Leseprobe

Das IT-Handbuch für Fachinformatiker ist Ihr zuverlässiger Begleiter während der Ausbildung und im beruflichen Alltag. Alle Themen werden anschaulich behandelt, so dass Sie umfassendes Wissen erhalten. Verschaffen Sie sich in dieser Leseprobe einen ersten Überblick.

Sascha Kersken
IT-Handbuch für Fachinformatiker – Der Ausbildungsbegleiter
1.304 Seiten, gebunden, 7. Auflage 2015
34,90 Euro, ISBN 978-3-8362-3473-3

www.rheinwerk-verlag.de/3756

Kapitel 4
Netzwerkgrundlagen

Jeder wandle für sich und wisse nichts von dem andern.
Wandern nur beide gerad', finden sich beide gewiss.
– Johann Wolfgang Goethe/Friedrich Schiller, Xenien

Internet und lokale Netzwerke haben die Bedeutung des Computers in den letzten Jahren revolutioniert. Viele Anwendungsprogramme kooperieren über das Netzwerk miteinander. Der Datenaustausch in und zwischen Unternehmen erfolgt fast ausschließlich per Vernetzung, und immer mehr Geschäftsabläufe erfolgen online. Da die Netzwerkfähigkeit zudem eine Grundfunktionalität aller modernen Betriebssysteme geworden ist, steht diese Einführung noch vor dem Kapitel über allgemeine Systemkonzepte.

Nach einer historischen und technischen Einführung erfahren Sie in diesem Kapitel das Wichtigste über gängige Netzwerkhardware; somit wird die Betrachtung der Hardware aus dem vorangegangenen Kapitel hier vervollständigt. Anschließend werden die Netzwerkprotokolle mit dem Hauptaugenmerk auf die seit Jahren dominierenden Internetprotokolle (TCP/IP) beleuchtet.

4.1 Einführung

In diesem Abschnitt erfahren Sie zunächst einmal, was Netzwerke eigentlich sind und was verschiedene Netzwerktypen voneinander unterscheidet. Anschließend wird die Entstehungsgeschichte lokaler Netze, der Datenfernübertragung und des Internets betrachtet.

4.1.1 Was ist ein Netzwerk?

Ein Netzwerk ist eine Verbindung mehrerer Computer zum Zweck des Datenaustauschs, für verteilte Anwendungen oder auch für die Kommunikation zwischen ihren Benutzern.

Im Lauf der Computergeschichte haben sich viele verschiedene Möglichkeiten der Verkabelung und der Kommunikationsstrukturen sowie zahlreiche Anwendungsgebiete entwickelt:

- Die Verkabelung oder allgemein die Hardwaregrundlage reicht von der Verwendung gewöhnlicher Telefonleitungen mit besonderen Verbindungsgeräten, den Modems, über speziell für die Anwendung in lokalen Netzwerken entwickelte Netzwerkkarten und Netz-

werkkabel bis hin zu Hochgeschwindigkeitsnetzen, etwa über Glasfaserkabel. Auch die diversen Möglichkeiten der drahtlosen Übertragung werden immer wichtiger.

- Kommunikationsstrukturen, definiert durch sogenannte *Netzwerkprotokolle*, gibt es unzählige. Viele sind von einem bestimmten Hersteller, einer Plattform oder einem Betriebssystem abhängig, andere – wie die Internetprotokollfamilie TCP/IP – sind offen, unabhängig und weitverbreitet.
- Was die Anwendungsgebiete angeht, reichen diese vom einfachen Dateiaustausch in Arbeitsgruppen über die gemeinsame Nutzung teurer Hard- und Software bis hin zu hochkomplexen, spezialisierten und verteilten Anwendungen.

Paketvermittelte Datenübertragung

Ein wesentliches Merkmal der meisten Netzwerkformen ist die Übertragung von Daten mithilfe sogenannter *Datenpakete*.

Um die *Paketvermittlung* (Packet Switching) zu verstehen, sollten Sie zunächst ihr Gegenteil, die *Schaltkreisvermittlung* (Circuit Switching) der herkömmlichen Telefonleitungen, betrachten. (Hinweis: Inzwischen gilt dies nicht mehr zwingend; durch die Einführung neuer Technik laufen auch immer mehr Telefonverbindungen hinter den Kulissen paketvermittelt ab – per *Voice over IP* (VoIP) sogar bis zum Endkunden. Durch geeignete Kommunikationsprotokolle wird aber dafür gesorgt, dass die Nutzer dies nicht bemerken.) Durch das Wählen einer bestimmten Rufnummer (oder früher durch die Handvermittlung) werden bestimmte Schalter geschlossen, die für die gesamte Dauer des Telefongesprächs eine feste Punkt-zu-Punkt-Verbindung zwischen beiden Stellen herstellen. Über diese dauerhafte Leitung können Sprache oder Daten in Echtzeit und in der korrekten Reihenfolge ohne Unterbrechung übertragen werden. Nachdem die Übertragung beendet ist, wird die Verbindung wieder abgebaut, und die betroffenen Leitungen stehen für andere Verbindungen zur Verfügung.

Ganz anders sieht es bei der Paketvermittlung aus: Zu keinem Zeitpunkt der Datenübertragung wird eine direkte Verbindung zwischen den beiden beteiligten Stellen hergestellt. Stattdessen sind beide nur indirekt über ein loses Netz von Vermittlungsstellen, *Router* genannt, miteinander verbunden. Damit auf diesem Weg Daten übertragen werden können, wird folgender Mechanismus verwendet:

- Die Daten werden in kleinere Einheiten unterteilt, die *Datenpakete*.
- Jedes einzelne Datenpaket wird mit der Absender- und der Empfängeradresse versehen.
- Der Absender übergibt jedes Datenpaket an den nächstgelegenen Router.
- Jeder beteiligte Router versucht, das Paket anhand der Empfängerangabe an den günstigsten Router weiterzuleiten, damit es letztlich an seinem Ziel ankommt.
- Der Empfänger nimmt die Datenpakete entgegen und interpretiert sie je nach Daten- und Übertragungsart auf irgendeine zwischen den beiden Stellen vereinbarte Art und Weise.

Zur reinen Paketvermittlung gehört zunächst einmal kein Mechanismus, der die vollständige Auslieferung aller Datenpakete garantiert. Es wird standardmäßig weder der Erfolg noch das Ausbleiben einer Paketlieferung gemeldet. Im Übrigen wird auch keine verbindliche Reihenfolge festgelegt. Da jedes einzelne Paket einen beliebigen Weg durch das Netzwerk nehmen kann, kommt mitunter ein später abgesendetes Paket noch vor einem früher versandten beim Empfänger an.

Um die potenziell unsichere Datenübertragung per Paketvermittlung für bestimmte Anwendungen zuverlässiger zu machen, wird zusätzlich eine Erfolgskontrolle implementiert. Außerdem werden die Pakete oft durchnummeriert, um die korrekte Reihenfolge wiederherzustellen. Allerdings haben solche Maßnahmen nichts mit der eigentlichen Paketvermittlung zu tun und müssen in diesem Zusammenhang nicht beachtet werden. In der Regel sind die Softwarekomponenten, die sich um die Übertragung der Datenpakete kümmern, gar nicht in der Lage, diese zusätzlichen Kontrollinformationen selbst auszuwerten.

4.1.2 Entstehung der Netzwerke

Wenn Sie sich die Geschichte der Computer anschauen, die in Kapitel 1, »Einführung«, skizziert wurde, fällt auf, dass die Verwendung von Netzwerken anfangs keinen Sinn ergeben hätte: Bei den frühen Großrechnern gab es keine standardisierte Software, die miteinander hätte kommunizieren können. Darüber hinaus wurden sie zunächst über Schalttafeln und später über Lochkarten bedient. Es gab also keine Echtzeitinteraktion zwischen Benutzer und Programm, sodass es erst recht abwegig war, verschiedene Computer miteinander interagieren zu lassen. Frühestens, als der Dialogbetrieb über Terminals (siehe Kapitel 3, »Hardware«, und Kapitel 5, »Betriebssystemgrundlagen«) eingeführt wurde, war an eine Vernetzung zu denken.

Geschichte des Internets

Der Anstoß für die Entwicklung eines Computernetzwerks kam aus einer eher unerwarteten Richtung: Die atomare Bedrohung des Kalten Krieges schürte die Angst der Verantwortlichen in Politik und Militär in den USA, im Fall eines Atomkriegs handlungsunfähig zu werden, weil die Übermittlung von Informationen nicht mehr funktionieren könnte. Es war schlichtweg zu riskant, sich auf einen einzigen Zentralcomputer mit Terminals zu verlassen. Deshalb begann 1969 der Betrieb eines experimentellen Netzes aus vier Computern an verschiedenen US-amerikanischen Universitäten. Federführend für das Projekt war die (Defense Department's) Advanced Research Projects Agency (ARPA, später auch DARPA), eine Forschungskommission des amerikanischen Verteidigungsministeriums, die 1957 als Reaktion auf den ersten sowjetischen Satelliten Sputnik gegründet worden war. Die USA wollten den Anschluss auf verschiedenen wichtigen Gebieten der Wissenschaft nicht verpassen – und

neben der Raumfahrt gehörte auch die Computertechnik zu diesen Gebieten. Folgerichtig hieß dieses erste Netzwerk ARPANET.

Allgemein sind bei der Betrachtung von Netzwerken immer mindestens zwei Ebenen zu unterscheiden: zum einen der Anwendungszweck des Netzwerks, zum anderen dessen technische Realisierung. Bei näherem Hinsehen sind sogar noch weitere solcher Ebenen auszumachen; diese sogenannten *Schichtenmodelle* werden in Abschnitt 4.2, »Funktionsebenen von Netzwerken«, besprochen. Interessanterweise stellt sich im Entwicklungsverlauf von Netzwerken manchmal heraus, dass der gewünschte Anwendungszweck technisch anders realisierbar ist, aber auch oft, dass eine bestimmte technische Realisation völlig anderen Anwendungen als der ursprünglich geplanten dienlich sein kann. Besonders in der Geschichte des Internets, dessen Vorläufer das ARPANET war, ist dies oft festzustellen.

Die ursprüngliche Anwendung dieses Netzes bestand lediglich darin, Datenbestände auf den unterschiedlichen angeschlossenen Computern automatisch zu synchronisieren, also einfach aus Sicherheitsgründen den gleichen Informationsbestand auf mehreren Rechnern bereitzuhalten.[1]

Grundgedanke der Vernetzung selbst war dabei besonders die Fähigkeit jedes beteiligten Computers, Daten, die nicht für ihn selbst bestimmt waren, sinnvoll weiterzuleiten. Daraus ergeben sich zwei unschätzbare organisatorische und technische Vorteile:

- Ein Computer muss nicht direkt mit demjenigen verbunden sein, mit dem er Daten austauschen soll.
- Der Ausfall oder die Überlastung eines bestimmten Verbindungswegs kann durch Alternativen kompensiert werden.

Auf diese Weise konnte das ursprüngliche Ziel, nämlich die Angriffs- und Ausfallsicherheit des Netzes zu gewährleisten, erreicht werden.

Schon unmittelbar nach der Einrichtung des ARPANETs begann die eingangs erwähnte Weiterentwicklung. Man stellte schnell fest, dass die technische Infrastruktur dieses Netzes für weit mehr Anwendungen zu nutzen war als das vergleichsweise langweilige automatische Synchronisieren von Datenbeständen. So kam bald eine benutzerorientierte Möglichkeit des Dateiaustauschs hinzu. Außerdem war es schon für gewöhnliche Konfigurationsaufgaben unerlässlich, einem entfernten Computer unmittelbar Anweisungen erteilen zu können. Dies war der Ausgangspunkt für die Entwicklung der Terminal-Emulation, also der Benutzung des eigenen Terminals für einen Computer, an den es nicht unmittelbar, sondern nur indirekt über das Netzwerk angeschlossen ist. Auch wenn diese Anwendungen noch nicht sofort ihre späteren Namen – FTP und Telnet – erhielten und die technischen Details

1 Auch heutige Serversysteme vervielfältigen wichtige Daten auf diese Weise automatisch. Das Verfahren wird *Replikation* genannt und kommt insbesondere bei Datenbank- oder Verzeichnisdienstservern zum Einsatz. In Kapitel 13, »Datenbanken«, wird es am Beispiel von MySQL beschrieben.

ihrer Implementierung sich noch weiterentwickelten, sind sie dennoch nach wie vor wichtige Nutzungsschwerpunkte des Internets.

Alles in allem wurde dieses Netzwerk schnell populär. Zwei Jahre nach seiner Einrichtung, im Jahr 1971, waren bereits 40 Computer an verschiedenen Universitäten und anderen staatlichen Forschungseinrichtungen angeschlossen, und es war bei Weitem nicht nur die militärische Nutzung von Interesse. Auch akademisch hatte das Netz viel zu bieten: Wissenschaftler sind darauf angewiesen, Daten auszutauschen; hier ergab sich eine Möglichkeit, dies sehr schnell und effektiv zu tun.

1972 wurde dann der bis dahin bedeutendste Dienst dieses Netzes erfunden: Ray Tomlinson, ein Mitarbeiter eines Ingenieurbüros in Kalifornien, verschickte die erste *E-Mail*. Bis heute zählt die E-Mail zu den erfolgreichsten und verbreitetsten Anwendungen des Netzes; sie kann sich nach dem viel jüngeren World Wide Web noch immer auf einem guten zweiten Platz in puncto Beliebtheit von Internetdiensten halten, und es ist auch nicht zu sehen, warum sich dies in absehbarer Zeit ändern sollte. Zwar gibt es offensichtliche Probleme wie das massenhafte Aufkommen von Spam (unerwünschten Werbemails) und Phishing (Mails, die dem Empfänger vorgaukeln, sie seien von bekannten Firmen, und ihn zum Beispiel zur Passworteingabe verleiten), aber es gibt auch noch immer keine allgemein verbreitete Alternative.

Das ursprüngliche ARPANET wuchs immer weiter. Zudem wurden nach dem gleichen Prinzip andere, ähnliche Netze konstruiert. Dies ist nicht zuletzt der Tatsache zu verdanken, dass alle Schritte, die zur Entwicklung des Netzes beigetragen haben, von Anfang an sorgfältig dokumentiert und der Öffentlichkeit zugänglich gemacht wurden. Dieser Dokumentationsstil ist bis heute beibehalten worden; die entsprechenden Dokumente heißen *RFC* (Request For Comments, etwa »Bitte um Kommentare«).

Es gibt bis heute über 7.000 solcher RFC-Dokumente, die alle online zur Verfügung stehen, zum Beispiel unter *http://www.rfc-editor.org/rfc-index2.html*. Die meisten sind technische Beschreibungen von Entwürfen, Protokollen und Verfahrensweisen; nur wenige (in der Regel mit dem Datum 1. April) nehmen sich nicht ganz so ernst – zum Beispiel RFC 2324, in dem das Protokoll HTCPCP zur Steuerung vernetzter Kaffeemaschinen vorgeschlagen wird, oder RFC 1300, ein nettes Gedicht über Namen und Begriffe, die im Zuge der Computer- und Netzwerkentwicklung ihre ursprüngliche Bedeutung verändert haben.

Alle Personen, Institutionen und Unternehmen, die etwas Entscheidendes zum ARPANET und späteren Internet beigetragen haben, haben dies in solchen Dokumenten erläutert. Dies ermöglicht es jedem beliebigen Hersteller von Hard- oder Software, mit seinen Produkten diese Standards zu unterstützen, denn sie gehören keinem einzelnen Hersteller und keiner bestimmten Person, und niemand kann den Zugriff darauf beschränken oder Lizenzgebühren fordern – ein entscheidender Grund dafür, warum die Protokolle des Internets heute vom Personal Computer bis zum Großrechner überall dominieren.

In den 80er-Jahren schließlich wurde der militärisch genutzte Teil des ARPANETs als MilNet von ihm abgetrennt, und das restliche ARPANET wurde mit dem NSFNet, dem Netz der National Science Foundation, und einigen anderen Netzwerken zum Internet zusammengeschlossen. Die kommerzielle Nutzung, heute Hauptverwendungsgebiet des Internets, ließ danach aber noch fast 15 Jahre auf sich warten. Denn die Anwendungen des Internets waren zwar robust und wenig störanfällig, aber alles andere als benutzerfreundlich. Abgesehen davon waren die ersten Personal Computer, die in der zweiten Hälfte der 70er-Jahre auftauchten, weder konzeptionell noch von der Leistung her in der Lage, mit den Internetprotokollen etwas anzufangen.

Recht früh wurde dagegen die Datenfernübertragung (DFÜ), also der Datenaustausch über Telefonleitungen, für Home- und Personal Computer eingeführt. Seit Ende der 70er-Jahre wurden sogenannte *Akustikkoppler* verwendet: Geräte, die an den Computer angeschlossen wurden und auf die einfach der Telefonhörer gelegt werden musste. Diese langsamen und störanfälligen Apparate wurden bald durch Modems ersetzt, die eine direkte Verbindung zwischen Computer und Telefonleitung zuließen und im Laufe der Jahre allmählich schneller und zuverlässiger wurden. Hauptanwendungsgebiete waren auf der einen Seite die sogenannten *Mailboxen*, also Informations- und Datenangebote für Computer, die eine direkte Telefonverbindung zum Mailboxrechner herstellten. Auf der anderen Seite entstanden in den 80er-Jahren die meisten kommerziellen *Online-Dienste* wie CompuServe, AOL oder in Deutschland BTX (Vorläufer von T-Online), das zunächst über spezielle Terminals anstelle von PCs mit einer bestimmten Software genutzt wurde.

Die Entwicklung des Internets vom exklusiven Wissenschaftlernetz zum Massenmedium nahm ihren Anfang erst 1989 in der Schweiz, am Europäischen Forschungsinstitut für Kernphysik (CERN) in Genf. Dort machte sich der britische Informatiker *Tim Berners-Lee Gedanken* darüber, wie man Netzwerke, besonders das Internet, für den einfachen und effizienten Zugriff auf wissenschaftliche Dokumente nutzen könnte. Ergebnis dieser Arbeit war die Grundidee des *World Wide Webs*, eines hypertextbasierten Informationssystems, das die Infrastruktur des Internets zur Datenübermittlung nutzen sollte.

Hypertext ist nichts anderes als Text mit integrierten Querverweisen, die automatisch funktionieren. Mit anderen Worten: Durch Anklicken des Querverweises, der in diesem Zusammenhang *Hyperlink* heißt, stellt der Text selbst – beziehungsweise das System, das diesen darstellt – die Verbindung mit dem verknüpften Dokument her.

Nun war Hypertext 1989 gewiss nichts Neues. Versuche damit reichen zurück bis in die 50er-Jahre, in Hilfesystemen war er in den 80er-Jahren bereits Alltag. Neu war nur seine Nutzung über ein Netzwerk, genauer gesagt, über das Internet.

So entstand ein äußerst effektives Informationssystem für Wissenschaftler, die auf diese Weise ihre Forschungsergebnisse miteinander austauschten. Der Prototyp dieses Systems, das World Wide Web heißen sollte, umfasste im Einzelnen die folgenden Bestandteile:

- ein spezielles neues Anwendungsprotokoll, das *Hypertext Transfer Protocol* (HTTP)
- einen Serverdienst, der in der Lage ist, Anfragen, die in der Sprache des HTTP formuliert sind, auszuliefern
- eine neu geschaffene Formatierungs- und Beschreibungssprache für solche Hypertext-Dokumente, die *Hypertext Markup Language* (HTML)
- ein Anzeigeprogramm für entsprechend formatierte Dokumente, den Browser

1991 wurde das System der Öffentlichkeit vorgestellt. Es wurde praktisch von Anfang an nicht nur zu ernsthaften wissenschaftlichen Zwecken genutzt, sondern allgemein zur Veröffentlichung von Text, Bildern und den verschiedensten Themen. Zunächst war die Nutzung des Systems beschränkt auf wissenschaftliches Personal sowie interessierte Studenten. Sie störten sich nicht am mangelnden Komfort der ersten Browser oder den geringen Layoutfähigkeiten der ersten HTML-Versionen. Als jedoch immer mehr private Benutzer dazukamen – was durch das allmähliche Entstehen kommerzieller Internetprovider und Browser für PC-Betriebssysteme wie Windows oder Mac OS gefördert wurde –, änderte sich dies. Der berühmt gewordene »Browserkrieg« zwischen Netscape und Microsoft schuf letztlich Fakten, die niemand für möglich oder auch nur wünschenswert gehalten hätte, die jedoch bis heute das Wesen des World Wide Web bestimmen.

Zwei Merkmale sind hier besonders wichtig:

- Die Seitenbeschreibungssprache HTML wurde immer mehr für die Definition des Seitenlayouts statt nur für die Struktur genutzt. Für Websites, die ein möglichst großes Publikum erreichen sollen, das weniger technisch und mehr inhaltlich interessiert ist, ist das Layout wichtiger als die Struktur. (Inzwischen kommt für das Layout allerdings praktisch nur noch CSS zum Einsatz, und HTML konzentriert sich wieder – wie ursprünglich beabsichtigt – auf die Dokumentstruktur.)
- Der Anteil kommerzieller Websites am gesamten Bestand wurde immer größer und überwiegt heute bei Weitem; das Angebot im Web ist den Rundfunkmedien wie etwa dem Fernsehen ähnlicher geworden. Während Tim Berners-Lee sich ursprünglich ein Netz vorgestellt hatte, in dem alle Teilnehmer sowohl Anbieter als auch Konsumenten von Inhalten sein sollten, wird das Web heutzutage von vielen weitgehend passiv als Medium genutzt. Erst die kollaborativen »Web-2.0«-Tools wie Blogs, Wikis, soziale Netzwerke und andere kommen Berners-Lees eigentlichen Ideen näher, wobei die gleichzeitig zu beobachtende Kommerzialisierung samt Aufkauf der wichtigsten Sites durch große Medienkonzerne sicherlich nicht in seinem Sinne ist.

Lokale Netze

Einen vollkommen anderen Anstoß zur Entwicklung von Netzwerken gab das Aufkommen des sogenannten *Outsourcings* in der Computertechnik, also der Verlagerung der Rechenleistung von einem Zentralcomputer auf den einzelnen Schreibtisch.

Die fortschreitende Ausstattung von Büros mit Personal Computern führte mangels anderer Optionen zunächst zur Blüte des »*Turnschuhnetzwerks*« (englisch *Sneakernet*): Anwender liefen mit Datenträgern bewaffnet durch das ganze Gebäude, um Daten miteinander auszutauschen oder zum Beispiel einen speziellen Drucker zu verwenden. Auch zwischen verschiedenen Unternehmen erfreute sich der sogenannte *Datenträgeraustausch* großer Beliebtheit: Die Datensätze von Geschäftsvorfällen wurden auf Disketten oder Magnetbändern zwischen den einzelnen Unternehmen hin und her gereicht.

Lokale Firmennetzwerke wurden zwar bereits Mitte der 70er-Jahre bei XEROX PARC erfunden, aber erst Ende der 80er-Jahre rückten sie stärker ins allgemeine Interesse. Es war ein Bedürfnis der Anwender von PCs, miteinander Daten auszutauschen, einfach deshalb, weil die meisten Vorgänge der Datenverarbeitung von mehreren Mitarbeitern erledigt werden. So entstanden viele verschiedene Arten der Netzwerkhardware. Neben dem bereits genannten Ethernet mit seinen vielfältigen Varianten gab es beispielsweise auch Token Ring von IBM, ARCnet oder auch einfache serielle Direktverbindungen zwischen Computern über die sogenannten *Nullmodemkabel*.

Was die Software angeht, wurden die eigentlich nicht dafür geeigneten PC-Betriebssysteme um Netzwerkfähigkeiten erweitert. Hinzu kamen spezielle Betriebssysteme für Server, also solche Rechner, die anderen im Netzwerk verschiedene Ressourcen zur Verfügung stellen. Bekannt sind hier etwa Novell NetWare, IBM OS/2 oder später auch Windows NT Server.

Wenn Sie in diesem Zusammenhang Linux und andere Unix-Varianten vermissen, dann liegt das daran, dass Unix als PC-Betriebssystem und als Serversystem für PC-Netzwerke erst einige Jahre später populär wurde. Ein gewisses Grundverständnis für Unix ist übrigens unerlässlich, um die Funktionsweise der Internetprotokolle nachvollziehen zu können. Einige Grundlagen dieses Systems werden in Kapitel 5, »Betriebssystemgrundlagen«, und Kapitel 7, »Linux«, erläutert.

4.2 Funktionsebenen von Netzwerken

Wie bereits in der Einleitung mehrfach angedeutet wurde, besteht ein gewisses Problem beim Verständnis von Netzwerken darin, dass einige sehr verschiedene Aspekte zu ihrem Funktionieren beitragen. Schon ganz zu Beginn haben Sie eine grobe Unterteilung in die drei Ebenen Verkabelung oder allgemeine Netzwerkhardware, Kommunikationsstrukturen oder Netzwerkprotokolle und schließlich Anwendungen eines Netzwerks kennengelernt.

Eine so ungenaue Einteilung lässt die grundsätzliche Schwierigkeit erkennen, reicht aber nicht ganz aus, um Netzwerke in all ihren Bestandteilen zu begreifen, und schon gar nicht, um verschiedene Arten von Netzwerken miteinander zu vergleichen. Auch die Tatsache, dass ein und dieselbe Komponente auf einer bestimmten Ebene wahlweise mit mehreren unterschiedlichen Elementen einer anderen Funktionsebene zusammenarbeiten kann, wird so noch nicht transparent genug.

4.2.1 Das OSI-Referenzmodell

Um die Ebenen, die ein Netzwerk ausmachen, ganz genau auseinanderhalten zu können, bedient man sich sogenannter *Schichtenmodelle* (Layer Models). Das bekannteste und verbreitetste von ihnen ist das OSI-Referenzmodell der internationalen Standardisierungsorganisation ISO. OSI steht für »Open Systems Interconnect«, also etwa »Verbindung zwischen offenen Systemen«. Das Modell wurde 1978 entworfen und besteht aus sieben übereinander angeordneten Schichten, die jeweils einen Aspekt der Netzwerkkommunikation beschreiben. Ganz unten ist die Hardware angesiedelt, ganz oben befindet sich die Anwendung des Netzes. Hier zunächst die Schichten des OSI-Modells im Überblick, die Beschreibung folgt im Anschluss daran:

1. Bit-Übertragungsschicht (Physical Layer)
2. Sicherungsschicht (Data Link Layer)
3. Vermittlungsschicht (Network Layer)
4. Transportschicht (Transport Layer)
5. Kommunikationssteuerungsschicht (Session Layer)
6. Darstellungsschicht (Presentation Layer)
7. Anwendungsschicht (Application Layer)

Die Bezeichnung *OSI-Referenzmodell* deutet bereits darauf hin, dass es sich nicht um einen Standard handelt, der konkrete Netzwerkprotokolle definiert. Das OSI-Modell definiert nur die Funktionen der einzelnen Schichten und ist somit ein Schema zur Definition solcher Standards, beispielsweise für die im weiteren Verlauf des Kapitels vorgestellten IEEE-802.3-Standards. Jeder Standard deckt dabei immer nur Teilaspekte des OSI-Modells ab.

Die Bedeutung der einzelnen Schichten des OSI-Modells

1. Die *Bit-Übertragungsschicht* oder auch *physikalische Schicht* beschreibt nur, wie die reine Übertragung der Daten elektrisch beziehungsweise allgemein physikalisch erfolgt. OSI-basierte Netzwerkstandards beschreiben in dieser untersten Schicht die Struktur der Signale. Dazu gehören unter anderem die folgenden Aspekte:
   - zulässiger Amplitudenbereich
   - Versand- und Empfangsmethoden für Bit-Folgen
   - Operationen zur Umwandlung dieser Bit-Folgen in Daten für die nächsthöhere Schicht (und umgekehrt)
   - Verarbeitungsgeschwindigkeit der Bit-Folgen
   - Start- und Stoppsignale
   - Erkennung beziehungsweise Unterscheidung der Signale bei gemeinsam genutzten Medien
   - Übertragungseigenschaften der Medien (Kabel, Lichtwellenleiter, Funk oder Ähnliches)

Die Medien selbst sowie Netzwerkkarten oder Onboard-Netzwerkchips sind kein Bestandteil der Definitionen auf der ersten Schicht. Die Hersteller müssen selbst dafür Sorge tragen, dass ihre Produkte den Spezifikationen genügen.

2. Die *Sicherungsschicht* beschreibt alle Vorkehrungen, die dafür sorgen, dass aus den einzelnen zu übertragenden Bits, also dem reinen physikalischen Stromfluss, ein verlässlicher Datenfluss wird. Dazu gehören die beiden Teilaufgaben Media Access Control (MAC) – die Regelung des Datenverkehrs, wenn mehrere Geräte den gleichen Kanal verwenden – sowie Logical Link Control (LLC), wobei es um die Herstellung und Aufrechterhaltung von Verbindungen zwischen den Geräten geht.

   Viele Protokolle dieser Schicht implementieren eine Fehlerkontrolle, bei Ethernet wird zum Beispiel CRC verwendet. In manchen Fällen wird auch Quality of Service (QoS), eine Art Prioritätsinformation, benutzt. In der Sicherungsschicht werden die Bit-Folgen in Einheiten einer bestimmten Größe unterteilt und mit einem Header aus Metainformationen versehen. Je nach Standard werden auf dieser Ebene unterschiedliche Namen für diese Datenpakete verwendet. Bei Ethernet und Token Ring ist beispielsweise von *Frames* die Rede, bei ATM von *Zellen*. Der Payload (Nutzdateninhalt) eines Frames beziehungsweise einer Zelle beginnt in aller Regel mit dem Header eines hineinverschachtelten Pakets der nächsthöheren Schicht. Es kann aber auch vorkommen, dass Pakete verschiedener Protokolle der zweiten Schicht ineinander verschachtelt werden. Dies ist zum Beispiel bei PPP over Ethernet, PPP over ATM oder ATM over SDH der Fall.

3. Die *Netzwerkschicht* oder *Vermittlungsschicht* definiert diejenigen Komponenten und Protokolle des Netzwerks, die an der indirekten Verbindung von Computern beteiligt sind. Hier ist sogenanntes *Routing* erforderlich, das Weiterleiten von Daten in andere logische oder auch physikalisch inkompatible Netzwerke. So gehören zum Beispiel auch alle diejenigen Protokolle zur Netzwerkschicht, die die logischen Computeradressen der höheren Schichten in die physikalischen Adressen umsetzen, bei Ethernet zum Beispiel ARP. Auch auf der Netzwerkschicht werden die Daten in Pakete unterteilt, deren Namen sich je nach konkretem Protokoll unterscheiden. Das mit Abstand verbreitetste Protokoll dieser Ebene, das im weiteren Verlauf des Kapitels noch ausführlich vorgestellte IP-Protokoll, bezeichnet sie als *IP-Datagramme*.

4. Die Protokolle der *Transportschicht* lassen sich in verbindungsorientierte Protokolle wie TCP und verbindungslose Protokolle wie etwa UDP unterteilen. Auf dieser Schicht werden vielfältige Aufgaben erledigt. Ein wichtiger Aspekt sind Multiplex-Mechanismen, die die Anbindung der Datenpakete an konkrete Prozesse auf den kommunizierenden Rechnern ermöglichen, bei TCP und UDP beispielsweise über Portnummern, bei SPX über Connection-IDs. Verbindungsorientierte Transportprotokolle wie TCP oder SPX sind zudem meist mit einer Fluss- und Fehlerkontrolle ausgestattet, um zu gewährleisten, dass Pakete vollständig am Ziel ankommen und dort in der richtigen Reihenfolge verarbeitet werden. Auch auf der vierten Schicht verwenden verschiedene Protokolle jeweils eigene Bezeichnungen für die Datenpakete; so ist etwa von *UDP-Datagrammen*, *TCP-Sequenzen* und *SPX-Paketen* die Rede.

5. Die *Kommunikationssteuerungsschicht* oder *Sitzungsschicht* stellt die Kommunikation zwischen kooperierenden Anwendungen oder Prozessen auf verschiedenen Rechnern sicher.

6. Die *Darstellungs-* oder *Präsentationsschicht* dient der Konvertierung und Übertragung von Datenformaten, Zeichensätzen, grafischen Anweisungen und Dateidiensten.

7. Die *Anwendungsschicht* schließlich definiert die unmittelbare Kommunikation zwischen den Benutzeroberflächen der Anwendungsprogramme, kümmert sich also um die Verwendung derjenigen Dienste über das Netzwerk, die Benutzer unmittelbar zu Gesicht bekommen.[2]

Da das OSI-Modell eine Zusammenstellung von möglichen Fähigkeiten für viele verschiedene Arten von Netzwerken darstellt, kann es natürlich vorkommen, dass die eine oder andere Schicht in einem bestimmten Netzwerk wichtiger ist als eine andere oder dass zum Beispiel ein Protokoll Funktionen zweier Schichten abdeckt oder auch nur eine Teilfunktion einer Schicht erbringt. Um diese Umstände deutlich zu machen, wendet man häufig anders aufgeteilte Schichtenmodelle mit meist weniger, selten mehr Schichten an – dies gerade dann, wenn es um die konkrete Beschreibung einer bestimmten Art von Netzwerk geht.

4.2.2 Das Schichtenmodell der Internetprotokolle

Im Bereich der TCP/IP-Netzwerkprotokolle, die unter dem Betriebssystem Unix und im Internet den Standard darstellen, wird zum Beispiel häufig ein Modell aus nur vier Schichten verwendet. Dies wird dem Wesen dieser Protokolle auch wesentlich eher gerecht als das OSI-Modell, denn die Internetprotokolle sind bereits einige Jahre älter als OSI. Es handelt sich um das Schichtenmodell des ursprünglichen ARPANETs, das vom US-Verteidigungsministerium finanziert wurde. Deshalb wird es meist als *DoD-Modell* (»Department of Defense«), manchmal auch als *DDN-Modell* (»Department of Defense Network«) bezeichnet.

Die vier Schichten bei TCP/IP-Netzwerken nach dem DDN Standard Protocol Handbook sind:

1. Netzzugangsschicht (Network Access Layer oder Link Layer)
2. Internetschicht (Internet Layer)
3. Host-zu-Host-Transportschicht (Host-to-Host Transport Layer oder einfach Transport Layer)
4. Anwendungsschicht (Application Layer)

2 Halb scherzhaft wird der Anwender des vernetzten Rechners manchmal als *achte Schicht* bezeichnet, und Probleme, die durch fehlerhafte Benutzung entstehen, nennt man entsprechend *Layer-8-Fehler*.

Diese vier Schichten sind den konkreten Gegebenheiten von TCP/IP-Netzwerken angepasst, bei denen es zum Beispiel nur theoretisch möglich ist, von einer separaten Sitzungsschicht zu sprechen.

Das OSI-Referenzmodell kann mit dem DDN-Schichtenmodell deshalb nur grob in Beziehung gesetzt werden. Tabelle 4.1 zeigt, wie ein solcher Vergleich ungefähr aussehen könnte.

OSI-Modell	DDN-Modell
7. Anwendungsschicht	4. Anwendungsschicht
6. Darstellungsschicht	
5. Sitzungsschicht	
4. Transportschicht	3. Host-zu-Host-Transportschicht
3. Vermittlungsschicht	2. Internetschicht
2. Sicherungsschicht	1. Netzzugangsschicht
1. Bit-Übertragungsschicht	

Tabelle 4.1 Vergleich zwischen dem OSI-Referenzmodell und dem DDN-Schichtenmodell der Internetprotokolle

Die Bedeutung der Schichten des DDN-Modells

1. Die Netzzugangsschicht (*Network Access Layer* oder *Link Layer*) beschreibt, wie die physikalische Datenübertragung erfolgt. Die Aufgaben, die auf dieser Schicht zu erledigen sind, werden durch viele recht unterschiedliche Protokolle erbracht, einfach deshalb, weil es kaum eine Sorte von Netzwerkhardware gibt, auf der die Internetprotokolle noch nicht implementiert worden wären. Die eigentlichen Kernprotokolle, zu denen besonders die Namensgeber der Protokollfamilie gehören – also das Transmission Control Protocol (TCP) und das Internet Protocol (IP) –, kümmern sich überhaupt nicht um die physikalischen Verhältnisse. Damit dies möglich ist, müssen auf dieser untersten Schicht die Bit-Übertragung und die Transportsicherung zuverlässig zur Verfügung gestellt werden.

   Auf diese Weise entspricht die Netzzugangsschicht der Internetprotokolle den beiden untersten Schichten von OSI.

