlich voneinander unabhängig sind, dabei aber über definierte Schnittstellen miteinander kommunizieren.

Jede Schicht kommuniziert über ein **schichtenspezifisches Protokoll** mit der entsprechenden Schicht auf dem anderen System (**logischer Datenfluss**), indem es Daten an die darunterliegende Schicht weiterleitet (**physikalischer Datenfluss**) bzw. von ihr erhält.

Ein einfaches grundlegendes Schichtenmodell für die Kommunikation kann folgendermaßen aussehen:

Schicht	Aufgabe
Anwendungsschicht	Nutzung des Anwendungsprogramms
Transportschicht	Zuverlässige Übertragung, Vollständigkeit der Daten
Vermittlungsschicht	Weitervermittlung von Paketen und die Wegewahl (Routing)
Netzzugangsschicht	Technik zur Datenübertragung

Die genaue Kenntnis der Schichtung des im Bereich der Netzwerkkommunikation häufig verwendeten ISO/OSI Modells oder des TCP/IP-Protokollstapels ist laut Lehrplan nicht gefordert. Die verwendeten Protokolle und die Aufgaben der einzelnen Schichten können aber durchaus im Rahmen von Beispielaufgaben besprochen werden.

Vertiefende Hinweise: In den Beispielimplementierungen in Java (z.B. p10\_chatserver) umfasst das Protokoll neben den Regeln zur Verbindungsaufnahme (richtiger Start des Programms, Anmeldung) auch Festlegungen in Bezug auf die Datenübertragung. Die Java-Klassen ServerSocket und Socket implementieren das sogenannte TCP/IP-Protokoll und garantieren damit die vollständige und fehlerfreie Übertragung der Daten in der richtigen Reihenfolge. Im Gegensatz dazu ist z.B. beim UDP-Protokoll (User Datagram Protocol), das häufig zur Übertragung von Bild und Ton eingesetzt wird, die Übertragung der Datenpakete in der richtigen Reihenfolge nicht zugesichert. Kommt z.B. bei der Übertragung eines Videos ein Datenpaket, das zu einem früheren Zwischenbild gehört, zu spät an, so wird es einfach verworfen, da es bei einer solchen Übertragung wichtiger ist, den zeitlichen Ablauf eines Films korrekt wiederzugeben, als dass etwa einige Einzelbilder fehlen.

### 2.2.3.2 Möglicher Unterrichtsablauf

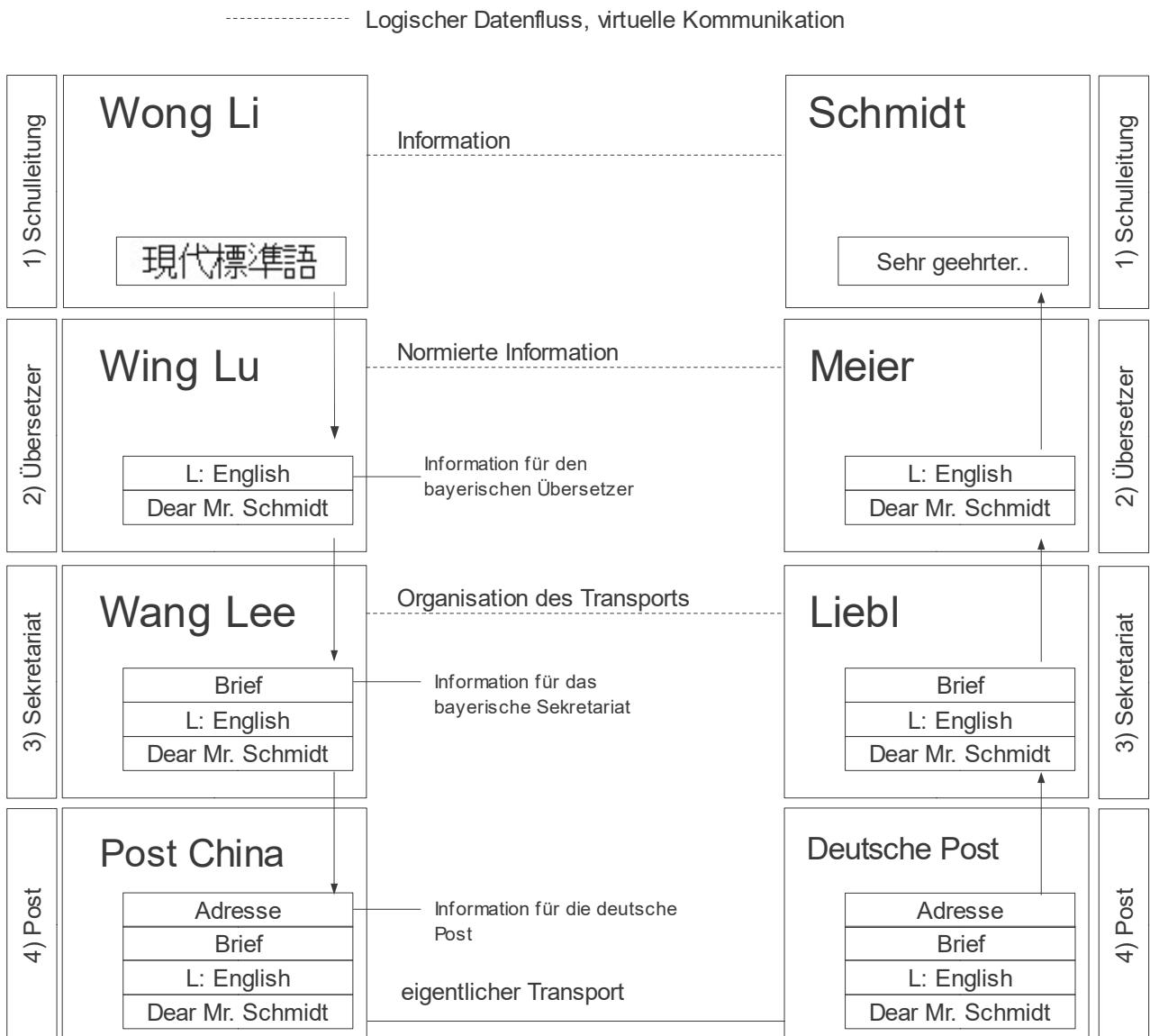
Ausgehend von folgender Einstiegsaufgabe wird ein erstes Modell einer Kommunikation in Schichten entwickelt:

#### Aufgabe 1: E-Mail-Kommunikation

Im Rahmen eines Schüleraustauschprojekts will die Schulleiterin Wong Li einer japanischen Schule mit dem Schulleiter Herrn Schmidt eines bayerischen Gymnasiums kommunizieren. Glücklicherweise verfügen beide Schulleiter über Kollegen, die die jeweilige Landessprache ins Englische übersetzen können. Deren Übersetzungen werden über das Sekretariat per Post an die jeweilige Schule versandt.

- Stellen Sie in einem Diagramm die jeweiligen Kommunikationspartner gegenüber und beschreiben Sie die Regeln, nach denen die Partner kommunizieren.
- Über welche Stationen gelangt eine Nachricht von Frau Wong an Herrn Schmidt? Welche Fähigkeiten müssen dabei die beteiligten Personen mitbringen, welche Informationen geben sie an den jeweiligen Adressaten weiter?

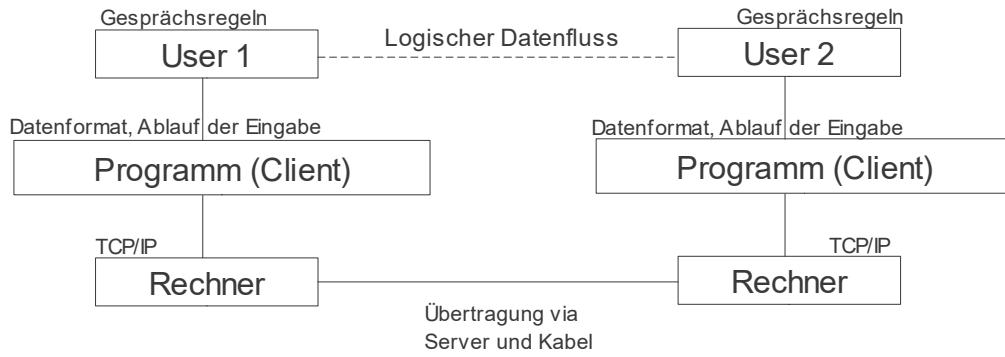
Die Diskussion über die beiden Teilaufgaben könnte zu folgendem Modell führen:



Es lassen sich in diesem Beispiel vier Schichten erkennen. Jede Schicht erledigt dabei eine ihr fest zugewiesene Aufgabe und stellt entsprechend festgelegte Schnittstellen zu den Schichten über und unter ihr bereit. Vom eigentlichen Weg der Daten braucht die Schicht dabei nichts zu wissen. Eine Schicht dieses sogenannten Protokollstapels kommuniziert in einem **schichtspezifischen Protokoll** mit der entsprechenden Schicht auf dem anderen System (**logischer Datenfluss**), indem es Daten an die darunter liegende Schicht weiterleitet (**physischer Datenfluss**) bzw. von ihr erhält.

Nach der Besprechung der Aufgabenlösung kann das Schichtenmodell allgemein definiert werden.

Zur Vertiefung wird die Chatkommunikation unter dem Aspekt des Schichtenmodells untersucht.



Im Unterrichtsgespräch werden die beteiligten Schichten erarbeitet und ihre jeweiligen Aufgaben identifiziert. Der Aspekt der schichtenspezifischen Protokolle wird durch die Gegenüberstellung von „Gesprächsregeln“ als Teil des Protokolls der Anwenderschicht und des TCP/IP-Protokolls, mit dem die Rechner miteinander kommunizieren, deutlich. Im vorliegenden Beispiel beinhaltet das Protokoll auf oberster Ebene die Gesprächsregeln (Netiquette), die beim Chatten zu beachten sind. Auf Ebene des Programms (ebenfalls Anwendungsschicht) sind Datenformate und Eingabeablauf geregelt. Auf der untersten Schicht sorgt ein entsprechendes Protokoll für die zuverlässige und fehlerfreie Übertragung der Daten (Transport-, Vermittlungs- und Netzzugangsschicht).

Hinweis: Für die Erläuterung des TCP/IP-Protokolls reicht es aus, die Aufgaben zu erarbeiten, die für eine fehlerfreie Übertragung nötig sind (Transport, Vermittlung, Übertragung). In der Aufgabe zu TCP/IP-Prokollstapel kann dieses Wissen noch vertieft werden. Auf die Einzelheiten des Protokolls braucht nicht eingegangen zu werden.

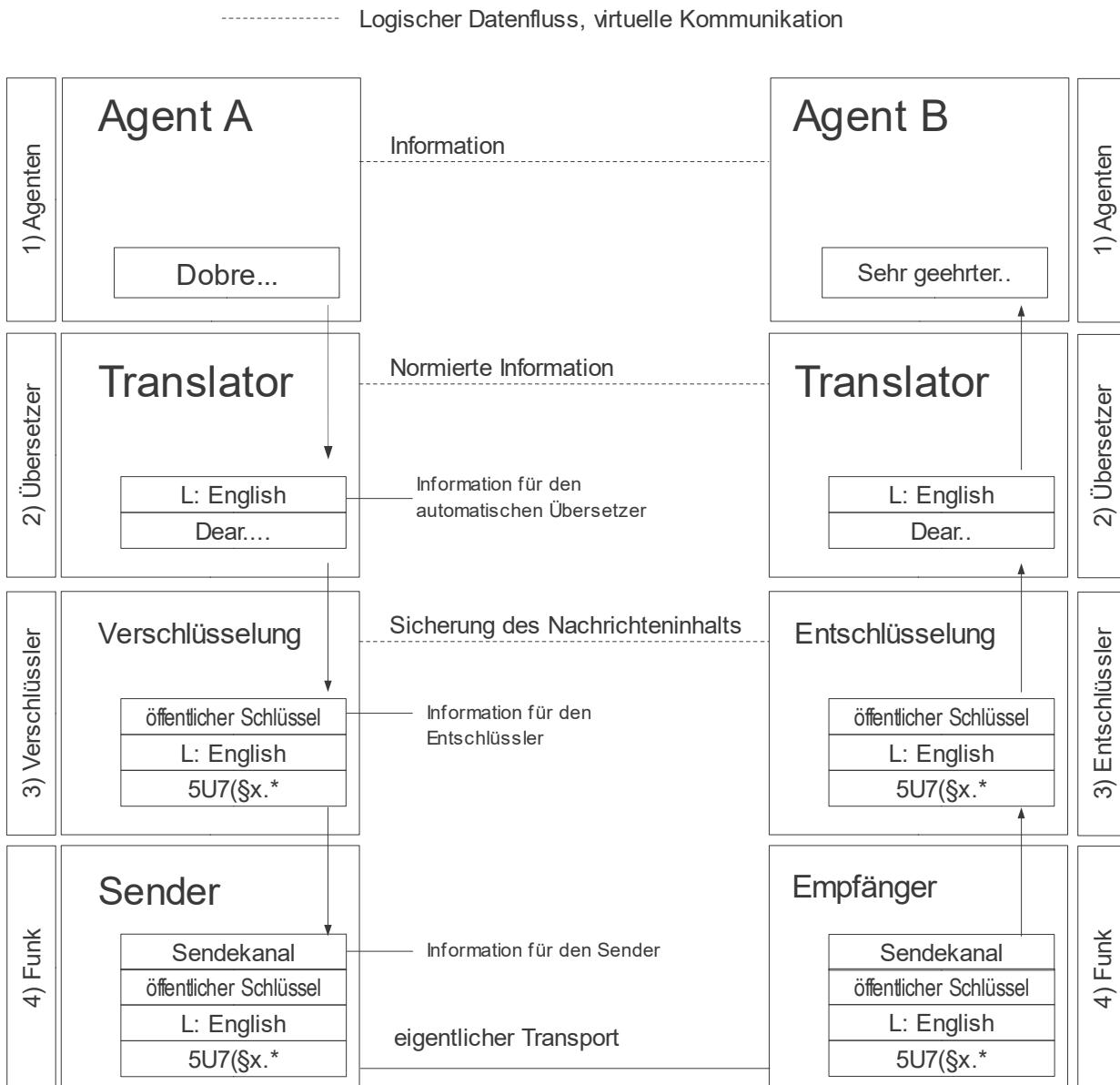
### Mögliche weitere Aufgaben:

#### Aufgabe: Verschlüsselte Kommunikation

Zwei Geheimagenten verschiedener Länder kommunizieren über abhörsichere Kommunikationsgeräte. Die eingegebene Nachricht wird automatisch ins Englische übersetzt und verschlüsselt übertragen.

Geben Sie ein Schichtenmodell mit (mindestens) 4 Schichten für diesen Kommunikationsablauf an und erklären Sie anhand dieses Modells beispielhaft den Ablauf der Kommunikation zwischen den Partnern.

#### Lösungsvorschlag:



### Aufgabe: TCP/IP-Protokollstapel

Ein bewährtes Schichtenmodell zur Beschreibung von Kommunikation innerhalb von Rechnernetzen ist der sogenannte TCP/IP-Protokollstapel. Recherchieren Sie, welche Schichten zur Beschreibung der Kommunikation verwendet werden. Welche Aufgaben haben dabei die einzelnen Schichten? Nennen Sie Protokolle, die auf den jeweiligen Schichten verwendet werden.

### Lösungsvorschlag:

Schicht	Aufgabe	Protokolle
Anwendungsschicht	Protokolle, die mit Anwendungsprogrammen zusammenarbeiten	HTTP, FTP, SMTP, POP, Telnet
Transportschicht	Zuverlässige Übertragung, Vollständigkeit der Daten	TCP

<b>Vermittlungsschicht</b>	Weitervermittlung von Paketen und die Wegewahl (Routing)	IP (Ipv4, IPv6)
<b>Netzzugangsschicht</b>	Technik zur Datenübertragung	Ethernet, Token Bus, Token Ring, FDDI

## 2.3 Modellierung einfacher paralleler Prozesse

Lp: Die Schüler erkennen z. B. bei der Analyse der Aufgaben eines Mailservers, dass Rechner anfallende Aufträge oft parallel bearbeiten und dennoch einen geordneten Ablauf gewährleisten müssen.

### 2.3.1 Inhaltliche Grundlagen

Im folgenden wird unter „Prozess“ ein ablaufender Teil eines Programms verstanden. Laufen mehrere Programme oder Teilprogramme gleichzeitig, so spricht man von **parallelen Prozessen**.

Um Wartezeiten des Prozessors zu nutzen, können bei allen Betriebssystemen mehrere Prozesse parallel ausgeführt werden. Man unterscheidet dabei zwischen der parallelen Ausführung von Programmen (schwergewichtige Prozesse) und von Programmteilen (leichtgewichtige Prozesse). Der Wechsel zwischen leichtgewichtigen Prozessen (sogenannten Threads) erfolgt sehr schnell. Der Zugriff auf gemeinsame Ressourcen ist möglich.

Hinweis: In Abhängigkeit des Prozessors und des jeweiligen Betriebssystems laufen Prozesse oft „quasiparallel“ ab. Die Nutzung des Prozessors durch die Programme erfolgt sequentiell, die Zuteilung von Prozessorzeit an die laufenden Prozesse erfolgt jedoch wechselweise unter Verwendung betriebssystemspezifischer Schedulingstrategien, so dass der Eindruck entsteht, Programme oder Programmteile würden parallel laufen. Im folgenden Kapitel werden solche Prozesse unabhängig von der Umsetzung im jeweiligen Betriebssystem kurz als „parallel“ bezeichnet.

Unter **nebenläufigen Prozessen** versteht man parallele Prozesse, die sich möglicherweise gegenseitig beeinflussen, weil sie auf gemeinsame Ressourcen zugreifen.

Hinweis: In der Literatur werden die Begriffe „nebenläufig“ und „parallel“ oft nicht klar unterschieden bzw. in unterschiedlicher Bedeutung verwendet.

### 2.3.2 Umsetzungshinweise

Mit der Klasse Thread kann in Java ein leichtgewichtiger paralleler Prozess innerhalb eines Programms leicht realisiert werden. Dazu muss die Methode *run()* einer eigenen von Thread abgeleiteten Klasse implementiert werden.

Beispiel:

```
class MeinThread extends Thread // damit lassen sich eigene
                               // Thread-Objekte erstellen
{
    // Nötige Attribute, Konstruktor, Hilfsmethoden etc.
    public void run () // Hier beginnt der Thread seine Arbeit
    {
        // Arbeitsauftrag des Threads
    }
}

// Erzeugen und aktivieren des Threads
Thread t = new MeinThread (); // Das Objekt wird angelegt
t.start (); // Der Thread wird in die Verwaltung aufgenommen; sobald
           // er an der Reihe ist, wird mit Ausführung
           // der Methode run begonnen
```

### 2.3.3 Möglicher Unterrichtsablauf

In den bisherigen Versionen der Klasse SERVER kann sich immer nur ein Client mit dem Server verbinden, da am Server nur ein Prozess für den (exklusiven!) Client aktiv ist.

In diesem Kapitel wird die Implementierung der Client-Server-Kommunikation fortgesetzt. Der Server wird weiter verbessert, so dass auch mehrere Clients parallel bedient werden können. In der nachfolgenden Aufgabe werden sich die Schülerinnen und Schüler dieses Problems bewusst. Es kann auch nur Teilaufgabe 1a) und 1d) unter Verwendung des fertigen Programms ([p04\\_mehrclients\\_loesung](#)) bearbeitet werden, wenn man auf Programmieraufgaben verzichten möchte.

Beim Test des Programms „Wetterauskunft“ in der nachfolgenden Aufgabe erkennen die Schülerinnen und Schüler, dass eine parallele Bearbeitung von Anfragen zum Wetter notwendig ist, und machen sich in kleinen Implementierungsaufgaben mit dem Quelltext des Programms vertraut. Die Implementierungen bereiten zudem die Parallelisierung des Serverprozesses vor, die anschließend im Unterrichtsgespräch gemeinsam durchgeführt bzw. studiert werden kann.

#### Aufgabe 1: Mehrere Clients ([p04\\_mehrclients\\_aufgabe](#))

Für diese Aufgabe sind drei verschiedene Rechner nötig. Arbeiten Sie in Gruppen.

- a) Testen Sie den Wetterauskunftsserver ([p03\\_wetterauskunft](#)) unter Verwendung von drei Rechnern. Starten Sie dazu auf einem Rechner das Serverprogramm (p03\_starteServer.bat) und verbinden Sie sich über einen weiteren Rechner mit der Wetterauskunft. Was passiert, wenn sich ein zweiter Client anmelden möchte? Erläutern Sie die Problematik mithilfe eines Sequenzdiagramms.
- b) Die vorliegende Implementierung „Wetterauskunft“ soll so verändert werden, dass der Client durch die Eingabe von „beenden“ jederzeit beendet werden kann. Die Methode *HoleAntwort()* soll dazu unabhängig vom Zustand die Anfrage des Clients mit dem Stopp-Signal „Server [stopClient]:“ beantworten. Verwenden Sie dazu die Quellcodevorgabe ([p04\\_mehrclients\\_aufgabe](#))
- c) Ergänzen Sie den Quellcode der unvollständig implementierten Klasse SERVER2 ([p04\\_mehrclients\\_aufgabe](#)) an den gekennzeichneten Stellen und zeichnen Sie ein Klassendiagramm.
- d) Testen Sie das Programmverhalten Ihrer Implementierung, wenn sich zunächst nacheinander zwei Clients (client1 und client2) anmelden und anschließend client1 beendet wird.