2. Die Internetschicht (*Internet Layer*), die im Wesentlichen der Vermittlungsschicht des OSI-Modells ähnelt, kümmert sich um die logische Adressierung der Rechner im Netz, durch die die grundsätzliche Identifizierbarkeit des jeweiligen Rechners sichergestellt wird. Eine weitere wichtige Aufgabe auf dieser Ebene ist das Routing, also die Weiterleitung von Daten über verschiedene physikalisch und/oder logisch getrennte Netze hinweg. Grundlage dieser Tätigkeiten ist das IP-Protokoll (Internet Protocol). Es definiert die IP-Adressen, 32 (in einer neueren Version 128) Bit breite Nummern, die den einzelnen Rechnern zugewiesen werden und die der Unterscheidung der einzelnen Netzwerke und der Rechner in diesen Netzen dienen. Außerdem versieht es jedes Datenpaket mit einem Header, also einer Zusatzinformation, die insbesondere die Quelladresse des sendenden Rechners und die Zieladresse des empfangenden Hosts enthält. Ein Datenpaket dieser Ebene wird als *Datagramm* bezeichnet.

3. Die Host-zu-Host-Transportschicht (*Host-to-Host Transport Layer*) kümmert sich um den zuverlässigen Datenaustausch zwischen den kommunizierenden Rechnern. Im Wesentlichen sind hier zwei verschiedene Protokolle verantwortlich (neben anderen, selten verwendeten). Diese beiden Protokolle werden jedoch niemals gleichzeitig, sondern immer alternativ verwendet. Das einfachere und weniger robuste *UDP* (User Datagram Protocol) stellt einen schlichten und wenig datenintensiven Mechanismus zur Verfügung, der die direkte Nutzung der IP-Datagramme für die Host-zu-Host-Kommunikation erlaubt. Dabei wird keine virtuelle Verbindung zwischen den beiden Rechnern hergestellt; es findet also keine Kontrolle über einen kontinuierlichen Datenstrom statt. Das erheblich komplexere *TCP* (Transmission Control Protocol) hat zwar einen deutlich größeren Overhead (Daten-Mehraufwand durch Verwaltungsinformationen) als UDP, stellt aber dafür einen zuverlässigen Transportdienst dar: Es wird eine virtuelle Verbindung zwischen den beiden Hosts hergestellt. Sie besteht darin, dass die Datenpakete durchnummeriert werden und eine Übertragungskontrolle und eventuelle Neuübertragung jedes einzelnen Pakets stattfinden. Ob eine Anwendung nun UDP oder TCP verwendet, ist ihre eigene Entscheidung. Allgemein benutzen Dienste, die kontinuierlich größere Datenmengen transportieren müssen, eher TCP, während etwa Verwaltungs- und Konfigurationsdienste zu UDP tendieren.

   Im Vergleich zum OSI-Modell nimmt die Host-zu-Host-Transportschicht insbesondere die Aufgaben der OSI-Transportschicht wahr; je nach konkretem Protokoll können auch Funktionen der Sitzungsschicht ausgemacht werden.

   Der Begriff *Host* (Gastgeber) bezeichnet übrigens jeden Computer, der an ein Netzwerk angeschlossen ist und mit anderen Geräten kommuniziert. Es ist keine Bezeichnung für einen expliziten Dienstleistungsrechner, dieser (oder vielmehr die darauf ausgeführte Software) wird *Server* genannt. Der Host muss lediglich vom Router abgegrenzt werden, der Pakete nicht für sich selbst entgegennimmt, sondern an andere Netze weiterleitet. Da das Routing jedoch eine Ebene weiter unten stattfindet, ist es ein auf dieser Schicht unsichtbares Detail – die Transportschicht ist nur für Rechner relevant, die Daten für den Eigenbedarf benötigen.[3]

3 Router verständigen sich allerdings auch mithilfe der im weiteren Verlauf dieses Kapitels vorgestellten Routing-Protokolle miteinander; diese Protokolle werden durchaus auf der Host-zu-Host-Transportschicht ausgeführt. Trotzdem ist dieses technische Detail für die eigentlichen Anwendungshosts uninteressant und unsichtbar.

4. Die Anwendungsschicht (*Application Layer*) schließlich definiert die Kommunikation zwischen den Anwendungsprogrammen auf den einzelnen Rechnern; hier arbeiten Protokolle wie HTTP für Webserver, FTP zur Dateiübertragung oder SMTP für den E-Mail-Versand. Die Schicht entspricht im Wesentlichen der gleichnamigen obersten Schicht des OSI-Referenzmodells, wobei auch einige Komponenten von dessen Darstellungsschicht mit hineinspielen. Beispielsweise bedarf HTML-Code, der von einer Webserver-Anwendung ausgeliefert wird, der Interpretation durch einen Browser; hier entspräche der HTML-Code selbst eher der Darstellungsschicht, die Browseranwendung aber der OSI-Anwendungsschicht. Sitzungsmanagement ist dagegen vom ursprünglichen Design her gar nicht vorgesehen; falls es benötigt wird, muss es durch die Anwendungen selbst bereitgestellt werden. In Kapitel 19, »Webserveranwendungen«, erfahren Sie beispielsweise, wie Sie mithilfe der Programmiersprache PHP Websessions verwalten.

4.2.3 Netzwerkkommunikation über die Schichten eines Schichtenmodells

In diesem Abschnitt wird erläutert, wie die Kommunikation über die Schichten von Schichtenmodellen funktioniert. Dazu werden zwei Beispiele angeboten: Das erste ist ein Alltagsbeispiel, das mit Computernetzwerken nichts zu tun hat, während das zweite ein einfaches Beispiel der Netzwerkkommunikation darstellt.

Ein Alltagsbeispiel

Neben der Datenübertragung im Netzwerk lassen sich auch völlig andere Arten der Kommunikation in Schichten gliedern. Beispielsweise kann die Kommunikation zwischen Gesprächspartnern am Telefon folgendermaßen unterteilt werden:

1. Die beiden Telefone sind physikalisch über eine Telefonleitung miteinander verbunden.
2. Die Verbindung zwischen den Telefonanschlüssen kommt dadurch zustande, dass einer der beiden Teilnehmer die eindeutige Nummer des anderen wählt und der andere das Gespräch annimmt.
3. Über die Telefonleitung werden Informationen in Form von elektromagnetischen Impulsen übertragen, beim klassischen Telefonnetz analog, bei ISDN, Mobilfunk oder VoIP dagegen digital.
4. An den beiden Endpunkten der Kommunikation sprechen die Gesprächspartner in ihre jeweilige Sprechmuschel hinein; die akustischen Signale werden in elektromagnetische Impulse umgewandelt (bei den digitalen Varianten kommt noch die Analog-Digital-Wandlung hinzu). Umgekehrt hört ein Teilnehmer aus der Hörmuschel wiederum akustische Signale, die aus den übertragenen Impulsen zurückverwandelt wurden.
5. Die akustischen Signale, die die Gesprächspartner miteinander austauschen, werden zu Silben, Wörtern und schließlich Sätzen kombiniert.

6. Aus den einzelnen Bestandteilen der Sprache ergibt sich schließlich der eigentliche Inhalt der Nachrichten, die miteinander ausgetauscht werden.

Möglicherweise besteht dieses Kommunikationsmodell sogar aus noch mehr Schichten: Angenommen, die beiden Gesprächspartner sind leitende Angestellte oder gar Direktoren eines großen Unternehmens. Dann wird in aller Regel auf beiden Seiten eine Sekretärin die Gesprächsvermittlung vornehmen, möglicherweise ist sogar noch eine Telefonzentrale involviert. Noch komplizierter wird es beispielsweise, wenn Dolmetscher mitwirken.

Dieses einfache Beispiel zeigt deutlich, dass alle Ebenen, die sich oberhalb der untersten, also der physikalischen, Ebene befinden, nur Abstraktionen darstellen, durch die den übertragenen Signalen jeweils ein neuer Sinnzusammenhang zugeordnet wird. Die eigentliche Verbindung erfolgt nämlich stets nur auf dieser untersten Ebene! Für jedes Schichtenmodell gilt daher zusammenfassend Folgendes:

Die Daten, die über einen Kommunikationskanal übertragen werden sollen, werden auf der Seite des Senders zunächst Schicht für Schicht nach unten weitergereicht und jeweils mit den spezifischen Zusatzinformationen für diese Schicht versehen. Schließlich werden sie über die unterste Schicht, die eigentliche physikalische Verbindung, übertragen. Auf der Empfängerseite werden sie dann wieder schichtweise nach oben weitergeleitet. Jede Schicht ermittelt die für sie bestimmten Informationen und regelt die Weiterleitung an die nächsthöhere Schicht.

Was in dem Telefonbeispiel geschieht, wird schematisch in Abbildung 4.1 dargestellt: Während die Gesprächspartner den Eindruck haben, in einem direkten Gespräch miteinander zu kommunizieren, geschieht in Wirklichkeit etwas erheblich Komplexeres: Die gesprochene Sprache (die eigentlich aus Silben beziehungsweise einzelnen Lauten besteht, die letztlich einfach nur Schallwellen sind) wird in eine andere Form von Information umgewandelt, über eine elektrische Leitung übertragen und auf der Empfängerseite wieder zusammengesetzt.

Wichtig ist außerdem, dass jede Schicht immer nur die für sie selbst bestimmten Informationen auswertet. Beim Empfang von Daten haben die niedrigeren Schichten überhaupt erst dafür gesorgt, dass die Informationen auf der entsprechenden Schicht angekommen sind, die Spezialinformationen der höheren Schichten sind der aktuellen Schicht dagegen unbekannt. Sie muss lediglich anhand ihrer eigenen Informationen dafür sorgen, dass die Daten an die korrekte Stelle einer höheren Schicht ausgeliefert werden. Auf diese Weise ist jede Schicht virtuell mit der Schicht der gleichen Stufe auf der anderen Seite verbunden; das tatsächliche Zustandekommen dieser Verbindung ist für die jeweilige Schicht dagegen absolut unsichtbar.

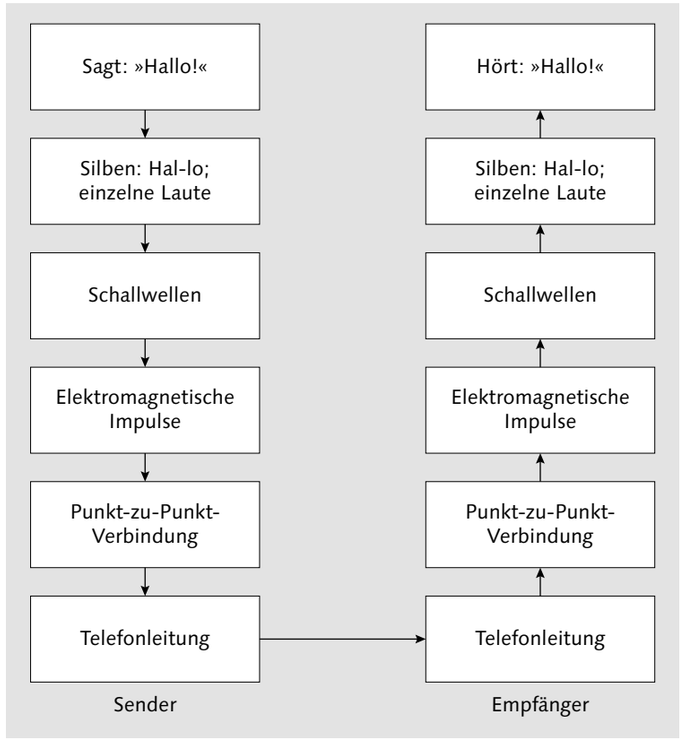


Abbildung 4.1 Schichtenmodell eines Telefongesprächs. Die tatsächliche Verbindung besteht nur auf der Ebene der Telefonleitung!

Ein Netzwerkbeispiel

Dieses zweite Beispiel – der Versand einer E-Mail an den Rheinwerk Verlag – zeigt, wie sich die beim Telefonbeispiel erläuterten Sachverhalte wieder auf Netzwerke übertragen lassen. Schematisch geschieht Folgendes:

1. In meinem E-Mail-Programm, zum Beispiel Mozilla Thunderbird, verfasse ich den eigentlichen Inhalt der Mail, als Empfänger setze ich *info@rheinwerk-verlag.de* ein. Nachdem ich alles fertig geschrieben habe, drücke ich auf den Absendebutton.
2. Da eine E-Mail eine in sich geschlossene Dateneinheit darstellt, die in ihrer ursprünglichen Reihenfolge beim Empfänger ankommen muss, wird der Transport durch das TCP-Protokoll übernommen, dessen eingebaute Datenflusskontrolle dafür sorgt, dass alle Daten vollständig und in der richtigen Reihenfolge übertragen werden.
3. Die Datenpakete, die durch das TCP-Protokoll angelegt wurden, werden nun durch das IP-Protokoll mit der korrekten Absender- und Empfängeradresse versehen. Diese Adressen

haben nichts mit den nur auf der Anwendungsebene wichtigen E-Mail-Adressen zu tun. Vielmehr geht es darum, dass mein Rechner die Daten an den zuständigen Mailserver beziehungsweise an einen Vermittlungsrechner versendet.

4. Die fertig adressierten Datenpakete werden nun dem eigentlichen physikalischen Netzwerk anvertraut und entsprechend übertragen.

Auf der Empfängerseite – also auf dem Serverrechner, der das Postfach *info@rheinwerk-verlag.de* verwaltet – kommen die Daten dann folgendermaßen an:

1. Über die physikalische Netzwerkverbindung des Serverrechners treffen Datenpakete ein.
2. Auf der Ebene des IP-Protokolls werden die Datenpakete nach den zuständigen Transportdiensten sortiert und an diese weitergereicht – die E-Mail wird dem TCP-Dienst übergeben.
3. Der TCP-Dienst stellt fest, dass die entsprechenden Datenpakete für den Mailserver (gemeint ist das Programm, nicht der Rechner selbst) bestimmt sind, und reicht sie an diesen weiter.
4. Der Mailserver wertet die E-Mail-Adresse des Empfängers aus und speichert die Mail in dem Postfach *info@rheinwerk-verlag.de*. Dort kann sie jederzeit vom berechtigten Empfänger abgeholt werden.

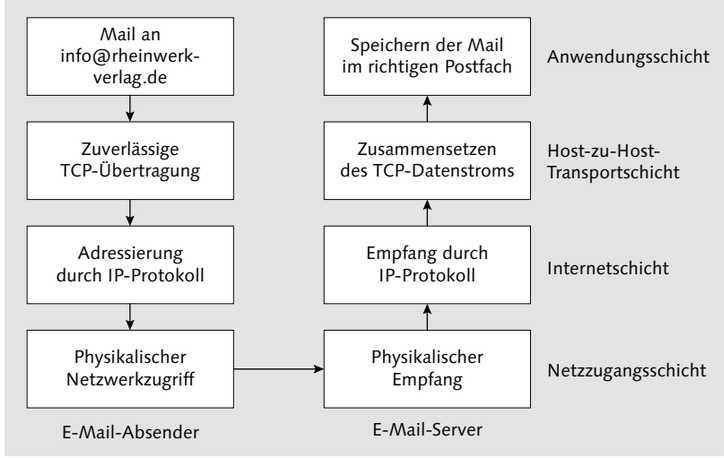


Abbildung 4.2 Übertragung einer E-Mail vom E-Mail-Programm des Absenders auf den Postfachserver des Empfängers. In der Regel sind allerdings mehrere Zwischenstationen beteiligt.

Die Übertragung der E-Mail vom empfangenden Server an das E-Mail-Programm des eigentlichen Empfängers funktioniert im Großen und Ganzen genauso, obwohl auf der Anwendungsebene ein anderes Protokoll zum Einsatz kommt.

Abbildung 4.2 zeigt noch einmal schematisch, wie die Übertragung funktioniert. Mehr über die grundlegenden E-Mail-Protokolle erfahren Sie in Abschnitt 4.6.5, »Verschiedene Internetanwendungsprotokolle«.

4.3 Klassifizierung von Netzwerken

Nachdem Sie nun mithilfe der Schichtenmodelle eine Möglichkeit kennengelernt haben, unterschiedliche Netzwerke in ihren Funktionen miteinander zu vergleichen, sollten Sie auch verstehen, worin sie sich unterscheiden. Es gibt diverse Unterscheidungsmerkmale, die zwar nicht genau den Schichten der Modelle entsprechen, aber doch ebenfalls mehrere Aspekte der einzelnen Netzwerke betreffen. Es handelt sich um die Unterscheidung nach der Reichweite des Netzwerks, der physikalischen Grundstruktur oder Topologie und zuletzt nach der zentralen oder dezentralen Verwendung des jeweiligen Netzes.

4.3.1 Die Reichweite des Netzwerks

Bei der Unterteilung der Netzwerke entsprechend der Reichweite – also nach der geografischen Größenordnung, die das Netzwerk überbrückt – werden insgesamt vier Stufen unterschieden:

- Das *Local Area Network* (LAN) – lokales Netzwerk – beschreibt ein Netzwerk, das an ein einzelnes zusammenhängendes Areal gebunden ist, also etwa einen Raum, ein Gebäude oder maximal ein zusammenhängendes (Firmen-)Gelände. LANs sind heutzutage in Wirtschaftsunternehmen, Schulen und Universitäten oder anderen Organisationen und Instituten weitverbreitet.
- Das *Metropolitan Area Network* (MAN) – Stadtgebietsnetzwerk – bezeichnet ein Netz, das eine Stadt, Gemeinde oder auch eine Region umfasst. Ein Beispiel wären die verschiedenen eigenen Netze von NetCologne in Köln. Die Ausdehnung für ein MAN liegt bei 100 km und mehr.
- Das *Wide Area Network* (WAN) – Fernnetzwerk – ist ein Netz, das mehrere Städte, eine ganze Region oder sogar ein ganzes Land umfasst. In Deutschland gibt es beispielsweise das Deutsche Forschungsnetz (DFN).
- Das *Global Area Network* (GAN) – weltweites Netzwerk – ist über mehrere Länder, einen ganzen Kontinent oder sogar die ganze Welt verbreitet. Das bei Weitem größte GAN ist heutzutage natürlich das Internet – im engeren Sinne ist ein GAN allerdings ein homogenes Netzwerk, während das Internet aus zahllosen Einzelnetzen mit unterschiedlichen Architekturen zusammengesetzt ist.

Es sei noch angemerkt, dass die drei Netzwerkarten, die größere Entfernungen überbrücken – also MAN, WAN und GAN – oftmals einfach unter dem Sammelnamen WAN zusammengefasst werden. Dies umso mehr, als alle drei Typen von Fernnetzen im Wesentlichen die gleiche Art von Technologie verwenden – oder genauer gesagt: Alle Arten von Technologien für Fernnetze werden von allen drei Netzarten verwendet.

Es gibt Fernnetze, die Wählleitungen, also einfache Telefonverbindungen, verwenden, sowohl das klassische Analog- als auch das digitale ISDN-Netz. In größeren Städten sind die diversen DSL-Dienste am verbreitetsten, bei denen durch die Verwendung besonders hochfrequenter Signale über die normalen Kupferdrähte der Telefonleitungen wesentlich höhere Datenübertragungsraten erzielt werden. Zu einer besonderen Form der Wählleitung zählen Verbindungen über die digitalen GSM-Mobilfunknetze und deren Nachfolger GPRS, EDGE, UMTS und LTE. Daneben existieren unterschiedliche Arten von Standleitungen, die für besonders häufig beanspruchte oder besonders zuverlässige Leitungen verwendet werden. Dabei gibt es unter anderem spezielle DSL-Standleitungen oder Glasfasernetze. Auch drahtlose Übertragung, etwa über Funk- oder Satellitenverbindungen, spielt eine immer größere Rolle.

Zu beachten ist allerdings, dass DSL, Wireless LAN und Mobilfunknetze im Wesentlichen die Technologien für den Zugang einzelner Hosts zu einem MAN oder WAN darstellen. Im Backbone-Bereich, also in der eigentlichen Netzwerkinfrastruktur, kommen vor allem Zeitmultiplexing-Verfahren über Glasfasernetze zum Einsatz. Sprache, Video und sonstige Daten werden dabei über Gigabit-Ethernet, SDH/SONET, ATM oder manchmal auch Frame-Relay übertragen. Eine zunehmende Bedeutung erlangten in den letzten Jahren auch DWDM-Verfahren (Dense Wavelength Division Multiplexing). Dabei werden über ein und denselben Lichtwellenleiter mehrere Signale mit unterschiedlicher Wellenlänge gleichzeitig versandt, was für extrem hohe Datenraten sorgt.

Lokale Netzwerke verwenden ebenfalls viele unterschiedliche technische Übertragungsarten. Allein für Ethernet, die häufigste Form der lokalen Vernetzung, werden unterschiedliche Arten von Koaxial-, Twisted-Pair- oder Glasfaserkabeln benutzt. Diese zeichnen sich durch verschiedene Übertragungsgeschwindigkeiten, mögliche maximale Entfernungen und natürlich auch unterschiedliche Kosten aus. Wireless LAN – der Betrieb von lokalen Netzwerken ohne Kabel über Funk, Infrarot oder Mikrowellen – erfreut sich auch zunehmender Beliebtheit. Abgesehen davon, gibt es neben Ethernet viele andere Formen lokaler Netzwerke.

Die technologischen Grundlagen der Verkabelung, der Signalübermittlung und des Netzzugangs werden in Abschnitt 4.4, »Netzwerkkarten, Netzwerkkabel und Netzzugangsverfahren«, ausführlicher besprochen.

4.3.2 Die Netzwerktopologie

Die *Topologie* eines Netzwerks beschreibt, in welcher physikalischen Grundform die einzelnen Geräte organisiert sind. Manche Arten von Netzwerkhardware setzen eine bestimmte Topologie voraus, andere überlassen dem Einrichtenden die Entscheidung zwischen mehreren Möglichkeiten. Topologie ist normalerweise eine Eigenschaft lokaler Netzwerke oder gar einzelner Netzsegmente. Die meisten Fernnetze verbinden ohnehin nicht einzelne Rechner, sondern ganze Netzwerke an unterschiedlichen Orten miteinander.

Es werden im Wesentlichen folgende Grundformen unterschieden:

- Die *Bustopologie* beschreibt ein Netzwerk, bei dem die einzelnen Knoten (Anschlüsse) hintereinander an einem einzelnen Kabelstrang angeschlossen sind, dessen Enden nicht miteinander verbunden werden dürfen (Sonst würde es sich um eine Ringtopologie handeln!). Häufig werden die beiden Enden des Kabelstrangs durch Abschlusswiderstände (Terminatoren) abgeschlossen. Ein Beispiel für echte busförmige Netzwerke ist Ethernet über Koaxialkabel.
- Die *Sterntopologie* ist die Form eines Netzes, bei dem alle Knoten mit jeweils eigenem Kabel an einem zentralen Gerät miteinander verbunden werden. Dieses zentrale Bindeglied heißt, je nach seiner genauen Funktionsweise, *Hub* oder *Switch*. Die Sterntopologie wird zum Beispiel von Ethernet über Twisted-Pair-Kabel verwendet.
- Die *Ringtopologie* ähnelt der Bustopologie insofern, als auch hier alle Knoten an einem zentralen Strang aufgereiht sind. Dieser zentrale Kabelstrang bildet jedoch einen geschlossenen Ring. Daraus ergibt sich automatisch eine Datenstromrichtung, in die die Datenpakete grundsätzlich weitergereicht werden. Bekanntestes Beispiel der ringförmigen Vernetzung ist *Token Ring*.
- Die *Baumtopologie* schließlich ist eher ein Standard für den Zusammenschluss verschiedener Netzsegmente. Von einem zentralen Kabelstrang, gewissermaßen dem »Stamm« des Baums, gehen nach beliebigen Richtungen einzelne Verästelungen ab, an denen entweder eine einzelne Station oder ein ganzes Netz hängt.

Wichtig ist zu guter Letzt, dass ein Unterschied zwischen einer physikalischen und einer logischen Topologie bestehen kann, denn die äußere Form der Verkabelung (physikalische Topologie) kann einfach aus praktischen Erwägungen heraus gewählt worden sein, obwohl von der Funktion her eine völlig andere Struktur herrscht, nämlich die logische Topologie.

Ein gutes Beispiel für eine unterschiedliche physikalische und logische Struktur sind neuere Token-Ring-Varianten: Die eigentliche Vernetzung erfolgt sternförmig, logisch gesehen, handelt es sich jedoch um einen Ring. Auch Ethernet über Twisted-Pair-Kabel verwendet – physikalisch gesehen – die Sterntopologie, die logische Funktionsweise hängt von der Art des zentralen Verteilers ab: Ein Hub erzeugt letztlich die Funktion eines busförmigen Netzes, da es einen durchgehenden Strang enthält, an dem alle Stationen angeschlossen sind. Ein Switch dagegen stellt jeweils eine gesonderte Verbindung zwischen zwei Stationen her, die miteinander Daten austauschen; mithin handelt es sich hier auch logisch um die echte Sternform.

4.3.3 Der Zentralisierungsgrad des Netzwerks

Ein weiteres wichtiges Kriterium bei der Einteilung von Netzwerken in unterschiedliche Gruppen ist die Frage nach der Arbeitsaufteilung in ihnen. Kleine Arbeitsgruppen, die jeweils mit ihren Arbeitsplatzrechnern untereinander Dateien austauschen möchten, haben hier

sicherlich andere Bedürfnisse als riesige Organisationen, in denen Tausende von Anwendern auf bestimmte Datenbestände zugreifen müssen. Deshalb werden die sogenannten *Client-Server-Netzwerke*, in denen zentrale Dienstleistungsrechner, die Server, arbeiten, von den Peer-to-Peer-Netzwerken unterschieden, in denen die einzelnen Computer gleichberechtigt Ressourcen freigeben und verwenden können.

- Das *Client-Server-Netzwerk* unterscheidet generell zwei Arten von beteiligten Rechnern: Der Server (Dienstleister) ist ein Computer, der den Arbeitsstationen der einzelnen Anwender an zentraler Stelle Ressourcen und Funktionen zur Verfügung stellt; der Client (Kunde) nimmt diese Dienstleistungen in Anspruch. Die Dienste, die von Servern angeboten werden, sind sehr vielfältig: Sie reichen vom einfachen Dateiserver, der Dateien im Netzwerk verteilt oder Festplattenplatz für andere freigibt, über Druckserver, Mail- und andere Kommunikationsserver bis hin zu ganz speziellen Diensten wie Datenbank- oder Anwendungsserver.
- Das *Peer-to-Peer-Netzwerk* besteht aus prinzipiell gleichberechtigten Arbeitsplatzrechnern (*peer* heißt etwa »Kollege«). Jeder Anwender ist in der Lage, Ressourcen seines eigenen Rechners an andere im Netzwerk freizugeben. Das heißt, dass alle Rechner im Netz bis zu einem gewissen Grad Serverdienste wahrnehmen.

In der Praxis sind allerdings Mischformen häufiger anzutreffen als reine Client-Server- oder absolute Peer-to-Peer-Netze. Beispielsweise könnte man sich in einem Unternehmen die folgende Situation vorstellen: Aufgaben wie die direkte Kommunikation (E-Mail), der Zugang zum Internet (über einen Proxyserver oder einfach einen Router) oder Lösungen zum Backup (Datensicherung) werden durch zentrale Server zur Verfügung gestellt; der Zugang zu Dateien innerhalb der Abteilungen oder auf Drucker der Kollegen innerhalb eines Büros wird dagegen im Peer-to-Peer-Verfahren, unter Umgehung von Servern, geregelt.

Wichtig ist außerdem, zu verstehen, dass die Begriffe *Client* und *Server* im engeren Sinne nicht unbedingt spezifische Rechner, sondern besondere Softwarekomponenten bezeichnen.

Ein *Server* ist einfach ein Programm, das meist automatisch gestartet wird und im Hintergrund darauf »lauert«, irgendeine Dienstleistung zur Verfügung zu stellen. Allgemeiner werden solche Programme zum Beispiel im Unix-Umfeld als *Daemon* bezeichnet, unter Windows NT und seinen Nachfolgern (Windows 2000, XP, Vista, Windows 7, 8 und 10) heißen sie Dienst (*Service*). Grundsätzlich kann ein solcher Serverdienst auf jedem beliebigen Rechner laufen – vorausgesetzt natürlich, er ist für die Hardwareplattform und das Betriebssystem dieses Rechners bestimmt. Der Grund für den Einsatz besonders leistungsfähiger Hardware (eben der Serverhardware) und spezialisierter Betriebssysteme liegt einfach in ihrer höheren Belastbarkeit, wenn viele Benutzer gleichzeitig diese Dienste benötigen.

Ein *Client* ist zunächst eine Software, die in der Lage ist, mit der Serversoftware zu kommunizieren; üblicherweise stellt sie dem Benutzer auch eine Schnittstelle zur Verfügung, um diese Kommunikation in Anspruch zu nehmen. So ist beispielsweise ein Webbrowser ein Client für das HTTP-Anwendungsprotokoll, er kommuniziert also mit HTTP-Servern. Interessanter-

weise können Webserver und Browser auch beide auf demselben Rechner laufen. Dies ist nützlich, um Webanwendungen zunächst lokal auszuprobieren.

Arten von Servern

Im Folgenden sollen einige Serverarten genauer vorgestellt werden. Sie sollten auf jeden Fall das zuvor Gesagte im Hinterkopf behalten: Es spielt überhaupt keine Rolle für die allgemeine Funktion, ob ein Serverdienst, also die Software, die diesen Dienst zur Verfügung stellt,

- mit anderen Diensten zusammen auf dem gleichen Rechner läuft,
- allein auf einem separaten Serverrechner ausgeführt wird oder
- sogar auf mehrere Server verteilt ist, weil ansonsten die Belastung zu groß wäre.

Letzteres ist insbesondere im Bereich öffentlicher WWW-Server sehr häufig zu finden, da populäre Sites wie etwa Suchmaschinen, Nachrichtenportale oder große Webshops sehr viel Datenverkehr zu verkraften haben. Hier werden sogenannte *Load-Balancing-Systeme* eingesetzt, die die hereinstürmenden Anfragen automatisch möglichst gerecht auf mehrere physikalische Server verteilen.

Im Wesentlichen gibt es die folgenden wichtigen Arten von Serverdiensten:

- Fileserver
- Printserver
- Mailserver
- Webserver
- Verzeichnisdienst-Server
- Anwendungsserver und Serveranwendungen

In den folgenden Abschnitten wird jeder dieser Servertypen kurz vorgestellt; in späteren Kapiteln erhalten Sie auch konkrete Beispiele für viele von ihnen.

Fileserver

Der Fileserver (Dateiserver) stellt anderen Rechnern im Netzwerk freigegebene Verzeichnisse zur Verfügung. Auf diese Weise können sich die Anwender über einen zentralen Austauschpunkt gegenseitig Dateien zukommen lassen. Der Fileserver ist relativ stark an ein bestimmtes Betriebssystem oder eine Plattform gebunden. Erst allmählich setzen sich neuere Möglichkeiten durch, die in der Lage sind, auch unterschiedliche Rechner gleichzeitig zu bedienen. Denn die Besonderheit eines Fileservers ist, dass die Benutzer ihn völlig transparent genau so benutzen können wie die lokalen Dateisysteme ihres Arbeitsplatzrechners. In einem idealen (lokalen) Netzwerk sollte es dem normalen Anwender vollkommen egal sein, ob seine Dateien am Arbeitsplatz oder auf einem Fileserver zu finden sind.

Sehr wichtig ist im Zusammenhang mit Fileservern die Verwaltung von Zugriffsrechten, da nicht jede Datei für alle Benutzer gedacht ist.

Der Internetdienst FTP (File Transfer Protocol) ist übrigens kein vollwertiger Fileserver, sondern dient lediglich der einfachen Dateiübertragung. Die Informationen über die Dateien des entfernten Rechners sind nicht vollständig genug, um das Äquivalent eines Dateisystems abzubilden.

Informationen über Fileserver für die verschiedenen Systemplattformen finden Sie in Kapitel 6, »Windows«, Kapitel 7, »Linux«, und Kapitel 8, »OS X«. FTP wird dagegen zusammen mit anderen Arten von Internetservern in Kapitel 15, »Weitere Internet-Serverdienste«, behandelt.

Printserver

Der Printserver (oder Druckserver) erlaubt mehreren Anwendern beziehungsweise Arbeitsstationen den gemeinsamen Zugriff auf einen Drucker. Die größte Herausforderung besteht darin, den einzelnen Arbeitsstationen automatisch den passenden Druckertreiber für ihr jeweiliges Betriebssystem zur Verfügung zu stellen, sodass diese den Drucker einfach verwenden können, ohne dass der Treiber zuvor noch einmal lokal installiert werden müsste.

Der Betrieb von Printservern ist besonders in Windows-Netzwerken weitverbreitet, da hier der Drucker gewöhnlich über ein USB-Kabel an einen einzelnen Rechner angeschlossen wird. Dieser Rechner wird dann so eingerichtet, dass er den Zugriff auf den Drucker auch den anderen Computern erlaubt.

Bei anderen Plattformen gibt es das Problem in dieser Form seltener. In klassischen Macintosh-Netzwerken ist es beispielsweise üblich – und viel bedienungsfreundlicher –, den Drucker unmittelbar per Ethernet ans Netzwerk anzuschließen, denn damit ist er automatisch für alle freigegeben.

In heterogenen Netzen war es bis vor wenigen Jahren verhältnismäßig schwierig, über Betriebssystemgrenzen hinweg gemeinsam auf einen Drucker zuzugreifen. Inzwischen ist jedoch beispielsweise das Drucksystem CUPS für alle Unix-Varianten verfügbar, das sogar Windows-Clients relativ problemlos bedienen kann.

Da Druckserver, genau wie Dateiserver, an das jeweilige Betriebssystem gebunden sind, werden die wichtigsten Varianten in Kapitel 6, »Windows«, und Kapitel 7, »Linux«, beschrieben.

Mailserver

Ein Server für elektronische Post (E-Mail) muss nicht immer bei einem Internetprovider installiert sein, sondern kann auch im lokalen Netz seinen Dienst verrichten. Denn erstens ist es in Unternehmen oder Organisationen oft von Vorteil, wenn die Mitarbeiter untereinander per E-Mail kommunizieren können, und zweitens ist es manchmal schon allein deshalb erforderlich, einen internen Mailserver zu betreiben, weil der Zugang zum Internet aus

Sicherheitsgründen stark eingeschränkt ist und etwa die Kommunikation eines Arbeitsplatzrechners mit einem externen Mailserver gar nicht zulässt.

Obwohl im Lauf der Netzwerkentwicklungsgeschichte verschiedene Formen der elektronischen Post entstanden sind, gibt es heute eigentlich keine Alternative mehr zu Internet-E-Mail. Diese verwendet verschiedene Serverdienste zum Senden und Empfangen der E-Mail: Das SMTP-Protokoll (Simple Mail Transport Protocol) bestimmt, wie zu versendende E-Mails zu transportieren sind; POP3 (Post Office Protocol Version 3) oder das modernere, komfortablere IMAP (Internet Message Access Protocol) beschreiben ein Benutzerkonto (Postfach) für eingehende E-Mails sowie den Vorgang der »Abholung«.

Rein theoretisch kann Internet-E-Mail direkt zum einzelnen Host gesendet werden. Das ist aber insofern problematisch, als normale Arbeitsplatzrechner manchmal ausgeschaltet werden und private Einzelplatzrechner meist nur temporär über Wählleitungen mit dem Internet verbunden sind. Dies ist überhaupt der wichtigste Grund dafür, warum sich Posteingangsserver etabliert haben, auf denen die Mail für einen bestimmten Anwender im Prinzip vorgehalten wird, bis dieser sie abruft.

Da die gewöhnliche Form der E-Mail auf den Standard-Internetprotokollen aufsetzt, gibt es übrigens kein Problem, sie plattform- und betriebssystemübergreifend zu verwenden.

Webserver

Ein Webserver (die exakte Bezeichnung ist eigentlich *HTTP-Server*) liefert auf Anfrage Webseiten über ein Netzwerk aus. In der Regel ist dieses Netzwerk das Internet. In den lokalen Netzen von Unternehmen und Institutionen setzt sich diese Form der Informationsübermittlung aber auch immer mehr durch. Ein solches lokales Netz, das Technologien und Dienste der Internetprotokolle verwendet, wird *Intranet* genannt. Der Anwender verwendet ein Anzeigeprogramm für Webseiten, den sogenannten *Browser*, um Webseiten anzufordern, zu betrachten und auch, um die enthaltenen Hyperlinks – also Verknüpfungen zu anderen Dokumenten auf dem gleichen oder einem anderen Server – per Mausklick zu folgen.

Webseiten sind prinzipiell Textdokumente, die in der Strukturierungssprache HTML geschrieben werden. Viele dieser Dokumente liegen statisch auf dem Server und werden einfach auf Anfrage ausgeliefert. Eine wachsende Anzahl solcher Dokumente wird aber auch aus Vorlagen und dynamischen Daten, etwa aus einer Datenbank, kombiniert und dann an den anfragenden Host geschickt. Diese Entwicklung ist für Websites mit umfangreichem, schnell wechselndem Inhalt, etwa Online-Tageszeitungen oder die Kataloge in E-Commerce-Sites, unvermeidlich.

Webserver sind im Übrigen schon von ihrer Grundidee her dafür gedacht, Clients unter vielen verschiedenen Betriebssystemen zu bedienen. Falls es Inkompatibilitäten geben sollte, liegt das höchstens daran, dass bei der Erstellung des HTML-Codes Steuerbefehle verwendet wurden, die nicht jeder Browser versteht.

Der praktische Einsatz eines Webservers wird in Kapitel 14, »Server für Webanwendungen«, am wichtigsten Beispiel Apache beschrieben; die Kapitel 18, »Webseitenerstellung mit (X)HTML und CSS«, bis 20, »JavaScript und Ajax«, kümmern sich dagegen um die Erstellung von Webinhalten und -anwendungen.

Verzeichnisdienst-Server

Verzeichnisdienste (*Directory Services*) gewinnen in der IT seit längerer Zeit stark an Bedeutung. Ein Verzeichnis ist in diesem Zusammenhang kein Dateisystem, sondern ein datenbankähnlicher, standardisierter Katalog von Benutzern, Computern, Peripheriegeräten und Rechten in einem Netzwerk. Durch den Eintrag in das Verzeichnis können diese Informationen netzwerkweit abgerufen werden, sodass Verzeichnisdienste eine praktische Grundlage für zahlreiche Dienste legen, die in einer größeren Netzwerkumgebung die Arbeit der Administratoren und das Leben der Anwender erleichtern. Hier nur einige Beispiele:

- automatisierte Softwareverteilung und -installation
- mobile Benutzerprofile (Roaming User Profiles)
- zentralisierte Anmeldedienste (Single Sign-on)
- rechner-, benutzer- und eigenschaftsbasierte Rechtekontrolle

In Kapitel 15, »Weitere Internet-Serverdienste«, wird OpenLDAP als Praxisbeispiel für einen Verzeichnisdienst vorgestellt.

Anwendungsserver und Serveranwendungen

Ein Anwendungsserver (*Application Server*) erlaubt den Benutzern die Verwendung von Anwendungsprogrammen, die sich eigentlich auf dem Server befinden, über das Netzwerk.

Bei der einfachsten Form des Anwendungsservers liegt der Datenbestand der Anwendung auf den Datenträgern des Servers, die Anwendung wird über das Netzwerk in den Arbeitsspeicher des Clients geladen und dort lokal ausgeführt. Der Unterschied zum Fileserver ist hier minimal: Es muss der Anwendung lediglich klar sein, dass eventuell notwendige Zusatzkomponenten oder Konfigurationsdaten nicht auf dem Rechner liegen, auf dem sie ausgeführt wird, sondern auf der Maschine, von der sie geladen wurde.

Bei vielen normalen Einzelplatz-Anwendungsprogrammen kann eine solche Einstellung vorgenommen werden. Diese Verwendung von Software hat vor allem zwei Vorteile: Erstens kann es weniger Arbeit bedeuten, ein Programm einmal auf dem Server statt auf mehreren Arbeitsplatzrechnern zu installieren, und zweitens können Kosten gespart werden – die meisten Softwarelizenzen gelten jeweils pro Rechner, auf dem das jeweilige Programm installiert ist. Wird eine Anwendung auf mehreren Rechnern genutzt, aber nicht gleichzeitig, kann die Software auf dem Server installiert werden; damit werden die Lizenzgebühren dann nur einmal fällig.

Bei komplexeren Formen von Anwendungsservern werden Teile des Programms – oder unter Umständen auch das ganze Programm – direkt auf dem Server ausgeführt. Die möglichen Gründe dafür sind im Einzelfall genauso vielfältig wie die verschiedenen Formen der Umsetzung. Beispielsweise ist es bei großen Datenbanken üblich, dass der Datenbestand als solcher auf einem Server liegt, ebenso die grundlegende Datenverwaltungssoftware. Auf den Clients existieren dann in der Regel sogenannte *Frontends*, also Softwarekomponenten, die den Benutzern eine Bedienoberfläche für die eigentliche Datenbank bereitstellen. (Den Gegenbegriff zum Frontend bildet das *Backend*, wobei es sich um einen nur für spezielle, angemeldete Benutzer zugänglichen Teil des Clients handelt, der der Verwaltung des Servers dient.)

Noch einen Schritt weiter gehen die sogenannten *verteilten Anwendungen* oder *Enterprise-Anwendungen*. Sie basieren in der Regel auf einem oder mehreren Datenbankservern für den Datenbestand, einem Anwendungsserver für die Geschäftsabläufe und diversen Client-Frontends (sowohl native Programme für bestimmte Betriebssysteme als auch Webanwendungen).

Eine andere Form der Serveranwendung existiert bei der Verwendung der sogenannten *Terminal-Server*. Die einfachste Form, der Internetdienst Telnet, stellt dem Anwender eine Konsolenoberfläche zur Verfügung, über die sich von fern auf dem Server selbst mithilfe von Kommandoeingabe arbeiten lässt. Das heißt, die Ein- und Ausgabe zeilenorientierter Kommandos und Anwendungsprogramme erfolgt auf dem Client, die eigentliche Ausführung auf dem Server – der eigene Rechner wird so zu einem Terminal für den entfernten Server.

Eine sehr merkwürdige Form der Serversoftware ist in diesem Zusammenhang der aus dem Unix-Bereich stammende X-Window-Server oder einfach X-Server (siehe Kapitel 7, »Linux«). Die Bezeichnung *Server* für diese Software erscheint zunächst sehr irreführend, handelt es sich doch einfach um die Grundlage der grafischen Benutzeroberfläche (GUI) unter Unix. Der X-Server stellt den Anwendungsprogrammen seine Dienste zur Verfügung, die darauf zugreifen, um Fenster und andere Komponenten des GUIs darzustellen.

Die Tatsache, dass hier ein Dienst verfügbar gemacht wird, ist es übrigens, die den Begriff *Server* rechtfertigt. Dabei müssen Anwendung und X-Server auch nicht unbedingt auf dem gleichen Rechner laufen. Erstaunlicherweise läuft aber der X-Server auf dem Anwendungsclient! Denn da die Programmausführung auf dem entfernten Rechner stattfindet, aber die grafische Darstellung auf dem lokalen Rechner, muss dieser Dienst hier angeboten werden.

Terminal-Server gibt es auch unter Windows Server 2012 R2, dem angekündigten Server 2016 und anderen Microsoft-Systemen; auch hier läuft die eigentliche Anwendung auf dem Server, der Client erlaubt deren Bedienung und Anzeige. Das Angebot solcher Anwendungsdienste über das Internet wird allmählich beliebter. Ein ASP (Application Service Provider) lässt Anwendungen wie beispielsweise Bürosoftware auf seinen Servern laufen; über eine spezielle Clientsoftware oder sogar über einen Webbrowser kann der Kunde darauf zugreifen und die angebotene Software von der ganzen Welt aus benutzen. Eine wesentlich einfachere Form solcher Serveranwendungen, die über das Web verwendet werden und die Sie

wahrscheinlich gut kennen, ist der weitverbreitete webbasierte E-Mail-Dienst mit diversen Zusatzfunktionen, wie ihn GMX, Google Mail oder WEB.DE anbieten.

Die Quintessenz der Verwendung von Anwendungsservern ist die von einigen Firmen (Sun Microsystems, Oracle) seit Jahren angestrebte Abschaffung der gewöhnlichen Personal Computer und deren Ersatz durch sogenannte *Thin Clients* – Rechner ohne Festplatte, die ihr Betriebssystem und die Anwendungsprogramme vollständig aus dem Netzwerk oder aus dem Internet beziehen. Allerdings konnte sich das Konzept bisher nicht recht durchsetzen. Das Hauptargument der entsprechenden Unternehmen, nämlich die geringeren Kosten, lässt sich angesichts des massiven Preisverfalls bei den »ausgewachsenen« PCs nicht aufrechterhalten.

Inzwischen sind es auf der Clientseite eher die Tablets, die den gewöhnlichen PCs den Rang ablaufen. Zudem bieten immer mehr Hardware- und Betriebssystemhersteller bereits ab Werk Cloud-Dienste zur Daten- und Anwendungsspeicherung.

4.4 Netzwerkkarten, Netzwerkkabel und Netzzugangsverfahren

Im Laufe der Entwicklungsgeschichte der Netzwerke, die in diesem Kapitel bereits skizziert wurde, haben sich viele verschiedene Formen der Netzwerkhardware entwickelt. Jede von ihnen hatte zum Zeitpunkt ihrer Entstehung ihre Berechtigung, und dennoch haben sich einige auf breiter Front durchgesetzt, während andere schnell wieder vom Markt verschwunden sind. Die verbreitetste Art der Netzwerkhardware ist heute Ethernet in seinen vielfältigen Varianten.

Analog zu den zuvor beschriebenen Schichtenmodellen – vor allem dem standardisierten OSI-Referenzmodell – gibt es auch Standards, die speziell die Netzwerkhardware und den Netzzugang betreffen, also die beiden untersten Ebenen des OSI-Modells. Die umfangreichste Sammlung ist IEEE 802 des Institute of Electrical and Electronical Engineers. Die Nummer 802 bezeichnet Jahr und Monat der ursprünglichen Festlegung, nämlich den Februar 1980. Innerhalb dieser Sammlung existiert eine Reihe verschiedener Unterstandards beziehungsweise Arbeitsgruppen. Zu den wichtigsten gehören 802.1 (allgemeine Netzwerkstandards), 802.3 (Netzzugangsverfahren CSMA/CD, besonders Ethernet) und 802.11 (drahtlose Netze). Tabelle 4.2 zeigt die vollständige Liste. Einige dieser Standards werden im Folgenden näher beschrieben.

IEEE-Gruppe	Bezeichnung
802.1	Internetworking
802.2	Logical Link Control (LLC)
802.3	CSMA/CD, Ethernet

Tabelle 4.2 Die IEEE-802-Arbeitsgruppen im Überblick

IEEE-Gruppe	Bezeichnung
802.3u	Fast Ethernet
802.3z	Gigabit Ethernet über Glasfaser
802.3ab	Gigabit Ethernet über Twisted Pair
802.4	Token-Bus-Zugriffsverfahren
802.5	Token-Ring-Zugriffsverfahren
802.6	Metropolitan Area Network (MAN)
802.7	Breitbandübertragungstechnologie
802.8	Glasfaserübertragungstechnologie
802.9	integrierte Sprach- und Datendienste
802.10	Netzwerksicherheit
802.11	drahtlose Netze
802.12	Demand-Priority-Verfahren
802.14	Breitband-Kabelfernsehen (CATV)
802.15	Wireless Personal Area Network (WPAN)
802.16	Broadband Wireless Access (BWA)
802.17	Resilient Packet Ring (RPR)
802.18	Radio Regulatory Technical Advisory Group (RRTAG)
802.19	Coexistence TAG
802.20	drahtlose Breitbandnetze
802.21	medienunabhängiges Handover
802.22	drahtlose Regionalnetze (WRAN)
802.23	Emergency Services Working Group
802.24	Smart Grid TAG
802.25	Omni-Range Area Network (Gruppe im Aufbau)
802.30	100BaseX, 100BaseT, Fast Ethernet

Tabelle 4.2 Die IEEE-802-Arbeitsgruppen im Überblick (Forts.)

4.4.1 Die verschiedenen Ethernet-Standards

Ethernet ist heute der verbreitetste Standard für lokale Netze (LANs). Zehntausende von Herstellern weltweit unterstützen diese Art von Netzwerken mit ihrer Hard- und Software.

Jede Ethernet-Schnittstelle, also die Netzwerkkarte oder der fest eingebaute Anschluss, ist mit einer weltweit einmaligen Identifikationsnummer ausgestattet, der sogenannten *MAC-Adresse* (für Media Access Control, einer der beiden Bestandteile der OSI-Netzzugangsschicht). Es handelt sich um eine 48 Bit lange Zahl, die in sechs hexadezimalen Blöcken zwischen 0 und 255 (00 bis FF hex) geschrieben wird, zum Beispiel `00-A0-C9-E8-5F-64`.

Die Datenpakete – auf der Netzzugangsschicht *Frames* genannt – werden mit den MAC-Adressen der sendenden und der empfangenden Station versehen und in der Regel an alle Stationen im Segment versandt. Jede Station überprüft daraufhin, ob die Daten für sie bestimmt sind. Im Übrigen kann man Ethernet-Schnittstellen auch in den »Promiscuous Mode« schalten, in dem sie ohne Unterschied alle Daten entgegennehmen. Auf diese Weise kann der gesamte Datenverkehr in einem Netzsegment überwacht werden.

Die MAC-Adresse wird normalerweise nicht über das jeweilige Teilnetz hinaus weiter verbreitet.[4] Nach außen ergäbe ihre Verwendung auch keinen Sinn, da das nächste Teilnetz auf einer Route womöglich noch nicht einmal zum Ethernet-Standard gehört.

Das Netzzugangsverfahren CSMA/CD

Es ist wichtig, zu verstehen, dass mit dem Namen *Ethernet* gar keine einheitliche Netzwerkhardware bezeichnet wird. Vielmehr handelt es sich um einen Sammelnamen für diverse Netzwerkstandards, die ein bestimmtes Netzzugangsverfahren verwenden. Insofern sind alle Ethernet-Varianten auf der OSI-Schicht 2 identisch, unterscheiden sich aber auf der untersten Schicht.

Als der Vorläufer von Ethernet Ende der 60er-Jahre des letzten Jahrhunderts an der Universität von Hawaii konzipiert wurde (anfangs unter dem geografisch passenden Namen ALOHANet), handelte es sich zunächst um Datenfunk. Diesem Umstand ist übrigens auch der endgültige Name zu verdanken: *ether*, zu Deutsch Äther, ist das gedachte Medium, durch das sich Funkwellen fortpflanzen. Erst in den 70er-Jahren wurde dasselbe Netzzugangsverfahren auch für die Datenübertragung per Kabel eingesetzt, und zwar zunächst über Koaxialkabel.

Das gemeinsame Netzzugangsverfahren aller Ethernet-Formen trägt den Namen CSMA/CD: Carrier Sense Multiple Access with Collision Detection. Schematisch gesehen, funktioniert dieses Verfahren wie folgt:

1. Ein Gerät, das Daten senden möchte, lauscht den Netzabschnitt ab, um festzustellen, ob dieser gerade frei ist, ob also gerade kein anderes Gerät sendet (*Carrier Sense*).

4 Ausnahme: Die IP-Weiterentwicklung IPv6 benutzt die MAC-Adresse als Teil der 128 Bit langen IP-Adresse.

2. Wurde in Schritt 1 festgestellt, dass der Netzabschnitt frei ist, beginnt die Station mit dem Senden der Daten. Möglicherweise hat auch eine andere Station festgestellt, dass das Netz frei ist, und beginnt gleichzeitig ebenfalls mit dem Senden (*Multiple Access*).
3. Falls auf die beschriebene Art und Weise zwei Stationen gleichzeitig mit dem Senden begonnen haben, findet eine sogenannte *Datenkollision* statt, die von den beteiligten Stationen entdeckt wird (*Collision Detection*). Eine Station, die eine Kollision bemerkt, stellt das Senden von Nutzdaten ein und versendet stattdessen eine Warnmeldung (Jam Signal).
4. Eine Station, die wegen einer Datenkollision das Senden abgebrochen hat, beginnt nach einer zufällig gewählten Zeitspanne von wenigen Millisekunden erneut mit dem Senden. Genau diese Zufälligkeit der Zeitspanne, die nach einem komplizierten Verfahren berechnet wird, ist enorm wichtig, damit die beiden Stationen beim nächsten Versuch nicht wieder genau gleichzeitig mit dem Senden beginnen.

Das große Problem von Ethernet besteht darin, dass das CSMA/CD-Verfahren umso ineffektiver wird, je frequentierter der jeweilige Netzabschnitt ist: Ab einem gewissen Grenzwert überschreitet die Anzahl der Datenkollisionen die Menge der Nutzdaten. Heutzutage umgeht man dieses Problem in der Regel durch die Verwendung sogenannter *Switches*, die für zwei miteinander kommunizierende Stationen jeweils eine exklusive Punkt-zu-Punkt-Verbindung einrichten. Wo diese Möglichkeit aufgrund veralteter, inkompatibler Hardware nicht zur Verfügung steht, muss ein Netz mit viel Datenverkehr stattdessen segmentiert, also in kleinere Abschnitte unterteilt werden.

Ethernet-Hardware

Die Bezeichnungen der verschiedenen Arten der Hardware, die für Ethernet-Netzwerke verwendet werden, setzen sich aus der Übertragungsgeschwindigkeit des jeweiligen Netzes in MBit/s und einer spezifischen Bezeichnung für den Kabeltyp oder die maximal zulässige Kabellänge zusammen.

Wie bereits erwähnt, waren Koaxialkabel die ersten für Ethernet verwendeten Kabel.[5] Der Aufbau dieser Kabel ist folgender: Im Zentrum befindet sich ein leitender Draht, der von einer Isolationsschicht umgeben ist, darüber befindet sich ein weiterer Ring aus leitendem Metall und außen natürlich wiederum eine Isolationsschicht. Das bekannteste Alltagsbeispiel für ein Koaxialkabel ist ein handelsübliches Fernsehantennenkabel.

Es gibt zwei Arten von Koaxialkabeln, die für Ethernet eingesetzt wurden:

- 10Base2: dünnes schwarzes Koaxialkabel
  Die 10 steht für die maximale Datenübertragungsgeschwindigkeit des Netzes, in diesem

5 Die Verwendung von Koaxialkabeln für Ethernet ist weitgehend historisch – aus Gründen des Leseflusses habe ich mich aber entschlossen, die Beschreibungen im Präsens zu belassen. Interessant ist die Entwicklung allemal; sie erklärt, warum bei Ethernet viele Dinge so und nicht anders gelöst wurden.

Fall 10 MBit/s. Die nähere Spezifikation, die durch die 2 angegeben wird, betrifft die maximal zulässige Gesamtlänge eines 10Base2-Netzsegments von etwa 200 Metern (eigentlich 200 Yard, was ca. 185 Metern entspricht). In einem Segment dürfen sich maximal 30 Stationen befinden. Um eine größere Entfernung zu überbrücken oder mehr Stationen zu betreiben, muss eine Signalverstärkung durch sogenannte *Repeater* vorgenommen werden.

Alternative Bezeichnungen für diese Ethernet-Form sind *Thinnet Coaxial* oder *Cheapernet*, weil es sich früher um die billigste Art der Vernetzung handelte.

An der Netzwerkkarte wird an eine BNC-Buchse ein T-Adapter angeschlossen. An dessen beiden Seiten werden wiederum über BNC-Stecker die Koaxialkabel angeschlossen, die zu den T-Stücken der Netzwerkkarten der benachbarten Rechner führen. Der Mindestabstand zwischen zwei T-Stücken, also die minimale Länge eines einzelnen Kabels, beträgt 50 cm. Das Netzwerk ist in einer Bustopologie organisiert; die T-Stücke des ersten und des letzten Rechners im Netzwerk werden auf je einer Seite mit einem Abschlusswiderstand oder Terminator versehen.

- 10Base5: dickes gelbes Koaxialkabel
  Der Vorteil dieser auch *Thicknet Coaxial* genannten Variante besteht in der größeren zulässigen Länge des Netzsegments, nämlich – wie die Zahl 5 vermuten lässt – 500 Yard (knapp 460 m). Andererseits ist dieses erheblich dickere Kabel weniger flexibel als das dünnere 10Base2. Beispielsweise ist es schwieriger, solche Kabel durch verwinkelte Kabelkanäle zu ziehen.

  Auf dem Kabel sitzen bei dieser Ethernet-Form sogenannte *Transceiver*, die über 15-polige Buchsen an die Netzwerkkarten angeschlossen werden. Zwischen zwei Transceivern muss ein Mindestabstand von 2,5 Metern eingehalten werden; das Kabel enthält ab Werk Markierungen in diesem Abstand. Die Transceiver werden an diesen Stellen einfach in das Kabel hineingebohrt (deshalb werden sie als *Vampirabzweige* bezeichnet). In einem Segment dürfen sich maximal 100 davon befinden. Auch dieses Netz ist busförmig, und beide Enden müssen durch Abschlusswiderstände terminiert werden.

Heutzutage wird Ethernet fast immer über Twisted-Pair-Kabel betrieben. Bei dieser Kabelsorte handelt es sich um einen verdrillten Kupfer-Zweidrahtleiter: Je zwei isolierte Kupferdrähte werden umeinandergewickelt. Dies verhindert die gegenseitige Beeinträchtigung der Signalqualität, die bei parallel zueinander verlaufenden Kabeln durch die elektromagnetischen Felder aufträte. In einem Twisted-Pair-Kabel verlaufen üblicherweise vier, manchmal auch acht solcher Doppeladern nebeneinander. Sie enden auf beiden Seiten in einem RJ-45-Stecker, der auch für ISDN-Anschlüsse verwendet wird. Bekannt sind solche Kabel vor allem durch ihre Verwendung als Telefonleitungen.

Man unterscheidet zwei verschiedene Grundarten von Twisted-Pair-Kabeln: UTP oder Unshielded Twisted Pair ist ein nicht abgeschirmter Zweidrahtleiter, STP (Shielded Twisted Pair)

ein abgeschirmter, der eine höhere Signalqualität aufweist, sodass er zum Beispiel größere Entfernungen überbrücken kann.

Außerdem werden Twisted-Pair-Kabel in verschiedene Kategorien unterteilt, die unterschiedliche Bandbreiten (gemessen in Megahertz) und entsprechend verschiedene maximale Übertragungsraten zulassen. Diese sind in Tabelle 4.3 aufgelistet.

Kategorie	Bandbreite	Verwendungszweck
1	nicht festgelegt	Telefonie
2	4 MHz	ISDN
3	10 MHz	Ethernet; Token Ring
4	16 MHz	verschiedene
5	100 MHz	Fast Ethernet; allgemeiner Standard
6	200 MHz	verschiedene
7	600 MHz	verschiedene

Tabelle 4.3 Die verschiedenen Kategorien von Twisted-Pair-Kabeln

Alle über Twisted Pair verkabelten Arten von Ethernet weisen eine sternförmige Topologie auf, zumindest im physischen Sinn: Alle Stationen werden jeweils über ein eigenständiges Kabel an einen zentralen Verteiler angeschlossen. Der Vorteil dieser Form der Vernetzung besteht grundsätzlich darin, dass der Ausfall einer einzelnen Verbindung zwischen einem Rechner und dem Verteiler nicht zur Unterbrechung des gesamten Netzes führt, wie es beim busförmigen Koaxialkabel-Ethernet der Fall ist.

Der zentrale Verteiler wird in seiner einfacheren Form *Hub* genannt, die etwas teurere, aber leistungsfähigere Bauweise heißt *Switching Hub* oder kurz *Switch*. Die innere Struktur des Hubs ist letztlich busförmig, sodass es genau wie bei der Vernetzung über Koaxialkabel zu Datenkollisionen kommen kann. Ein Switch stellt dagegen für zwei Stationen, die miteinander kommunizieren möchten, eine exklusive Punkt-zu-Punkt-Verbindung bereit. Dies geschieht dadurch, dass ein Switch die MAC-Adressen aller Schnittstellen zwischenspeichert, an die er bereits Daten ausgeliefert hat, und auf diese Weise die restlichen Stationen nicht mehr mit Daten behelligen muss, die gar nicht für sie bestimmt sind. Da die Preise für Netzwerkzubehör in den letzten Jahren stark gesunken sind, gibt es eigentlich keinen Grund mehr, etwas anderes als einen Switch einzusetzen.

Bei einem Hub teilen sich alle Stationen die gesamte Übertragungsgeschwindigkeit, beim Switch steht sie dagegen jeder einzelnen Verbindung zur Verfügung.

Im Übrigen gibt es besondere Hubs, die als *Bridges* bezeichnet werden. Sie verbinden Ethernet-Netzwerke verschiedenen Typs miteinander, beispielsweise besitzen sie eine Reihe von

RJ-45-Ports für Twisted-Pair-Kabel und zusätzlich einen Anschluss für 10Base2-BNC-Kabel; oder sie unterstützen einfach verschiedene maximale Übertragungsgeschwindigkeiten.

Hubs oder Switches weisen in der Regel 5 bis 24 Anschlüsse (Ports) auf, an die jeweils ein Gerät angeschlossen werden kann. Um Netzwerke mit mehr Geräten zu betreiben, sind diese Geräte kaskadierbar: Die meisten Hubs oder Switches besitzen einen speziellen Port, den sogenannten *Uplink-Port*, der über ein Kabel mit einem normalen Port eines weiteren Verteilers verbunden werden kann. Bei vielen Hubs/Switches kann ein einzelner Port über einen Schalter zwischen »Normal« und »Uplink« umgeschaltet werden.

Die einzige Ausnahme von der allgemeinen Regel, dass ein Hub oder Switch benötigt wird, bildet der Sonderfall, in dem nur zwei Rechner miteinander vernetzt werden sollen: Die beiden Stationen können unmittelbar über ein sogenanntes *Crosslink-Kabel* verbunden werden. Dieses spezielle Kabel besitzt überkreuzte Anschlusspaare anstelle der geradlinig verlaufenden bei normalen Twisted-Pair-Kabeln.

Historisch betrachtet, existieren zwei Arten von Ethernet über Twisted Pair, die unterschiedliche Übertragungsgeschwindigkeiten unterstützen:

- 10BaseT: Die Datenübertragungsrate beträgt 10 MBit/s.
- 100BaseT (auch *Fast Ethernet* genannt): Daten werden mit bis zu 100 MBit/s übertragen; dazu sind mindestens UTP-Kabel der Kategorie 5 erforderlich. Genauer gesagt, gibt es zwei Unterarten: 100BaseTX ist voll kompatibel mit 10BaseT, sodass das Netz schrittweise umgerüstet werden kann. 100BaseT4 verwendet dagegen alle vier Kupferdrahtpaare eines Twisted-Pair-Kabels und ist mit den anderen Standards inkompatibel; in der Praxis spielt es keine Rolle mehr.

Die meisten Netzwerkkarten, Hubs und Switches, die heute verkauft werden, unterstützen beide Übertragungsraten. Der zu verwendende Wert kann bei vielen Netzwerkkarten per Software eingestellt werden, häufiger wird er automatisch gewählt. Natürlich sollten Sie prinzipiell darauf achten, keine reine 10-MBit-Hardware mehr zu kaufen. Aber möglicherweise hat 100-MBit-Hardware der ersten Generation, die nicht auf 10 MBit/s heruntergeschaltet werden kann, sogar noch schlimmere Einschränkungen zur Folge. Zwar ist es bei normalen Standard-PCs ein Leichtes, die Netzwerkkarte gegen ein neueres Modell auszutauschen, um die Kompatibilität zu einer aktualisierten Netzwerkumgebung aufrechtzuerhalten, aber bei anderen Geräten wie beispielsweise Netzwerkdruckern oder kompakten Router-Boxen ist das eventuell nicht möglich. Solche Geräte sind mit einem reinen 100er-Netz eventuell nicht mehr kompatibel.

Noch neuere Formen von Ethernet erreichen Übertragungsraten von 1.000 MBit/s (Gigabit-Ethernet), entweder über Lichtwellenleiter (1000BaseFL für »Fiber Logic«) oder über mehradrige Twisted-Pair-Kabel (1000BaseTX). Bereits entwickelt, aber noch nicht weitverbreitet, sind Ethernet-Varianten mit 10 oder gar 100 GBit/s – anfangs nur über verschiedene Arten von Lichtwellenleitern, aber inzwischen ebenfalls über Twisted Pair.

4.4.2 Drahtlose Netze

Schon seit sehr langer Zeit werden über drahtlose Technologien wie Funk, Mikrowellen, Satellit oder Infrarot nicht nur Sprache, Radio- und Fernsehsignale, sondern auch Daten übertragen. Die digitale (!) Datenübertragung per Funk war sogar die erste Anwendung der drahtlosen Nachrichtentechnik überhaupt: Der Funkpionier Guglielmo Marconi erfand die drahtlose Telegrafie mithilfe des binären Morse-Alphabets[6] lange vor dem Sprechfunk.

Im Bereich der Netzwerke gibt es immer mehr Anwendungsfälle, bei denen sich der Einsatz drahtloser Techniken anbietet. Die folgenden Beispiele können als Anhaltspunkte dienen:

- In Privathaushalten wird WLAN inzwischen häufiger eingesetzt als kabelbasierte Netze. Da viele Menschen Laptops und/oder WLAN-fähige Mobiltelefone und Tablets haben, ist dies auch viel praktischer. Für den Internetzugang kommen entsprechend oft WLAN-DSL-Router zum Einsatz, die eine Verbindung zwischen dem Internet und den Endgeräten vermitteln.
- In einem Unternehmen werden viele Außendienstmitarbeiter beschäftigt. Sie sind mit Notebooks ausgestattet und kommen nur gelegentlich in die Firmenzentrale.
- Eine Firma zieht in ein denkmalgeschütztes Haus ein, an dessen Bausubstanz nichts geändert werden darf – an das Verlegen von Kabelkanälen oder gar das Aufstemmen von Wänden für die Vernetzung ist nicht zu denken.
- Zwischen zwei Gebäuden eines Unternehmens verläuft eine öffentliche Straße; für die Überbrückung durch ein Kabel müsste ein langfristiges Genehmigungsverfahren mit ungewissem Ausgang eingeleitet werden.
- Auf LAN-Partys (Treffen von Netzwerkspielern), Messen, Kongressen oder ähnlichen Veranstaltungen müssen Unmengen von Computern für kurze Zeit vernetzt werden.

Für den Betrieb drahtloser Netzwerke kommen die verschiedensten Übertragungsmethoden zum Einsatz. Sie lassen sich nach folgenden Kriterien unterscheiden oder für den praktischen Einsatz auswählen:

- Welche maximale Entfernung zwischen zwei Stationen muss überbrückt werden?
- Besteht zwischen den einzelnen Standorten Sichtkontakt, oder befinden sich Wände oder andere Hindernisse zwischen ihnen?
- Soll eine freie Funkfrequenz genutzt werden, oder kann es auch eine lizenzpflichtige sein (Letzteres kann teuer werden)?
- Sind die vernetzten Geräte selbst stationär oder mobil?

6 Ganz und gar binär ist das Morsealphabet übrigens nicht: Neben »Lang« und »Kurz« muss die Pause zwischen zwei Zeichen als drittes mögliches Signal betrachtet werden, da die einzelnen Zeichen (übrigens gemäß ihrer Häufigkeit in englischen Texten) aus unterschiedlich vielen Einzelsignalen bestehen.

Diese diversen Auswahlkriterien zeigen bereits, dass es so etwas wie »das« drahtlose Netz nicht gibt. Für jeden Anwendungszweck bieten sich verschiedene Lösungen an, die sorgfältig geprüft werden müssen.

Genau wie bei der verkabelten Konkurrenz lassen sich auch hier verschiedene Kategorien von Reichweiten unterscheiden. Das WLAN (Wireless LAN, auch *WiFi* genannt) nach IEEE 802.11 ist ein drahtloses Netz für den Nahbereich, also für die Vernetzung innerhalb einer einzelnen Institution. Das WWAN (Wireless Wide Area Network) dagegen ist ein drahtloses Fernnetzwerk. Dazu zählen unter anderem Satellitenverbindungen.

In diesem Abschnitt wird nur das 802.11-kompatible WLAN beschrieben, da es sich seit seiner Einführung 1997 sehr schnell verbreitet hat und heute von allen Wireless-Technologien am häufigsten eingesetzt wird. 802.11 besteht aus mehreren Unterstandards, die sich in den Punkten Frequenzspektrum, Übertragungsrate und Funktechnologie unterscheiden. Sie alle werden jedoch über Funk betrieben; eine ursprünglich ebenfalls spezifizierte Infrarotvariante hat sich nicht durchgesetzt. Infrarot wird größtenteils für den drahtlosen Anschluss von Peripheriegeräten wie Mäusen oder Tastaturen verwendet. Tabelle 4.4 zeigt eine Übersicht über die wichtigsten gebräuchlichen 802.11-Varianten.

Standard	Frequenzbereich	Übertragungsrate	Funktechnik
802.11	2,4 GHz	1 oder 2 MBit/s	FHSS/DSSS
802.11a	5 GHz	bis zu 54 MBit/s	OFDM
802.11b	2,4 GHz	5,5/11/22 MBit/s	HR/DSSS
802.11g	2,4 GHz	bis zu 54 MBit/s	OFDM
802.11n	2,4 und 5 GHz	bis zu 600 MBit/s	MIMO

Tabelle 4.4 Verschiedene Varianten von IEEE 802.11

Die Trägerfrequenz von 2,4 GHz wird vor allem deshalb am häufigsten verwendet, weil sie nicht lizenzpflichtig ist. Es handelt sich nämlich um diejenige Frequenz, mit der Mikrowellenherde arbeiten, da diese Wellenlänge Wassermoleküle am effektivsten erhitzt.

Die diversen Funkverfahren arbeiten alle mit verschiedenen Varianten der Frequency-Hopping-Methode, die auch im Mobilfunk eingesetzt wird: Nach einem bestimmten Schema werden die Funkwellen über mehrere Frequenzen übertragen, die mehrmals in der Sekunde wechseln. Dies ist erheblich weniger störanfällig als die Verwendung einer einzelnen Frequenz. Die grundlegende Technik wurde Mitte der 30er-Jahre von der österreichischen Schauspielerin Hedy Lamarr erfunden. Ihr damaliger Ehemann war Rüstungsfabrikant, und diese Funktechnik sollte helfen, Torpedos der Alliierten fernzusteuern, ohne dass die Signale abgefangen und verfälscht werden konnten. Im Einzelnen werden folgende Verfahren unterschieden:

- FHSS (Frequency Hopping Spread Spectrum): Die Frequenzen wechseln nach einem zufälligen Muster.
- DSSS (Direct Sequence Spread Spectrum): Es werden erheblich mehr Einzelfrequenzen verwendet; die Verteilung erfolgt nach einem komplexen mathematischen Verfahren.
- HR/DSSS (High Rate Direct Sequence Spread Spectrum): Entspricht DSSS mit speziellen Erweiterungen, die eine höhere Übertragungsrate ermöglichen.
- OFDM (Orthogonal Frequency Division Multiplexing): Jeder Kanal wird in mehrere Teilkanäle unterteilt, die Signale werden über alle Teilkanäle parallel übertragen. Aus diesem Grund ist OFDM das Übertragungsverfahren mit der höchsten Datenrate, andererseits aber auch das aufwendigste, sodass die entsprechende Hardware noch vor wenigen Jahren vergleichsweise teuer war.
- MIMO (Multiple Input/Multiple Output): Im Wesentlichen eine nochmals verbesserte OFDM-Variante, die wiederum erheblich höhere Übertragungsraten ermöglicht. Die Datenübertragung kann gleichzeitig über mehrere Frequenzbänder erfolgen.

Der größte Teil der Wireless-LAN-Hardware, der momentan verkauft wird, basiert auf den Standards 802.11b und 802.11g (die meisten Geräte unterstützen wahlweise beide). Die Preise für Hardware dieser Variante sind in den letzten Jahren stark gefallen. Ein WLAN-Adapter ist inzwischen ab etwa 20 € erhältlich, sowohl als PCI-Karte als auch als PCMCIA- oder USB-Adapter. Außerdem sind Notebooks (und meist auch Desktop-PCs) ab Werk standardmäßig mit einer WLAN-Schnittstelle ausgestattet. Vorreiter dürften das PowerBook und das iBook von Apple gewesen sein; Apple fördert diese Technologie unter dem Namen AirPort seit vielen Jahren.

Als Netzzugangsverfahren in 802.11-Netzen kommt CSMA/CA zum Einsatz (Carrier Sense Multiple Access with Collision Avoidance) – wie der Name vermuten lässt, werden Datenkollisionen von vornherein vermieden. Anders als bei CSMA/CD sendet eine Station, die ein freies Übertragungsmedium (in diesem Fall den entsprechenden Funkkanal) vorfindet, nicht einfach ihre Daten, sondern eine Sendeanforderung (RTS). Daraufhin warten andere sendebereite Stationen; und die erste Station, die das RTS gesendet hat, sendet ihre Daten, nachdem ihr die Empfängerstation ihre Empfangsbereitschaft (CTS) signalisiert hat. Abgeschlossen wird die Datenübertragung durch ein ACK-Signal, daraufhin kann die nächste Station ihren Sendewunsch bekannt geben.