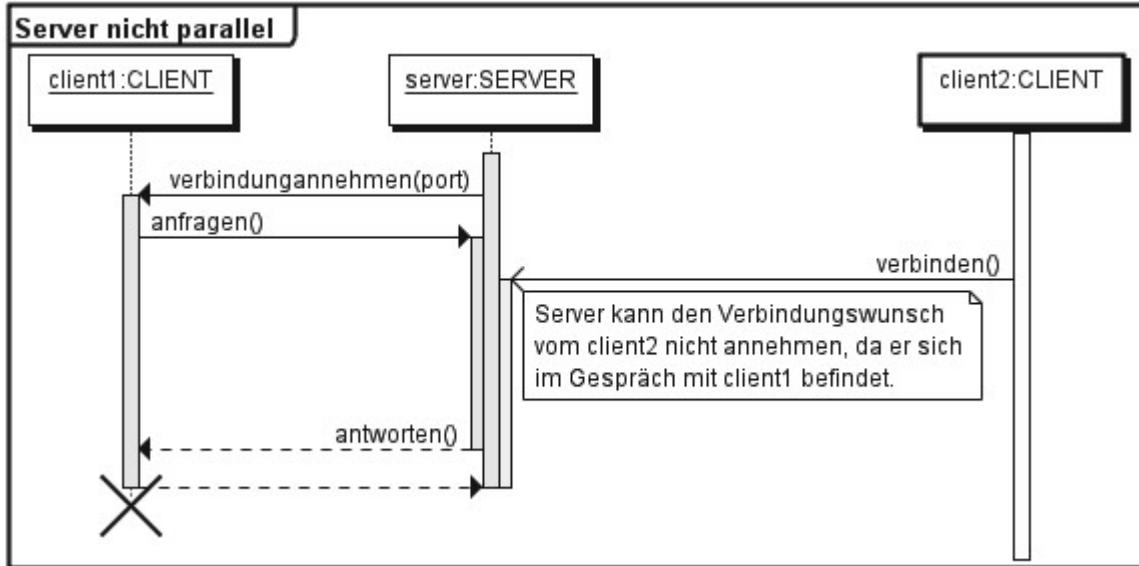
#### Lösungshinweise: ([p04\\_mehrclients\\_loesung](#))

zu a) Ein zweiter Client kann sich beim Programm „Wetterauskunft“ nicht mit dem Server verbinden, weil der Server die Verbindungen nicht parallel bearbeiten kann. Da beim Beenden des verbundenen Clients auch der Server beendet wird, scheitert die Verbindung des zweiten Clients, auch wenn der verbundene Client die Verbindung freigibt.

zu d) Durch das implementierte Stoppsignal wird die Clientverbindung beendet, ohne dass das Serverprogramm beendet wird. Ein Clientprogramm, das auf eine Verbindung wartet, kann nun die Verbindung zum Server aufnehmen.

Hinweis: Bei allen nachfolgenden Implementierungen soll künftig die Serververbindung des Clients beendet werden, falls die empfangene Serverbotschaft mit der Zeichenfolge „Server [stopClient]“ beginnt.

Die Schülerinnen und Schüler erkennen schnell, dass es darauf ankommt, dass der Server nach einer Verbindungsannahme sofort wieder für eine weitere Verbindung zur Verfügung steht, da es sonst zu langen Wartezeiten kommt, weil die Clientanfragen nur nacheinander abgearbeitet werden können.



Die eigentliche Arbeit erledigt der Server innerhalb der Wiederholung im Konstruktor und ist währenddessen nicht für eine neue Verbindung bereit. Die zu wiederholende Sequenz muss also künftig ausgelagert sein. Alle Attribute und Methoden(-teile), die einem Client zugeordnet sind, werden dazu in einer Klasse CLIENTPROZESS gekapselt.

Als Aufgabe oder im Unterrichtsgespräch erfolgt der Umbau des Servercodes. Im Quelltext der Klasse SERVER (p04\_mehrclients\_loesung) sollen die Schülerinnen und Schüler zunächst die clientspezifischen Elemente (im Lösungsvorschlag gelb) markieren. Die Variable *zustand* und das WETTERVERHALTEN-Objekt darf dabei nicht vergessen werden. Das Warten auf einen neuen Client wird in die Methode *AufNeuenClientWarten()* übertragen. Bei diesen Überlegungen wird auch deutlich, warum die unterschiedlichen Aufgabenteile (Annehmen und Verwalten der Verbindungen, Arbeit mit den Verbindungen) auf zwei Klassen aufgeteilt werden sollten.

Der (teilweise) bearbeitete Quelltext der Klasse SERVER könnte so aussehen:

- Alle markierten Zeilen (gelb) werden in die Klasse CLIENTPROZESS übertragen (Sequenzanteile in die dortige Methode *run()*), die von der Klasse Thread erbt, damit die Methode *run()* nebenläufig ausgeführt werden kann.
- **Fett** markierte Teile müssen angepasst werden: das Stop-Signal für das Beenden des Clientprozesses lautet künftig „**Server[stopClient]**“.
- *Kursiv* markierte Teile werden nicht mehr benötigt

```

//Schnittstelle zur Netzwerkprotokoll-Implementierung...
ServerSocket serverSocket = null; //.. des Servers
Socket clientSocket = null; //.. des Clients
PrintWriter zumClient = null; //Datenstrom zum Client
BufferedReader vomClient = null; //Datenstrom vom Client

WETTERVERHALTEN2 kkp; //Zustandsdiagramm des Clients
String clientBotschaft = null; //Botschaft von Client zum Server
String serverAntwort = null; //Botschaft vom Server zum Client

public SERVER2() throws IOException {
  
```

```

ServerStarten();
ClientVerbindungStarten(); //auf Client warten und verbinden

do {//lesen und antworten

    clientBotschaft = vomClient.readLine();
    serverAntwort = kkp.HoleAntwort(clientBotschaft);
    zumClient.println(serverAntwort);

    if (serverAntwort.startsWith("Server[stopClient]:")) {
        ClientVerbindungBeenden(); //Verbindung schließen
        ClientVerbindungStarten(); //auf neue Verbindung warten
    }

} while (!serverAntwort.startsWith("Server[stopServer]:"));

ClientVerbindungBeenden();
ServerStoppen();
}

private void ClientVerbindungStarten() throws IOException {

    clientSocket = serverSocket.accept(); //Warten auf die Verbindung
    zumClient = new PrintWriter(clientSocket.getOutputStream(), true);
    vomClient = new BufferedReader(new InputStreamReader(
        clientSocket.getInputStream()));
    //Protokoll-Klasse zur Ermittlung der Serverantworten
    kkp = new WETTERVERHALTEN2();
    //Begrüßung
    serverAntwort = kkp.HoleAntwort("");
    zumClient.println(serverAntwort);
    System.out.println("Client verbunden");
}

private void ClientVerbindungBeenden() throws IOException {...}

```

Der Programmteil der Klasse SERVER reduziert sich dabei deutlich; das SERVER-Objekt braucht in einer Methode *AufNeuenClientWarten()* nur noch auf einen neuen Client zu warten und für diesen einen neuen Thread zu starten.

Der Umbau des Servercodes kann im Unterrichtsgespräch diskutiert und auch durchgeführt werden. Da der Clientcode unverändert bleibt, können sich die Schülerinnen und Schüler sofort mit dem Server verbinden und die Implementierung testen (☞ p05\_mehrclientsparallel).

Hinweis: In der vorliegenden Implementierung wird der Einfachheit halber auf ein Server-Endesignal verzichtet. Für eine Implementierung wäre die Einführung einer Variablen nötig, dessen Wert in Abhängigkeit der vom Client gesendeten Nachricht durch den jeweiligen Clientprozess zugewiesen wird. Die Umsetzung eignet sich als Zusatzaufgabe im nächsten Kapitel, weil dadurch möglicherweise ein weiterer kritischer Abschnitt entsteht.

Anschließend kann der Kern der jetzigen Implementierung des Servers in Pseudocode formuliert und mit dem ersten naiven Ansatz aus Kapitel 2.2.2.1 verglichen werden.

erster Ansatz	Servercode nach Umbau
<p>Starten des Servers</p> <p>Warten auf eine Clientverbindung und Herstellen der Verbindung</p> <p>wiederhole Client-Botschaft lesen; Antwort ermitteln und senden; bis Server-Endesignal gesetzt</p> <p>Clientverbindung beenden Server stoppen</p>	<p>Starten des Servers</p> <p>wiederhole immer Warten auf eine Clientverbindung und Herstellen der Verbindung Auslagern der Verbindung in Thread endewiederhole</p>

Auch die frühen Implementierungen des Serververhaltens der Beispiele p01\_wie\_gehts und p02\_wiegehts2 sind leicht auf den neuen Server anpassbar und könnten als Zusatzaufgabe umgesetzt werden.

Dass Clientanfragen durch Auslagerung der Kommunikationsbehandlung in einen parallel aufgeführten Programmteil „quasi“ gleichzeitig behandelt werden können, ist auch für Buchungssysteme wichtig. Im Unterschied zur behandelten „Wetterauskunft“ greifen die Clientprozesse im Buchungssystem des Projekts p06\_platzbuchung lesend und schreibend auf die Ressource *platzanzahl* zu.

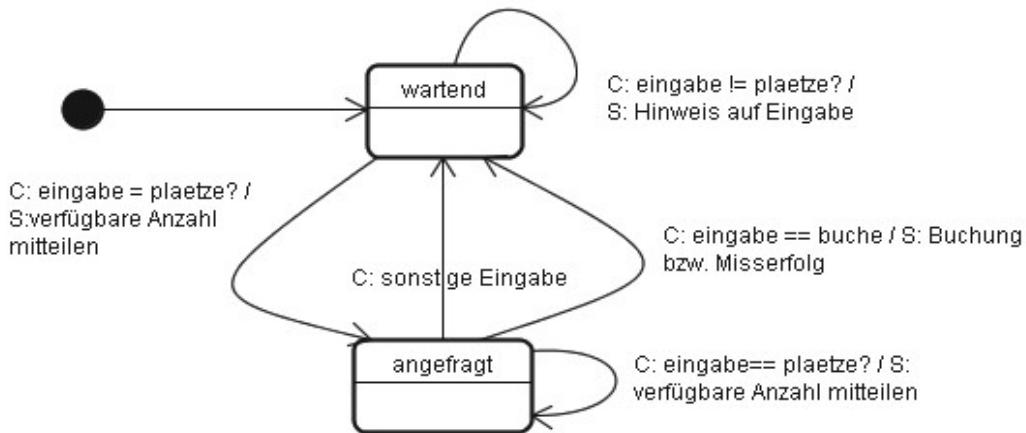
Zur Vertiefung implementieren die Schülerinnen und Schüler ausgehend vom Quelltext der Wetterauskunft ein Platzbuchungssystem und sind dadurch bereits mit dem Motivationsbeispiel des nächsten Kapitels vertraut, mit dessen Hilfe die Probleme beim gemeinsamen Zugriff durch parallele Prozesse auf Ressourcen aufzeigt werden.

### Aufgabe 2: Buchungssystem (p06\_platzbuchung\_aufgabe)

Über einen Server soll eine Platzbuchung nach folgenden Regeln vorgenommen werden können:

Nach der Verbindungsherstellung muss der Client mindestens einmal die Anzahl der freien Plätze (Attribut *plaezeverhalten* des Servers) durch Senden der Zeichenkette „plaetze?“ erfragen, erst anschließend kann ein Platz gebucht (sende „buche“) werden. Nach einer Buchung kann erst erneut gebucht werden, wenn wiederum mindestens einmal eine Anfrage nach den verfügbaren Plätzen erfolgt ist.