Das einfachste denkbare 802.11-WLAN besteht nur aus mehreren Rechnern mit entsprechender Schnittstelle, die auf direktem Weg miteinander kommunizieren. Ein solcher Aufbau wird als *Basic Service Set* (BSS) bezeichnet. Die Entfernung zwischen zwei beliebigen Stationen darf die maximale Reichweite des Funksignals nicht überschreiten, da jede Station die Signale nur senden und empfangen, aber nicht verstärken und weiterleiten kann. Da ein solches Netzwerk nicht mit anderen Netzen kommunizieren kann, wird es als *unabhängiges BSS* (Independent BSS oder kurz IBSS) bezeichnet. Derartige Netzwerke sind sinnvoll für die sogenannte *Ad-hoc-Vernetzung* temporärer Zusammenkünfte wie Messen oder LAN-Partys.

Ein wenig komplexer wird der Aufbau eines BSS, wenn ein Access Point hinzugefügt wird. Im Grunde funktioniert ein Access Point wie ein Ethernet-Hub, denn sobald er vorhanden ist, kommunizieren die Stationen nicht mehr direkt miteinander, sondern senden die Frames an den Access Point, der sie an den gewünschten Empfänger weitergibt. Die Identifikation der einzelnen Stationen erfolgt wie bei Ethernet anhand einer 48 Bit langen MAC-Adresse. Ein BSS mit einem Access Point wird als *Infrastruktur-BSS* bezeichnet. Für die Reichweite des Netzes ist nur noch die Entfernung zwischen einer Station und dem Access Point ausschlaggebend.

Die wichtigste Aufgabe eines Access Points besteht in seiner Funktion als Bridge. Er verbindet das WLAN mit einem Backbone-Netzwerk – meistens Twisted-Pair-Ethernet. Auf diese Weise kann das WLAN mit stationären Teilen des Netzes verbunden werden oder Zugang zu Servern und Routern erhalten, ohne dass diese selbst mit WLAN-Schnittstellen ausgestattet werden müssten.

Im Übrigen bildet ein Verbund aus miteinander vernetzten Access Points (entweder ebenfalls über Funk oder über Ethernet) ein sogenanntes *Extended Service Set* (ESS). Eine Station kann sich innerhalb eines ESS frei bewegen, weil die Access Points einander darüber auf dem Laufenden halten, welche Stationen sich gerade in ihrem Bereich befinden. Eine Station kann immer nur genau mit einem Access Point verbunden sein; sobald das Signal eines anderen Access Points stärker wird als das des bisherigen, meldet die Station sich bei ihrem alten Access Point ab und bei dem neuen an. Auf diese Weise werden Frames immer über den jeweils aktuellen Access Point an eine Station gesendet.

Ein zusätzlicher Nutzen von Access Points besteht darin, dass sie in der Lage sind, Frames zu puffern, die an bestimmte Stationen adressiert sind. Gerade Notebooks schalten im Standby-Modus oft auch die WLAN-Schnittstelle ab, um Strom zu sparen; sobald die Verbindung wieder aufgebaut wird, werden die zwischengespeicherten Frames ausgeliefert.

Das ESS-Modell wird immer häufiger für öffentlich verfügbare Netzwerkzugänge eingesetzt. In Bahnhöfen, Flughäfen oder Gaststätten stehen öffentlich zunehmend WLAN-Access-Points (auch *Hotspots* genannt) zur Verfügung, in die sich Notebook-Benutzer ohne Weiteres einwählen können. Mittlerweile werden sogar die ersten Innenstädte fast flächendeckend mit einander überlappenden Access Points ausgestattet. Irgendwann könnte ein ähnlich dichtes Netz entstehen, wie es die Mobilfunkzellen inzwischen bilden.

Eine der größten Herausforderungen beim Einsatz von Wireless-Technologien bleibt die Sicherheit. Es ist zwar auch nicht weiter schwierig, das Signal von Ethernet-Kabeln abzuhören, aber immerhin ist es vergleichsweise einfach, den physikalischen Zugang zu ihnen zu kontrollieren. Bei WLAN kann dagegen im Grunde genommen jeder die Signale mit einer kompatiblen Antenne auffangen und analysieren, um unberechtigt Informationen zu erhalten oder gar zu manipulieren. Das gilt umso mehr, als man die Grenzen der Funkreichweite niemals ganz genau auf die Größe des zu vernetzenden Gebäudes oder Geländes abstimmen kann; es ist also durchaus möglich, die Funkwellen außen zu empfangen.

Um ein Mindestmaß an Sicherheit zu gewährleisten, bot die ursprüngliche 802.11-Spezifikation eine optionale Verschlüsselung der Frames an. Allerdings ist diese Methode nicht besonders sicher; Sicherheitsexperten haben bereits bewiesen, dass die Verschlüsselung verhältnismäßig leicht zu knacken ist. Schon der Name dieser Technik, *WEP* (Wired Equivalent Privacy), sagt allzu deutlich aus, dass es nicht um mehr geht, als etwa dasselbe Maß an Sicherheit zu gewährleisten wie beim rein physikalischen Schutz verkabelter Netzwerke. Der Hauptverwendungszweck besteht auch gar nicht in der Geheimhaltung, sondern in der Abgrenzung eines Wireless-Netzes von benachbarten Netzen: Es ist ärgerlich, wenn jedes vorbeifahrende Fahrzeug, in dem sich zufälligerweise ein Laptop mit 802.11-Schnittstelle befindet, diesen vorübergehend automatisch ins Netz einbucht und wieder daraus verschwindet. Dies lässt sich allerdings zuverlässiger verhindern, indem der Access Point mit einer Whitelist zugelassener MAC-Adressen konfiguriert wird.

Inzwischen stehen mit WPA und WPA2 (WiFi Protected Access) stark verbesserte WLAN-Verschlüsselungsverfahren zur Verfügung.

4.5 Datenfernübertragung

Nachdem im vorangegangenen Abschnitt die verschiedenen Formen der LAN-Vernetzung und der WANs über Standleitungen beschrieben wurden, sollen nun diverse Verfahren der Datenfernübertragung (DFÜ) geschildert werden. Wie bereits angesprochen, wurde DFÜ bereits eingesetzt, als sie lediglich der Punkt-zu-Punkt-Kommunikation zwischen einzelnen Rechnern über eine direkte Telefonverbindung diente. Heute geht es in der Regel darum, den Zugang zu einem bestehenden Netzwerk oder (über einen kommerziellen Provider) zum Internet herzustellen.

Die erste Generation der DFÜ-Hardware, der umständliche und störanfällige Akustikkoppler, muss hier nicht mehr beschrieben werden. Die drei wesentlichen Technologien sind heute Modems für den Netzwerkzugang über analoge Telefonleitungen, der Zugang über die digitale Telefonleitung ISDN sowie Hochfrequenz-Verbindungen über verschiedene DSL-Dienste. Diese verschiedenen Zugangsverfahren werden im Folgenden dargestellt.

Eine Gemeinsamkeit aller DFÜ-Netzwerkverbindungen besteht in der Notwendigkeit, die Datenübertragung über diese Leitungen zu standardisieren und bestimmte Grundlagen für die Protokolle der Vermittlungsschicht zu schaffen. Dafür werden spezielle Protokolle verwendet, die den Netzzugang über relativ langsame serielle Leitungen ermöglichen. Das traditionelle Protokoll für die Vernetzung über Wählleitungen war SLIP (Serial Line Interface Protocol). Allerdings besitzt es eine Reihe organisatorischer und technischer Mängel und wurde deshalb weitgehend durch PPP (Point-to-Point Protocol) ersetzt.

PPP kümmert sich um die Authentifizierung des Benutzers nach der Einwahl, indem Benutzername und Passwort übermittelt werden; anschließend verhandeln die beiden direkt miteinander verbundenen Punkte die eigentlichen Netzwerkdetails. Eine der wesentlichsten Fähigkeiten des Protokolls für Internetverbindungen besteht darin, dass der Einwahlknoten dem anwählenden Rechner automatisch eine IP-Adresse zuweisen kann, über die diese Netzwerkschnittstelle im gesamten Internet identifiziert wird.

Im Einzelnen erfolgen bei PPP also die folgenden Schritte:

- Wird eine Wählleitung (analog oder ISDN) verwendet, stellt der Rechner des Benutzers über die entsprechende Schnittstelle eine Telefonverbindung her. Falls die Leitung besetzt sein sollte, werden spezielle frei konfigurierbare Maßnahmen getroffen; in der Regel erfolgt nach einer gewissen Wartezeit ein erneuter Wählversuch. Bei DSL-Leitungen wird ebenfalls die Verbindung aktiviert, auch wenn man dies nicht als *Wählen* im klassischen Sinne bezeichnen kann.
- Der Einwahlknoten verlangt eine Authentifizierung, in der Regel in Form von Benutzername und Passwort. Die meisten PPP-Implementierungen in modernen Betriebssystemen übermitteln diese Daten nach einmaliger Konfiguration automatisch, ohne Zutun des Benutzers.
- Nachdem die Daten überprüft wurden, erfolgen entweder die Ablehnung des Benutzers und der Verbindungsabbau, oder die Netzwerkparameter werden ausgehandelt. Auch wenn PPP als Netzzugangsgrundlage für alle möglichen Protokolle der Vermittlungsschicht dienen kann, wird heute fast nur noch TCP/IP aufgesetzt. Zu diesem Zweck weist der PPP-Knotenpunkt des Internetproviders der seriellen Verbindung auf der Einwahlseite eine IP-Adresse zu, eine im gesamten Internet einmalige Identifikationsnummer. Ihr Konzept wird im nächsten Abschnitt genau beschrieben.

4.5.1 Netzwerkzugang per Modem (analoge Telefonleitung)

Für Modems wurden im Laufe der Zeit viele verschiedene Standards entwickelt, die sich insbesondere bezüglich ihrer maximalen Datenübertragungsrate voneinander unterscheiden. Der aktuelle Standard heißt V.90 und überträgt bis zu 56.600 Bit/s. Gemessen an den üblichen Geschwindigkeiten fest verdrahteter Netzwerke, ist das natürlich sehr langsam, aber kein Vergleich zu den Modem-Geschwindigkeiten vergangener Jahrzehnte. Rein physikalisch scheint mit 56,6 KBit/s die Leistungsgrenze erreicht zu sein; etwas höhere Übertragungsraten lassen sich durch die heutzutage häufig verwendete Datenkomprimierung erzielen – je nach übertragener Datenart etwa bis zur doppelten Leistung.

Das Wort »Modem« ist eine Zusammensetzung aus den Abkürzungen für *Modulator* und *Demodulator*, weil es die digitalen Signale des Computers in frequenzmodulierte Analogimpulse umwandelt, diese über die Telefonleitung überträgt und am Ziel wieder zurückverwandelt. Zu diesem Zweck muss es auf der einen Seite mit dem Computer verbunden werden, zum Beispiel über USB, klassisch auch über die alte serielle Schnittstelle oder als PCI-Steckkarte. Auf der anderen Seite wird das Modem über einen TAE-Stecker an die Telefonbuchse angeschlossen. Praktisch sind in diesem Zusammenhang Dreifach-TAE-Dosen, die

leicht nachgerüstet werden können: Sie verfügen über einen speziellen Anschluss (TAE-F) für ein Telefon in der Mitte und zwei Anschlüsse (TAE-N) für Zusatzgeräte – Faxgerät, Anrufbeantworter oder eben Modem – außen.

Das Modem wird vom Computer über ein einfaches ASCII-basiertes Protokoll gesteuert. Heutzutage verwenden praktisch alle Modems den sogenannten *Hayes-Befehlssatz* (benannt nach einem längst vergessenen Modem-Hersteller). Da die Befehle dieses Protokolls alle mit der Zeichenfolge »AT« beginnen, wird er mitunter auch als *AT-Befehlssatz* bezeichnet. Wichtige Befehle sind etwa folgende:

- `ATDT <Rufnummer>`: DT steht für »Dial Tone« – eine Rufnummer wird im Tonwahlverfahren (Mehrfrequenzverfahren) angewählt.
- `ATDP <Rufnummer>`: »Dial Pulse« – eine Rufnummer wird im Pulswahlverfahren angewählt (heute sehr selten).
- `ATH`: »Hangup« – die Telefonverbindung wird unterbrochen, es wird »aufgelegt«.
- `ATZ`: Das Modem wird auf den Einschaltzustand zurückgesetzt (Reset).

Das Tonwahlverfahren verwendet mehrere Töne unterschiedlicher Frequenzen, die zusammen die verschiedenen Ziffern und Funktionen des Telefons repräsentieren. Das Pulswahlverfahren sendet dagegen eine Reihe von »Klicktönen« – einen für eine 1, zwei für eine 2 etc., bis zehn für eine 0. Seitdem alle Vermittlungsstellen in den deutschen Telefonnetzen digital sind, benötigt niemand mehr das langsamere und unzuverlässigere Pulswahlverfahren. Verwechseln Sie übrigens digitale Vermittlung nicht mit digitaler Signalübertragung – Letztere findet beispielsweise bei ISDN statt, das im nächsten Abschnitt behandelt wird.

Bevor eine Datenkommunikation überhaupt denkbar ist, müssen sich beide Seiten darüber einig sein, auf welche Weise sie die aufeinanderfolgenden einzelnen Bits überhaupt als Daten-Bits interpretieren sollen, was Daten-Bits, Stopp-Bits und eventuelle Parity-Bits angeht. Die meisten Interneteinwahlpunkte verwenden heutzutage den Standard 8N1 (acht Daten-Bits, kein Parity-Bit, ein Stopp-Bit). Dies muss in den Modem-Konfigurationsdaten eingetragen werden. Eine Beschreibung der verschiedenen Formen der seriellen Datenübertragung finden Sie in Kapitel 3, »Hardware«.

Nach der Herstellung der eigentlichen Telefonverbindung findet der sogenannte *Handshake* (»Handschlag«) zwischen den beiden Gegenstellen statt. Es wird eine Übertragungskapazität ausgehandelt, die beide Seiten verwenden können. Erst nachdem die grundlegende Datenübertragung funktioniert, wird PPP eingesetzt, um die eigentliche Netzwerkverbindung über die Telefonverbindung herzustellen, wie hier bereits beschrieben wurde.

4.5.2 ISDN

Das Integrated Services Digital Network (etwa »Digitalnetzwerk mit integrierten Diensten«) oder kurz ISDN wurde in den 80er-Jahren von verschiedenen europäischen Telefongesellschaften eingeführt; die aktivste von ihnen dürfte die in Deutschland damals noch zuständige Deutsche Bundespost gewesen sein. Es handelt sich im Prinzip um die Übertragung digitaler Signale über klassische Kupferdraht-Telefonleitungen. Einem reinen Telefonkunden bietet ISDN zunächst die folgenden unmittelbaren Vorteile:

- Es werden zwei voneinander unabhängige Kanäle zur Verfügung gestellt; über beide kann gleichzeitig telefoniert, gefaxt oder Datenfernübertragung betrieben werden.
- Die Rufnummer eines Anrufers, der ebenfalls ISDN verwendet, wird übermittelt und im Display eines entsprechend ausgerüsteten Telefons angezeigt (dies funktioniert natürlich auch bei Standardtelefonleitungen mit Digitalvermittlung).
- Während eine Verbindung besteht, kann ein weiterer Anruf angenommen werden. Entweder wird der jeweils andere Gesprächspartner in den Wartezustand versetzt (Makeln), oder der neue Anrufer wird mit dem bisherigen in dieselbe Verbindung aufgenommen (Dreierkonferenz).

Inzwischen stehen die meisten dieser Dienste dank der flächendeckend digitalen Vermittlung in Deutschland auch Analogkunden zur Verfügung; lediglich die beiden separaten Kanäle bleiben ISDN vorbehalten. Im Übrigen erhalten ISDN-Benutzer üblicherweise drei unabhängige Rufnummern (bei einigen Telefongesellschaften sogar noch mehr), die frei auf die jeweiligen Geräte verteilt werden können.

Technisch betrachtet, werden sogar drei Kanäle zur Verfügung gestellt; die beiden B- Kanäle übertragen Telefongespräche, Faxe oder Daten mit jeweils 64 KBit/s, während der D-Kanal Dienstinformationen wie Rufnummernübermittlung oder Anklopfen mit 16 KBit/s überträgt. Für Internetverbindungen und andere Datenübertragungsmethoden besteht die Möglichkeit, beide B-Kanäle zu bündeln und auf diese Weise insgesamt eine Datenübertragungsrate von 128 KBit/s zu gewährleisten; natürlich entstehen dafür auch doppelte Kosten.

In der Praxis funktioniert ISDN folgendermaßen: An die normale TAE-Telefonsteckdose wird ein spezielles ISDN-Endgerät namens *NTBA* angeschlossen. Es stellt zwei sogenannte *SO-Basisanschlüsse* zur Verfügung. Diese verwenden die auch vom Twisted-Pair-Ethernet bekannten RJ-45-Stecker. An jeden dieser Anschlüsse kann ein ISDN-Endgerät angeschlossen werden, beispielsweise ein Telefon, ein Faxgerät oder ein ISDN-Adapter zur Computerdatenübertragung. Damit niemand seinen kompletten Telekommunikations-Gerätepark umstellen muss, werden als spezielle Form von ISDN-Endgeräten sogenannte *TK-Anlagen* angeboten, die wiederum den Anschluss analoger Endgeräte ermöglichen. Einige TK-Anlagen bieten auch durchgeschleifte S_0-Anschlüsse an, beispielsweise um ISDN-Geräte an einer praktischeren Stelle anzuschließen.

Natürlich sollten Sie nicht versuchen, ein Analogmodem an eine TK-Anlage anzuschließen – es würde nicht etwa nur den eigentlichen Vorteil der ISDN-Datenübertragung zunichte machen, sondern funktioniert gar nicht. Da die übertragenen Daten nach außen wie ISDN aussehen, würde die Gegenstelle ihre Antworten mit einer Übertragungsgeschwindigkeit übermitteln, die das Modem nicht verarbeiten kann.

Um einen Computer mit ISDN zu verbinden, werden stattdessen verschiedene Formen von ISDN-Adaptern angeboten: als PCI-Steckkarten (früher gab es sogar ISA-Modelle) oder externe USB-Geräte. Mittlerweile werden einige externe ISDN-Geräte auch mit integrierter TK-Anlagen-Funktion angeboten. Eine interne ISDN-Karte sieht genauso aus wie eine moderne Ethernet-Karte, und es kann leicht passieren, dass man das Twisted-Pair-Netzwerkkabel mit seinem baugleichen Stecker in die ISDN-Karte steckt und umgekehrt (selbstverständlich geschieht in diesem Fall gar nichts). Jedenfalls muss normalerweise ein ISDN-Kabel vom Anschluss des ISDN-Adapters zu einem S0-Anschluss verlaufen.

Der Unterschied zwischen den Übertragungsraten eines heutigen Modems (56,6 KBit/s) und Ein-Kanal-ISDN (64 KBit/s) mag Ihnen nicht besonders groß erscheinen. Als jedoch das Internet für Privatkunden interessant zu werden begann, lag die Übertragungsrate der meisten Modems bei 9.600 oder 14.400 Bit/s; erst allmählich kamen Geräte mit 28.800 Bit/s hinzu. Abgesehen davon, besitzt ISDN noch heute einen weiteren Vorteil gegenüber Modem-Verbindungen: Der Verbindungsaufbau geht fast ohne Verzögerung vonstatten, während es bei Modems zu Wartezeiten von etlichen Sekunden kommen kann, bis die Leitung bereit ist.

Neben der Verbindung zu einem Internetprovider, die mittlerweile wohl die häufigste über ISDN genutzte Dienstleistung ist, war über Jahre hinweg auch die direkte Verbindung zwischen Computern für die ISDN-Datenübertragung üblich. Sehr viele Macintosh-Benutzer verwendeten dafür regelmäßig die Software Leonardo, während Windows-Benutzern beispielsweise das Programm Fritz!Data zur Verfügung stand, das mit der in Deutschland besonders populären Fritz!Card der Berliner Firma AVM geliefert wurde. Auch viele Mailbox/BBS-Systeme wurden in der zweiten Hälfte der 90er-Jahre auf ISDN umgestellt oder um eine ISDN-Einwahlmöglichkeit erweitert.

Insgesamt lässt sich feststellen, dass ISDN erst mit dem Aufkommen von Internetzugängen in Firmen und Privathaushalten wirklich populär wurde. Zuvor wurde es manchmal als Telefonleitung für Firmen eingesetzt, allerdings nicht annähernd so häufig, wie die Telefongesellschaften sich dies erhofft hatten. Als ISDN dann schließlich immer öfter benutzt wurde, reichte seine Übertragungsrate immer mehr Nutzern nicht mehr aus; das Bedürfnis nach multimediafähigen Breitband-Verbindungen wuchs deutlich. Dies führte zur Einführung der im nächsten Abschnitt vorgestellten DSL-Dienste.

4.5.3 DSL-Dienste

DSL ist die Abkürzung für »Digital Subscriber Line« (etwa »digitale Abonnement-Leitung«). Der Name soll verdeutlichen, dass es sich de facto um eine Standleitung anstelle einer Wählleitung handelt. Zur Einführung von DSL kam es, da es durch die allmähliche Verbesserung der Qualität von Telefonleitungen möglich wurde, Signale hoher Frequenz zu übertragen. Auch die meisten DSL-Dienste verwenden also genau wie Modem- und ISDN-Verbindungen die klassischen Kupferleitungen der Telefongesellschaften, die allerdings immer häufiger durch Glasfaserleitungen ersetzt oder ergänzt werden.

DSL-Varianten

Es existieren zwei grundsätzliche Varianten von DSL: Bei *Symmetric DSL* (SDSL) sind die Übertragungsraten für ankommende und ausgehende Daten identisch, bei *Asymmetric DSL* (ADSL) ist die ankommende Übertragungsrate höher als die ausgehende. SDSL ist eher für mittlere bis große Unternehmen geeignet, die nicht nur permanent auf das Internet zugreifen, um im Web zu recherchieren oder ihre E-Mails zu lesen, sondern bei denen auch eine Menge Datenausgänge stattfinden – beispielsweise für den Vor-Ort-Betrieb eigener Web- oder Mailserver oder für den Direktzugriff auf das Firmennetzwerk durch externe Mitarbeiter. ADSL dagegen wird häufiger von Privatkunden oder kleineren Firmen eingesetzt, die recht hohe Datenmengen aus dem Internet herunterladen, aber nur verhältnismäßig wenige und eher kleine Uploads durchführen.

Übliche ADSL-Angebote wie T-DSL der Deutschen Telekom stellten ursprünglich eine Download-Rate von 1.024 KBit/s und eine Upload-Rate von 128 KBit/s zur Verfügung, teilweise sogar noch weniger. Aktuelle ADSL-Anschlüsse des klassischen Typs sind dagegen mit Download-Geschwindigkeiten von 2 bis 8 MBit/s ausgestattet. Auch die umgekehrte Datenrate wurde entsprechend vervielfacht. Die neueren Typen ADSL2 und ADSL2+, die allerdings eine nicht überall verfügbare hohe Leitungsqualität und spezielle Hardware benötigen, schaffen sogar 16 beziehungsweise 25 MBit/s im Download.

SDSL-Lösungen werden von sehr vielen kommerziellen Providern angeboten und stellen je nach Bedarf viele verschiedene Übertragungsraten von 1.024 KBit/s bis hin zu mehreren MBit/s zur Verfügung. Sie sind deutlich teurer als ADSL-Angebote mit der gleichen oder gar mit einer höheren Übertragungsrate, weil die entsprechende Technik aufwendiger ist.

Anders als bei Modem- oder ISDN-Angeboten werden die Gebühren für DSL-Anschlüsse in der Regel nicht nach der Nutzungsdauer berechnet, sondern als sogenannte *Flatrate* für beliebig lange Online-Zeiten. Einige Provider verwenden allerdings eine Volumenbeschränkung, das heißt, ohne Aufpreis darf monatlich nur eine bestimmte Datenmenge transferiert werden.

Neben den DSL-Angeboten, die über normale Telefonleitungen laufen, werden seit einiger Zeit auch spezielle Lösungen angeboten. Eine davon ist die Internetverbindung über das Glasfaserkabel des Kabelfernsehens. Da dieses Kabel für das Passivmedium Fernsehen erfunden wurde, besitzt es in seiner ursprünglichen Version keine Rückkanal-Fähigkeit. Es können Daten empfangen, aber nicht gesendet werden; noch nicht einmal die Anforderung einer URL kann abgesetzt werden. Erst allmählich wird der Rückkanal derjenigen Kabelnetze nachgerüstet, die die Deutsche Telekom bereits verkauft hat.

Eine andere Lösung ist besonders interessant für kleine Gemeinden, die so weit von der nächsten größeren Stadt entfernt liegen, dass sich eine Nachrüstung der bestehenden Telefonleitungen oder Fernsehkabelnetze nicht lohnt: die Kommunikation mit einem Satelliten über eine Parabolantenne. Diese Lösung bietet beispielsweise die Firma Teles unter dem

Namen skyDSL an. Die Datenübertragungsrate beträgt bis zu 24.000 KBit/s. Über den Satelliten ist allerdings nur der Datenempfang möglich; Anfragen und andere Sendevorgänge erfolgen über Analogmodem, ISDN oder GSM-Mobilfunk.

ADSL anschließen

An den TAE-Anschluss eines ADSL-Kunden wird ein sogenannter *Splitter* angeschlossen – eine Frequenzweiche, die die hochfrequenten DSL-Signale und die niedrigfrequenten normalen Telefonsignale voneinander trennt. Den Ausgang für die Telefonsignale bietet wiederum ein TAE-Anschluss, an den entweder ein Analogtelefon oder ein NTBA angeschlossen wird, je nachdem, ob ADSL mit einem Analog- oder mit einem ISDN-Telefonanschluss kombiniert wird.

Den Ausgang für die speziellen ADSL-Signale bietet eine Twisted-Pair-Buchse vom Typ RJ-11. An diesen Anschluss wird in der Regel ein ADSL-Modem angeschlossen, das dann über USB oder Twisted-Pair-Ethernet mit dem Computer verbunden wird. Natürlich ist die Bezeichnung *ADSL-Modem* technisch gesehen Unfug. Bei DSL findet keinerlei Analog-Digital-Umwandlung statt. Dennoch ist der Begriff *Modem* für das Gerät weitverbreitet, weil es den Computer mit einer seriellen Fernleitung verbindet. Beim Anschluss über eine Ethernet-Schnittstelle kommt eine spezielle PPP-Variante namens *PPPoE* (PPP over Ethernet) zum Einsatz.

Anstelle der reinen ADSL-Modems zum Anschluss eines einzelnen Rechners werden inzwischen meist DSL-Router verwendet, die gleich einem gesamten Netzwerk per Ethernet, WLAN oder beidem den Internetzugang bereitstellen. Inzwischen erlauben auch die meisten Provider den Einsatz solcher Router; früher waren die günstigsten Tarife dagegen vielfach auf einen Einzelrechner beschränkt. Die jüngste Generation von DSL-Routern verzichtet auch gleich auf einen externen Splitter und bringt diese Funktion selbst mit; eventuelle Analog- oder ISDN-Telefone werden direkt an den Router angeschlossen.

4.5.4 Internetzugänge über Mobilfunk

Neben den hier behandelten stationären DFÜ-Verbindungen werden auch diejenigen über Mobilfunk immer wichtiger. Über die seit den 90er-Jahren errichteten GSM-Netze (in Deutschland beispielsweise D1, D2, E-Plus etc.) kam ursprünglich vor allem ein Verfahren namens *GPRS* (General Packet Radio Service) zum Einsatz; es wurde mehrfach in Details verbessert, bietet aber noch immer keine allzu hohen Datentransferraten – sie liegen bei 53,6 KBit/s im Download und 26,8 KBit/s im Upload. Neuere Verfahren sind zum Teil erheblich schneller:

- EDGE: Download 217,6 KBit/s, Upload 108,8 KBit/s
- UMTS: Download und Upload 384 KBit/s

- HSPA: Download (HSDPA) 7,2 MBit/s, Upload (HSUPA) 1,4 MBit/s
- LTE (Long-Term Evolution) ist ein neuer Standard, den die meisten Geräte der neuesten Generation und viele Mobilfunkanbieter, insbesondere in den Industrieländern, unterstützen. In der ersten Ausbaustufe wurden Übertragungsraten bis 100 MBit/s realisiert, die später auf 150 MBit/s erhöht wurden. Seit 2014 sind im Rahmen der nächsten Ausbaustufe (LTE-Advanced) auch Übertragungen im Gigabit-Bereich möglich.

Alternativ werden die verschiedenen mobilen Datenübertragungsstandards mit Generationsnummern bezeichnet: 1G bezeichnet historische Mobilfunknetze vor der Handy-Revolution der 90er-Jahre, 2G ist der GSM-Mobilfunk, 2.5G ist GPRS, und 3G steht für EDGE, UMTS und HSPA. LTE wurde im Marketing zwar mit dem Modebegriff *4G* beworben, streng genommen, handelt es sich jedoch um 3.9G als Weiterentwicklung von 3G. Erst LTE-Advanced erfüllt die volle 4G-Spezifikationen.

Mobilfunkzugänge können entweder für Web-, E-Mail- und andere Netzwerksoftware auf dem Mobiltelefon selbst verwendet werden, oder aber das Handy dient – beispielsweise über Bluetooth – als Mobilfunkmodem für einen Laptop oder ein Nur-WLAN-Tablet (der Fachbegriff dafür lautet *Tethering*). Speziell für UMTS oder HSPA gibt es auch eigenständige Netzwerkzugangsgeräte, die per USB an den Rechner angeschlossen werden und beinahe überall einen Internetzugang mit annehmbarer Geschwindigkeit bieten.

Um die Vorteile eines solchen Anschlusses wirklich zu nutzen, sollte aus Kostengründen ein Datenflatrate-Vertrag mit dem Mobilfunkanbieter abgeschlossen werden. Beachten Sie aber, dass diese Zugänge sehr oft eine Transfervolumenbeschränkung enthalten. Nachdem das Volumen für den entsprechenden Monat aufgebraucht ist, steht in der Regel nur noch eine langsamere Verbindung wie etwa EDGE zur Verfügung.[7] Glücklicherweise verfügen praktisch alle modernen Smartphones (und Tablets sowieso) auch über WLAN, sodass zu Hause oder am Arbeitsplatz meist keine Datenübertragung über Mobilfunk erforderlich ist.

Für den Urlaub oder ähnliche Gelegenheiten werden auch Prepaid-SIM-Karten oder USB-Sticks angeboten. Auf keinen Fall sollten Sie den Fehler machen, im Ausland ungeprüft das sogenannte *Daten-Roaming* Ihres Mobiltelefons zu aktivieren, da es immer noch recht teuer ist. Früher waren die Preise geradezu irrational (zum Beispiel 1 Euro pro 50 Kilobyte übertragener Daten), aber aufgrund von Vorschriften der EU-Kommission werden die Preise nun schrittweise gesenkt, von 0,70 € pro Megabyte im Jahr 2012 bis 0,05 € pro Megabyte im Jahr 2016.

7 Die Telekom kündigte im Frühjahr 2013 an, solche Regelungen künftig auch für DSL-Anschlüsse einführen zu wollen. Für Empörung sorgte dabei insbesondere, dass sie ihre eigenen Web-TV- und Unterhaltungsangebote nicht in dieses Volumen einzurechnen gedenkt (mögliche Verletzung der Netzneutralität). Nach sehr viel Kritik und Spott hat die Telekom die Pläne im Dezember 2013 wieder aufgegeben und die Drossel-Klausel aus allen Festnetzverträgen gestrichen. Unverbesserliche – meist kleinere – Provider, die daran festhalten, gibt es aber noch; einen Überblick erhalten Sie unter *http://werdrosselt.de*.

4.6 Die TCP/IP-Protokollfamilie

Nach einigen halbherzigen Versuchen, das OSI-Referenzmodell durch konkrete Protokolle tatsächlich zu implementieren, bemerkte man letzten Endes, dass die bereits Jahre zuvor entwickelten Internetprotokolle hervorragend als flexible, skalierbare und universelle Netzwerkprotokollfamilie einsetzbar sind. Die rasante Ausbreitung des Internets und die freie Verfügbarkeit sorgten dafür, dass diese Protokolle heute häufiger als jeder andere Protokollstapel eingesetzt werden.

Abbildung 4.3 zeigt eine konkrete Version des zuvor bereits vorgestellten TCP/IP-Protokollstapels: Auf jeder Ebene sind einige der Protokolle zu erkennen, die dort arbeiten können. Die meisten davon werden in den folgenden Abschnitten genau erläutert; die Netzzugangsprotokolle der untersten Schicht wurden ebenfalls bereits vorgestellt. Ganz zuletzt habe ich zusätzlich einige Beispiele für die Hardware angegeben, auch wenn sie kein Teil des eigentlichen TCP/IP-Stapels ist.

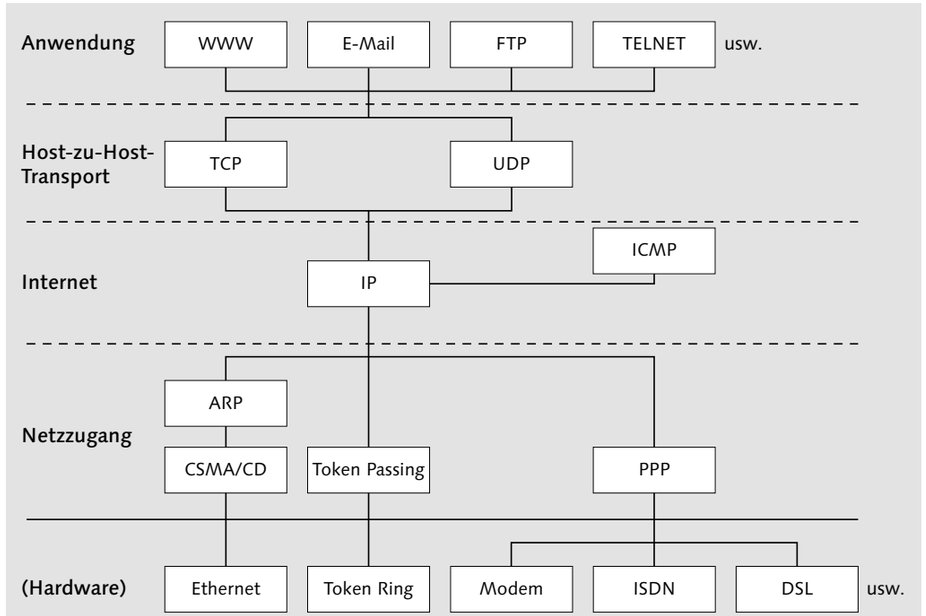


Abbildung 4.3 Der TCP/IP-Protokollstapel

Zwischen der Hardware und dem Netzzugang auf der einen und den anwendungsorientierten Protokollen auf der anderen Seite befinden sich die Protokolle der Vermittlungs- und der Transportschicht. Insgesamt werden alle Protokolle, die auf den verschiedenen Ebenen eines Schichtenmodells zusammenarbeiten, als *Protokollstapel* oder auch *Protokollfamilie* bezeichnet. Allerdings konzentriert sich der Schwerpunkt von TCP/IP auf die beiden mittleren

Ebenen des Internetprotokollstapels. Sie können zum einen auf fast jeden beliebigen Netzzugang aufsetzen, zum anderen wurde beinahe jede ernst zu nehmende Netzwerkanwendung inzwischen für diesen Protokollstapel umgesetzt – abgesehen von den klassischen Internetanwendungen, die ohnehin dafür geschrieben wurden.

Die Protokolle der mittleren Schichten sind dafür verantwortlich, dass Daten zuverlässig über verschiedene Teilnetze oder Netzwerksegmente hinweg übertragen werden können oder auch über Netze, die verschiedene Hardware oder Netzzugangsverfahren verwenden:

- Die Protokolle der Internetschicht regeln die Adressierung der Rechner und die Übertragung der Daten an den korrekten Rechner im Netzwerk. Darüber hinaus kümmern sie sich darum, dass Daten bei Bedarf in andere Teilnetze weitergeleitet werden, übernehmen also das sogenannte *Routing*.
- Auf der Host-zu-Host-Transportschicht werden die Daten in Pakete unterteilt und mit der Information versehen, welche Anwendung auf dem einen Host diese Daten an welche Anwendung auf dem anderen sendet.

Die Bezeichnung *TCP/IP* kombiniert die Namen der beiden wichtigsten Bestandteile des Protokollstapels: das Internet Protocol (IP) auf der Internetschicht und das Transmission Control Protocol (TCP), das am häufigsten verwendete Protokoll der Transportebene. In den folgenden Abschnitten werden diese Protokolle näher vorgestellt, anschließend wird die technische Seite einiger wichtiger Internetanwendungsprotokolle beleuchtet.

4.6.1 Netzzugang in TCP/IP-Netzwerken

Die unterste Ebene des Internetschichtenmodells ist der Netzzugang, nicht die Netzwerkhardware. Dies garantiert, dass sich die Internetprotokolle auf fast jeder beliebigen Hardware implementieren lassen, und in der Tat ist dies geschehen: In allen Formen von LANs wie Ethernet oder Token Ring, in WANs über Wähl- und Standleitungen wie auch über die meisten Formen drahtloser Netzen – überall laufen diese Protokolle. Dies ist ein weiterer guter Grund dafür, dass sich die Protokolle des Internets als Standard für die Netzwerkkommunikation durchsetzen konnten.

Im Grunde wird innerhalb der Spezifikation der TCP/IP-Protokolle nicht einmal der Netzzugang im OSI-Sinn beschrieben, sondern lediglich die Zusammenarbeit des IP-Protokolls, das sich innerhalb des Internetprotokollstapels um Adressierung und Routing kümmert, mit verschiedenen Netzzugangsverfahren.

An dieser Stelle sollen nur zwei der wichtigsten Internetnetzzugangsverfahren genannt werden: das für den Zugriff auf Ethernet verwendete Address Resolution Protocol (ARP) und das Point-to-Point Protocol (PPP), das für serielle Verbindungen über Modem, ISDN oder DSL eingesetzt wird. PPP wurde in diesem Kapitel bereits ausführlich beschrieben. Hier das Wichtigste zu ARP:

Das Address Resolution Protocol, beschrieben in RFC 826, übernimmt – kurz gesagt – die Umsetzung der vom Netzwerkadministrator vergebenen IP-Adressen in die vorgegebenen Hardwareadressen der Netzwerkschnittstellen.

Da die IP-Adresse den einzelnen Hosts willkürlich zugeteilt wird, kann sie auf der Netzzugangsschicht nicht bekannt sein: Auf einer bestimmten Schicht eines Protokollstapels werden die Steuerdaten der höher gelegenen Ebenen nicht ausgewertet, sondern als gewöhnliche Nutzdaten betrachtet. Deshalb kann beispielsweise eine Netzwerkkarte oder ein Hub nicht anhand der IP-Adresse entscheiden, für welche Station ein Datenpaket bestimmt ist; die Netzwerkhardware nimmt diese Adresse nicht einmal wahr.

Nachdem die IP-Software auf einem Host oder Router anhand der Empfänger-IP-Adresse festgestellt hat, dass die Daten überhaupt für das eigene Netz bestimmt sind, wird der ARP-Prozess gestartet, um diese IP-Adresse in die MAC-Adresse der Empfängerschnittstelle umzusetzen. Zu diesem Zweck sendet ein Rechner, der das ARP-Protokoll ausführt (beinahe jeder Rechner, der TCP/IP über Ethernet betreibt), ein sogenanntes *Broadcast-Datenpaket* in das Netzwerk. Es handelt sich um ein Datenpaket mit einer speziellen Empfängeradresse, das an alle Rechner im Netzwerk übertragen wird. Der Rechner, der seine eigene IP-Adresse im Inhalt dieses Pakets erkennt, antwortet als Einziger auf diese Anfrage und versendet seine eigene MAC-Adresse. Auf diese Weise wird ermittelt, für welchen Rechner das Datenpaket bestimmt ist.

In Ausnahmefällen kann ein Rechner auch die MAC-Adressen anderer Stationen zwischenspeichern und in Vertretung antworten.

4.6.2 IP-Adressen, Datagramme und Routing

Auf der Internet- oder Vermittlungsschicht des Internetprotokollstapels arbeitet das Internet Protocol (IP). Als *Internet* wird in diesem Zusammenhang jedes Netzwerk bezeichnet, das diese Protokollfamilie verwendet. Dies verdeutlicht den Umstand, dass die Internetprotokolle dem Datenaustausch über mehrere physikalische Netzwerke hinweg dienen können. Spezielle Rechner, die mindestens zwei Netzwerkschnittstellen besitzen, leiten die Daten zwischen diesen Netzen weiter. Sie werden als *IP-Router* oder *IP-Gateways* bezeichnet. Im engeren Sinne ist ein Router ein Rechner, der Daten zwischen zwei Netzen des gleichen physikalischen Typs weiterleitet; ein Gateway verbindet dagegen zwei physikalisch verschiedene Netze oder arbeitet gar auf Anwendungsebene (*Application Level Gateway*). Die beiden Begriffe werden jedoch oft synonym verwendet.

Eine IP-Adresse des klassischen Typs – der Version IPv4 gemäß RFC 791 – ist eine 32 Bit lange Zahl. Sie wird üblicherweise in vier durch Punkte getrennte Dezimalzahlen zwischen 0 und 255 geschrieben. Allerdings ist die Logik, die einer solchen Adresse zugrunde liegt, besser verständlich, wenn diese binär notiert wird:

Eine typische IP-Adresse wäre etwa `11000010000100010101000111000001`, in 8-Bit-Gruppen getrennt, ergibt sich daraus `11000010 00010001 01010001 11000001`; dies lautet in der gängigen Schreibweise dann `194.17.81.193`.

IP-Adressklassen

IP-Adressen bestehen aus zwei Komponenten: dem Netzwerkteil und dem Hostteil. Der Netzwerkteil gibt an, in welchem Netz sich der entsprechende Rechner befindet, während der Hostteil den einzelnen Rechner innerhalb dieses Netzes identifiziert.

Es gibt verschiedene Arten von IP-Adressen, die sich bezüglich der Länge des Netzwerk- beziehungsweise Hostteils voneinander unterscheiden. Traditionell wurden die verfügbaren Adressen in feste Klassen unterteilt. Bereits 1993 wurde die Klasseneinteilung aufgegeben und durch CIDR ersetzt (RFC 1518 und 1519), da sie für die rasant steigende Anzahl von Hosts erheblich zu unflexibel war. Aus diesem Grund wurde das CIDR-Verfahren entwickelt, das dynamisch zwischen dem Netzwerk- und dem Hostteil einer Adresse trennt. Dennoch sollen an dieser Stelle zuerst die ursprünglichen Klassen vorgestellt werden, denn auf diese Weise wird vieles leichter verständlich. Beachten Sie aber, dass IP-Adressen heutzutage immer über CIDR vergeben werden. Dabei wird neben der IP-Adresse auch die im Folgenden definierte Teilnetzmaske mitgeliefert, die in der historischen Klassenlogik nicht benötigt wurde.

Zu welcher Klasse eine IP-Adresse gehörte, zeigte sich an den Bits, die am weitesten links standen:

- Klasse A: Das erste Bit ist 0, folglich liegt die erste 8-Bit-Gruppe zwischen 0 und 127.
- Klasse B: Die ersten beiden Bits lauten 10; die erste Gruppe liegt im Bereich 128 bis 191.
- Klasse C: Die ersten drei Bits sind 110, sodass die erste Gruppe zwischen 192 bis 223 liegt.
- Klasse D: Die ersten vier Bits sind 1110; die Adressen beginnen mit 224 bis 239.

Die restlichen Adressen, die mit 240 bis 255 anfangen, wurden weder im Klassenbereich noch über CIDR vergeben und sind für zukünftige Anwendungszwecke reserviert. Diejenigen mit der Anfangssequenz 11110 (Start-Byte 240 bis 247) wurden manchmal trotzdem als *Klasse E* bezeichnet.

Je nach Klasse ist der Teil, der das Netzwerk kennzeichnet, unterschiedlich lang, entsprechend existieren unterschiedlich viele Netze der verschiedenen Klassen. Die Bits, die ganz rechts in der Adresse stehen und nicht zum Netzwerkteil gehören, sind die Host-Bits. Je nach Länge des Netzwerkteils bleiben unterschiedlich viele Bits für den Hostteil übrig, sodass die Höchstzahl der Rechner in einem Netz variiert.

Tabelle 4.5 zeigt die wichtigsten Informationen zu den einzelnen Klassen im Überblick. In der Spalte »Netzwerk-Bits« stehen jeweils zwei Werte. Der erste stellt die Anzahl von Bits dar, die insgesamt den Netzwerkteil bilden. Da die Grenzen zwischen Netzwerk- und Hostteil an den Byte-Grenzen verlaufen, handelt es sich je nach Klasse um 1 bis 3 Byte. Da jedoch die Bits am Anfang der Adresse – wie zuvor gezeigt – die Klasse angeben, besteht die praktisch nutz-

bare Netzwerkangabe nur aus 7, 14 beziehungsweise 21 Bit. Der Rest der Adresse bildet den Hostteil, der je nach Klasse unterschiedlich groß ausfällt.

Klasse	Adressbereich	Netzwerk-Bits	Host-Bits	Anzahl Netze	Adressen pro Netz
A	0.0.0.0 bis 127.255.255.255	8 (7)	24	128	16,7 Mio.
B	128.0.0.0 bis 191.255.255.255	16 (14)	16	16.384	65.536
C	192.0.0.0 bis 223.255.255.255	24 (21)	8	2.097.152	256
D	224.0.0.0 bis 239.255.255.255	spezieller Bereich der Multicast-Adressen			

Tabelle 4.5 Die IP-Adressklassen

Innerhalb eines einzelnen Netzes – egal, welcher Klasse – stehen die erste und die letzte mögliche Adresse nicht als Hostadressen zur Verfügung: Die niedrigste Adresse identifiziert das gesamte Netz als solches nach außen hin, aber keinen speziellen Host; die höchste Adresse ist die sogenannte *Broadcast-Adresse*: Werden Datenpakete innerhalb des Netzes an diese Adresse gesendet, werden sie von jedem Host empfangen.

Zum Beispiel bilden die Adressen, die mit 18.x.x.x beginnen, das Klasse-A-Netzwerk 18.0.0.0 mit der Broadcast-Adresse 18.255.255.255 und Hostadressen von 18.0.0.1 bis 18.255.255.254. Dieses Netz kann theoretisch bis zu 16.777.214 Hosts beherbergen (2^{24}–2).

Die Adressen, die mit 162.21.x.x anfangen, befinden sich in dem Klasse-B-Netzwerk 162.21.0.0, dessen Broadcast-Adresse 162.21.255.255 lautet. Es kann bis zu 65.534 Hosts (2^{16}–2) mit den Adressen 162.21.0.1 bis 162.21.255.254 enthalten.

Letztes Beispiel: Adressen, die mit 201.30.9.x beginnen, liegen in dem Klasse-C-Netz 201.30.9.0 mit der Broadcast-Adresse 201.30.9.255; die 254 möglichen Hostadressen (2^8–2) sind 201.30.9.1 bis 201.30.9.254.

Die sogenannten *Multicast-Adressen* der Pseudoklasse D nehmen eine Sonderstellung ein: Eine Multicast-Gruppe ist eine auf beliebige Netze verteilte Gruppe von Hosts, die sich dieselbe Multicast-IP-Adresse teilen. Dies ermöglicht einen erheblich ökonomischeren Versand von Daten, da sie nicht mehr je einmal pro empfangendem Host versendet werden, sondern nur noch kopiert werden müssen, wo Empfängerrechner in unterschiedlichen Teilnetzen liegen. Aus diesem Grund ist Multicasting eine zukunftsträchtige Technologie für dateninten sive Anwendungen wie etwa Videokonferenzen. Im Gegensatz dazu werden die individuellen Hostadressen als *Unicast-Adressen* bezeichnet.

Die Verteilung der IP-Adressen

Alle Adressen des IPv4-Adressraums werden von der Internet Assigned Numbers Association (IANA) verwaltet. Falls Sie jedoch für bestimmte Anwendungen in Ihrem Unternehmen eine oder mehrere feste IP-Adressen benötigen, dann sollten Sie sich in der Regel an einen Internetprovider und nicht an die IANA selbst wenden.

Die 128 Netze der Klasse A sind bereits alle vergeben; in der Regel an große internationale Unternehmen aus dem Elektronik- und Computerbereich sowie an US-amerikanische Staats-, Militär- und Bildungsinstitutionen. Beispielsweise gehört das Netz 17.0.0.0 der Firma Apple, 18.0.0.0 dem Massachusetts Institute of Technology (MIT) und 19.0.0.0 der Ford Motor Company.

Die 16.384 Klasse-B-Netze sind ebenfalls weitgehend vergeben, insbesondere an US-amerikanische Unternehmen und Internetprovider.

Die mehr als zwei Millionen Netze der Klasse C schließlich sind inzwischen ebenfalls überwiegend belegt. Die meisten von ihnen gehören Unternehmen und Internetprovidern, die nicht in den USA ansässig sind, sondern etwa in Europa oder Asien. Da solche Institutionen oft mehr als 254 Hosts in ihrem Netz betreiben, wird ihnen häufig ein größerer Block aufeinanderfolgender Klasse-C-Netze zugewiesen.

Die aktuelle Verteilung der IPv4-Adressen können Sie auf der Website der IANA unter *http://www.iana.org/assignments/ipv4-address-space* einsehen.

Als das Konzept der IP-Adressen entstand, konnte niemand auch nur ansatzweise erahnen, welche Dimensionen das Internet einmal annehmen würde. Deshalb glaubten die ursprünglichen Entwickler, dass sie es sich leisten könnten, den Adressraum relativ großzügig aufzuteilen – bedenken Sie etwa, dass die Hälfte des Adressraums für die überaus ineffektiven Klasse-A-Adressen vergeudet wird. Um die drohende Verknappung der IP-Adressen zu verhindern oder zumindest zu verzögern, bis eine Alternative gefunden würde, wurden einige Adressbereiche zur Verwendung in privaten Netzwerken freigegeben, die nicht (oder nicht direkt) mit dem Internet verbunden sind. Es handelt sich um die folgenden Blöcke:

- das Klasse-A-Netz 10.0.0.0
- die 16 Klasse-B-Netze 172.16.0.0 bis 172.31.0.0
- die 256 Klasse-C-Netze 192.168.0.0 bis 192.168.255.0

Ein weiterer Block, der erst später freigegeben wurde, ist das Klasse-B-Netz 169.254.0.0, das einem besonderen Verwendungszweck vorbehalten ist: Moderne TCP/IP-Implementierungen in fast allen Betriebssystemen verwenden dieses Netz für »link local« – eine Möglichkeit, sich automatisch selbst IP-Adressen zuzuweisen, falls wider Erwarten keine Verbindung zu einem DHCP-Server hergestellt werden kann, der eigentlich für die automatische Zuweisung von Adressen zuständig wäre.

Zu guter Letzt existieren noch einige Netze mit anderen speziellen Bedeutungen:

- Die Adresse 0.0.0.0 kann innerhalb eines Netzes verwendet werden, um sich auf das aktuelle Netz selbst zu beziehen.
- Das Klasse-A-Netz 127.0.0.0 beherbergt den sogenannten *Loopback-Bereich*: Über das Loopback-Interface, eine virtuelle Netzwerkschnittstelle mit der Adresse 127.0.0.1, kann ein Host mit sich selbst Netzwerkkommunikation betreiben. Dies ist zum Beispiel nützlich, um während der Programmierung von Client-Server-Anwendungen sowohl das Client- als auch das Serverprogramm auf dem lokalen Host laufen zu lassen.
- Schließlich wird die Adresse 255.255.255.255 als universelle Broadcast-Adresse verwendet: Ein Datenpaket, das an diese Adresse gesendet wird, wird wie beim normalen Broadcast von allen Hosts im Netzwerk empfangen. Nützlich ist diese Einrichtung für Schnittstellen, die ihre IP-Adresse dynamisch beziehen, da sie bei Inbetriebnahme in der Regel noch nicht einmal wissen, in welchem Netz sie sich eigentlich befinden. Auf diese Weise erhalten sie überhaupt erst die Möglichkeit, die Zuteilung einer Adresse anzufordern.

Die Vergabe der privaten Adressbereiche ist in RFC 1918 geregelt; die Festlegung der anderen speziellen Adressbereiche findet sich in RFC 3330.

Supernetting, Subnetting und CIDR

In der neueren Entwicklungsgeschichte des Internets hat sich herausgestellt, dass die traditionellen Adressklassen nicht für alle Anwendungsbereiche flexibel genug sind. Deshalb wurde ein neues Schema entwickelt, das die Trennlinie zwischen Netz- und Hostteil der Adressen an einer beliebigen Bit-Grenze ermöglicht. Das in RFC 1519 beschriebene Verfahren heißt Classless Inter-Domain Routing (CIDR).

Die folgenden beiden Anwendungsbeispiele verdeutlichen typische Probleme mit der alten Klassenlogik, die mithilfe von CIDR gelöst werden können:

- Ein Unternehmen besitzt das Klasse-B-Netzwerk 139.17.0.0. Es wäre jedoch wünschenswert, wenn die vier Filialen des Unternehmens jeweils unabhängige Netze betreiben könnten. Dazu soll das vorhandene Netz in vier Teile unterteilt werden – ein Fall für das sogenannte *Subnetting*.
- Ein vor Kurzem neu gegründeter europäischer Internetprovider hat die 1.024 Klasse-C-Netze 203.16.0.0 bis 203.19.255.0 erhalten. Das Unternehmen möchte diese Netze als ein großes Netz verwalten, da dies die dynamische Zuteilung an Kunden bei der Einwahl erheblich vereinfacht. Eine solche Zusammenfassung von Netzen wird *Supernetting* genannt.

Das Prinzip von CIDR basiert darauf, dass die traditionellen Byte-Grenzen zwischen Netz- und Hostteil völlig aufgehoben werden. Deshalb ist die Größe des Netzes bei einem CIDR nicht mehr am Beginn der Adresse zu erkennen. Stattdessen wird die Anzahl der Bits, die den

Netzwerkteil der Adresse bilden, durch einen Slash getrennt hinter der Netzwerkadresse notiert. Zum Beispiel wird das Klasse-A-Netz 14.0.0.0 zu 14.0.0.0/8.

Eine alternative Darstellungsform für die Grenze zwischen Netz- und Hostteil bei CIDR-Adressen – insbesondere in der IP-Konfiguration der meisten Betriebssysteme – stellt die Teilnetzmaske (Subnet Mask) dar. In dieser Maske werden für die Bits des Netzwerkteils am Anfang der Adresse Einsen notiert, für die Bits des Hostteils am Ende der Adresse dagegen Nullen. Genau wie die IP-Adresse selbst wird auch die Teilnetzmaske in vier dezimalen 8-Bit-Blöcken geschrieben.

Wie bereits erwähnt, wird die Klasseneinteilung bereits seit Mitte der 90er-Jahre nicht mehr verwendet. Deshalb sind alle modernen Netzwerke auf sie angewiesen, da sie allein den Netzwerkteil einer Adresse kennzeichnet. Alle modernen Betriebssysteme verwenden Protokollaufrufe mit Teilnetzmaske. Die Netzwerkadresse wird dabei mithilfe einer Bitweise-Und-Operation aus IP-Adresse und Teilnetzmaske berechnet. Hier ein Beispiel:

IP-Adresse: 192.168.0.37
Teilnetzmaske: 255.255.255.0
Berechnete Netzwerkadresse: 192.168.0.0

Hier zum Vergleich die Binärdarstellung:

```
  11000000 10101000 00000000 00100101
& 11111111 11111111 11111111 00000000
-----------------------------------
  11000000 10101000 00000000 00000000
```

Tabelle 4.6 zeigt Beispiele für die Schreibweise der ursprünglichen klassenbasierten Adressen nach CIDR-Logik sowie ihre Teilnetzmasken.

Klasse	Beispielnetz	CIDR-Adresse	Teilnetzmaske
A	17.0.0.0	17.0.0.0/8	255.0.0.0
B	167.18.0.0	167.18.0.0/16	255.255.0.0
C	195.21.92.0	195.21.92.0/24	255.255.255.0

Tabelle 4.6 Die traditionellen IP-Adressklassen in CIDR-Darstellung

Das Subnetting aus dem ersten Beispiel, die Unterteilung des Netzes 139.17.0.0/16 in vier gleich große Teilnetze, kann folgendermaßen durchgeführt werden:

- Da die 65.536 rechnerischen Adressen in vier Teile unterteilt werden sollen, sind zwei weitere Bits für den Netzwerkteil der Adresse erforderlich ($4 = 2^2$).
- Da das ursprüngliche Klasse-B-Netz einen 16 Bit (zwei Byte) langen Netzwerkteil besitzt, erfolgt die Unterteilung der vier Adressbereiche nach Bit 18, also nach dem zweiten Bit des

dritten Bytes; die vier neuen Netze sind demnach 139.17.0.0/18, 139.17.64.0/18, 139.17.128.0/18 sowie 139.17.192.0/18.

Tabelle 4.7 zeigt die Eigenschaften der vier neuen Netze.

Netzwerk	1. Hostadresse	letzte Hostadresse	Broadcast-Adresse	Teilnetzmaske
139.17.0.0/18	139.17.0.1	139.17.63.254	139.17.63.255	255.255.192.0
139.17.64.0/18	139.17.64.1	139.17.127.254	139.17.127.255	255.255.192.0
139.17.128.0/18	139.17.128.1	139.17.191.254	139.17.191.255	255.255.192.0
139.17.192.0/18	139.17.192.1	139.17.255.254	139.17.255.255	255.255.192.0

Tabelle 4.7 Subnetting – Unterteilung des Netzes 139.17.0.0/16 in vier gleich große Teilnetze

Im zweiten Beispiel geht es um Supernetting, also um die Zusammenfassung einzelner Netze zu einem größeren Gesamtnetz. Die Netze 203.16.0.0/24 bis 203.19.255.0/24 sollen zu einem einzigen Netz verbunden werden. Diese Aufgabe lässt sich auf folgende Weise lösen:

- Es werden 1.024 Klasse-C-Netze miteinander verbunden. 256 Netze der Klasse C ergäben einfach ein Gesamtnetz von der Größe eines Klasse-B-Netzwerks. Beispielsweise würde die Vereinigung der Netze 203.16.0.0/24 bis 203.16.255.0/24 das neue Netz 203.16.0.0/16 erzeugen. Um das gewünschte Netz der vierfachen Größe zu erhalten, muss die Grenze zwischen Netz- und Hostteil noch um zwei Bits weiter nach links verschoben werden.
- Die Adresse wird zwei Bits links von der Klasse-B-Grenze, also vor dem vorletzten Bit des zweiten Bytes, unterteilt. Daraus ergibt sich die Netzwerkadresse 203.16.0.0/14 mit der Teilnetzmaske 255.252.0.0. Die Broadcast-Adresse des neuen Netzes ist 203.19.255.255; die möglichen Hostadressen reichen von 203.16.0.1 bis 203.19.255.255.

Im Allgemeinen bietet es sich an, die Teilnetzmaske des ursprünglichen Netzes, das aufgeteilt oder mit mehreren verbunden werden soll, zunächst in die Binärdarstellung umzurechnen. In dieser Schreibweise fällt es am leichtesten, die Grenze zwischen Netz- und Hostteil um die gewünschte Anzahl von Bits nach links oder nach rechts zu verschieben. Anschließend können Sie die Maske wieder in die vier üblichen 8-Bit-Gruppen unterteilen und in Dezimalzahlen umrechnen.

Diese Vorgehensweise soll im Folgenden an zwei weiteren Beispielen demonstriert werden. Das Klasse-B-Netzwerk 146.20.0.0/16 soll in acht Teilnetze unterteilt werden:

- Die ursprüngliche Netzmaske ist 255.255.0.0.
- In binärer Darstellung entspricht dies 11111111 11111111 00000000 00000000.
- Eine Aufteilung in acht Netze erfolgt durch eine Verschiebung der Grenze zwischen den beiden Adressteilen um drei Stellen ($8 = 2^3$) nach rechts.

- Die neue Netzmaske in binärer Schreibweise ist 11111111 11111111 11100000 00000000.
- Nach der erneuten Umrechnung in die dezimale Vierergruppen-Darstellung ergibt sich 255.255.224.0.
- Entsprechend ergeben sich die folgenden acht Netze:
  - 146.20.0.0/19
  - 146.20.32.0/19
  - 146.20.64.0/19
  - 146.20.96.0/19
  - 146.20.128.0/19
  - 146.20.160.0/19
  - 146.20.192.0/19
  - 146.20.224.0/19

Die vier Klasse-C-Netzwerke 190.16.0.0/24 bis 190.16.3.0/24 sollen zu einem gemeinsamen Netz verbunden werden:

- Die Teilnetzmaske der vier Netze lautet jeweils 255.255.255.0.
- Binär geschrieben, ergibt sich daraus 11111111 11111111 11111111 00000000.
- Die Zusammenfassung vier solcher Netze erfordert eine Verschiebung der Adressgrenze um zwei Bits ($4 = 2^2$) nach links.
- In Binärdarstellung lautet die neue Maske 11111111 11111111 11111100 00000000.
- Wird diese Maske wieder in Dezimalschreibweise umgerechnet, resultiert daraus 255.255.252.0.
- Das neue Netz besitzt die CIDR-Adresse 190.16.0.0/22.

Die folgenden Tabellen zeigen in übersichtlicher Form, wie die Aufteilung der alten IP-Adressklassen in verschiedene Anzahlen von Teilnetzen funktioniert. In Tabelle 4.8 wird die Klasse A behandelt. Die – rein rechnerisch mögliche – Zusammenfassung mehrerer Klasse-A-Netze durch Supernetting wird in der Praxis nicht durchgeführt, weil erstens wohl niemand mehr als 16,7 Millionen Hosts in einem Teilnetz betreiben möchte und zweitens bereits alle Klasse-A-Netze an einzelne Betreiber vergeben wurden.

Netzwerk-Bits	Host-Bits	Anzahl Teilnetze	Anzahl Hosts	Teilnetzmaske
8	24	1	16.777.214	255.0.0.0
9	23	2	8.388.606	255.128.0.0
10	22	4	4.194.302	255.192.0.0
11	21	8	2.097.150	255.224.0.0

Tabelle 4.8 Bildung von CIDR-Teilnetzen aus einem Klasse-A-Netz

Netzwerk-Bits	Host-Bits	Anzahl Teilnetze	Anzahl Hosts	Teilnetzmaske
12	20	16	1.048.574	255.240.0.0
13	19	32	524.286	255.248.0.0
14	18	64	262.142	255.252.0.0
15	17	128	131.070	255.254.0.0
16	16	256	65.534	255.255.0.0
17	15	512	32.766	255.255.128.0
18	14	1.024	16.382	255.255.192.0
19	13	2.048	8.190	255.255.224.0
20	12	4.096	4.094	255.255.240.0
21	11	8.192	2.046	255.255.248.0
22	10	16.384	1.022	255.255.252.0
23	9	32.768	510	255.255.254.0
24	8	65.536	254	255.255.255.0
25	7	131.072	126	255.255.255.128
26	6	262.144	62	255.255.255.192
27	5	524.288	30	255.255.255.224
28	4	1.048.576	14	255.255.255.240
29	3	2.097.152	6	255.255.255.248
30	2	4.194.302	2	255.255.255.252

Tabelle 4.8 Bildung von CIDR-Teilnetzen aus einem Klasse-A-Netz (Forts.)

Tabelle 4.9 zeigt die Aufteilung eines Klasse-B-Netzes in beliebig kleine Teilnetze.

Netzwerk-Bits	Host-Bits	Anzahl Teilnetze	Anzahl Hosts	Teilnetzmaske
16	16	1	65.534	255.255.0.0
17	15	2	32.766	255.255.128.0
18	14	4	16.382	255.255.192.0

Tabelle 4.9 Bildung von CIDR-Teilnetzen aus einem Klasse-B-Netz

Netzwerk-Bits	Host-Bits	Anzahl Teilnetze	Anzahl Hosts	Teilnetzmaske
19	13	8	8.190	255.255.224.0
20	12	16	4.094	255.255.240.0
21	11	32	2.046	255.255.248.0
22	10	64	1.022	255.255.252.0
23	9	128	510	255.255.254.0
24	8	256	254	255.255.255.0
25	7	512	126	255.255.255.128
26	6	1.024	62	255.255.255.192
27	5	2.048	30	255.255.255.224
28	4	4.096	14	255.255.255.240
29	3	8.192	6	255.255.255.248
30	2	16.384	2	255.255.255.252

Tabelle 4.9 Bildung von CIDR-Teilnetzen aus einem Klasse-B-Netz (Forts.)

Tabelle 4.10 demonstriert schließlich, wie die Unterteilung eines Klasse-C-Netzes erfolgt. In kleineren Unternehmen könnte es durchaus praktisch sein, ein solches – ohnehin kleines – Netzwerk weiter zu unterteilen.

Netzwerk-Bits	Host-Bits	Anzahl Teilnetze	Anzahl Hosts	Teilnetzmaske
24	8	1	254	255.255.255.0
25	7	2	126	255.255.255.128
26	6	4	62	255.255.255.192
27	5	8	30	255.255.255.224
28	4	16	14	255.255.255.240
29	3	32	6	255.255.255.248
30	2	64	2	255.255.255.252

Tabelle 4.10 Bildung von CIDR-Teilnetzen aus einem Klasse-C-Netz

In der Praxis ermöglicht CIDR bereits einen erheblich flexibleren Netzwerkaufbau als die Verwendung der alten Klassen. Doch auch diese Verfahrensweise kann immer noch ungüns-

tige Ergebnisse zur Folge haben, wenn Teilnetze mit erheblich unterschiedlichen Größen benötigt werden: Das größte benötigte Teilnetz bestimmt die Größe aller anderen; selbst das kleinste belegt eine Menge von Adressen, die es womöglich niemals benötigen wird.

Aus diesem Grund wurde das VLSM-Konzept (Variable Length Subnet Mask) eingeführt. Es handelt sich um ein spezielles Subnetting-Verfahren, bei dem ein vorhandenes Netz nicht mehr in gleich große, sondern in verschieden große Teilnetze unterteilt wird. Jedem dieser Teilnetze wird eine individuelle Teilnetzmaske zugewiesen.

Das grundlegende Prinzip von VLSM besteht darin, vom kleinsten benötigten Teilnetz auszugehen und die entsprechenden größeren Netze aus Blöcken solcher kleinsten Teilnetze zu bilden, denen dann höhere Teilnetzmasken zugewiesen werden. Angenommen etwa, bei der Aufteilung eines Klasse-B-Netzes mit seinen 65.534 Hostadressen besitzt das kleinste gewünschte Teilnetz zwölf Hosts, das größte etwa 500. Für die zwölf Hosts ist mindestens ein Netz mit der Teilnetzmaske 255.255.255.240 erforderlich, das 14 Hostadressen bietet. Aus diesen kleinen Teilnetzen können dann entsprechend größere aufgebaut werden, wobei die Grenzen zwischen den Netzen der Logik der jeweiligen Netzmaske entsprechen müssen.

An dieser Stelle soll ein einfaches Beispiel genügen: Ein Unternehmen betreibt das öffentliche Klasse-C-Netz 196.17.41.0/24. Dieses Netz soll auf die drei Abteilungen der Firma aufgeteilt werden; die beiden Router und die drei Server sollen ein viertes separates Teilnetz bilden. Tabelle 4.11 zeigt die klassische Aufteilung des Netzes in vier gleich große Teile nach CIDR-Logik.

Bereich	Anzahl Hosts	Teilnetz	maximale Hosts	freie Adressen
Server/Router	5	196.17.41.0/26	62	57
Verwaltung	20	196.17.41.64/26	62	42
Programmierung	61	196.17.41.128/26	62	1
Design	30	196.17.41.192/26	62	32

Tabelle 4.11 Aufteilung des Netzes 196.17.41.0/24 in vier Teile nach dem CIDR-Schema

Es ist leicht zu erkennen, dass zwei der Teilnetze – Server/Router und Verwaltung – vollkommen überdimensioniert sind, während zumindest das Teilnetz der Programmierabteilung beinahe seine Belastungsgrenze erreicht hat. Stellen Sie sich vor, es werden noch zwei weitere Hosts in diese Abteilung aufgenommen: Schon wäre das Teilnetz zu klein, und es müsste über eine andere Verteilung nachgedacht werden. In diesem Beispiel könnte sie nur noch darin bestehen, zwei der anderen Bereiche zusammenzulegen, um den Programmierbereich zu vergrößern.

Eine komplexere, aber für den konkreten Anwendungsfall sinnvollere Aufteilung des Netzes mithilfe der VLSM-Technik zeigt Tabelle 4.12.

Bereich	Anzahl Hosts	Teilnetz	maximale Hosts	freie Adressen
Server/Router	5	196.17.41.0/27	30	25
Verwaltung	20	196.17.41.32/27	30	10
Design	30	196.17.41.64/26	62	32
Programmierung	61	196.17.41.128/25	126	65

Tabelle 4.12 Flexible Aufteilung des Netzes 196.17.41.0/24 in vier Teile nach dem VLSM-Schema

Für die IP-Konfiguration eines einzelnen Hosts macht es keinen Unterschied, ob das Teilnetz, in dem er sich befindet, nach der alten Klassenlogik, nach dem CIDR-Verfahren oder nach der VLSM-Methode konfiguriert wurde: In jedem Fall wird im Konfigurationsdialog des jeweiligen Betriebssystems die korrekte Teilnetzmaske eingestellt. Spezielle Unterstützung für VLSM benötigen lediglich Router, die in dem betroffenen Netz eingesetzt werden. Die meisten neueren Routing-Protokolle bieten diese Unterstützung.

Die Übertragung von IP-Datagrammen

Auf der Internetschicht des TCP/IP-Protokollstapels, auf der das IP-Protokoll arbeitet, werden die Datenpakete, wie bereits erwähnt, als *Datagramme* bezeichnet. Um die Datenübertragung mithilfe des IP-Protokolls genau zu erläutern, soll an dieser Stelle zunächst der IP-Header vorgestellt werden. Er enthält die Steuerdaten, die das IP-Protokoll zu einem Datenpaket hinzufügt, das ihm vom übergeordneten Transportprotokoll übergeben wird.

Der IPv4-Protokoll-Header wird wie das gesamte Protokoll in RFC 791 definiert. Seine Länge beträgt mindestens 20 Byte, dazu können bis zu 40 Byte Optionen kommen. Tabelle 4.13 zeigt den genauen Aufbau.

Byte	0		1	2	3
0	Version	IHL	Type of Service	Paketgesamtlänge	
4	Identifikation			Flags	Fragment-Offset
8	Time to Live		Protokoll	Header-Prüfsumme	
12	Quelladresse				
16	Zieladresse				
20	Optionen				Padding
...	eventuell weitere Optionen				

Tabelle 4.13 Aufbau des IPv4-Datagramm-Headers

Die einzelnen Daten des IP-Headers sind folgende:

- Version (4 Bit): Die Versionsnummer des IP-Protokolls, die das Paket verwendet – bei IPv4, wie der Name schon sagt, die Version 4.
- IHL (4 Bit): Internet Header Length; die Länge des Internet-Headers in 32-Bit-Wörtern (entsprechen den Zeilen in Tabelle 4.13). Der kleinste mögliche Wert beträgt 5.
- Type of Service (8 Bit): Ein Code, der die Art des Datenpakets bestimmt. Bestimmte Sorten von Paketen, etwa für den Austausch von Routing- oder Statusinformationen, werden von bestimmten Netzen bevorzugt weitergeleitet.
- Paketgesamtlänge (16 Bit): Die Gesamtlänge des Datagramms in Bytes, Header und Nutzdaten.
- Identifikation (16 Bit): Ein durch den Absender frei definierbarer Identifikationswert, der beispielsweise das Zusammensetzen fragmentierter Datagramme ermöglicht.
- Flags (3 Bit): Kontrollflags, die die Paketfragmentierung regeln. Das erste Bit ist reserviert und muss immer 0 sein, das zweite (DF) bestimmt, ob das Paket fragmentiert werden darf (Wert 1) oder nicht (0), das dritte (MF) regelt, ob dieses Paket das letzte Fragment (0) ist oder ob weitere Fragmente folgen (1).
- Fragment-Offset (13 Bit): Dieser Wert (angegeben in 64-Bit-Blöcken) legt fest, an welcher Stelle in einem Gesamtpaket dieses Paket steht, falls es sich um ein Fragment handelt. Das erste Fragment oder ein nicht fragmentiertes Paket erhält den Wert 0.
- Time to Live (8 Bit): Der TTL-Mechanismus sorgt dafür, dass Datagramme nicht endlos im Internet weitergeleitet werden, falls die Empfängerstation nicht gefunden wird. Jeder Router, der ein Datagramm weiterleitet, zieht von diesem Wert 1 ab; wird der Wert 0 erreicht, leitet der betreffende Router das Paket nicht mehr weiter, sondern verwirft es.
- Protokoll (8 Bit): Die hier gespeicherte Nummer legt fest, für welches Transportprotokoll der Inhalt des Datagramms bestimmt ist, zum Beispiel 6 für TCP oder 17 für UDP. Diese beiden wichtigsten Transportprotokolle werden im nächsten Abschnitt beschrieben.
- Header-Prüfsumme (16 Bit): Die Prüfsumme stellt eine einfache Plausibilitätskontrolle für den Datagramm-Header zur Verfügung. Ein Paket, dessen Header-Prüfsumme nicht korrekt ist, wird nicht akzeptiert und muss erneut versendet werden.
- Quelladresse und Zieladresse (je 32 Bit): Die IP-Adressen von Absender und Empfänger. IP-Adressen wurden zuvor bereits ausführlich behandelt.
- Optionen (variable Länge): Die meisten IP-Datagramme werden ohne zusätzliche Optionen versandt, da Absender- und Empfängerhost sowie alle auf dem Weg befindlichen Router die jeweils verwendeten Optionen unterstützen müssen. Zu den verfügbaren Optionen gehören unter anderem Sicherheitsfeatures und spezielle Streaming-Funktionen.