- Verwenden Sie die Zustände *wartend* und *angefragt* und implementieren Sie ausgehend von der KLASSE WETTERVERHALTEN2 der vorherigen Aufgabe eine Klasse PLATZBUCHUNG nach folgendem Zustandsdiagramm.



- b) Ergänzen Sie den Quellcode der Klasse SERVER4 des Projekts p06\_platzbuchung\_aufgabe. Auf das Attribut *platzanzahl* des Servers soll über die Methode *PlaetzeVerfuegbar()* zugegriffen werden. Die eigentliche Buchung erfolgt über die Methode *PlaetzeBuchen(int anzahl)*, die den Wert *false* zurückgibt, falls die Buchung nicht möglich ist.
- c) Testen Sie Ihre Implementierung. Kann Ihrer Meinung nach eine Überbuchung auftreten?
- d) *Zusatzaufgabe für besonders Schnelle:* Erweitern Sie die Implementierung so, dass ein gebuchter Platz wieder storniert werden kann, solange der Client nicht beendet wird.

Die Schülerinnen und Schüler können einsehen, dass vor der eigentlichen Aktion (Buchung oder Stornierung) die Anzahl der freien Plätze nochmals überprüft werden muss. Diese Überprüfung bewahrt noch nicht vor Synchronisationsproblemen, gibt aber einen ersten Hinweis auf die Problematik, die im nächsten Kapitel durch eine Buchungssimulation gezeigt wird.

**Lösung:** [p06\\_platzbuchung\\_loesung](#)

### Zusätzliche Aufgabenmöglichkeiten und Implementierungen

Falls man bei besonders schnellen und leistungsstarken Schülergruppen die Implementierung des Chatserver weiter studieren bzw. umsetzen möchte, können weitere Aufgaben bearbeitet werden. Dieser Teil ist jedoch keineswegs verpflichtend.

Aufgabe 3 führt die maximale Anzahl an Verbindungen als weitere Ressource ein, die von den parallelen Clientprozessen verwendet wird. Falls diese Aufgabe bearbeitet wird, so kann bei der Behandlung von Synchronisationsproblemen im nächsten Kapitel zusätzlich auf dieses Beispiel zurückgegriffen werden.

In der Aufgabe 4 wird die Notwendigkeit einer Parallelisierung auf Clientseite problematisiert, die für ein funktionierendes Chatprogramm notwendig ist.

### Aufgabe 3: maximale Anzahl von Verbindungen ([zp1\\_maximaleclientanzahl](#))

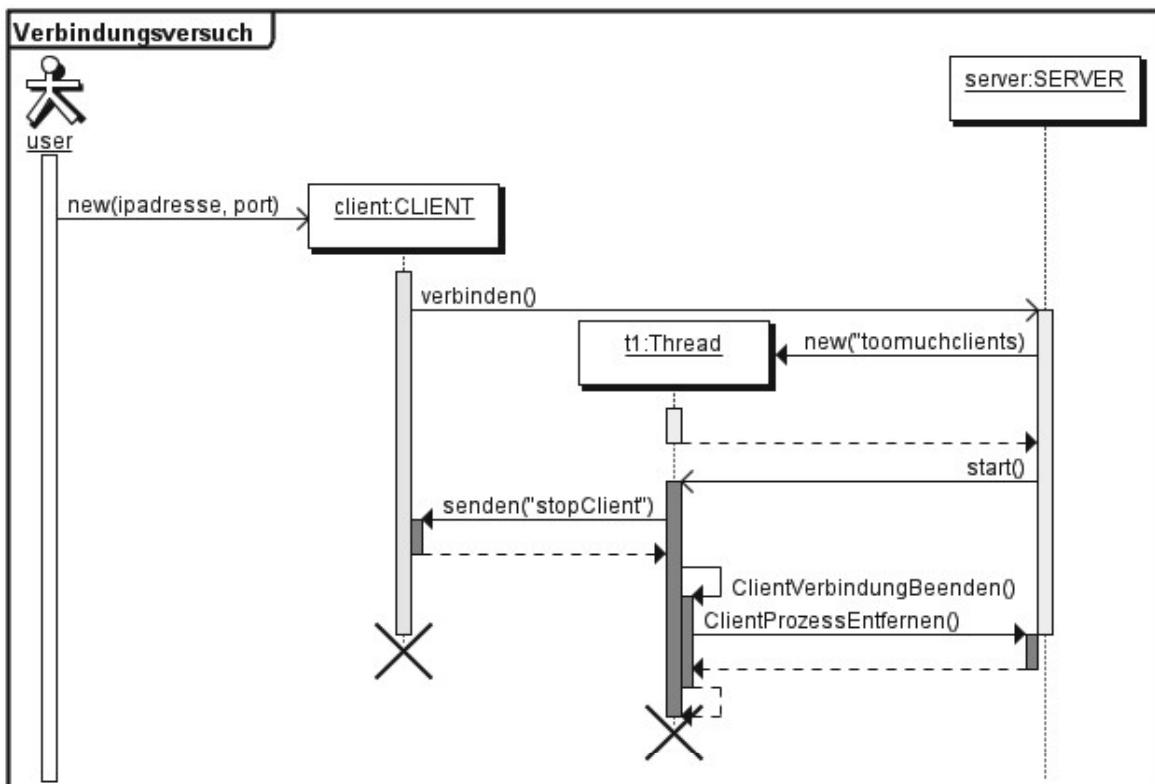
In der bisherigen Programmversion (p05\_mehrclientsparallel) ist dem Server nicht bekannt, wie viele Clients bereits mit ihm verbunden sind. Dies könnte zu Engpässen der Ressourcen führen, wenn sich eine große Anzahl von Clients mit dem Server verbindet. Deshalb wird in der Praxis die Anzahl der Verbindung beschränkt.

Erweitern Sie die Klasse SERVER so, dass

- der Server die Anzahl der Verbindungen in einem Attribut *clientanzahl* speichert;
- der Server bei mehr als 2 (zu Testzwecken, später mehr) Verbindungen keine weitere Verbindung mehr zulässt (die Verbindung wird zunächst aufgebaut, der Client über die maximale Anzahl benachrichtigt und anschließend die Verbindung durch Senden der Antwort „Server [stopClient]“ beendet);
- der Server beim Verbinden und Beenden eines Clients die aktuelle Zahl der Verbindungen ausgibt.

Ein Verbindungsversuch läuft prinzipiell folgendermaßen ab:

Damit der Server dem Client mitteilen kann, dass keine (dauerhafte) Verbindung aufgebaut werden kann, muss der Client mit dem Server kurzzeitig verbunden sein. In der vorliegenden Implementierung sendet der Server nach Überprüfung der Verbindungsanzahl sofort das Stoppsignal zum Client, der sich darauf hin beendet. Der serverseitige Prozess des Clients trägt sich selbst aus der Clientprozessliste aus und wird dann beendet.



### Stopp-Bedingungen für den Client und den Clientprozess des Servers:

Der Client und der Clientprozess soll künftig immer gestoppt werden, wenn die Serverantwort mit „Server[stopClient]“ beginnt.

**Lösungshinweis:**  zp1\_maximaleclientanzahl

**Aufgabe 4: Modellierung eines Chatservers** ( p10\_chatserver)

Da bei einer Chatsitzung nicht bekannt ist, wann eine Nachricht vom Server eintrifft, muss der Client stets empfangsbereit sein. Bei der Realisierung eines Chatservers ist deshalb eine weitere Parallelisierung auf Clientseite nötig. Betrachtet man den Quellcode der Klasse CLIENT, so fällt

auf, dass das Clientprogramm entweder auf die Tastatureingabe oder auf eine Servernachricht wartet.

Modellieren Sie ausgehend vom vorliegenden Buchungsserver einen Chatserver.

Erstellen Sie dafür

- a) ein Klassendiagramm der beteiligten Klassen;
- b) ein Sequenzdiagramm, das den vergeblichen Anmeldeversuch eines Clients zeigt (maximale Zahl der Verbindungen erreicht);
- c) ein Sequenzdiagramm eines erfolgreichen Anmeldeversuchs eines Clients;
- d) ein Zustandsdiagramm für einen Clientprozess.

Folgende Aspekte sollen bei der Modellierung berücksichtigt werden:

- Eine Nachricht soll immer an alle Chatteilnehmer versandt werden.
- Für einen Clientprozess gibt es die Zustände: wartend, angemeldet.
- Ein Client soll einen eindeutigen Namen haben; erst nach der Anmeldung, bei der ein gültiger Name angeben werden muss, kann gechattet werden.
- Ein Client muss zu jeder Zeit Servernachrichten empfangen können.
- Es gibt eine Grenze für die Anzahl der Verbindungen.

**Lösungshinweis:**  p10\_chatserver

## 2.4 Synchronisation von Prozessen

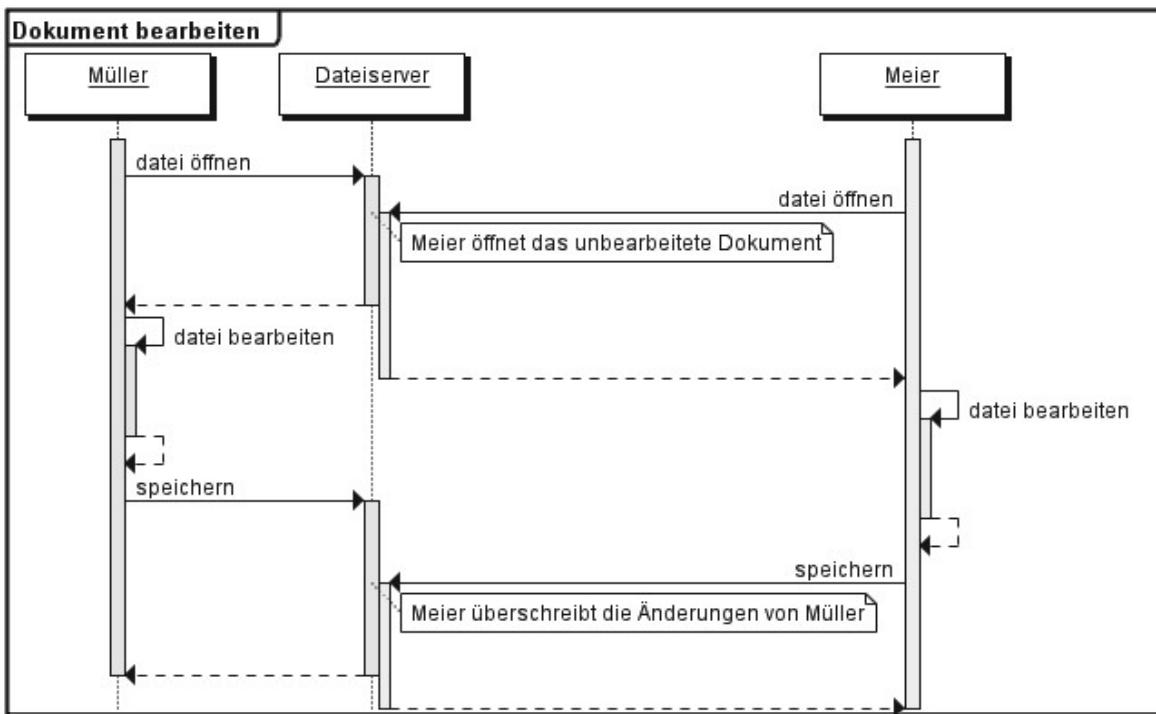
Lp: [Die Schülerinnen und Schüler] erarbeiten Lösungsansätze für die Synchronisation derartiger Vorgänge an Beispielen aus dem täglichen Leben wie der Regelung des Gegenverkehrs durch eine einspurige Engstelle. Beim Übertragen dieser Ansätze auf die Prozesse im Computer wird ihnen bewusst, dass es bei jedem dieser Verfahren bestimmte Sequenzen gibt, die nicht von mehreren Prozessen gleichzeitig ausgeführt werden dürfen.

### 2.4.1 Kritischer Abschnitt, Monitore

#### 2.4.1.1 Inhaltliche Grundlagen

Der Zugriff verschiedener Prozesse auf gemeinsame Ressourcen muss geregelt werden, damit es nicht zu Inkonsistenzen kommt.

Beispiel: Zwei Sekretärinnen sollen gemeinsam an einem Dokument arbeiten, das auf einem Server liegt. Wird die Veränderung des Dokuments (=Ressource) von beiden nicht abgesprochen (oder durch entsprechenden Schreibschutz geregelt), so könnte es vorkommen, dass eine Sekretärin die Änderungen der anderen überschreibt. Ein Sequenzdiagramm verdeutlicht das Problem.



Die einfachste Lösung des Problems besteht darin, dass die Datei exklusiv einer Bearbeiterin oder einem Bearbeiter zugeteilt wird, also nur einmal geöffnet und nur wechselseitig bearbeitet werden kann.

In der Informatik wird der Programmteil, der von verschiedenen Prozessen nur im wechselseitigem Ausschluss ausgeführt werden darf, als kritischer Abschnitt bezeichnet.

Als **kritischer Abschnitt** wird ein Programmabschnitt eines Prozesses bezeichnet, in dem Betriebsmittel (z.B. Daten, Verbindungen, Geräte usw.) verwendet werden und der nicht zeitlich verzahnt zu Programmabschnitten anderer Prozesse ausgeführt werden darf, die die gleichen Betriebsmittel ebenfalls verwenden. Andernfalls kann es zu inkonsistenten Zuständen der Betriebsmittel kommen.

Für den wechselseitigen Ausschluss von Prozessen im kritischen Abschnitt stellen höhere Programmiersprachen sogenannte **Monitore** zur Verfügung. Ein Monitor besteht aus einem oder mehreren Programmabschnitten, die immer nur exklusiv von einem einzigen Thread ausgeführt werden dürfen. Wollen weitere Threads diese Programmteile betreten, müssen sie warten. Im Falle des aktiven Wartens versuchen wartende Threads immer wieder aktiv, den Monitor zu betreten. Beim passiven Warten werden die Threads dagegen in einen Wartezustand versetzt und benachrichtigt, wenn der aktuell im Monitor befindende Thread den kritischen Abschnitt verlässt. Sobald ein Thread den Monitor betreten hat, werden die weiteren Bewerber um den Monitor abgewiesen und erneut in den Wartezustand versetzt.

## Verklemmung

Falls der Zugriff auf mehr als eine Ressource synchronisiert werden muss, kann es zur Verklemmung (Dead Lock) kommen. Dabei sichern sich verschiedene Prozesse zunächst nur einen Teil der Ressourcen, die sie für ihre Aufgabe benötigen. Eine Verklemmung tritt auf, wenn die

Prozesse weitere zur Beendigung ihrer Arbeit nötige Ressourcen nicht bekommen können, ihrerseits aber die bereits reservierten Ressourcen nicht wieder frei geben.

Dies kann in folgender Definition formalisiert werden:

**Verklemmung (Dead Lock):**

Eine Menge von Prozessen  $\{P_1, \dots, P_n\}$  sind in einer **Verklemmung**, wenn jeder dieser Prozesse auf ein Ereignis wartet, das nur von einem anderen Prozess (aus dieser Menge) ausgelöst werden kann.

Hinweis: Das Ereignis ist typischerweise die Freigabe einer benötigten Ressource.

### 2.4.1.2 Umsetzungshinweise

#### Buchungssimulation

Das Buchungssystem des vorherigen Kapitels kontrolliert die Anzahl der verfügbaren Plätze. Bevor ein Platz gebucht werden kann, wird seine Verfügbarkeit überprüft.

Für die Simulation (☞ p07\_buchungssimulation) wird eine leicht modifizierte Version des Buchungssystems aus dem vorherigen Kapitel (☞ p06\_platzbuchung\_loesung) verwendet, bei der die Tastatureingabe in der Klasse CLIENT durch eine zufällige Auswahl aus den Eingaben „plaetze?“ und „buche“ ersetzt wurde.

#### Veränderungen in der Klasse CLIENT:

```
String[] eingaben = {"plaetze?", "buche"};

//Simulation !!! statt clientEingabe
int index = (int) Math.round(Math.random());
clientEingabe = eingaben[index];
//
```

In der Methode *PlaetzeBuchen()* des Servers wird die Anzahl der verfügbaren Plätze periodisch auf den Wert *platzkontingent* gesetzt, falls keine Plätze mehr verfügbar sind. Damit startet die Simulation quasi wieder „von vorne“. Außerdem wird ein verzögter Zugriff auf die Platzanzahl simuliert:

#### Veränderungen in der Klasse SERVER:

```
public boolean PlaetzeBuchen(int anzahl) {

    boolean plaeetzebuchbar = (anzahl <= plaeetzevorhanden);

    // das Ermitteln, ob Plaetze buchbar sind braucht Zeit ...
    try {
        Thread.sleep((int)(Math.random() * 100));
    } catch (InterruptedException e) { }

    if (plaeetzebuchbar) {
        plaeetzevorhanden = plaeetzevorhanden - anzahl;
        System.out.println(plaeetzevorhanden + " Plaetze vorhanden.");

        if (plaeetzevorhanden < 0) {//bei Überbuchung abbrechen.
            System.exit(1);
        }
        if (plaeetzevorhanden == 0) {//Kontingent wieder auffüllen
            plaeetzevorhanden = platzkontingent;
        }
        return true;
    } else {
        return false;
    }
}
```

Hinweise:

- Die zusätzlich hinzugekommenen Zeilen sind farblich hinterlegt.
- `Thread.sleep(zeitangabe)` muss nicht die angegebene Zeit lang warten, sondern kann im Extremfall auch sofort zurückkehren, falls das System nicht ausgelastet ist.

### Monitorkonzept

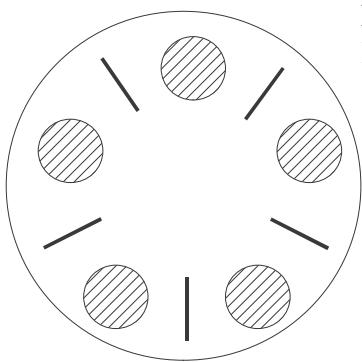
In Java ist das Monitorkonzept auf Objektebene implementiert, d. h. ein Objekt kann den wechselseitiger Ausschluss bei Zugriffen auf eine gemeinsam genutzte Datenstruktur garantieren. Die Deklaration der zu schützenden Methoden als „`synchronized`“ genügt. Synchronisierte Methoden benötigen in der Regel wesentlich mehr Ausführungszeit als nicht synchronisierte Methoden, man wird also möglichst nur die Methoden schützen, die kritische Abschnitte enthalten.

Will man, wie im Beispiel des ChatServers und den Zusatzprojekten `zp2_maximaleclientanzahl_mit_Liste` und `zp3_platzbuchung_mit_maxclientanzahl`, die Referenzen auf die verbundenen Clients in einem Feld speichern, ergibt sich ein weiteres Synchronisationsproblem. Da sich in den vorliegenden Implementierungen die Clientprozesse selbst wieder aus diesem Feld austragen, darf der Zugriff auf dieses Feld nur im wechselseitigem Ausschluss erfolgen. Die Methoden `ClientProzessHinzufuegen` und `ClientProzessEntfernen` müssen deshalb ebenfalls synchronisiert werden.

### Simulation „Die speisenden Philosophen“:

An einem runden Tisch sitzen fünf Philosophen, die abwechselnd speisen und denken.

Ein Philosoph kann nur zu speisen beginnen, falls er das rechts und links von ihm liegende Stäbchen von den vorhandenen fünf Stäbchen an sich nehmen konnte.



Es kommt zur Verklemmung, wenn *jeder* der Philosophen z. B. das links von ihm liegende Stäbchen bereits an sich genommen hat und auf das rechts vom ihm liegende Stäbchen wartet. In der beiliegenden Simulation führt das dazu, dass jeder der Philosophen nur noch „denkt“ (das genommene Stäbchen ist vor dem Philosophennamen vermerkt).

```
[L] Philosoph 4 denkt
[L] Philosoph 3 denkt
[L] Philosoph 0 denkt
[L] Philosoph 2 denkt
[L] Philosoph 1 denkt
[L] Philosoph 4 denkt
```

Die Verklemmung kann nur gelöst werden, falls einer der Philosophen sein Stäbchen wieder her gibt, obwohl er auf das zweite Stäbchen wartet. Eine Verklemmung ließe sich nur verhindern, wenn der Philosoph beide Stäbchen ausschließlich gleichzeitig anfordern würde und beide Stäbchen auch nur gleichzeitig zugeteilt würden.

Hinweis: Die Möglichkeit einer Verklemmung bei Verwendung mehrerer Ressourcen wird an dieser Stelle nur angesprochen. Keinesfalls sollen für das beiliegende Programm Lösungen zur Vermeidung der Verklemmung implementiert werden. Eine weitere typische Deadlock-Situation findet man bei „Rechts vor Links“ im Straßenverkehr.

#### 2.4.1.3 Möglicher Unterrichtsablauf

Mit Hinweis auf das Platzbuchungsprogramm aus dem vorherigen Kapitel werden die Schülerinnen und Schüler mit der Simulation vertraut gemacht. Für die weitere Vorgehensweise reicht die

Erläuterung aus, dass die Benutzereingaben (`plaetze?` und `buche`) zufallsgesteuert vom Clientprogramm simuliert werden und die Buchungen zurückgesetzt werden, wenn die Plätze ausgebucht sind, damit der Buchungsvorgang von Neuem beginnen kann. Der Quellcode muss hierbei nicht analysiert werden.