Das Problem der Paketfragmentierung entsteht dadurch, dass verschiedene physikalische Netzarten unterschiedliche Maximallängen für Datenpakete erlauben. Dieser Wert, der als *Maximum Transmission Unit* (MTU) bezeichnet wird, kann bei einigen Netzwerkschnittstellen per Software konfiguriert werden, bei anderen ist er vom Hersteller vorgegeben. Werden nun Datagramme aus einem Netz mit einer bestimmten MTU in ein anderes Netz mit einer kleineren MTU weitergeleitet, dann müssen die Daten in kleinere Pakete »umgepackt« werden. Wie bereits beschrieben, werden sie dazu mit Fragmentierungsinformationen versehen, damit sie später wieder richtig zusammengesetzt werden können.

Solange Quell- und Zieladresse im gleichen Netzwerk liegen, ist die Übertragung der Datagramme sehr einfach: Je nach Netzwerkart wird auf die passende Art (bei Ethernet zum Beispiel über ARP) diejenige Schnittstelle ermittelt, für die die Daten bestimmt sind. Anschließend wird das Datagramm an den korrekten Empfänger übermittelt. Dieser liest den IP-Header des Pakets, setzt eventuelle Fragmente wieder richtig zusammen und übermittelt das Paket an das Transportprotokoll, dessen Nummer im Header angegeben ist. Wie der Transportdienst mit den Daten umgeht, erfahren Sie im nächsten Abschnitt.

IPv6

Bereits in der ersten Hälfte der 90er-Jahre wurde damit gerechnet, dass sehr bald keine weiteren IPv4-Adressen mehr verfügbar sein würden. Dass dies viel länger als gedacht noch nicht der Fall war, lag an der Einführung von CIDR, VLSM und NAT (Letzteres wird im übernächsten Unterabschnitt, »Weitere IP-Dienste«, beschrieben). Da das Internet aber weiterhin wächst, ist es nur noch eine Frage der Zeit, bis die Anzahl der Adressen endgültig erschöpft ist; was die Zuteilung an Dienstleister angeht, ist dies sogar jetzt schon der Fall.

Deshalb wurde schon vor einigen Jahren mit der Arbeit an einem Nachfolger für das IPv4-Protokoll begonnen, der vor allem einen größeren Adressraum durch längere IP-Adressen besitzen sollte. Letzten Endes entschieden die Entwickler sich für Adressen von 128 Bit Länge. Dies ergibt theoretisch mehr als $3{,}4 \times 10^{38}$ verschiedene Adressen! Damit erscheint der Adressraum mehr als überdimensioniert; offensichtlich kann man damit jedem einzelnen Sandkorn auf unserem Planeten mehrere eigene IP-Adressen zuweisen. Letzten Endes geht es allerdings eher darum, beinahe beliebig viele Netze von sehr unterschiedlicher Größe einrichten zu können. Abgesehen davon, werden immer mehr tragbare Geräte entwickelt, die mit Netzwerken verbunden werden – etwa dynamisch über öffentliche WLAN-Access-Points.

Die aktuelle Version des neuen IP-Protokolls wird in RFC 2460 beschrieben. Da die Version 5 für Experimente mit Multicasting verwendet wurde, lautet die Versionsnummer des Protokolls IPv6; während seiner Entwicklung wurde es auch manchmal als *IPng* (für »next generation«) bezeichnet, zum Beispiel in RFC 1752, das den ersten Arbeitsentwurf beschreibt. Die IPv6-Adresse wird nicht in 8-Bit-Dezimalgruppen geschrieben wie bei IPv4; mit 16 Gruppen wäre sie ein wenig unhandlich. Stattdessen schreibt man acht vierstellige Hexadezimalgruppen, die durch Doppelpunkte getrennt werden. Eine IPv6-Adresse sieht zum Beispiel folgendermaßen aus:

4A29:30B4:0031:0000:0000:0092:1A3B:3394

Eine zulässige Verkürzung besteht darin, führende Nullen in einem Block wegzulassen sowie Blöcke, die nur aus Nullen bestehen, durch zwei aufeinanderfolgende Doppelpunkte zu ersetzen. Kurz gefasst, lautet die Beispieladresse also 4A29:30B4:31::92:1A3B:3394. Um die Adresse eindeutig zu halten, darf eine solche Verkürzung innerhalb einer Adresse nur einmal vorgenommen werden.

Genau wie IPv4-Adressen werden auch die neuen IPv6-Adressen in zwei Teile unterteilt: Links steht ein Präfix, dahinter ein Individualteil, der dem Hostteil der IPv4-Adresse entspricht. Das Präfix gibt allerdings nicht das einzelne Netz an, zu dem die Adresse gehört, sondern informiert über den Adresstyp. Da die Präfixe wie bei IPv4 unterschiedliche Längen aufweisen können, wird das Präfix zusammen mit seiner Bit-Anzahl angegeben. Tabelle 4.14 gibt Ihnen einen Überblick über die verschiedenen Adressblöcke und ihre Verwendung.

Präfix	Verwendung
0::0/8	Reserviert für spezielle Anwendungen.
100::0/8	noch nicht zugeordnet
200::0/7	Abbildung von NSAP-Adressen
400::0/7	Abbildung von IPX-Adressen
600::0/7	noch nicht zugeordnet
800::0/5	noch nicht zugeordnet
1000::0/4	noch nicht zugeordnet
2000::0/3	global eindeutige Adressen
6000::0/3	noch nicht zugeordnet
8000::0/3	noch nicht zugeordnet
A000::0/3	noch nicht zugeordnet
C000::0/3	noch nicht zugeordnet
E000::0/4	noch nicht zugeordnet
F000::0/5	noch nicht zugeordnet
F800::0/6	noch nicht zugeordnet
FE00::0/7	noch nicht zugeordnet
FE00::0/9	noch nicht zugeordnet

Tabelle 4.14 IPv6-Adressbereiche und -Präfixe

Präfix	Verwendung
FE80::0/10	auf eine Verbindung begrenzte Adressen
FEC0::0/10	auf eine Einrichtung begrenzte Adressen
FF00::0/8	Multicast-Adressen

Tabelle 4.14 IPv6-Adressbereiche und -Präfixe (Forts.)

Die typischste Form von IPv6-Adressen, deren Stil am ehesten den öffentlich gerouteten IPv4-Adressen entspricht, ist die globale Unicast-Adresse. Ihre Struktur ist in RFC 2374 festgelegt und sieht folgendermaßen aus:

- externes Routing-Präfix (48 Bit)
- Site-Topologie (üblicherweise 16 Bit)
- Schnittstellen-Identifikationsnummer (normalerweise 64 Bit); wird in der Regel automatisch generiert, oft aus der MAC-Adresse der Schnittstelle oder aus der bisherigen IPv4-Adresse

Der IPv6-Datagramm-Header wurde gegenüber dem IPv4-Header erheblich vereinfacht. Durch die Auslagerung eventueller Optionen in sogenannte *Erweiterungs-Header* wird die Länge des Basis-Headers auf genau 320 Bit (40 Byte) festgelegt; einige Felder des IPv4-Headers wurden entfernt, weil sie keine Bedeutung mehr haben. Tabelle 4.15 zeigt den genauen Aufbau des IPv6-Headers.

Byte	0	1	2	3
0	Version	Klasse		
4	Payload Length		Next Header	Hop Limit
8 12 16 20	Quelladresse			
24 28 32 36	Zieladresse			

Tabelle 4.15 Aufbau des IPv6-Datagramm-Headers

Hier die Bedeutung der einzelnen Felder des Headers:

- Version (4 Bit): Die Versionsnummer des IP-Protokolls; hier natürlich 6.
- Klasse (8 Bit): Dieses Feld gibt die Priorität an, mit der das Datagramm übertragen werden soll. Es ist noch nicht abschließend geklärt, wie die entsprechenden Werte aussehen sollen.
- Flow Label (20 Bit): Ein Erweiterungsfeld, in das ein von 0 verschiedener Wert eingetragen wird, wenn IPv6-Router das Datagramm auf besondere Weise behandeln sollen. Es dient vor allem der Implementierung der Quality-of-Service-Funktionalität, mit deren Hilfe Paketsorten voneinander unterschieden werden, um beispielsweise Echtzeitanwendungen wie Streaming, Multimedia oder Videokonferenzen zu unterstützen.
- Payload Length (16 Bit): die Länge der Nutzdaten, die auf den Header folgen.
- Next Header (8 Bit): Der Wert in diesem Feld gibt den Typ des ersten Erweiterungs-Headers an, falls einer vorhanden ist. Es gibt bisher sechs Arten von Erweiterungs-Headern; eine Übersicht finden Sie in Tabelle 4.16.
- Hop Limit (8 Bit): Ein neuer Name für die Time-to-Live-Funktion: Jeder Router zieht von dem ursprünglichen Wert 1 ab; bei Erreichen des Wertes 0 wird das Paket verworfen.
- Quell- und Zieladresse (je 128 Bit): Die Adresse des Absenders und des Empfängers; genau wie bei IPv4, nur entsprechend der Protokollspezifikation 128 statt 32 Bit lang.

Header	Next-Header-Code	Beschreibung
Hop-by-Hop Options Header	0	Optionen, die bei jedem Routing-Schritt ausgeführt werden müssen
Routing Header	43	Festlegung der Router, über die das Paket geleitet werden soll
Fragment Header	44	Der Absender muss bei IPv6 die MTU herausfinden und Fragmente selbst bilden; die Fragmentinformationen befinden sich hier.
Authentication Header	51	Authentifizierung des Absenders gegenüber dem Empfänger
Encapsulating Security Payload Header	50	Dient der Verschlüsselung des Datagramms (IPv6).
Destination Options Header	60	Optionen, die nur für den Zielhost bestimmt sind
Upper-Layer Header	59	Header einer höheren Schicht; aus IPv6-Sicht also Nutzdaten

Tabelle 4.16 Die verschiedenen Typen von IPv6-Erweiterungs-Headern

Das größte Problem, das der sofortigen Einführung von IPv6 noch im Wege steht, ist ein organisatorisches: Zum einen kann man nicht einfach über Nacht flächendeckend umsteigen, da in diesem Fall die IP-Treiber aller Hosts und Router weltweit gewechselt werden müssten, was vollkommen illusorisch ist – zumal viele ältere Hardwarekomponenten, Betriebssysteme und Programme IPv6 gar nicht unterstützen und ihre Hersteller auch nicht vorhaben, diese Unterstützung nachträglich zu implementieren. Zum anderen ist es aber auch nicht möglich, gleichzeitig einen Teil des Internets mit IPv4 und einen anderen mit IPv6 zu betreiben und auf diese Weise allmählich auf die neue Version umzusteigen, da die beiden Adressierungsschemata miteinander inkompatibel sind.

Die Lösung, die letzten Endes gefunden wurde, besteht in der Tunnelung von IPv6-Paketen durch das klassische IPv4-Netzwerk. *Tunnelung* bedeutet nichts anderes, als dass jedes IPv6-Datagramm in ein IPv4-Datagramm verpackt wird. Das heißt, das IPv6-Paket bildet aus der Sicht des IPv4-Pakets die Nutzdaten, die mit einem v4-Header versehen werden. Am jeweiligen Zielpunkt, an dem wiederum IPv6 verfügbar ist, wird das Version-4-Datagramm »ausgepackt« und gemäß den Header-Daten weiterverarbeitet. IPv6-Tunnel-Dienste werden mittlerweile auch von mehreren kommerziellen und verschiedenen freien Anbietern, den Tunnel-Brokern, zur Verfügung gestellt.

IP-Routing

Komplizierter, aber auch interessanter wird es, wenn IP-Datagramme nicht für einen Host im lokalen Netz bestimmt sind, sondern für ein anderes Netzwerk. In diesem Fall muss das Paket an einen Router übergeben werden, der es weiterleitet. Die meisten Daten, die im Internet übertragen werden, passieren eine Vielzahl solcher Router, bis sie schließlich ihr Ziel erreichen. Um das Konzept des IP-Routings verstehen zu können, müssen Sie verschiedene Aspekte betrachten. Insbesondere ist die Frage von Bedeutung, auf welche Art und Weise überhaupt das korrekte Empfängernetzwerk gefunden wird.

Wichtig ist, dass man zwei Arten der Paketweiterleitung unterscheiden muss. Die reine Weiterleitung wird als *IP-Forwarding* bezeichnet; dabei sind nur zwei mögliche Netzwerkschnittstellen betroffen, sodass Quelle und Ziel jeweils feststehen. *Routing* im engeren Sinne beschreibt dagegen Verfahren, bei denen Entscheidungen zur Weiterleitung über verschiedene Wege an ein bestimmtes Ziel getroffen werden. Ein Router muss beide Verfahren beherrschen, sodass im Alltag oft nicht zwischen ihnen unterschieden wird. In LANs findet jedoch oft nur Forwarding, aber kein echtes Routing statt, da meist ohnehin nicht mehrere Router zur Auswahl stehen. Sowohl Forwarding als auch Routing lässt sich übrigens entweder statisch über festgelegte Tabellen oder dynamisch mithilfe von Protokollen erledigen.

Bei einem einzelnen Host können üblicherweise zwei Arten von Routern angegeben werden: zum einen die Router, die Daten in ein bestimmtes Fremdnetzwerk weiterleiten, und zum anderen das Default-Gateway (der Standard-Router), das alle Daten entgegennimmt, die weder für das lokale Netz noch für ein Netz mit einem speziellen Router bestimmt sind.

Beachten Sie übrigens, dass der Begriff *Gateway* zweideutig ist: Das Wort *Default-Gateway* beim IP-Forwarding oder Routing bezeichnet wie erwähnt den Standard-Router. Im Allgemeinen steht Gateway dagegen für einen Verbindungsrechner, der über sämtliche OSI-Schichten arbeitet und deshalb genauer als *Application Level Gateway* bezeichnet wird.

Bei einem privaten PC oder DSL-Router, der über eine Wählleitung mit dem Internet verbunden ist, besteht in der Regel nur eine Verbindung zu einem einzelnen Router des Providers. Welcher das ist, wird jedoch bei der Einwahl in das Netzwerk des Providers bestimmt, da auch die IP-Adresse bei jeder Einwahl dynamisch zugeteilt wird. Je nachdem, welche Adresse dem Host zugeteilt wird, ist möglicherweise ein anderer Router zuständig. Deshalb wird der Router bei der IP-Konfiguration des DFÜ-Netzwerkzugangs nicht fest angegeben, sondern durch das Einwahlprotokoll (üblicherweise PPP) mitgeteilt.

Anders sieht es dagegen oft bei Workstations in Unternehmen aus, die an ein lokales Netzwerk angeschlossen sind: Sämtliche Netzwerkkommunikation, sowohl mit dem lokalen Netz als auch mit dem Internet, findet über ein und dieselbe LAN-Schnittstelle statt, meistens über Ethernet. Innerhalb des LAN besitzt der Router für die Verknüpfung zum Internet eine bekannte IP-Adresse, die bei der IP-Konfiguration des Hosts angegeben wird. Mitunter besteht die Netzwerkinfrastruktur eines größeren Unternehmens auch aus mehreren Einzelnetzen, die über interne Router miteinander vernetzt werden. In einem solchen Fall wird häufig der Router, der zu dem anderen lokalen Netz führt, als Router für dieses konkrete Netz angegeben, während der Internetrouter (dessen Zielnetz »alle anderen Netze« sind) als Standard-Router eingerichtet wird. Für den letzteren – routingtechnisch relativ interessanten – Fall sehen Sie hier ein Beispiel:

In einem Unternehmen bestehen die beiden lokalen Netze 196.87.98.0/24 und 196.87.99.0/24. Das erste Netz wird von der Grafikabteilung verwendet, das zweite von den Softwareentwicklern. In Abbildung 4.4 wird der Aufbau dieses Netzes dargestellt.

Das Netzwerk der Grafikabteilung enthält die folgenden drei Rechner:

- `zeus` (196.87.98.3)
- `aphrodite` (196.87.98.4)
- `hermes` (196.87.98.5)

Zum Netzwerk der Entwicklungsabteilung gehören die drei folgenden Hosts:

- `newton` (196.87.99.7)
- `curie` (196.87.99.8)
- `einstein` (196.87.99.9)

Zwischen den beiden lokalen Netzen befindet sich ein Router, dessen Schnittstelle im Netz der Grafikabteilung die IP-Adresse 196.87.98.1 besitzt. Seiner anderen Schnittstelle für die Entwicklungsabteilung wurde die Adresse 196.87.99.2 zugewiesen. Ein zweiter Router verbin-

det die Entwicklungsabteilung mit dem Internet. Seine lokale Schnittstelle wurde mit der IP-Adresse 196.87.99.1 konfiguriert; die Adresse für die Internetschnittstelle wird vom Internetprovider dynamisch zugewiesen.

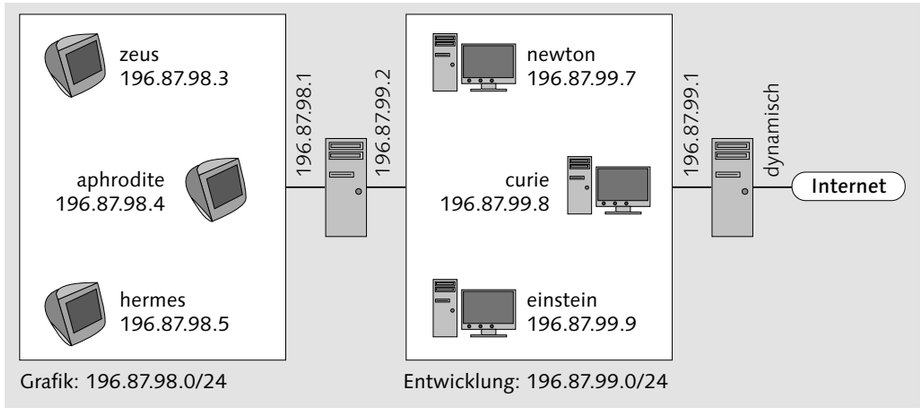


Abbildung 4.4 Verbindung zwischen zwei verschiedenen lokalen Netzen und dem Internet über zwei Router

Interessant ist nun die Routing-Konfiguration der einzelnen Hosts. Die drei Rechner im Entwicklernetzwerk kennen zwei verschiedene Router: Der Standard-Router ist 196.87.99.1, als spezieller Router für Datenpakete an das Netz 196.87.98.0 wird 196.87.99.2 angegeben. Dagegen kennen die drei Hosts im Grafiknetzwerk nur einen einzigen Router, nämlich 196.87.98.1, der als Standard-Router eingerichtet wird. Ob Datenpakete jenseits dieses Routers für das Netz 196.87.99.0 oder für das Internet bestimmt sind, muss der Router selbst entscheiden; die Rechner schicken ihm einfach alle Datagramme, die nicht für das lokale Netz verwendet werden sollen.

Angenommen, »aphrodite« möchte auf Daten zugreifen, die »newton« bereitstellt. Die Daten sind offensichtlich nicht für das Netz 196.87.98.0 bestimmt, deshalb werden sie dem Router übergeben. Dieser erkennt, dass sie für das Netz 196.87.99.0 bestimmt sind, an das er unmittelbar angeschlossen ist. Er kann die Daten direkt an den Zielhost ausliefern.

Will dagegen »zeus« auf Daten aus dem Internet zugreifen, muss der Standard-Router des Grafiknetzes erkennen, dass die Daten nicht für das andere Netz bestimmt sind, an das er selbst angeschlossen ist, und sie an den nächsten Router weiterreichen.

Ein wenig anders verhält es sich, wenn ein Rechner aus dem Entwicklernetz wie »curie« auf »zeus« zugreifen möchte. Es ist bereits in der Routing-Konfiguration von »curie« bekannt, dass ein bestimmter Router, nämlich 196.87.99.2, verwendet werden soll. Ebenso weiß beispielsweise »einstein«, dass Zugriffe auf das Internet über den Router 196.87.99.1 erfolgen müssen.

Damit ein Host weiß, wohin er Datenpakete eigentlich schicken muss, um ein bestimmtes Netz zu erreichen, müssen die einzelnen Router in seiner Netzwerkkonfiguration angegeben werden – dies funktioniert je nach Betriebssystem unterschiedlich. Das Ergebnis dieser Konfiguration ist eine Routing-Tabelle, die ebenfalls je nach System unterschiedlich aussieht. Angenommen, alle Rechner im zuvor gezeigten Beispielnetzwerk liefen unter Unix-Varianten (die Grafikrechner unter OS X, die Entwicklercomputer unter Linux). Dann sähe die Routing-Tabelle von »curie«, die durch den Unix-Befehl `netstat rn` angezeigt werden kann[8], so aus:

```
$ netstat -rn
Routing Tables
Destination Gateway FlagsRefcntUseInterface
127.0.0.1 127.0.0.1 UH 1 132lo0
196.87.99.0 196.87.99.8 U26490411e0
196.87.98.0 196.87.99.2 UG 0 0le0
default 196.87.99.1 UG 0 0le0
```

Die erste Zeile (Zieladresse 127.0.0.1) beschreibt das Erreichen der Loopback-Adresse: Das Interface (Netzwerkschnittstelle) ist `lo0` (»local loopback«). Das Flag `H` zeigt an, dass es sich um eine Route zum Erreichen eines einzelnen Hosts handelt. Das Flag `U` dagegen steht für »up« und bedeutet, dass die Route zurzeit intakt ist.

In der nächsten Zeile wird das lokale Netzwerk angegeben, in dem sich »curie« selbst befindet. Deshalb wird als Gateway einfach die IP-Adresse von »curie« ausgegeben. Das Interface `le0` ist die erste (und in diesem Fall einzige) Ethernet-Schnittstelle des Rechners.

Die dritte Zeile beschreibt die Route in das Grafiknetzwerk über den Router, dessen Adresse im Entwicklernetz 196.87.99.2 lautet. Das Flag `G` steht für »Gateway«, also für die Tatsache, dass für diese Route die Dienste eines Routers in Anspruch genommen werden.

In der letzten Zeile wird schließlich 196.87.99.1 als Default-Gateway angegeben, also als Router für alle Ziele, die nicht explizit in der Routing-Tabelle auftauchen.

Die Routing-Tabelle von »hermes« sieht einfacher aus:

```
Routing Tables
Destination GatewayFlagsRefcnt UseInterface
127.0.0.1 127.0.0.1UH 1 132lo0
196.87.98.0 196.87.98.5U26490411e0
default 196.87.98.1UG 0 0le0
```

Da das Grafiknetz nur einen Router kennt, gibt es nur den Loopback-Eintrag, die Information für das lokale Netz und schließlich den Default-Eintrag für alle anderen Netze.

8 Näheres über die einfachen TCP/IP-Dienstprogramme erfahren Sie für die jeweilige Systemplattform in Kapitel 6, »Windows«, beziehungsweise Kapitel 7, »Linux«.

Auf diese Weise werden Daten durch das gesamte Internet geroutet. Jedes Mal, wenn ein Router passiert wird, erfolgt ein sogenannter *Hop* der Daten. Wegen des TTL-Feldes von 8 Bit Größe, das im IP-Header enthalten ist und bereits beschrieben wurde, erreicht ein Datagramm sein Ziel stets mit höchstens 255 Hops – oder eben gar nicht.

Damit IP-Datenpakete ihr Ziel überhaupt erreichen können, muss im Prinzip jeder einzelne Router im gesamten Internet darüber Bescheid wissen, wie er jedes beliebige Netz erreichen kann. Zu diesem Zweck unterhält auch jeder Router Routing-Tabellen, die den bereits für die einzelnen Hosts gezeigten ähnlich sehen. Da das Internet ein Zusammenschluss aus vielen einzelnen Netzwerken ist, müssen diese Tabellen jedoch ständig aktualisiert werden, denn es ergeben sich häufig Konfigurationsänderungen, weil neue Netze hinzukommen oder vorhandene geändert oder aufgegeben werden. Es wäre absolut unzumutbar, diese Konfigurationsänderungen ständig manuell auf dem aktuellen Stand zu halten, was deshalb auch seit vielen Jahren nicht mehr üblich ist (außer innerhalb sehr kleiner Netze wie in dem Beispiel zuvor, in denen sich die Routing-Einstellungen selten ändern müssen).

Die Router im Internet müssen deshalb ständig Informationen darüber austauschen, an welche anderen Netzwerke sie jeweils Daten vermitteln. Sie müssen komplexe Routing-Entscheidungen treffen, indem sie den Aufwand und die Kosten verschiedener Routen vergleichen und die Pakete eben nicht direkt ans Ziel, sondern auf dem derzeit günstigsten Weg weiterleiten, damit diese nicht nur sicher, sondern auch möglichst schnell ihr Ziel erreichen. Dieses eigentliche Routing ist erheblich dynamischer als das einfache Forwarding, sodass die Routing-Informationen ständig aktualisiert werden müssen. Auf diese Weise kann ein Paket bei Ausfall oder auch nur starker Belastung einer bestimmten Route über eine andere Route umgeleitet werden. Zu diesem Zweck wurde eine Reihe verschiedener Routing-Protokolle entwickelt, mit deren Hilfe dies möglich wird. Jedes dieser Routing-Protokolle besitzt andere Eigenschaften, außerdem wird nicht jedes dieser Protokolle von jedem Hersteller unterstützt.

Zunächst muss zwischen zwei Arten von Routing unterschieden werden: dem Routing innerhalb zusammenhängender Netze eines einzelnen Betreibers (Interior Routing), der innerhalb dieses Bereichs frei über die Konfiguration entscheiden kann, und dem Routing zwischen voneinander unabhängigen derartigen Bereichen (Exterior Routing). Alle zusammenhängenden Netze eines Betreibers werden als *autonome Systeme* (Autonomous Systems, abgekürzt AS) bezeichnet. Einige Routing-Protokolle, etwa das veraltete RIP oder das aktuellere OSPF, dienen dem Routing innerhalb von autonomen Systemen, während andere, vor allem BGP, für das Routing zwischen den Grenzen autonomer Systeme zuständig sind. Diese drei genannten Routing-Protokolle werden im weiteren Verlauf des Kapitels kurz vorgestellt.

Wenn ein Router ein Routing-Protokoll ausführt, dann teilt er den benachbarten Routern mit, an welche Netze er Daten weiterleitet. Die meisten Routing-Protokolle machen außerdem Angaben über die »Kosten«, die für das Erreichen eines bestimmten Netzes kalkuliert

werden müssen. Der Begriff *Kosten* hat nichts mit dem Preis zu tun, sondern bestimmt vor allem, über wie viele Hops ein bestimmtes Netzwerk durch den jeweiligen Router erreicht werden kann. Allerdings gibt es auch die Möglichkeit, die Kostenangaben willkürlich zu manipulieren – je nachdem, wie »gern« ein Router Daten an ein bestimmtes Netzwerk übermitteln soll. Wenn ein Router bestimmen muss, an welchen benachbarten Router er die Daten für ein bestimmtes Netz übergeben soll, sucht er sich denjenigen aus, der für dieses Netz geringere Kosten angibt. Diese Kostendaten werden auch als die *Metrik* des Routings bezeichnet.

Auf diese Weise wird versucht, die Datenströme zwischen den verschiedenen Backbone-Netzwerken möglichst gleichmäßig zu verteilen, außerdem bestehen verschiedene Arten von Verträgen oder Vereinbarungen zwischen den Netzbetreibern, was die Weiterleitung von Daten bestimmter anderer Netzwerke betrifft. Beispielsweise gab es in Deutschland in den 90er-Jahren einen mehrjährigen Streit zwischen dem Deutschen Forschungsnetz (DFN), dem Betreiber der deutschen Universitätsnetze, und den kommerziellen Internetprovidern. Es ging um die Frage, wer wem mehr Datenverkehr aus dem jeweils anderen Netz zumutete. Erst durch die Einführung neuer zentraler Datenaustauschpunkte wie dem DE-CIX konnte der Konflikt beigelegt werden.

Hier einige wichtige Routing-Protokolle im Überblick:

- **Routing Information Protocol (RIP)**
  Das Routing Information Protocol (RIP) wird auf Unix-Routern durch den Routing Daemon (`routed`) ausgeführt. Beim Start von `routed` wird eine Anfrage ausgesendet. Alle anderen Router, die innerhalb desselben autonomen Systems ebenfalls `routed` ausführen, beantworten diese Anfrage durch Update-Pakete. Darin sind die Zieladressen aus den Routing-Tabellen der anderen Router und deren jeweilige Metrik enthalten.

  Enthält ein Update-Paket die Routen zu Netzen, die noch gar nicht bekannt sind, fügt der Router sie zu seiner Routing-Tabelle hinzu. Außerdem werden Routen ersetzt, falls ein Update-Paket die Information enthält, dass ein bestimmtes Netzwerk über einen anderen Router mit geringeren Kosten zu erreichen ist.

  Ein Router, auf dem `routed` läuft, sendet ebenfalls Update-Pakete, und zwar in der Regel alle 30 Sekunden. Erhält ein Router von einem anderen Router mehrere Male keine Update-Pakete mehr (häufig beträgt die Wartezeit 180 Sekunden), löscht er alle Einträge aus seiner Routing-Tabelle, die diesen Router verwenden. Außerdem werden diejenigen Einträge gelöscht, deren Kosten mehr als 15 Hops betragen. Letzteres beschränkt RIP auf kleinere autonome Systeme.

  RIP interpretiert IP-Adressen streng nach der alten Klassenlogik und beherrscht weder CIDR noch VLSM. Dies ist der Hauptgrund, warum es immer seltener verwendet wird.

  Außerdem besteht das Problem, dass durch den plötzlichen Ausfall von Routern Konfigurationsfehler entstehen können: Alle Netze, die ursprünglich nur durch den ausgefallenen Router erreicht werden konnten, sind nun gar nicht mehr erreichbar. Dies spricht

sich jedoch nur allmählich herum, da ein Router zwar zunächst alle Routen entfernt, die durch den ausgefallenen Router führten, von den anderen jedoch wieder die Route zu dem Netz lernt, das nun nicht mehr erreichbar ist. Bei einem Update-Intervall von 30 Sekunden kann es recht lange dauern, bis die Router die Entfernung zu dem nicht mehr verfügbaren Netz auf die nicht mehr relevanten 16 Hops »hochgeschaukelt« haben.

Um dieses Szenario zu verhindern, wird eine Technik namens *Split Horizon* verwendet: Ein Router bietet Routing-Informationen nicht über die Verbindung an, über die er sie gelernt hat. Eine Erweiterung dieses Verfahrens ist *Poison Reverse*; hier wird den Routern, von denen eine bestimmte Verbindung gelernt wurde, aktiv die »Unendlich-Metrik« 16 angegeben.

Einige Probleme von RIP werden in der neueren Version RIP-2, die in RFC 1723 beschrieben wird, beseitigt; vor allem arbeitet diese Version mit CIDR-Adressierung.

- **Open Shortest Path First (OSPF)**
  Das in RFC 2178 beschriebene Open-Shortest-Path-First-Protokoll (OSPF) ist ein sogenanntes *Link-State-Protokoll*: Der einzelne Router speichert einen gerichteten Graphen des Netzwerks aus seiner jeweiligen Sicht. Ein gerichteter Graph ist eine Art Baumdiagramm mit dem lokalen Router als Wurzel; sein Aufbau erfolgt nach dem Shortest-Path-First-Algorithmus von Dijkstra: Die Kosten des lokalen Routers selbst werden mit 0 angegeben; von diesem zweigen die Routen zu den Nachbarn baumförmig ab, dann wiederum zu deren Nachbarn etc. In einem zweiten Schritt wird der Link-State-Graph optimiert. Falls mehrere Routen zu einem Ziel vorhanden sind, beispielsweise eine direkte und eine indirekte, wird jeweils die weniger kostengünstige Route entfernt.

  Um die Link-State-Datenbank klein zu halten, werden größere autonome Systeme in kleinere Einheiten unterteilt, die *Areas*. Nur vereinzelte Router, die sogenannten *Bereichsgrenzrouter*, werden von den Routern innerhalb einer Area als Verbindung in andere Areas betrachtet.

  Ein OSPF-Router gewinnt seine Erkenntnisse über die benachbarten Router, indem er sogenannte *Hello-Pakete* aussendet. Diese enthalten seine eigene Adresse und die Information, von welchen benachbarten Routern er bereits Routing-Daten erhalten hat. Ein Router, der ein Hello-Paket erhält, trägt den Absender dieses Pakets als Nachbarn in seinen eigenen Link-State-Graphen ein. Die Hello-Pakete werden in regelmäßigen Abständen ausgesandt, um den Nachbarn mitzuteilen, dass der Router noch bereit ist. Erhält ein Router keine weiteren Pakete von einem bestimmten Nachbarn, geht er davon aus, dass dieser nicht mehr zur Verfügung steht, entfernt ihn aus seiner Link-State-Datenbank und informiert das Netzwerk darüber.

  OSPF-Router geben Daten über ihre Nachbarn an das gesamte Netzwerk weiter, indem sie Link State Advertisements (LSA) über alle ihre Netzwerkschnittstellen versenden. Der Empfänger eines LSA-Pakets leitet es weiter, indem er es ebenfalls über alle seine Schnittstellen versendet – mit Ausnahme derjenigen, über die er es empfangen hat. Dieses

Verfahren der schnellen Verbreitung von Informationen über ein Netzwerk wird als *Flooding* bezeichnet.

- **Border Gateway Protocol (BGP)**
  Anders als bei den beiden zuvor behandelten Routing-Protokollen handelt es sich beim Border Gateway Protocol (BGP) um ein externes Routing-Protokoll, das Verbindungen zwischen verschiedenen autonomen Systemen regelt. Vom Standpunkt des externen Routings aus erscheinen die autonomen Systeme selbst als in sich geschlossene Gebilde, die nicht näher differenziert werden. BGP wird nur von den Bereichsgrenzroutern der autonomen Systeme ausgeführt, also in der Regel lediglich bei Internetprovidern oder großen Backbone-Netzbetreibern. Die meisten Firmennetze sind dagegen Teil eines autonomen Systems, das von einem Provider betrieben wird, führen also lediglich interne Routing-Protokolle wie OSPF aus.

  Die benachbarten BGP-Router, *Peers* genannt, kommunizieren über eine zuverlässige TCP-Verbindung, die über den dafür vorgesehenen TCP-Port 179 abgewickelt wird. Es wird stets eine vollständige Route mit allen ihren Knotenpunkten angegeben. Dies unterscheidet BGP von den meisten internen Routing-Protokollen, die nur die Verbindungen zu ihren unmittelbaren Nachbarn angeben. Aus diesem Grund wird BGP als *Pfadvektor-Protokoll* bezeichnet.

  Wird das erste Mal eine Verbindung zu einem Peer hergestellt, dann werden über sogenannte *Update-Pakete* die vollständigen Routing-Tabellen ausgetauscht, danach werden nur noch Änderungen mitgeteilt. Außerdem werden in regelmäßigen Abständen KEEPALIVE-Pakete versandt, falls keine Änderungen vorliegen, um den Peers mitzuteilen, dass der Router noch einsatzbereit ist.

Weitere IP-Dienste

In fast allen modernen TCP/IP-Netzwerken – insbesondere in lokalen Firmennetzen, die mit dem Internet verbunden sind – spielen zwei weitere Protokolle eine wichtige Rolle: DHCP dient dazu, den Rechnern im Netzwerk automatisch IP-Adressen zuzuweisen, während das NAT-Protokoll meist vom Standard-Router ausgeführt wird und die im Internet unbrauchbaren privaten IP-Adressen mit öffentlichen überschreibt und umgekehrt. Diese beiden Protokolle sollen hier näher vorgestellt werden.