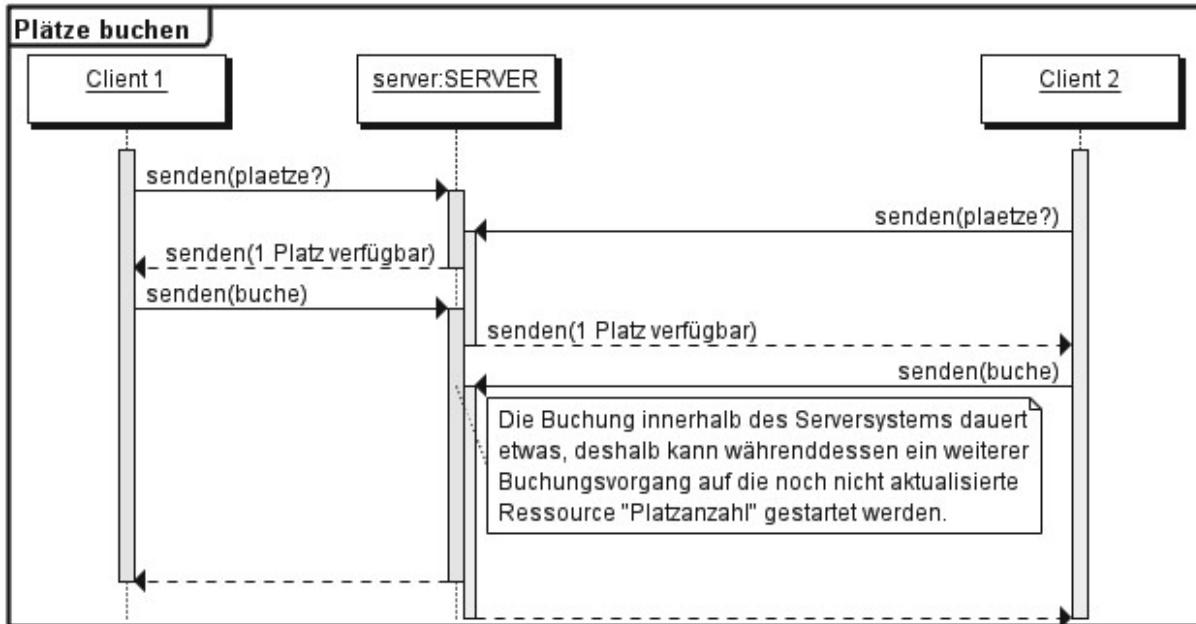
In der Simulation wird die korrekte Arbeitsweise des Buchungssystems getestet.

Die Simulation (p07\_buchungssimulation) bricht mit einer Überbuchung ab, wenn man mehrere Clients mit dem Server verbindet. Im Beispiel buchen zwei Clients je einen Platz, obwohl keine Plätze mehr verfügbar sind (0 Plätze verfügbar). Die Statusmeldung zeigt deshalb eine vorhandene Platzzahl von -1.

```
5 Plaetze vorhanden.
4 Plaetze vorhanden.
3 Plaetze vorhanden.
2 Plaetze vorhanden.
1 Plaetze vorhanden.
-1 Plaetze vorhanden.
Üerbuchung ! Beenden mit Return...
-1 Plaetze vorhanden.
Üerbuchung ! Beenden mit Return...
```

Die Schülerinnen und Schüler erkennen, dass die Ursache des Problems darin besteht, dass verschiedene Threads (CLIENTPROZESS) auf die gleiche Ressource (Plaetze) zugreifen wollen.

Ein Sequenzdiagramm kann dies verdeutlichen:



Die Kennzeichnung der Methode `PlaetzeBuchen()` mit dem Schlüsselwort „**synchronized**“ verhindert, dass mehrere Prozesse diese Methode gleichzeitig betreten können. Eine Überbuchung ist dadurch künftig ausgeschlossen.

Nachdem sich die Schülerinnen und Schüler überzeugen konnten, dass die Wiederholung der Buchungssimulation mit verändertem Quelltext offensichtlich nicht mehr zu einer Überbuchung führt, wird die Definition eines kritischen Abschnitts formuliert (siehe inhaltliche Grundlagen).

Da in Programmiersprachen (wie z. B. Java), die das Monitorkonzept implementieren, ein wechselseitiger Ausschluss von Prozessen leicht zu realisieren ist, kommt es vor allem darauf an, die kritischen Abschnitte zu erkennen. Dies soll nochmals bei der Bahnplatzbuchung geübt werden.

### Identifizierung des kritischen Abschnitts bei der Bahnplatzbuchung

Aus Gründen der Übersichtlichkeit wird im Beispiel p08\_bahnsimulation auf die Client-Server-Kommunikation verzichtet. Nach der Demonstration des Programms sollen die Schülerinnen und Schüler den kritischen Abschnitt identifizieren, den Quellcode verändern und sich davon überzeugen, dass in der korrigierten Version keine Doppelbuchung auftritt. Die Prozesse der Kunden rufen die Methode *nochFrei()* und *getName()* auf. In der Methode *nochFrei()* befindet sich der kritische Abschnitt.

### Aufgabe 1: Bahnsimulation

Untersuchen Sie den Quellcode des Programms Bahnsimulation (p08\_bahnsimulation) und identifizieren Sie den kritischen Abschnitt des Programms. Korrigieren Sie den Quellcode und überzeugen Sie sich anschließend, dass keine Doppelbuchungen mehr auftreten. Begründen Sie, wieso es ohne Maßnahmen zu Fehlbuchungen kommen kann.

#### Lösungshinweise:

In der vorliegenden Simulation ist der Zugriff auf die freien Plätze nicht synchronisiert. Nach einem Durchlauf findet man im Buchungsprotokoll doppelte Belegungen:

```
Kunde 1: Platz 34 in der Vogtlandbahn reserviert
Kunde 2: Platz 35 in der Vogtlandbahn reserviert
Kunde 1: Platz 35 in der Vogtlandbahn reserviert
Kunde 2: Platz 36 in der Vogtlandbahn reserviert
Kunde 1: Platz 36 in der Vogtlandbahn reserviert
```

Synchronisiert man die Methode *nochFrei()* der Klasse ZUG, so tritt keine Doppelbuchung mehr auf.

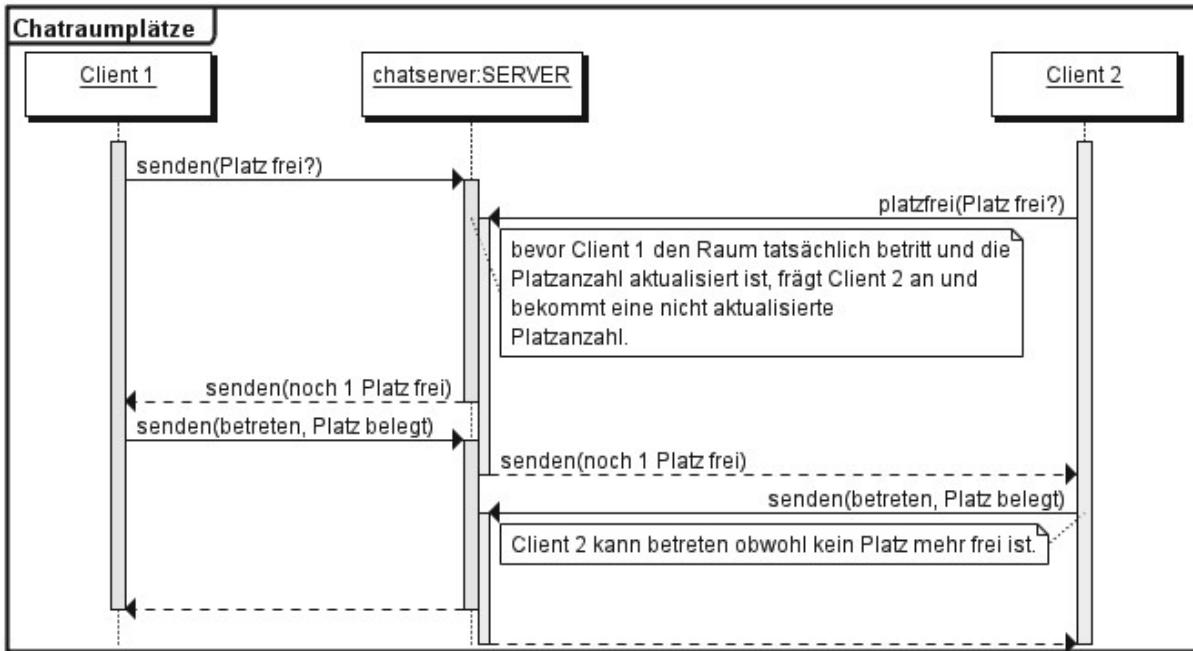
Ein ähnliches Problem tritt auf, falls die Anzahl der Teilnehmer innerhalb eines Chatraums begrenzt ist. Dies kann zur weiteren Vertiefung untersucht werden.

### Aufgabe 2: Chaträume mit einer festen Anzahl von freien Plätzen

Die Anzahl der Plätze in einem Chatraum soll auf 20 begrenzt werden. Obwohl durch das Programm die Anzahl der Teilnehmer kontrolliert wird, stellen die Entwickler in der Testphase des Programms fest, dass plötzlich mehr als 20 Teilnehmer in einem Raum vorhanden sind.

Erläutern Sie unter Verwendung eines Sequenzdiagramms und eines geeigneten Codefragments (Pseudocode genügt), wie es dazu kommen kann, dass mehr als 20 Teilnehmer in den Raum gelangen, und welche Maßnahmen dagegen ergriffen werden können.

### Lösungshinweise:



Kritischer serverseitiger Abschnitt im Pseudocode:

```

falls noch Plätze im Chatraum frei sind
    füge Client zu den Chatraummitgliedern hinzu
    aktualisiere Anzahl der freien Plätze
endefalls

```

Falls die Aufgabe 4 im Kapitel 2.3.3 behandelt wurde, kann optional bei der Besprechung der Lösung von Aufgabe 2 die vorliegende Implementierung zur Begrenzung der Anzahl der Clientverbindungen ( `zp2_maximaleclientanzahl_mit_Liste`) untersucht werden. Bei der Prüfung des Serverquelltextes könnte man in der Methode `ClientProzessHinzufuegen()` einen kritischen Abschnitt finden:

```

private void ClientProzessHinzufuegen(Socket clientSocket) throws IOException {
    if (clientprozesse.size() < maximaleclientanzahl) {
        CLIENTPROZESS1 clientprozess = new CLIENTPROZESS1(clientSocket, this, "");
        clientprozess.start();
        clientprozesse.add(clientprozess);
        System.out.println(clientprozesse.size());
    } else {...}
}

```

Ab hier ist die  
Listenlänge erst wieder  
aktuell!

Tatsächlich erfolgt hier ein nicht synchronisierter Zugriff auf die Liste der Clients. In der Zeit zwischen der Prüfung der Listenlänge und dem Hinzufügen in die Liste könnte sich ein anderer Client eingetragen haben. Außerdem können sich die Clients beim Beenden jederzeit selbstständig aus der Liste austragen. Dadurch besteht die Gefahr eines inkonsistenten Zustands der Clientliste. Die Methode `ClientProzessHinzufuegen()` muss synchronisiert werden. Aber auch die Methode `ClientProzessEntfernen()` muss synchronisiert sein:

```

public synchronized void ClientProzessEntfernen(CLIENTPROZESS clientprozess) {
    clientprozesse.remove(clientprozess);
}

```

```
        System.out.println("Clientverbindung beendet ");
        System.out.println(clientprozesse.size() + " Verbindung(en)");

    }
```

Die Methoden *remove* und *add* der Clientliste dürfen ebenfalls nicht parallel ausgeführt werden. Dies ist hier durch den Monitor der umgebenden Methoden sichergestellt.