Das in RFC 2131 und 2132 definierte Dynamic Host Configuration Protocol (DHCP) dient dazu, einem Host automatisch TCP/IP-Konfigurationsdaten zuzuweisen. Es ist eine Erweiterung des älteren Bootstrap Protocols (BOOTP). Ein Host, der seine Netzwerkparameter über DHCP beziehen möchte, sendet bei Inbetriebnahme eine Broadcast-Anfrage namens *BOOTREQUEST* an die allgemeine Broadcast-Adresse 255.255.255.255. Der Rechner muss also noch nicht einmal wissen, in welchem Netzwerk er sich befindet – das ist beispielsweise ideal für ein Notebook, das manchmal an ein Heim- und manchmal an ein Büro-Netzwerk angeschlossen wird. Läuft in dem Netz ein DHCP-Server, antwortet er mit einem Satz von Konfigurationsparametern, mit denen der Host seine TCP/IP-Konfiguration vornimmt.

Das wichtigste Merkmal von DHCP besteht in der dynamischen Vergabe von IP-Adressen, die Netzwerkadministratoren das Leben erheblich erleichtert – insbesondere in solchen Netzwerken, in denen häufig Änderungen auftreten. Diese automatische Vergabe erfolgt in Form einer »Lease« (Pacht) mit beschränkter Gültigkeit. Ein Host, der ordnungsgemäß vom Netz abgemeldet wird (ein normaler Vorgang beim Herunterfahren moderner Betriebssysteme), gibt seine IP-Adresse selbst an den DHCP-Server zurück. Das Lease-Verfahren sorgt dagegen dafür, dass IP-Adressen auch dann wieder für den Server verfügbar werden, wenn ein Host unerwartet vom Netz getrennt oder unsachgemäß abgeschaltet wird. Bleibt ein Rechner über den Lease-Zeitraum hinaus im Netz aktiv, erfolgt in der Regel eine Verlängerung der Lease.

Auf dem DHCP-Server muss ein Teil der Adressen des Netzwerks, in dem er sich befindet, als DHCP-Pool konfiguriert werden, aus dem die Adressen automatisch an die anfragenden DHCP-Clients vergeben werden. Es muss darauf geachtet werden, genügend Adressen aus diesem Pool auszuschließen, weil eine Reihe von Internetdiensten eine feste IP-Adresse benötigt oder zumindest besser damit funktioniert.

Network Address Translation (NAT) ist eine relativ neue Entwicklung und löst dementsprechend ein modernes Problem: Immer mehr Netzwerke benötigen permanenten oder auch nur temporären Zugang zum Internet, obwohl sie mit den zuvor vorgestellten privaten IP-Adressen konfiguriert wurden. Es wäre bei der heutigen Anzahl von Internethosts und angeschlossenen Netzen auch gar nicht mehr möglich, allen angeschlossenen Netzwerken öffentliche IP-Adressen zuzuweisen. Da die privaten IP-Adressen jedoch nicht eindeutig sind, müssen sie beim Übergang ins Internet mit einer öffentlichen Adresse überschrieben werden und umgekehrt.

Eine aktuelle Form von NAT, die im Kernel moderner Unix-Systeme konfiguriert werden kann, wird auch als *IP-Masquerading* bezeichnet und geht noch einen Schritt weiter als NAT: Es ist nur eine externe IP-Adresse erforderlich; alle lokalen Adressen werden auf diese eine Adresse abgebildet. Unterschieden werden die Rechner in diesem Fall anhand der Client-Portnummer der Datenpakete, die zur Transportebene gehört und im nächsten Abschnitt näher beschrieben wird. Aus diesem Grund wird das echte Masquerading manchmal auch als *PAT* (Port Address Translation) bezeichnet. Diese spezielle Form von NAT verbirgt die Details des internen Netzwerks vor dem Internet, die einzelnen Rechner sind von außen nicht erreichbar. Dies ist ein angenehmer Nebeneffekt dieses Verfahrens, der zusätzlich der Sicherheit im Netzwerk dient.

Tabelle 4.17 zeigt ein Beispiel für klassisches NAT in einem privaten Netzwerk mit der Adresse 192.168.1.0/24. Jede interne IP-Adresse wird auf eine individuelle externe Adresse abgebildet.

Hostname	interne IP-Adresse	externe IP-Adresse
gandalf	192.168.1.4	204.81.92.6
Frodo	192.168.1.5	204.81.92.3
Bilbo	192.168.1.6	204.81.92.5

Tabelle 4.17 Beispiel für klassisches NAT

In Tabelle 4.18 wird dagegen für dasselbe Netzwerk ein Beispiel für IP-Masquerading (PAT) gezeigt. Das Konzept der Portnummern wird im weiteren Verlauf des Kapitels noch genauer beschrieben.

Hostname	interne IP-Adresse	externe IP-Adresse	externe Portnummer
Gandalf	192.168.1.4	204.81.92.4	22.191
Frodo	192.168.1.5		22.192
Bilbo	192.168.1.6		22.193

Tabelle 4.18 Beispiel für IP-Masquerading

Der Rechner, der NAT ausführt, ist üblicherweise derjenige Router, der das lokale Netz mit dem Netzwerk eines Providers und demzufolge mit dem Internet verbindet. NAT wird von allen gängigen Unix-Versionen sowie von Windows NT und seinen Nachfolgern unterstützt. Außerdem können die meisten ISDN- oder DSL-Kompakt-Router NAT ausführen. Eine nähere Beschreibung vieler Aspekte von NAT findet sich in RFC 3022.

Auf einem Linux-System kann NAT beispielsweise durch die Kernel-Firewall Netfilter/iptables bereitgestellt werden. Dieser Aspekt wird auch in Kapitel 21, »Computer- und Netzwerksicherheit«, erwähnt.

Die Windows-Desktop-Systeme enthalten einen eigenen NAT-Dienst namens *Internet Connection Sharing* (ICS). Dadurch fungiert eine Workstation als NAT-Router und ermöglich so die Nutzung ihres Internetzugangs durch andere Maschinen. Diese NAT-Variante gilt allerdings als unsicher und ist zudem erheblich unflexibler als IP-Masquerading, da sie automatisch erfolgt und keinerlei Einstellungen ermöglicht.

In manchen Netzwerken erfolgt der Zugriff auf Internetdienste nicht über Router, sondern über Gateways auf Anwendungsebene, die in der Öffentlichkeit als Stellvertreter (Proxys) des eigenen Rechners arbeiten. Solche Proxyserver gibt es für fast alle Dienste. Am bekanntesten sind Webproxys, die oft auch als Cache (Zwischenspeicher) für häufig aufgerufene Websites dienen. Die bekannteste Proxy- und Webcache-Software ist der Open-Source-Proxy *squid*; Microsoft bietet ebenfalls ein entsprechendes Produkt namens *ISA Server* an.

Auch der Webserver Apache (siehe Kapitel 14, »Server für Webanwendungen«) kann optional als Caching-Proxy für verschiedene Protokolle konfiguriert werden. Eine Anleitung dazu finden Sie auf den Webseiten zu meinem Buch »Apache 2.4« (4. Auflage: Bonn: Galileo Press 2012); konkret unter *http://buecher.lingoworld.de/apache2/mod_proxy.html*.

4.6.3 Transportprotokolle

Eine Anwendung, die Daten über ein TCP/IP-Netzwerk wie das Internet übertragen möchte, beauftragt zu diesem Zweck ein Transportprotokoll, also ein Protokoll der Host-zu-Host-Transportschicht des Internetschichtenmodells. Nachdem im letzten Abschnitt das IP-Routing erläutert wurde, sollten Sie auch verstehen, warum der Vorgang als *Host-zu-Host-Transport* bezeichnet wird: Router betrachten von den Datenpaketen, die sie weiterleiten sollen, immer nur den IP-Header und werten dessen Informationen aus. Aus der Sicht des IP-Protokolls existieren die Daten der Transportschicht nicht. Umgekehrt ist also das Routing ein Implementierungsdetail, das für die Protokolle der Transportschicht nicht sichtbar ist. Aus ihrer Sicht kann der Zielhost immer unmittelbar erreicht werden.

Um den Bedürfnissen verschiedener Anwendungen gerecht zu werden, wurden zwei verschiedene wichtige Transportprotokolle definiert. Das häufiger verwendete *TCP-Protokoll* (definiert in RFC 793), das einen Teil des Namens der Protokollfamilie ausmacht, stellt den zuverlässigen Transport von Datenpaketen in einer definierten Reihenfolge zur Verfügung. Dagegen bietet das *UDP-Protokoll* (RFC 768) die Möglichkeit, Daten auf Kosten der Zuverlässigkeit möglichst schnell zu transportieren.

Ein wenig zwischen Vermittlungs- und Transportschicht liegt das ICMP-Protokoll (Internet Control Message Protocol), das für den Versand spezieller Datagramme verwendet wird, mit deren Hilfe überprüft werden kann, ob ein entfernter Host im Netzwerk aktiv ist. Das entsprechende Dienstprogramm heißt *ping* und wird in Kapitel 6, »Windows«, und in Kapitel 7, »Linux«, für die jeweilige Systemplattform vorgestellt.

Die beiden Protokolle werden in den folgenden Abschnitten genauer beschrieben.

Das Transmission Control Protocol (TCP)

Wie Sie im letzten Abschnitt erfahren haben, werden IP-Datagramme jeweils individuell durch das Netzwerk geleitet. Deshalb kann auf der Basis von Datagrammen kein zuverlässiger Transport kontinuierlicher Datenströme erfolgen, weil es vollkommen normal ist, dass Datagramme nicht in der Reihenfolge ankommen, in der sie abgeschickt wurden. Außerdem ist es auch möglich, dass sie gar nicht ankommen, weil auf der Ebene des IP-Protokolls keine entsprechende Kontrolle durchgeführt wird.

Um nun über den potenziell unsicheren Weg der IP-Datagramme Daten zuverlässig durch das Netzwerk zu transportieren, wird auf dieser höher gelegenen Ebene eine Flusskontrolle implementiert: Im Wesentlichen werden die Datenpakete durch das TCP-Protokoll durch-

nummeriert, um die korrekte Reihenfolge aufrechtzuerhalten. Im Übrigen erwartet der ursprüngliche Absender für jedes einzelne Datenpaket eine Bestätigung; bleibt sie zu lange aus, versendet der Absender das entsprechende Paket einfach erneut.

Als Erstes sollten Sie sich den TCP-Paket-Header ansehen, der in Tabelle 4.19 gezeigt wird.

	0	1	2	3
0	Quellport		Zielport	
4	Sequenznummer			
8	Bestätigungsnummer			
12	Offset	reserviert	Flags	Fenster
16	Prüfsumme		Urgent-Zeiger	
20	Optionen			Padding

Tabelle 4.19 Aufbau des TCP-Datenpaket-Headers

Ein TCP-Datenpaket-Header besteht aus den folgenden Bestandteilen:

- Quellport (16 Bit): Ports stellen eine Methode zur Identifikation der konkreten Anwendungen zur Verfügung, die auf den beteiligten Hosts miteinander kommunizieren. Der Quellport ist die Portnummer des Absenders.
- Zielport (16 Bit): Dies ist entsprechend die Portnummer des Empfängers.
- Sequenznummer (32 Bit): Normalerweise gibt diese Nummer an, dem wievielten Byte der zu übertragenden Sequenz das erste Nutzdaten-Byte des Pakets entspricht. Ausnahme: Ist das SYN-Flag gesetzt, wird die Anfangssequenznummer (Initial Sequence Number, ISN) angegeben.
- Bestätigungsnummer (32 Bit): Das Start-Byte der Sequenz, deren Übertragung als Nächstes erwartet wird; ist nur bei gesetztem ACK-Bit von Bedeutung.
- Offset (4 Bit): Anzahl der 32-Bit-Wörter, aus denen der Header besteht; gibt entsprechend den Beginn der Nutzdaten im Paket an.
- Reserviert (6 Bit): Reserviert für zukünftige Anwendungen; muss 0 sein.
- Flags (6 Bit): verschiedene Statusbits; im Einzelnen:
  - URG: Urgent Data wird versandt; der Inhalt des Urgent-Zeigers muss beachtet werden.
  - ACK: Acknowledgement – das Bestätigungsfeld muss berücksichtigt werden.
  - PSH: Push-Funktion – ist dieses Bit gesetzt, wird die Pufferung des Pakets verhindert; es wird unmittelbar gesendet.
  - RST: Reset – Verbindung zurücksetzen

  - SYN: Sequenznummern synchronisieren
  - FIN: Ende der Sequenz; keine weiteren Daten vom Absender
- Fenster (16 Bit): Die Anzahl von Daten-Bytes, die der Absender des Pakets zu empfangen bereit ist; basiert unter anderem auf der IP-MTU der verwendeten Schnittstelle.
- Prüfsumme (16 Bit): Anhand dieser einfacheren Plausibilitätskontrolle kann die Korrektheit der übertragenen Daten überprüft werden.
- Urgent-Zeiger (16 Bit): Ein Zeiger auf das Byte der aktuellen Sequenz, das Urgent Data enthält. Wird nur ausgewertet, wenn das URG-Flag gesetzt ist.
- Optionen (variable Länge): Enthält verschiedene hersteller- und implementierungsabhängige Zusatzinformationen; stets ein Vielfaches von 8 Bit lang.

Zwischen den beiden Hosts, die über TCP kommunizieren, wird eine virtuelle Punkt-zu-Punkt-Verbindung hergestellt; aus diesem Grund wird TCP auch als *verbindungsorientiertes Protokoll* bezeichnet. Dies ermöglicht den Transport eines kontinuierlichen Datenstroms über die potenziell unzuverlässigen IP-Datagramme, in die die TCP-Pakete verpackt werden. Um die Datenübertragung einzuleiten, findet zunächst ein sogenannter *Drei-Wege-Handshake* statt: Drei spezielle Datenpakete ohne Nutzdateninhalt werden ausgetauscht. Der Host, der die Verbindung initiiert, sendet ein Paket mit gesetztem SYN-Bit an den Empfänger. Dieser schickt ein Paket zurück, bei dem SYN und ACK gesetzt sind, und erwartet wiederum eine Antwort, bei der nur das ACK-Flag gesetzt ist. Erst nachdem dies geschehen ist, beginnt die eigentliche Übertragung von Nutzdaten. Dieses Vorgehen garantiert, dass beide Hosts bereit sind, miteinander zu kommunizieren.

Anschließend sendet der Absender ein Paket nach dem anderen an den Empfängerhost, wobei die Sequenznummer stets um die im vorangegangenen Paket versandte Nutzdatenmenge erhöht wird. Der Empfänger beantwortet jedes empfangene Paket, dessen Prüfsumme mit dem Inhalt übereinstimmt, mit einem Bestätigungspaket, dessen ACK-Flag also gesetzt ist. Der Wert des Bestätigungsfeldes ist der Byte-Offset der nächsten Datensequenz, die der Empfänger erwartet, ist also die Summe aus Sequenznummer und Nutzdatenlänge des soeben empfangenen Pakets.

Erhält der Absender die Bestätigung nicht innerhalb einer definierten Zeit (Timeout), sendet er das entsprechende Paket unaufgefordert erneut. Dieses Verfahren wird *positive Bestätigung* (Positive Acknowledgement) genannt, da lediglich der Erfolg gemeldet wird; von einem Misserfolg wird automatisch ausgegangen, wenn keine Meldung erfolgt. Dieses Verfahren ist zuverlässiger als das Arbeiten mit Misserfolgsmeldungen: Kommt die Erfolgsmeldung nicht an, wird das Paket einfach erneut versandt, ansonsten gibt es aber keine schädlichen Folgen (außer dem geringen Mehraufwand für ein überflüssig versandtes Paket, falls einmal lediglich die Bestätigung verloren gegangen ist). Käme dagegen eine Misserfolgsmeldung nicht an, würde das betreffende Paket nicht erneut versandt und würde den Empfänger niemals erreichen.

Ein weiterer wichtiger Bestandteil von TCP-Paketen sind die beiden 16 Bit langen Portnummern. Jedes Paket kann anhand des Portnummern-Paares als zu einer bestimmten Sequenz und Anwendung gehörig identifiziert werden. Das ist auch absolut notwendig: Stellen Sie sich vor, Sie haben zwei Browserfenster geöffnet; in beiden werden gleichzeitig verschiedene Seiten von *www.rheinwerk-verlag.de* geladen. Anhand der IP-Adressen können die beiden Datenübertragungen nicht voneinander unterschieden werden, da die beiden Hosts, die hier miteinander kommunizieren, identisch sind. Es könnte also sehr leicht passieren, dass die Daten fehlerhaft zugeordnet werden und Sie zwei seltsame Mischungen der beiden Dokumente erhalten – ein Effekt wie in dem Horror-Klassiker »Die Fliege«!

Dieses Szenario kann deshalb nicht eintreten, weil die beiden Datenübertragungen nicht über dasselbe Paar von Portnummern erfolgen. In der Regel ist die Portnummer des Servers festgelegt, während der Client irgendeinen Port wählt, der gerade frei ist. Die untersten 1.024 Portnummern sind als sogenannte *Well-known Ports* für Standard-Serverdienste fest vergeben; für Clients wird eine zufällige Nummer (ein sogenannter *Ephemeral Port*) zwischen 1.024 und 65.535 verwendet. Beispielsweise benutzen Webserver, also HTTP-Server, üblicherweise den TCP-Port 80, FTP-Server den Port 21 und Telnet-Server den Port 23. Eine kleine Liste, die auch UDP betrifft, finden Sie in Tabelle 4.21.

In dem Beispiel mit den beiden Browserfenstern ist der Server-Port jeweils 80; die Client-Ports sind dagegen unterschiedlich, beispielsweise 16832 und 16723. Dies ist eine Verdeutlichung der Formulierung, dass nur die Portpaare und nicht die beiden einzelnen Ports unterschiedlich sein müssen, um Sequenzen voneinander abzugrenzen.

Gewöhnlich »lauscht« ein TCP-Serverdienst an seinem speziellen Port auf ankommende Verbindungsversuche. Unternimmt ein Client den Versuch, eine TCP-Verbindung mit dieser speziellen Portnummer als Empfängerport und einer zufälligen Nummer als Absender herzustellen, akzeptiert der Server dies nach den Regeln des Drei-Wege-Handshakes; eine Verbindung für den gegenseitigen Datenaustausch ist hergestellt.

Interessant ist schließlich das Thema Urgent Data: Manchmal muss ein Host einen anderen über einen besonderen Zustand informieren, beispielsweise eine Konfigurationsänderung oder einen vom Benutzer initiierten Abbruch mitteilen. Zu diesem Zweck wird das URG-Flag gesetzt; der Empfänger ermittelt daraufhin aus dem Urgent-Zeiger-Feld des Paket-Headers die Byte-Nummer innerhalb der Sequenz, in der sich diese dringlichen Daten befinden. Es handelt sich stets nur um ein einziges Byte, das auch als *Out-of-Bound-Byte* bezeichnet wird, weil es nicht zum gewöhnlichen Datenstrom gehört. Es ist also unmöglich, auf diesem Weg eine längere dringende Mitteilung zu versenden, aber immerhin besteht die Möglichkeit, bestimmte zwischen den Anwendungen vereinbarte Signale auszutauschen.

Das User Datagram Protocol (UDP)

Manche Anwendungen möchten auf den Komfort und die Sicherheit von TCP getrost verzichten, wenn sie die Daten dafür schneller ans Ziel befördern können. Die Möglichkeit eines

solchen möglichst schnellen Versands bietet das UDP-Protokoll. Ob eine Anwendung für ihre Datenübertragung nun TCP, UDP oder beide verwenden möchte, entscheidet sie selbst.

Stellen Sie sich als Beispiel ein Netzwerkspiel vor, eine virtuelle 3-D-Umgebung, in der Sie gegen Ihre Mitspieler »kämpfen« können. Ein solches Spiel ist ideal für die Erklärung des Nutzens beider Übertragungsarten geeignet: Bestimmte grundlegende Konfigurationsdaten (Lebt der Gegner überhaupt noch? Hat er auf mich geschossen?) sind entscheidend für den eigentlichen Spielverlauf und sollten deshalb zuverlässig über TCP übertragen werden. Dagegen sind bestimmte Details (Pose und Gesichtsausdruck der gegnerischen Spielfigur; die Position von Gegnern außerhalb des »Gesichtsfeldes« etc.) nicht so wichtig. Wenn überhaupt, sollten sie möglichst schnell übertragen werden. Fallen sie vorübergehend aus, schadet das auch nichts – ideale Kandidaten für die Übertragung mithilfe des schnelleren, aber weniger zuverlässigen UDP-Protokolls.

Der Hauptgrund, warum sich Daten über UDP schneller übertragen lassen als über TCP, ist der erheblich kleinere und weniger komplexe Paket-Header. Der Aufbau dieses Headers, der gerade einmal (unveränderlich) 64 Bit groß ist, wird in Tabelle 4.20 dargestellt.

Byte	0	1	2	3
0	Quellport		Zielport	
4	Länge		Prüfsumme	

Tabelle 4.20 Aufbau des UDP-Headers

Die einzelnen Header-Felder haben dieselbe Bedeutung wie die gleichnamigen Felder beim TCP-Protokoll. Mit der Länge ist hier die Länge des gesamten Pakets inklusive dieses Headers gemeint. Der Quellport wird häufig einfach auf 0 gesetzt: Da UDP dem schnellen Versand einer einzelnen Nachricht dient, auf die in der Regel keine Antwort erwartet wird, ist es nicht nötig, diese Information festzulegen. Der Zielport ist dagegen meist der festgelegte Port des UDP-Servers, an den das Paket verschickt wird. UDP wird für viele einfache Internetdienste verwendet: die Uhrzeitsynchronisation über ein Netzwerk, den Echo-Dienst zur Kontrolle der Funktionstüchtigkeit von Verbindungen oder entfernten Hosts etc.

Im Gegensatz zum verbindungsorientierten TCP wird UDP als *nachrichtenorientiertes Protokoll* bezeichnet, da es dem schnellen, verbindungslosen Versand einzelner Pakete in Form kurzer Meldungen dient. Dies erklärt auch den Namen des Protokolls: Einer Anwendung, die von diesem Transportdienst Gebrauch macht, wird der unmittelbare und leichtgewichtige Zugriff auf IP-Datagramme ermöglicht.

Die Portnummern für gängige Serverdienste (bei UDP spricht man häufig auch von *Servicenummern*) liegen wie bei TCP zwischen 0 und 1023. Sie werden genau wie öffentliche IP-Adressen von der IANA vergeben. In der Regel wird dieselbe Portnummer für beide Transportprotokolle verwendet, obwohl die meisten Anwendungen nur auf jeweils einem der bei-

den Protokolle laufen. Tabelle 4.21 zeigt einige häufig verwendete Beispiele mit ihrem offiziellen Namen und dem am häufigsten verwendeten Transportprotokoll. Die vollständige Liste aller öffentlichen Serverdienste finden Sie unter *http://www.iana.org/assignments/port-numbers*. Falls Sie ein Unix-System verwenden, steht eine ähnliche, möglicherweise weniger vollständige Liste in der Konfigurationsdatei */etc/services*.

Nummer	Transportprotokoll	Name	Beschreibung
7	TCP, UDP	`echo`	genaue Rückgabe der übermittelten Daten zur Kontrolle
13	TCP, UDP	`daytime`	Datum und Uhrzeit (RFC 867)
20	TCP	`ftp`	FTP-Datenstrom
21	TCP	`ftp`	FTP-Steuerung
22	TCP	`ssh`	Secure Shell – Telnet-Alternative mit Verschlüsselung
23	TCP	`telnet`	Terminal-Emulation
25	TCP	`smtp`	E-Mail-Versand
53	TCP, UDP	`domain`	Nameserver-Abfragen
80	TCP	`http`	Webserver
110	TCP	`pop3`	E-Mail-Postfach-Server (klassisch)
142	TCP	`imap`	E-Mail-Postfach-Server (modern)
443	TCP, UDP	`https`	SSL-verschlüsselte Webserver-Kommunikation

Tabelle 4.21 Einige TCP/UDP-Portnummern für gängige Dienste

4.6.4 Das Domain Name System (DNS)

Die Verwendung von IP-Adressen zum Erreichen entfernter Rechner ist ideal, solange in die Datenübertragung nur Computer involviert sind. Für die Verwendung durch Menschen sind sie weniger gut geeignet (es gibt zum Beispiel nur wenige Menschen, die sich Telefonnummern auf Anhieb besser merken können als die zugehörigen Namen). Aus diesem Grund ist es seit den Anfängen des Internets und seiner Vorläufer üblich, einen Mechanismus einzurichten, der den beteiligten Menschen die Verwendung benutzerfreundlicher Namen anstelle der unhandlichen IP-Adressen ermöglicht.

Als das ARPANET entwickelt wurde, behalf man sich mit einer einfachen Textdatei, die pro Zeile einen Hostnamen und eine IP-Adresse nebeneinander auflistete. Noch heute verwen-

den Unix-Rechner eine ähnliche Datei namens */etc/hosts*. Auch unter Windows ist das Verfahren bekannt. Hier befindet sich die Datei in *<Windows-Verzeichnis>\system32\drivers\etc* und heißt – untypisch für Windows – ebenfalls nur *hosts*, ohne Dateiendung. Allerdings wird dieses Verfahren heute immer seltener für die Namenszuordnung in lokalen Netzen eingesetzt, weil in immer mehr Firmennetzen DHCP verwendet wird.

Findet der Rechner einen Namenseintrag in seiner */etc/hosts*-Datei, wird er die entsprechende Adresse nicht mehr bei einem Nameserver nachfragen. Auf diese Weise können Sie natürlich Ihre Kollegen ein wenig ärgern: Machen Sie beispielsweise die IP-Adresse ausfindig, die zu *www.yahoo.de* gehört (aktuell 77.238.184.150), und tragen Sie in die Datei */etc/hosts* eines Kollegen etwa folgende Zeile ein:

```
77.238.184.150 www.google.de
```

Jedes Mal, wenn der Kollege nun die Suchmaschine Google anwenden möchte, wird er stattdessen bei Yahoo! landen, kann sich das aber zunächst beim besten Willen nicht erklären.

Früher wurde die Datei namens *hosts.txt* zentral verwaltet und regelmäßig unter den teilnehmenden Hosts im ARPANET ausgetauscht, um die Namensdaten aktuell zu halten. Als das Netz jedoch immer größer wurde, funktionierte dieses System nicht mehr, weil man mit den häufigen Änderungen nicht mehr nachkam und weil die gesamte Datei außerdem sehr umfangreich war und ihr Versand eine erhebliche Menge an Netzwerkverkehr erzeugte.

Schließlich wurde anstelle der einfachen Textdatei eine hierarchische, vernetzte Datenbank eingeführt, die bis heute ein verteiltes Netz von Nameservern bildet. Diese Server geben auf Anfrage Auskunft über die zu einem Hostnamen gehörende IP-Adresse oder umgekehrt. Außerdem leiten sie die Anfrage automatisch weiter, wenn sie selbst keine Antwort wissen. Das System wird als *Domain Name System* (DNS) bezeichnet und ist Thema einer ganzen Reihe von RFCs. Die wesentlichen Grundlagen werden in RFC 1034 und 1035 beschrieben.

Damit Hostnamen im gesamten Internet eindeutig sind, werden sie hierarchisch als sogenannte *Domainnamen* vergeben. Zu diesem Zweck wird ein Name aus immer spezialisierteren Bestandteilen zusammengesetzt; das System lässt sich mit einem Pfad in einem Dateisystem vergleichen. Allerdings besteht ein wesentlicher Unterschied: Beim Dateisystempfad steht der allgemeinste Name vorn und der speziellste hinten, während es beim Domainnamen genau umgekehrt ist.

Beispielsweise bedeutet der Unix-Pfad */home/sascha/hb_fachinfo/netzwerk/protokolle.txt*, dass sich die Datei *protokolle.txt* im Verzeichnis *netzwerk* befindet, einem Unterverzeichnis von *hb_fachinfo*, das wiederum dem Verzeichnis *sascha* untergeordnet ist. *sascha* ist seinerseits ein Unterverzeichnis von *home*, das schließlich direkt unter der Wurzel des Dateisystems (*/*) liegt.

Dagegen ist der Domainname *www.buecher.lingoworld.de* genau umgekehrt aufgebaut: Der Host/Dienst *www* liegt in der Domain *buecher*, einer Subdomain von *lingoworld* in der

Top-Level-Domain *de*. Die Wurzel des DNS-Systems selbst ist nicht sichtbar, weil ihr Name der leere String ist.

Auf der jeweiligen Ebene des DNS-Systems muss ein bestimmter Name einmalig sein. Beispielsweise kann es *buecher.lingoworld.de* nur einmal geben. Unterhalb dieser Domain können untergeordnete Domains (Subdomains) oder die Namen einzelner Hosts oder Serverdienste bestehen, beispielsweise *www.buecher.lingoworld.de*, *ftp.buecher.lingoworld.de* oder *neuheiten.buecher.lingoworld.de*. Im Übrigen dürfen dieselben Namen natürlich auf über- oder untergeordneten oder auch auf »Geschwister-Ebenen« existieren: Es kann die Website *www.lingoworld.de* ebenso geben wie etwa *www.download.lingoworld.de*. Selbstverständlich ist auch *www.buecher.de* kein Problem – es handelt sich um eine andere Domain unterhalb der Top-Level-Domain *de*.

Aus der Sicht der DNS-Administration wird jede Ebene eines solchen Namens auch als *Zone* bezeichnet, weil eine solche Ebene jeweils unabhängig von den übergeordneten Ebenen verwaltet wird. Beispielsweise kann der Administrator der Domain *lingoworld.de* Subdomains wie *buecher.lingoworld.de* oder *download.lingoworld.de* einrichten. Er kann die Verantwortung für eine Subdomain auch an jemand anderen delegieren, für den dann beispielsweise *buecher.lingoworld.de* wieder eine unabhängige Zone darstellt. Andererseits kann der Zonenverantwortliche für *lingoworld.de* nicht auf andere Zonen in der Domain *.de* zugreifen; beispielsweise geht ihn die Konfiguration der Zone *google.de* nichts an.

Die Infrastruktur der Domainnamen wird von den über das gesamte Internet verbreiteten Nameservern verwaltet. Diese führen alle ein Programm aus, das Anfragen nach Name-Adresse-Zuordnungen beantwortet, unbekannte Zuordnungen bei anderen Nameservern erfragt und dann meistens dauerhaft speichert. Die am häufigsten verwendete derartige Software heißt BIND (Berkeley Internet Name Domain) und läuft unter allen Unix-Varianten; sie wird in Kapitel 15, »Weitere Internet-Serverdienste«, vorgestellt.

Auf der obersten Ebene des DNS existiert die spezielle Zone, deren Name der leere String ist. Diese Zone wird durch die Root-Nameserver der ICANN verwaltet und enthält Verweise auf alle Top-Level-Domains. Davon gibt es organisatorisch gesehen zwei Sorten (auch wenn es keinen technischen Unterschied gibt):

- Die »Generic TLDs« (allgemeine Top-Level-Domains) wie *.com* oder *.org* unterteilen die jeweiligen Domains, die unter ihnen liegen, nach der Funktion ihrer Betreiber.
- Die »Country TLDs« oder »ccTLDs« (Länder-TLDs) sind dagegen für eine geografische Einteilung vorgesehen.

In der Praxis kommt es ohnehin zu einer Vermischung: Zum einen sind viele Generic TLDs mittlerweile für beliebige Betreiber verwendbar, zum anderen gibt es einige Länder-TLDs, die wegen ihrer spezifischen Abkürzung für bestimmte Branchen interessant sind – am bekanntesten ist in diesem Zusammenhang wohl der Südsee-Inselstaat Tuvalu mit seiner bei Fernsehsendern und Web-Videodiensten beliebten TLD *.tv*.

Tabelle 4.22 listet einige häufig verwendete Top-Level-Domains auf. Die mit Abstand meisten Betreiber-Domains enthält die Generic-TLD *.com*, gefolgt von der länderspezifischen Domain *.de* (Deutschland).

Top-Level-Domain	Bedeutung
Generic Top-Level-Domains	
.com	*commercial* (Firmen)
.org	*organization* (Organisationen und Vereine)
.net	*network* (Netzwerkbetreiber; Internetinfrastruktur)
.edu	*educational* (US-Schulen und -Universitäten)
.gov	*government* (US-Regierung, US-Behörden, öffentlicher Dienst)
.mil	*military* (US-Militär)
.info	*information* (allgemeine Informationsdienste)
.aero	*aeronautics* (Luftfahrtindustrie, Fluggesellschaften)
Länder-Top-Level-Domains	
.at	Österreich
.ch	Schweiz
.cn	Volksrepublik China
.de	Deutschland
.es	Spanien
.fr	Frankreich
.it	Italien
.jp	Japan
.ru	Russland
.tr	Türkei
.uk	Vereinigtes Königreich
.us	USA
.va	Vatikanstadt

Tabelle 4.22 Übersicht über einige wichtige Top-Level-Domains

Der jeweilige Haupt-Nameserver einer Top-Level-Domain enthält Verweise auf sämtliche unterhalb dieser Domain befindlichen Second-Level-Domains. Je nach konkreter TLD handelt es sich dabei entweder unmittelbar um die einzelnen Domains, die von Betreibern angemeldet werden können, oder eine Domain ist in sich noch einmal in Organisationsstrukturen unterteilt. Bei allen Generic-TLDs und den meisten Länder-TLDs ist Ersteres der Fall. Nur einige Länder-TLDs werden noch einmal organisatorisch unterteilt: Zum Beispiel verwendet das Vereinigte Königreich Unterteilungen wie *.co.uk* für Firmen, *.ac.uk* für Universitäten oder *.org.uk* für Vereine und Organisationen; auch die Türkei verwendet die entsprechenden Unterteilungen *.com.tr*, *.edu.tr* und *.org.tr*.

Aus Sicherheitsgründen sollten die Zonendaten für die Domain eines einzelnen Betreibers auf mindestens zwei voneinander unabhängigen (das heißt in verschiedenen autonomen Systemen befindlichen) Nameservern vorliegen. Die Daten müssen dafür nur auf einem der beiden Server erstellt werden, der als *primärer Master-Nameserver* bezeichnet wird; der andere – *Slave-Nameserver* genannt – repliziert sie automatisch. Größere Unternehmen und Institutionen verwalten in der Regel ihre eigenen Zonen. Der externe Slave-Nameserver mit denselben Zonendaten befindet sich in diesem Fall meist beim zuständigen Backbone-Provider, über den diese Betreiber mit dem Internet verbunden sind. Privatanwender oder kleinere Firmen besitzen dagegen zwar häufig eine eigene Domain (*www.meinname.de* ist werbewirksamer als so etwas wie *home.t-online.de/users/meinname*), unter dieser Domain laufen allerdings oft lediglich eine beim Provider gehostete Website und einige E-Mail-Adressen. In diesem Fall werden die Zonendaten meist beim Hosting-Provider und einem anderen Provider verwaltet; die beiden Provider stellen sich den Slave-Name-Service dann gegenseitig zur Verfügung.

Bei den meisten Einzelrechnern oder kleineren Firmennetzwerken besteht die ganze DNS-Konfiguration oft lediglich aus der Eingabe der IP-Adresse eines Nameservers des eigenen Providers; in vielen Fällen ist sogar dies unnötig, weil die Standard-Nameserver beim Verbindungsaufbau bekannt gegeben werden. Die Nameserver werden stets befragt, wenn Namensdaten erforderlich sind.

Gebe ich zum Beispiel in meinem Webbrowser »www.google.de« ein, überprüft dieser zunächst, ob er die IP-Adresse vielleicht bereits kennt. Andernfalls fragt er den Standard-Nameserver des Providers. Dieser weiß die Antwort entweder selbst und liefert sie unmittelbar zurück oder wendet sich an den übergeordneten Nameserver – in diesem Fall den für die Top-Level-Domain *.de* zuständigen Server. Der wiederum kennt die für die Domain *google.de* zuständigen Nameserver und leitet die Anfrage an den ersten von ihnen weiter. Dieser ermittelt die IP-Adresse des Dienstes *www.google.de* und gibt sie zurück. Nun weiß der Browser, welche IP-Adresse er verwenden muss. Außerdem speichert der Nameserver des Providers die gefundene Adresse ebenfalls in seinem Cache ab, um die nächste entsprechende Anfrage schneller beantworten zu können.

Insgesamt stellt das DNS ein leistungsfähiges, flexibles und effizientes System zur Verwaltung benutzerfreundlicher Hostnamen zur Verfügung. Es wird im gesamten Internet eingesetzt und zumindest clientseitig von jedem beliebigen Betriebssystem unterstützt. Allerdings handelt es sich nicht um die einzige Art und Weise der Namensverwaltung. Gerade in herstellerabhängigen lokalen Netzen werden Dienste wie der Windows-Namensdienst WINS oder das Network Information System (NIS) von Sun eingesetzt. Letzteres ist nicht nur ein Namens-, sondern auch ein einfacher Verzeichnisdienst.