### Zugriff auf mehr als eine Ressource (Busplatz und Stadtbesichtigung)

In Anlehnung an die Platzbuchung wird folgendes Problem untersucht und anschließend der Begriff Verklemmung erläutert.

Bei einer Busreise können wie in der vorherigen Aufgabe Plätze im Bus gebucht werden. Der Veranstalter bietet auch eine Besichtigung am Reiseziel an, die extra gebucht werden muss. Während ein Reiseteilnehmer den letzten Busplatz buchen konnte, hat sich ein anderer Teilnehmer derweil den letzten Platz der Stadtbesichtigung reserviert. Beide Teilnehmer wollen aber mit dem Bus fahren und an der Stadtbesichtigung teilnehmen. Obwohl nun Busplatzkontingent und Stadtbesichtigungsplätze komplett vergeben sind, kann kein korrekter Abschluss des Buchung erfolgen. Der Busplatzbesitzer gibt den Platz nicht her und versucht ständig einen Platz für die Stadtbesichtigung zu erhalten und umgekehrt.

Den Schülerinnen und Schülern wird klar, dass das Problem durch die Notwendigkeit der Reservierung zweier Ressourcen verursacht wird.

Dies wird in der Definition eines Deadlocks verallgemeinert.

Hinweis: In der Literatur findet man häufig zusätzliche Bedingungen, die erfüllt sein müssen, damit eine Verklemmung auftreten kann. Eine umfassende Behandlung dieser Bedingungen bzw. Strategien zur Verhinderung und Vermeidung von Verklemmungen ist nicht erforderlich. Es reicht, wenn der Begriff „Verklemmung“ an Beispielen erläutert und darüber hinaus allgemein definiert wird.

### Speisende Philosophen

Auch der Philosoph benötigt zwei Ressourcen (linkes und rechtes Stäbchen), die nacheinander angefordert werden, um zu essen. Bevor man die Simulation ( p09\_speisendePhilosophen) vorführt, sollen die Schülerinnen und Schüler die Möglichkeit der Verklemmung erkennen und die Verklemmungssituation näher beschreiben.

## 2.4.2 Erzeuger-Verbraucher-Problem, aktives und passives Warten

### 2.4.2.1 Inhaltliche Grundlagen

Falls ein Prozess vom Ergebnis eines anderen Prozesses abhängig ist, müssen die Prozesse neben der möglicherweise notwendigen Synchronisation von Zugriffen auf eine gemeinsame Ressource auch aufeinander warten können. Wird zum Beispiel bei der Platzvergabe auch eine Stornierung zugelassen, so könnte ein anderer Kunde zum Zuge kommen, sobald ein Platz frei wird. Dabei kann je nach Kundenverhalten zwischen aktivem und passivem Warten unterschieden werden: Aktives Warten liegt vor, wenn der Kunde ununterbrochen von sich aus wiederholte Platzanfragen stellt. Beim passiven Warten würden alle wartenden Kunden vom Buchungsserver benachrichtigt werden, falls ein Platz frei geworden ist.

Als Standardbeispiel zweier Prozesse, die aufeinander warten müssen, findet man in der Literatur das „Erzeuger-Verbraucher-Problem“:

In einem System nutzen verschiedene Prozesse eine gemeinsame Datenstruktur (Speicher), in der die sogenannten Erzeugerprozesse Elemente ablegen können. Verbraucherprozesse entnehmen der Datenstruktur diese Elemente und verarbeiten sie. Sie müssen warten, bis entsprechende Elemente zur Weiterverarbeitung vorliegen. Erzeugerprozesse hingegen können ihre Produkte nur ablegen, falls die Datenstruktur nicht bereits eine maximale Anzahl von Elementen (Speicher ist voll) enthält, sonst müssen sie ebenfalls warten.

### 2.4.2.2 Umsetzungshinweise

Das Erzeuger-Verbraucher-Modell kann bei folgendem Produktionsablauf betrachtet werden:

Zwei Roboter arbeiten nebenläufig innerhalb einer Produktionskette. Zur Übergabe der Ware von Roboter E (Erzeuger) an Roboter V (Verbraucher) wird ein Abstellplatz verwendet, der genau einen Artikel aufnehmen kann.

Da die Arbeitsgeschwindigkeit der Roboter auch noch von anderen Randbedingungen abhängt, kann es unterschiedlich lange dauern, bis Roboter V die Ware abholt. Da der Abstellplatz derweil belegt ist, muss Roboter E warten, bis er die neu erzeugte Ware dort zur Übergabe abstellen kann. Umgekehrt muss auch der Roboter V warten, bis auf dem Abstellplatz eine Kiste steht, die er weiterverarbeiten kann.

Zur Modellierung einer Simulation werden sowohl beim aktiven ( p11\_aktivesWarten) als auch beim passiven ( p12\_passivesWarten) Warten die Klassen ERZEUGER (Roboter E), VERBRAUCHER (Roboter V) und die KLASSE ABSTELLPLATZ verwendet.

#### Aktives Warten:

Falls die Methoden *abholen()* und *abstellen()* der Klasse ABSTELLPLATZ im Falle einer erfolgreichen Aktion wahr zurückgeben, setzen die Roboter ihre Arbeit fort. Andernfalls probieren sie erneut, ob sie die Ware abstellen bzw. abholen können.

ERZEUGER	VERBRAUCHER
<pre>erzeugen(); wiederhole immer   wenn ablegen() wahr liefert     erzeugen();   endewenn endewiederhole</pre>	<pre>wiederhole immer   wenn abholen() wahr liefert     weiterverarbeiten();   endewenn endewiederhole</pre>

ABSTELLPLATZ	
Methode ablegen	Methode abholen
<pre>wenn platzbelegt den Wert falsch hat   platzbelegt = wahr   gib wahr zurück //wurde abgelegt endewenn  gib falsch zurück</pre>	<pre>wenn platzbelegt den Wert wahr hat   platzbelegt = falsch   gib wahr zurück //wurde abgeholt endewenn  gib falsch zurück</pre>

Betrachtet man die Möglichkeit, dass mehrere Erzeuger oder mehrere Verbraucher vorhanden sind, so wird schnell klar, dass der Zugriff auf die Variable *platzbelegt* (allgemein auf die Ressource) synchronisiert erfolgen muss. Der kritische Abschnitt beginnt wie in den Beispielen zuvor mit der Prüfung des Variablenwerts von *platzbelegt*.

#### **Passives Warten:**

Aktives Warten kostet Rechenzeit und beansprucht dadurch unnötig Systemressourcen. Deshalb ist es besser, sich die wartenden Interessenten (ähnlich einer Warteliste) zu merken und zu benachrichtigen, falls die benötigte Ressource zur Verfügung steht. Dazu wird das Monitorkonzept um die Methoden *Warten()* und *Benachrichtigen()* erweitert.

Der Aufruf der Methode *Warten()* innerhalb des Monitors schickt den Thread in einen Wartezustand, der Thread wird auf die Liste der wartenden Threads gesetzt.

Der Aufruf der Methode *Benachrichtigen()* benachrichtigt alle wartenden Prozesse, dass sich der Ressourcenzustand geändert hat. Sobald der Monitor freigegeben ist, kann ein Prozess aus der Warteliste Zugriff auf die Ressource erhalten. Die anderen Prozesse werden durch den Aufruf der Methode *Warten()* wieder in den Wartezustand versetzt, falls die Ressource bereits erneut belegt ist.

ERZEUGER	VERBRAUCHER
wiederhole immer  erzeugen(); ablegen();  endewiederhole	wiederhole immer  abholen(); weiterverarbeiten();  endewiederhole

ABSTELLPLATZ	
Methode ablegen	Methode abholen

wiederhole solange platzbelegt den Wert wahr hat  warten(); ende wiederhole  platzbelegt = wahr; benachrichtigen();	wiederhole solange platzbelegt den Wert falsch hat  warten(); endewiederhole  platzbelegt = falsch; benachrichtigen();
---	--

#### **2.4.2.3 Möglicher Unterrichtsverlauf**

Ausgehend vom Beispiel der Platzbuchung werden mögliche Szenarien für die Platzvergabe im Falle einer Stornierung erarbeitet:

Beim **aktiven Warten** erhält derjenige den Platz, dessen Anfrage zufällig als erste nach der Stornierung eintrifft. Dazu ist es nötig, dass der Kunde die Anfrage ständig wiederholt.

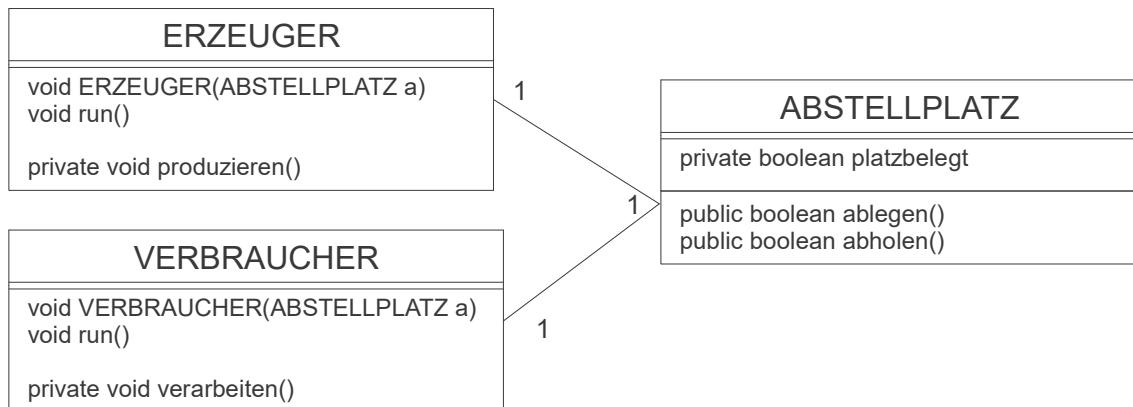
Beim **passiven Warten** werden die Interessenten in eine Liste vorgemerkt und beim Freiwerden eines Platzes benachrichtigt.

Zur weiteren Vertiefung der Begriffe wird anschließend nun das Beispiel zweier nebenläufig arbeitender Roboter innerhalb einer Produktionskette untersucht (siehe Abschnitt 2.4.2.1). Es werden zunächst nur ein Erzeuger (Roboter E) und ein Verbraucher (Roboter V) betrachtet.

### Aufgabe 1: Aktives Warten ([p11\\_aktivesWarten\\_aufgabe](#))

Zwei Roboter arbeiten innerhalb eines Produktionsablaufes zusammen: Dazu verpackt Roboter E die Ware in einen Karton und legt diesen auf einem Übergabeplatz ab. Von dort wird der Karton vom Roboter V abgeholt, etikettiert und weiter auf Paletten verladen. Da die Arbeitsgeschwindigkeit beider Roboter noch von anderen Randbedingungen abhängt, muss die Übergabe zeitlich geregelt werden.

- Formulieren Sie in Worten, wie die Übergabe zeitlich geregelt werden kann. Wer muss dabei auf wen warten?
- Im folgenden soll die Belegung des Übergabeplatzes modelliert werden. Die Klassen der Roboter E (Klasse ERZEUGER) und V (Klasse VERBRAUCHER) haben folgendes Klassendiagramm, der Abstellplatz wird mithilfe der Klasse ABSTELLPLATZ modelliert, für die Ihnen der Quellcode vorliegt. Ergänzen Sie den Quellcode der Methode *run()* der Klassen ERZEUGER und die Klasse VERBRAUCHER ([p11\\_aktivesWarten\\_aufgabe](#)).



- Warten die Roboter aktiv oder passiv aufeinander? Erläutern Sie Ihre Antwort.
- Erkunden Sie das Verhalten des Programms, falls mehrere Objekte der Klasse VERBRAUCHER eingesetzt werden. Erläutern Sie mögliche Ursachen für Programmfehler.

### Lösungshinweise zur Aufgabe 1: ([p11\\_aktivesWarten\\_loesung](#))

Das beiliegende Programm p11\_AktivesWarten\_aufgabe ist ein erster Implementierungsversuch zur Simulation des obigen Problems und wird als Quelltextfragment eingesetzt.

In der Aufgabe sollen die Schülerinnen und Schüler den Quelltext ergänzen und die fehlende Synchronisation der Methoden *abholen()* und *ablegen()* in dieser Implementierung erkennen und beheben. Werden *mehrere* Roboter eingesetzt, um die Ware am Abstellplatz abzuholen, führt dies schnell zur fehlerhaften Ausführung im Programm:

Es wird die Kiste zweimal hintereinander abgeholt bzw. abgelegt!

```

Platz wurde belegt
Platz wurde frei
Platz wurde frei
Platz wurde belegt
Platz wurde belegt
  
```

Nach der Identifikation des kritischen Abschnitts werden die Methoden *ablegen()* und *abholen()* synchronisiert. Die Schülerinnen und Schüler überzeugen sich nun von der korrekten Arbeitsweise des Programms, indem sie mehrere Erzeuger und Verbraucher einsetzen.

Das Programm soll nun so verändert werden, dass die Roboter passiv aufeinander warten (p12\_passivesWarten):

In Java hat jedes Objekt mit der Objektmethode *wait()* die Möglichkeit, Threads, die sich im aktuellen Monitor befinden, in den Wartezustand zu versetzen. Soll der Zugriff durch die wartenden Threads wieder möglich sein, so ruft das Objekt die Methode *notify()* oder *notifyAll()* auf. Der Aufruf von *notify()* bewirkt, dass genau einer der wartenden Threads benachrichtigt wird und den Zugriff auf den Monitor erhält. Warten mehrere Threads, so wird dieser zufällig ausgewählt. Beim Aufruf von *notifyAll()* werden alle wartenden Threads aus dem Wartezustand geholt, der Scheduler des Betriebssystems entscheidet dann, welcher Thread als erster Rechenzeit der CPU zugeteilt bekommt und damit den Monitor als erster betreten kann. Hat ein Thread den Monitor betreten, so werden die restlichen Threads wieder in die Warteposition versetzt, sobald sie den Zugriff auf den Monitor versuchen.

Der Zugriff auf den Lagerplatz erfolgt im Beispiel über die Methoden *ablegen()* und *abholen()*:

```
public synchronized void ablegen() {  
  
    while (platzbelegt) {  
        try {  
            wait();  
        } catch (InterruptedException e) {}  
    }  
    platzbelegt = true;  
    System.out.println("Platz wurde belegt");  
    notifyAll();  
}  
  
public synchronized void abholen() {  
  
    while (!platzbelegt) {  
        try {  
            wait();  
        } catch (InterruptedException e) {}  
    }  
    platzbelegt = false;  
    System.out.println("Platz wurde frei");  
    notifyAll();  
}
```

Eine weitere Implementierungsvariante kann für besonders gute Schülerinnen und Schüler darin bestehen, dass mehrere Erzeuger und Verbraucher, die durch ein Attribut *name* unterschieden werden, eingesetzt werden. Sowohl Erzeuger und Verbraucher könnten zudem eine verschiedene Anzahl an Kisten anliefern bzw. abholen (Abstellplatz nimmt beispielsweise bis zu 5 Kisten auf, Abholer können zwei oder auch drei Kisten mitnehmen).

## Anhang zu Kapitel 2: Praxis-Tipps

### A1 Verbindung mit Mailserver via Telnet

Bei den Betriebssystemen Windows Vista und höher muss der Telnet-Client extra aktiviert werden. Anleitungen dazu findet man im Internet.

## A2 Implementierung einer Client-Server-Kommunikation

### A2.1 Allgemeine Tipps und Vorgehensweise bei den Implementierungen von Client-Server-Kommunikation

- **Vorsicht „Firewall“:** Möglicherweise verhindert eine rechnereigene Firewall, dass sich zwei Rechner verbinden können. Weichen Sie gegebenenfalls auf andere Ports aus (oder schalten Sie die Firewall aus).
- **Zwei BlueJ-Instanzen:** Bei der Demonstration von Server- und Clientprogrammen auf nur einem Rechner durch den Lehrer muss das Programm BlueJ zweimal gestartet werden, da pro BlueJ-Instanz nur ein Konsolenfenster zur Verfügung steht. In Programmen wie z. B. Netbeans ist dies nicht nötig.
- **Javakonsolen-Problem:** Startet man ein executable JAR mit einem Doppelclick unter Windows, so wird standardmäßig `javaw.exe` verwendet, um das Programm auszuführen. Dabei erscheint kein Konsolenfenster. Deshalb werden zum Starten der Programme Batchdateien verwendet.
- **Problem mit Umlauten:** Bei der Kommunikation sollte möglichst auf Umlaute verzichtet werden, da es zu Konvertierungsproblemen kommen kann. Java codiert die Zeichen nach ISO-8859-1 (bzw. Unicode), während die Dos-Eingabeaufforderung aber Codepage 850 codierte Zeichen ausgibt. Die beiliegenden Batchdateien verwenden deshalb die Option `-Dfile.encoding=cp850` zum Start der Javaprogramme.

### A2.2 Übersicht über den Zusammenhang der beiliegenden Implementierungen der Client-Server-Projekte

p01_wiegehts	SERVER	SERVERVERHALTEN	CLIENT
p02_wiegehts2	SERVER Umbenennung von SERVERVERHALTEN zu SERVERVERHALTEN2	SERVERVERHALTEN2 verändertes Serververhalten lt. Aufgabe 2, Kapitel 2.2.2	CLIENT unverändert
p03_wetterverhalten (optionale Implementierung)	SERVER Umbenennung von SERVERVERHALTEN2 zu WETTERVERHALTEN	WETTERVERHALTEN	CLIENT unverändert
p04_mehrclients_aufgabe	SERVER2 für Quelltextergänzung	WETTERVERHALTEN2 für Quelltextergänzung	CLIENT2 für Quelltextergänzung
p04_mehrclients_loesung	SERVER2 Stopbedingung für Client: „Server[stopClient]:“	WETTERVERHALTEN2 Antwort auf „beenden“: „Server[stopClient]:“	CLIENT2 Bedingung zum Beenden: „Server[stopClient]:“
p05_mehrclientsparallel	SERVER3 Auslagerung der Serververarbeitung in einen nebenläufigen Prozess unter Verwendung der Klasse CLIENTPROZESS.	WETTERVERHALTEN2 unverändert	CLIENT2 unverändert
p06_platzbuchung_aufgabe	SERVER4	PLATZBUCHUNG	CLIENT2

	Implementierung der Methoden <i>PlaetzeVerfuegbar()</i> <i>PlaetzeBuchen()</i> neues Attribut plaetzevorhanden	Implementierung der Methode <i>HoleAntwort()</i>	unverändert
p06_platzbuchung_loesung	SERVER4 erhält die Methoden <i>PlaetzeVerfuegbar()</i> <i>PlaetzeBuchen()</i> und das Attribut plaetzevorhanden	PLATZBUCHUNG Die Methode <i>HoleAntwort()</i> ruft erstmalig auch Methoden des Servers auf	CLIENT2 Unverändert
p08_buchungssimulation	SERVER5 Das Platzkontingent in der Methode <i>PlaetzeVerfuegbar()</i> wird immer wieder aufgefrischt	PLATZBUCHUNG unverändert	CLIENT3 Die Eingabe des Benutzers wird zufalls- gesteuert simuliert.
p10_chatserver	CHATSERVER Unterscheidung zwischen angemeldeten Usern und Anzahl der Verbindungen Neue Methoden: <i>SpitznameVorhanden()</i> <i>PrivateNachrichtSenden()</i> <i>PrivateNachrichtSenden()</i> mit weiterer Signatur <i>AnAlleSenden()</i>  CLIENTPROZESS Attribut spitzname <i>SpitznameHolen()</i> <i>Senden()</i>	Das Verhalten wird komplett in die Klasse CLIENTPROZESS übertragen	CHATCLIENT Der Prozess zum Senden der Tastatureingabe wird erst gestartet, wenn die maximale Verbin- dungsanzahl nicht überschritten ist. Empfängt die Servernachrichten.  CLIENTSENDER Dieser nimmt Tastatureingabe entgegen und sendet diese.

### Optionale Programme

zp1_maximaleclientanzahl	SERVERZ3 Der Server erhält das Attribut maximaleclientanzahl und lässt nicht mehr Verbindungen zu.  CLIENTPROZESSZ1	WETTERVERHALTENZ2 bei der Servernachricht „toomuchclients“ gibt die Methode <i>HoleAntwort()</i> als Antwort die Abbruchbedingung „Server [stopClient]: ...“ zurück	CLIENT2 unverändert
zp2_maximaleclientanzahl_mi_t_Liste	SERVERZ4 Der Server verwaltet die Clientprozesse in einer Liste  CLIENTPROZESSZ1	WETTERVERHALTENZ2 Unverändert	CLIENT2 unverändert
zp3_platzbuchung_mit_maxclientanzahl	SERVERZ5 Der Server erhält das Attribut maximaleclientanzahl und lässt nicht mehr Verbindungen zu und verwaltet ein Platz-	PLATZBUCHUNG2 bei der Servernachricht „toomuchclients“ gibt die Methode <i>HoleAntwort()</i> als Antwort die Abbruchbedingung „Server [stopClient]: ...“ zurück	CLIENT2 unverändert

zp1_maximaleclientanzahl	<p>SERVERZ3 Der Server erhält das Attribut maximaleclientanzahl und lässt nicht mehr Verbindungen zu.</p> <p>CLIENTPROZESSZ1</p>	<p>WETTERVERHALTENZ2 bei der Servernachricht „toomuchclients“ gibt die Methode HoleAntwort() als Antwort die Abbruchbedingung „Server [stopClient]: ...“ zurück</p>	CLIENT2 unverändert
	<p>kontingent</p> <p>CLIENTPROZESSZ2</p> <p>WETTER-VERHALTENZ2 wird zu PLATZBUCHUNG2</p>		