4.6.5 Verschiedene Internetanwendungsprotokolle

Genau, wie beinahe jede Hardware die TCP/IP-Protokolle unterstützt, laufen auf der Anwendungsschicht des Internetprotokollstapels auch fast alle Arten von Netzwerkanwendungen. Dazu gehören unter anderem auch Anwendungsprotokolle, die ursprünglich für bestimmte herstellerabhängige Netzwerke konzipiert wurden, beispielsweise die Standard-Fileserver-Protokolle der diversen Betriebssysteme: Das unter Windows verwendete SMB-Protokoll (Server Message Blocks) wurde zunächst auf Microsofts eigenes NetBEUI-Netzwerk aufgesetzt; das von Apple konzipierte AppleShare lief ursprünglich nur unter AppleTalk. Mittlerweile werden diese speziellen Anwendungsprotokolle praktisch exklusiv auf TCP/IP aufgesetzt.

An dieser Stelle geht es lediglich darum, die grundlegende Funktionsweise einiger typischer Internetdienste auf der Ebene ihrer Protokolle zu beschreiben. Falls Sie also Details über die Verwendung von Internet-Client-Server-Diensten oder deren Konfiguration unter einem bestimmten Betriebssystem suchen, sollten Sie Kapitel 14, »Server für Webanwendungen«, und Kapitel 15, »Weitere Internet-Serverdienste«, lesen. Hier erfahren Sie dagegen eher, was hinter den Kulissen wirklich passiert, wenn Sie eine E-Mail versenden oder eine Webseite anfordern.

Das (für Administratoren und Programmierer) Angenehme an den meisten Internetanwendungsprotokollen ist, dass die Protokollbefehle in Form von Klartextwörtern in Englisch verschickt werden. Wenn Sie mit einem Packet-Sniffer oder einfach mit *telnet* die Inhalte der über das Netzwerk übertragenen Datenpakete kontrollieren, können Sie deshalb unmittelbar verstehen, was die verschiedenen Hosts miteinander »reden«. Auf diese Weise ist es verhältnismäßig einfach, Konfigurations- oder Programmierfehler auf der Ebene der Anwendungsprotokolle zu entdecken und zu beseitigen.

In der Regel bestehen die Anforderungen eines Internetanwendungsclients aus einzeiligen Befehlen, die vom Server mit einer Statusmeldung und manchmal auch mit der Lieferung konkreter Daten beantwortet werden. Sie können das Verhalten eines Clients simulieren, indem Sie mit einem Terminal-Programm wie *telnet* eine Verbindung zu dem passenden Host und Port aufbauen und die entsprechenden Befehle von Hand eintippen.

Telnet

Eine der ältesten Anwendungen des Internets ist die Terminal-Emulation: Ein Programm ermöglicht Ihnen über ein Terminal, das an Ihren Computer angeschlossen ist, die Arbeit an einem anderen Computer, zu dem eine Netzwerkverbindung besteht. Telnet ist eines der wichtigsten Werkzeuge für Systemadministratoren, die auf diese Weise entfernte Rechner verwalten, ohne sich physikalisch dorthin zu begeben (insbesondere an Wochenenden oder nach Feierabend schätzen Admins diese Möglichkeit, weil sie eventuelle Pannenhilfe von zu Hause aus erledigen können). Die Telnet-Spezifikation ist in RFC 854 festgelegt.

Fast alles, was an dieser Stelle über Telnet gesagt wird, gilt sinngemäß auch für SSH, die Secure Shell. Im Grunde genommen, handelt es sich dabei um eine sichere Variante von Telnet, die mit Verschlüsselung arbeitet. Der gravierendste Schönheitsfehler des klassischen Telnets besteht nämlich darin, dass es Daten im Klartext überträgt – und das gilt unter anderem auch für Passwörter und ähnlich sensible Daten. SSH ist nicht in einem RFC spezifiziert, denn obwohl es sich aus frei verfügbaren Komponenten zusammensetzt, ist die ursprüngliche Implementierung kommerziell. Nähere Informationen dazu erhalten Sie auf der Website *http://www.ssh.com*. Eine freie Implementierung, die in den meisten Linux- und anderen Unix-Systemen zum Einsatz kommt, ist OpenSSH (*http://www.openssh.com*).

Der Telnet-Server lauscht auf dem TCP-Port 23 auf eingehende Verbindungen (SSH verwendet Port 22). Wenn ein TCP-Verbindungsversuch erfolgt, wird der Benutzer am entfernten Host zunächst nach Benutzernamen und Passwort gefragt, bevor er tatsächlich arbeiten kann. Im Grunde genommen, wird dem jeweiligen Benutzer seine Standard-Unix-Shell zur Verfügung gestellt, an der er auch lokal auf dem entsprechenden Rechner arbeiten würde.

Telnet und SSH sind daher beliebig flexibel, was den Inhalt der in beide Richtungen übermittelten Daten angeht. Wenn Sie erst einmal mit dem Telnet-Server verbunden sind, können Sie jedes beliebige Programm auf dem entfernten Host ausführen, für das Sie Benutzerrechte besitzen. Dazu gehören auch solche Programme, die nicht zeilenorientiert, sondern mit einer Vollbildmaske arbeiten – zum Beispiel die klassischen Unix-Texteditoren *vi* und *Emacs*. Deshalb genügt die zeilenorientierte Kommunikation zwischen Client und Server bei Telnet nicht: In einem Vollbildprogramm kann jeder einzelne Tastendruck eine Bedeutung haben, die unmittelbar umgesetzt werden muss. Falls Sie Beispiele dafür sehen möchten, was Sie in einer SSH- oder Telnet-Sitzung eingeben können, lesen Sie einfach die Abschnitte über die Shell in Kapitel 7, »Linux«. Alles, was dort steht, gilt auch für den Fernzugriff.

Die einzige Art von Programmen, die nicht über Telnet ausgeführt werden können, sind solche, die auf einer grafischen Benutzeroberfläche laufen. Allerdings bietet die Unix-Welt auch für dieses Problem die passende Lösung: Sie benötigen auf Ihrem lokalen Rechner zusätzlich zu dem Telnet-Client einen X-Window-Server, der die grundlegenden Zeichenfunktionen für das GUI zur Verfügung stellt. Sobald dieser X-Server läuft, können Sie ein X-basiertes Anwendungsprogramm auf dem entfernten Server starten und Ihren eigenen Rechner als Ziel der grafischen Darstellung angeben (in der Regel mit dem Parameter `display`). Angenommen,

Ihr eigener Rechner besitzt im lokalen Netz die IP-Adresse 192.168.0.9. Dann können Sie in das Telnet-Programm, in dem eine Sitzung auf einem anderen Rechner läuft, Folgendes eingeben, um in Ihrem X-Server ein *xterm* (ein X-basiertes Terminal) zu starten:

```
# xterm -display 192.168.0.9:0.0
```

Der Zusatz `0.0` hinter der IP-Adresse bedeutet sinngemäß »erster X-Server, erster Bildschirm«. Wichtig ist, dass Sie den Begriff *X-Server* nicht falsch verstehen: Hier läuft der Server, der die grafische Oberfläche als Dienstleistung zur Verfügung stellt, auf Ihrem eigenen Rechner, während der Client das auf dem entfernten Rechner laufende Programm ist, dessen Ausgabe in Ihrem lokalen X-Window-System erfolgt. Näheres über die Konfiguration von X-Servern unter Unix erfahren Sie in Kapitel 7, »Linux«. Allerdings gibt es auch X-Server für andere Systeme, beispielsweise für Windows. Sie sind natürlich nicht für die grafische Darstellung lokaler Programme gedacht (das können die eingebauten GUIs von OS X oder Windows selbst gut genug), sondern für grafisch orientierte Programme, die auf entfernten Unix-Rechnern laufen.

Im Übrigen sollten Sie das Telnet-Protokoll, das die Terminal-Emulation bereitstellt, nicht mit dem Unix- und Windows-Dienstprogramm *telnet* verwechseln. Letzteres kann nämlich – wie bereits erwähnt – mit fast jedem Internetserver kommunizieren, wenn Sie die passenden Parameter (IP-Adresse beziehungsweise Hostname und Portnummer beziehungsweise Standarddienstname) eingeben.

FTP

Das File Transfer Protocol (FTP) ist beinahe das genaue Gegenteil von Telnet: ein aus ganz wenigen Befehlen bestehendes, klartextbasiertes, zeilenorientiertes Protokoll. Es gehört zu den frühesten Internetanwendungen überhaupt. Seine erste Definition steht in RFC 172 von 1971, die aktuelle Spezifikation befindet sich in RFC 959. Den reinen Datei-Download über FTP beherrscht heutzutage fast jeder Webbrowser; die meisten stellen auch die Verzeichnisansicht des entfernten Rechners übersichtlich und angenehm dar.

In der Praxis wird jedoch überwiegend ein grafisch orientierter FTP-Client verwendet, der dem lokalen Dateinavigator Ihres Betriebssystems idealerweise möglichst ähnlich sieht. Die häufigste Anwendung für ein solches Programm dürfte die Pflege einer eigenen Website sein. Dabei bearbeiten Sie die Daten in der Regel auf Ihrem eigenen Rechner und laden sie anschließend mithilfe eines solchen FTP-Programms auf den Server Ihres Hosting-Providers hoch, um sie zu veröffentlichen. Bekannte FTP-Clients sind beispielsweise WS_FTP für Windows oder Fetch für OS X. Auch in gängige Website-Editoren wie Adobe Dreamweaver sind FTP-Module eingebaut.

Falls Sie jedoch genau sehen möchten, wie FTP-Client (auf Ihrem eigenen Rechner) und -Server (auf dem entfernten Rechner) miteinander kommunizieren, können Sie das in Unix und Windows eingebaute Konsolenprogramm *ftp* verwenden. Die Befehle, die Sie auf der Clientseite eingeben, sind jeweils einzeilig und bestehen aus einem Schlüsselwort mit eventuellen

Parametern, gefolgt von einem Zeilenumbruch. Die Antwort des Servers ist zunächst eine Statusmeldung, die aus einer dreistelligen dezimalen Codenummer und einem Meldungstext besteht; häufig folgen auf die Statusmeldung zusätzliche Datenzeilen. Um dem Client das Ende einer solchen Datensequenz zu signalisieren, beginnt die letzte Zeile der Antwort des Servers wieder mit derselben Codenummer wie die erste.

Die folgenden Zeilen zeigen den Mitschnitt einer FTP-Sitzung mit dem Host *www.lingoworld.de*. Der Name *www* besagt natürlich, dass es sich eigentlich um einen Webserver handelt. Es ist durchaus üblich, dass Hosting-Provider den Webserver unmittelbar per FTP zugänglich machen, um die eigene Website hochzuladen:

```
> ftp www.lingoworld.de
Verbunden zu www.lingoworld.de.
220 FTP Server ready.
Benutzer (www.lingoworld.de:(none)): XXXXX
331 Password required for XXXXX.
Kennwort: [Eingabe wird nicht angezeigt]
230 User XXXXX logged in.
Ftp> pwd
257 "/" is current directory.
Ftp> cd extra
250 CWD command successful.
Ftp> ls
200 PORT command successful.
150 Opening ASCII mode data connection for file list.
test.txt
info.txt
226-Transfer complete.
226 Quotas off
21 Bytes empfangen in 0,01 Sekunden (2,10 KB/s)
Ftp> get test.txt
200 PORT command successful.
150 Opening ASCII mode data connection for test.txt (2589 bytes).
226 Transfer complete.
2718 Bytes empfangen in 0,04 Sekunden (67,95 KB/s)
Ftp> quit
221 Goodbye.
```

In dieser Sitzung wird zunächst die Anmeldung durchgeführt (der Benutzername wird angezeigt, allerdings habe ich ihn hier geändert; die Passworteingabe hat kein grafisches Feedback), anschließend werden die folgenden Befehle eingesetzt (die ersten drei entsprechen gleichnamigen Unix-Shell-Befehlen, siehe Kapitel 7, »Linux«):

- `pwd`: *print working directory* – aktuelles Arbeitsverzeichnis ausgeben
- `cd`: *change directory* – Verzeichnis auf dem entfernten Server wechseln
- `ls`: *list* – Verzeichnisinhalt anzeigen
- `get`: die angegebene Datei in das aktuelle lokale Verzeichnis herunterladen
- `quit`: die Sitzung und das FTP-Programm beenden

Weitere wichtige Befehle sind folgende:

- `put`: die angegebene Datei in das aktuelle entfernte Verzeichnis hochladen
- `binary`: umschalten in den Binärmodus
- `ascii`: umschalten in den ASCII-Modus
- `help`: eine Liste aller verfügbaren Befehle anzeigen

Es ist wichtig, dass Sie den Unterschied zwischen dem ASCII- und dem Binärmodus verstehen. Das ganze Problem hat damit zu tun, dass die verschiedenen Betriebssystementwickler sich nicht auf einen gemeinsamen Standard für Zeilenumbrüche in Textdateien einigen konnten. Wie in Kapitel 17, »Weitere Datei- und Datenformate«, genau erläutert wird, verwendet Unix das ASCII-Zeichen mit dem Code 10 (LF, Line Feed oder Zeilenvorschub), klassisches Mac OS das ASCII-Zeichen 13 (CR, Carriage Return oder Wagenrücklauf), und Windows sowie die meisten Netzwerkanwendungsprotokolle benutzen beide Zeichen hintereinander.

Im ASCII-Modus werden die Zeilenumbrüche innerhalb einer Datei bei der Übertragung jeweils umgewandelt, sodass beispielsweise die auf Ihrem Windows-Rechner gespeicherten Textdateien mit CR/LF auf dem entfernten Unix-Server mit dem für dessen Verhältnisse korrekten (Nur-)LF ankommen und umgekehrt. Sie sollten jedoch begreifen, dass dieses bei Textdateien recht segensreiche Feature bei Binärdateien wie Bildern oder Programmen den sicheren Tod zur Folge hat. Wird jedes Vorkommen des Byte-Wertes 10 durch die beiden Bytes 13 und 10 ersetzt oder umgekehrt, werden die Bytes in einer solchen Datei verändert und planlos verschoben! Natürlich ist eine auf diese Weise behandelte Bild-, Audio- oder Programmdatei unbrauchbar.

Die meisten grafisch orientierten FTP-Programme entscheiden je nach Dateityp passend selbst, ob sie ASCII- oder Binärübertragung verwenden sollen. Bei dem Konsolen-FTP-Programm müssen Sie für jede einzelne Datei selbst in den richtigen Modus umschalten. Das ist ein – aber nicht der einzige – Grund dafür, dass die Arbeit mit der Konsolenversion von FTP in der Praxis fast unzumutbar ist.

E-Mail

Die E-Mail, die sich unter dem Dach eines Mailclients wie Thunderbird, Outlook Express oder Apple Mail so einheitlich präsentiert, bedarf in Wirklichkeit der Zusammenarbeit mit mindestens zwei verschiedenen Servern. Der eine ist für den Versand von E-Mails zuständig und führt zu diesem Zweck das Protokoll SMTP (Simple Mail Transport Protocol) aus. Ein anderer

enthält das E-Mail-Postfach, in dem an Sie adressierte Nachrichten ankommen; dieser Dienst wird entweder von dem weitverbreiteten POP3-Protokoll (Post Office Protocol Version 3) oder dem komfortableren IMAP (Internet Message Access Protocol) versehen.

Wenn Sie eine E-Mail versenden möchten, wird diese an einen SMTP-Server übermittelt, der sich um die Weiterleitung der Nachricht an den Empfänger kümmert. SMTP, definiert in RFC 2821 (Neufassung von RFC 821), ist ähnlich wie FTP ein einfaches, textbasiertes Protokoll aus wenigen Befehlen; der zuständige Server wartet am TCP-Port 25 auf Verbindungen.

Einige SMTP-Server von Internetprovidern kontrollieren bis heute nicht die Identität des Absenders. Das Problem dabei ist, dass solche Server dadurch leicht für das Versenden von Spam verwendet werden können oder dass sogar jemand eine falsche Identität vortäuschen kann. Dabei sieht die SMTP-Spezifikation durchaus mehrere mögliche Authentifizierungsverfahren vor:

- Manche SMTP-Server überprüfen die IP-Adresse des Hosts, von dem die Verbindung initiiert wurde – ein ideales Verfahren für normale Internetprovider, die nur ihren eigenen Kunden Zugriff auf ihre SMTP-Server gewähren möchten.
- Eine andere Möglichkeit besteht darin, die Anmeldung am E-Mail-Empfangsserver desselben Providers als Voraussetzung für den E-Mail-Versand zu verlangen. Dieses Verfahren wird *SMTP after POP* genannt. Nachteil: Manche E-Mail-Clients können nicht damit umgehen, sodass man jedes Mal vor dem E-Mail-Versand auf MAIL EMPFANGEN klicken muss.
- Das sicherste Verfahren wurde erst nachträglich zu SMTP hinzugefügt (inzwischen ist es aber glücklicherweise flächendeckend verbreitet): Die persönliche Anmeldung beim SMTP-Server mit Benutzernamen und Passwort.

Sie können die Kommunikation mit einem SMTP-Server über das Programm *telnet* abwickeln. Eine solche Sitzung sieht beispielsweise folgendermaßen aus (die konkreten Namens- und Adressdaten habe ich anonymisiert):

```
> telnet smtp.myprovider.de smtp
220 smtp.myprovider.de ESMTP Mon, 13 Apr 2015 12:37:21 +0100
HELO
250 smtp.myprovider.de Hello[203.51.81.17]
MAIL From: absender@myprovider.de
250 <absender@myprovider.de> is syntactically correct
RCPT To: empfaenger@elsewhere.com
250 <empfaenger@elsewhere.com> verified
DATA
354 Enter message, ending with "." on a line by itself
FROM: Sascha <absender@myprovider.de>
To: Jack <empfaenger@elsewhere.com>
Subject: Gruesse

Hallo Jack,
hier ist wieder einmal Post für dich.
Viel Spaß damit!
Gruss, Sascha
.
250 OK id=18QdIY-00048Y-00
QUIT
221 smtp.myprovider.de closing connection.
```


In dieser kurzen Konversation werden die folgenden SMTP-Befehle verwendet:

- `HELO`: Mit diesem Befehl meldet sich der Client beim Server an; eventuell findet in diesem Zusammenhang die bereits beschriebene Überprüfung der Client-IP-Adresse statt. Manche SMTP-Server verlangen auch die Angabe eines Domainnamens hinter dem Befehl.
- `MAIL`: Dieser Befehl leitet die Erzeugung einer neuen Nachricht ein; der Empfänger muss im Format `From: E-Mail-Adresse` oder `From: Name <E-Mail-Adresse>` angegeben werden.
- `RCPT`: Gibt einen Empfänger im Format `To: E-Mail-Adresse` oder `To: Name <E-Mail-Adresse>` an.
- `DATA`: Alle folgenden Zeilen des Clients werden als Teil der eigentlichen E-Mail-Nachricht aufgefasst, bis eine Zeile folgt, die nur einen Punkt (`.`) enthält.
- `QUIT`: Die Sitzung wird hiermit beendet; alle bis zu diesem Zeitpunkt erzeugten E-Mail-Nachrichten werden versandt.

Die E-Mail-Nachricht selbst (zwischen `DATA` und der Abschlusszeile mit dem Punkt) ist eine klassische Textnachricht, deren Aufbau in RFC 822 (aktualisiert in RFC 2822) beschrieben wird. Prinzipiell besteht sie aus mehreren Header-Zeilen im Format `Feldname: Wert`, gefolgt von einer Leerzeile und dem eigentlichen Text. Der minimale Header enthält den Absender (`From`), den Empfänger (`To`) und einen Betreff (`Subject`). Absender und Empfänger dürfen wie bei den SMTP-Befehlen `MAIL` und `RCPT` diverse Formate besitzen. Weitere häufige Header-Felder sind die Kopienempfänger (`Cc` für »Carbon Copy«) sowie die unsichtbaren Kopienempfänger (`Bcc` für »Blind Carbon Copy«). Die normalen Kopienempfänger werden in der Nachricht angezeigt, die unsichtbaren nicht.

Ein alternatives Format für E-Mails, das heutzutage bereits häufiger verwendet wird als RFC 822, ist das MIME-Format. Die verschiedenen Aspekte von MIME werden in RFC 2045 bis 2049 dargelegt. Die Abkürzung MIME steht für »Multipurpose Internet Mail Extensions«. Es handelt sich um ein Format, das für den Versand beliebiger Text- und Binärdaten geeignet ist, sogar von verschiedenen Datentypen innerhalb derselben Nachricht.

Der MIME-Header ist eine Erweiterung des RFC-822-Headers. Die wichtigsten neuen Felder sind `Content-type`, das den Datentyp angibt, und `Content-Transfer-Encoding`, mit dessen Hilfe das Datenformat festgelegt wird. Ersteres beschreibt also den Inhalt der Nachricht, Letzte-

res die Form, in der sie versandt wird. Der Inhaltstyp (`Content-Type`), meist *MIME-Type* genannt, besteht aus zwei Bestandteilen, die durch einen Slash (/) voneinander getrennt werden: dem Haupttyp und dem genaueren Untertyp. Haupttypen sind beispielsweise `text` (ASCII-Text), `image` (Bilddaten), `audio` (Audiodaten), `video` (Digitalvideo) oder `application` (proprietäres Datenformat eines bestimmten Anwendungsprogramms). Tabelle 4.23 listet einige gängige MIME-Types auf. Die vollständige Liste aller registrierten Typen finden Sie online unter *http://www.iana.org/assignments/media-types/index.html*.

Typ	Beschreibung
`text/plain`	reiner Text ohne Formatierungsbefehle
`text/html`	HTML-Code
`text/xml`	XML-Code
`image/gif`	Bild vom Dateityp GIF
`image/jpeg`	Bild vom Dateityp JPEG
`image/png`	Bild vom Dateityp PNG
`audio/wav`	Sounddatei vom Typ Microsoft Wave
`audio/aiff`	Sounddatei vom Typ Apple AIFF
`audio/mpeg`	komprimierte Sounddatei vom Typ MP3
`video/avi`	Digitalvideo vom Typ Microsoft Video for Windows
`video/mov`	Digitalvideo vom Typ Apple QuickTime
`video/mpeg`	Digitalvideo vom Typ MPEG
`application/x-shockwave-flash`	komprimierter Adobe-Flash-Film (Dateiendung *.swf*)
`application/x-director`	komprimierter Adobe-Director-Film (Dateiendung *.dcr*)
`application/x-www-form-urlencoded`	POST-Formulardaten bei HTTP-Anfragen an Webserver (Näheres dazu erfahren Sie in Kapitel 13, »Datenbanken«, Kapitel 17, »Weitere Datei- und Datenformate«, und Kapitel 18, »Webseitenerstellung mit (X)HTML und CSS«)
`multipart/mixed`	»Umschlag« für mehrere MIME-Unterabschnitte

Tabelle 4.23 Einige gängige MIME-Datentypen

Typ	Beschreibung
`multipart/alternative`	»Umschlag« für denselben Inhalt in mehreren Alternativformaten
`multipart/form-data`	»Umschlag« für POST-Formulardaten einschließlich Datei-Uploads (siehe Kapitel 18, »Webseitenerstellung mit (X)HTML und CSS«)

Tabelle 4.23 Einige gängige MIME-Datentypen

Die drei letzten Typen in der Tabelle machen MIME besonders interessant: Ein MIME-Dokument vom Typ `multipart/mixed` kann beliebig viele Teile enthalten, die jeweils einen vollständigen MIME-Header besitzen und wiederum beliebige MIME-Types aufweisen können. Mithilfe dieser Technik werden in modernen E-Mail-Programmen Attachments (Dateianhänge) der Mail hinzugefügt. Dagegen wird ein Abschnitt vom Typ `multipart/alternative` eingesetzt, um denselben Inhalt in verschiedenen alternativen Darstellungsformen zu umschließen, beispielsweise ein Bild im GIF- und im PNG-Format oder (wahrscheinlich die häufigste Anwendung) den Text einer E-Mail-Nachricht einmal im einfachen Text- und einmal im HTML-Format. Abbildung 4.5 zeigt beispielhaft, wie eine MIME-Nachricht mit zwei Dateianhängen aufgebaut sein könnte.

Dasselbe Verfahren – in diesem Fall mit dem MIME-Type `multipart/form-data` – kommt bei Webformularen zum Einsatz, wenn diese außer gewöhnlichen Auswahl- oder Eingabefeldern auch Datei-Uploads unterstützen.

Der `Content-Transfer-Encoding`-Header gibt dem Empfängerclient einen Hinweis, auf welche Weise die ankommenden Daten zu interpretieren sind. Häufig verwendete Werte sind etwa folgende:

- `7bit`: Keine Codierung; eignet sich für 7-Bit-ASCII (englischer Text). Automatischer Zeilenumbruch nach spätestens 1.000 Zeichen.
- `8bit`: Keine Codierung; eignet sich für 8-Bit-Text (internationaler Text). Ebenfalls automatischer Zeilenumbruch nach spätestens 1.000 Zeichen.
- `binary`: Keine Codierung, es erfolgt kein automatischer Zeilenumbruch.
- `quoted-printable`: Spezielle Codierung von Sonderzeichen, die über 7-Bit-ASCII hinausgehen. Beispiel: »größer« wird zu »gr=FC=DFer« (die Codierung besteht aus einem Gleichheitszeichen, gefolgt von hexadezimalem Zeichencode).
- `base64`: Bevorzugte Codierung für Binärdateien. Ein spezieller Algorithmus packt die Daten 7-Bit-kompatibel um. Dieses Format ist auch dann nicht von Menschen lesbar, wenn Klartext codiert wird – aus »Hallo Welt!« wird beispielsweise `SGFsbG8gV2VsdCE=`.

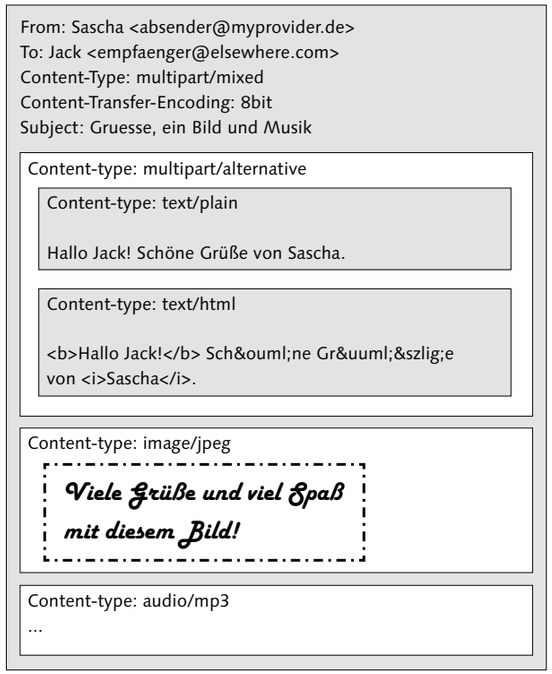


Abbildung 4.5 Beispiel für eine E-Mail im MIME-Multipart-Format

Die Codierungsformen `quoted-printable` und `base64` besitzen den Vorteil, dass die Mailnachricht formal kompatibel mit RFC 822 bleibt und entsprechend auch über alte Mailserver versandt und empfangen werden kann.

Der E-Mail-Empfang über einen POP3-Server erfolgt auf textbasierte Art, ähnlich wie bei SMTP. Der Server kommuniziert über den TCP-Port 110. Die Beschreibung von POP3 steht in RFC 1939. Zur Verdeutlichung hier wiederum eine Telnet-basierte Konversation mit einem (unkenntlich gemachten) POP3-Server:

```
# telnet pop.myprovider.de pop3
+OK POP3 server ready
USER absender
+OK
PASS XXXXX
+OK
LIST
1898
.
RETR 1
+OK 953 octets
```



```
Return-path: <empfanger@elsewhere.com>
Envelope-to: absender@myprovider.de
Delivery-date: Fri, 24 Apr 2015 03:08:24 +0100
Received: from [207.18.31.76] (helo=smtp.elsewhere.com)
 by mxng13.myprovider.de with esmtp (Exim 3.35 #1)
 id 18QeUU-000270-00
 for absender@myprovider.de; Fri, 24 Apr 2015
 03:08:18 +0100
Received: from box (xdsl-202-21-109-17.elsewhere.com
[202.21.109.17])
by smtp.elsewhere.com (Postfix) with SMTP id
CA500866C1
for <absender@myprovider.de>; Fri, 24 Apr 2015
03:08:14 +0100 (MET)
Message-ID: <001901c2aaf2$31ce81e0$0200a8c0@box>
From: "Jack" <empfaenger@elsewhere.com>
To: "Sascha" <absender@provider.de>
Subject: Gruesse
Date: Fri, 24 Apr 2015 03:14:30 +0100
MIME-Version: 1.0
Content-Type: text/plain;
charset="iso-8859-1"
Content-Transfer-Encoding: quoted-printable

Hi!
Wie geht's?
Alles klar?
Ciao.
.
DELE 1
+OK
QUIT
+OK
```

In dieser Sitzung kommen die folgenden POP3-Befehle zum Einsatz:

- `USER`: Angabe des Benutzernamens für die Anmeldung
- `PASS`: Angabe des Passworts für die Anmeldung
- `LIST`: nummerierte Liste der verfügbaren E-Mails mit der jeweiligen Länge in Bytes
- `RETR`: E-Mail mit der angegebenen Nummer empfangen
- `DELE`: E-Mail mit der angegebenen Nummer vom Server löschen
- `QUIT`: Sitzung beenden

Die meisten E-Mail-Programme führen `RETR` und `DELE` standardmäßig unmittelbar nacheinander durch, die Nachrichten verbleiben also in der Regel nicht auf dem Server. Bei IMAP-Servern ist es dagegen meist anders: Der besondere Vorteil des IMAP-Protokolls besteht darin, dass auf dem Mailserver selbst verschiedene Ordner eingerichtet werden können, um Mails dort zu verwalten. Auf diese Weise erleichtert IMAP die E-Mail-Verwaltung für mobile Benutzer. Die aktuelle Version von IMAP ist das in RFC 2060 dargestellte IMAP4. Ein IMAP-Server funktioniert ähnlich wie ein POP3-Server, verwendet allerdings den TCP-Port 142.

Eine weitere beliebte Form der E-Mail-Nutzung sind webbasierte Freemail-Dienste wie GMX oder Hotmail. Dabei handelt es sich um gewöhnliche POP-SMTP-Kombinationen, die über eine Website mit persönlicher Anmeldung zugänglich gemacht werden. Das Programm, das mit den E-Mail-Servern kommuniziert, läuft auf dem Webserver und wird dem Kunden per Browser zur Verfügung gestellt.

Newsgroups

Newsgroups als virtuelle »Schwarze Bretter« wurden 1979 eingeführt, um Gruppendiskussionen zwischen der Duke University und der University of North Carolina zu ermöglichen. Das System entwickelte sich im Laufe der Jahre zum weltweiten *Usenet* mit mehreren Zehntausend Newsgroups.

Das Usenet besteht aus einem losen Verbund von weltweit verteilten Newsservern, die das in RFC 977 festgelegte Protokoll NNTP (Network News Transport Protocol) ausführen. Wer einen Artikel in einer Newsgroup veröffentlichen möchte, sendet diesen an den TCP-Port 119 des nächstgelegenen Newsservers (in der Regel den des eigenen Providers); innerhalb von spätestens 24 Stunden dürfte die Nachricht jeden Newsserver weltweit, der die betreffende Newsgroup bereitstellt, erreicht haben.

Um Nachrichten in einer bestimmten Newsgroup zu lesen, müssen Sie diese abonnieren: Sie verbinden sich über Ihren Newsreader mit einem Newsserver, der diese Group anbietet, und aktivieren das Abonnement für die Group. Bei jedem Start Ihres Newsreaders werden nun zunächst die Header-Daten aller aktuellen Nachrichten in der Group heruntergeladen und angezeigt. Sobald Sie eine solche Kopfzeile anklicken, wird der eigentliche Inhalt der Nachricht geladen.

Eine Newsgroup-Nachricht ist ein RFC-822-kompatibles Dokument. Allerdings definiert das NNTP-Protokoll einige spezielle Header-Felder, die erforderlich sind, um die Nachrichten den verschiedenen Groups zuzuordnen und ihre Position in einem Thread, einem Diskussionsstrang, festzulegen. Die meisten Newsserver beherrschen im Übrigen die MIME-Erweiterungen, allerdings sind MIME-basierte HTML- oder gar Multimedia-Nachrichten in traditionellen Usenet-Newsgroups verpönt; es ist üblich, nur reinen Text zu verwenden.

Zu einer Newsgroup-Nachricht gehören die folgenden wichtigen Header-Felder (abgesehen von denjenigen, die bereits beim Thema SMTP beschrieben wurden):

- `Article`: Eine ID des Beitrags, bestehend aus einer Nummer und dem Namen der Newsgroup, in die der Beitrag gepostet wird. Diese Nummern können auf verschiedenen Newsservern unterschiedlich sein, da sie in der Reihenfolge vergeben werden, in der Nachrichten eintreffen.
- `Message-ID`: Eine unveränderliche und weltweit einmalige ID für diesen einen Beitrag über alle Newsgroups hinweg. Ermöglicht das quellgenaue Zitieren und Verlinken eines Postings.
- `Referrers`: Die Message-ID des ursprünglichen Newsgroup-Beitrags, auf den geantwortet wurde. Anhand dieses Feldes wird die Nachricht in einen Thread einsortiert.

Das Usenet besteht aus einer Reihe von Newsgroups mit hierarchisch gegliederten Namen. Ganz links steht dabei die allgemeine Oberkategorie, die nach rechts immer weiter spezialisiert wird (ähnlich wie in einem Dateisystem und andersherum als bei Domainnamen). Die Hauptkategorien sind beispielsweise *comp* für computerbezogene Themen, *rec* (*recreation*) für Freizeit, Sport und Spiel, *soc* für gesellschaftspolitische Themen oder *sci* (*science*) für die Welt der Naturwissenschaft und Technik. Innerhalb dieser Kategorien bestehen einzelne Groups wie *comp.lang.perl.modules* (Computer – Programmiersprachen – Perl – Module), *rec.autos.vw* (Freizeit – Autos – Volkswagen), *soc.religion.islam* (Gesellschaft – Religion – Islam) oder *sci.crypt.random-numbers* (Wissenschaft – Kryptografie – Zufallsgeneratoren). Im Übrigen gibt es Hauptkategorien, die auf ein bestimmtes Länderkürzel lauten, für Newsgroups, in denen eine bestimmte Sprache gesprochen wird (etwa die *de.**-Hierarchie für deutschsprachige Groups).

Da das traditionelle Usenet recht schwerfällig und konservativ ist, bedarf es beinahe endloser Diskussionen, bevor unter einer der klassischen Hauptkategorien eine neue Newsgroup eingerichtet wird. Aus diesem Grund wurde die spezielle *alt.**-Hierarchie eingeführt, unter der jeder neue Groups anlegen kann. Diese Hierarchie gehört nicht zum eigentlichen Usenet und enthält die bizarrsten, aber auch einige der interessantesten Newsgroups.

Die Beliebtheit der Newsgroups unter den Internetnutzern scheint allerdings bereits vor einigen Jahren ihren Zenit überschritten zu haben. Die größte Konkurrenz bilden heutzutage webbasierte Foren, in denen über speziellere Themen diskutiert wird und die nicht ganz so strenge Verhaltensregeln besitzen wie die Newsgroups oder insbesondere das klassische Usenet. Dennoch besteht die Möglichkeit, ohne spezielles News-Programm auf beliebige Newsgroups zuzugreifen: Der Suchmaschinenbetreiber Google kaufte vor einigen Jahren den webbasierten News-Dienst Deja.com und dessen Usenet-Archiv auf. Unter *http://groups.google.com* können Sie in fast allen jemals geschriebenen Newsgroup-Beiträgen recherchieren und sich darüber hinaus anmelden, um aktiv an Newsgroup-Diskussionen teilzunehmen.

Das World Wide Web

Das Web ist heute die dominierende Internetanwendung überhaupt, und zwar in einem solchen Maße, dass viele Leute das WWW mit dem gesamten Internet gleichsetzen. Wer auf das Web zugreifen möchte, verwendet dazu eine spezielle Clientsoftware, den sogenannten *Webbrowser*. Nach der Eingabe einer Dokumentadresse stellt der Browser eine TCP-Verbindung zu dem gewünschten Webserver her und fordert über das HTTP-Protokoll das gewünschte Dokument an. Das Dokument ist üblicherweise in der Seitenbeschreibungssprache HTML verfasst (unterstützt durch CSS und JavaScript), die der Browser interpretiert und in eine auf bestimmte Art und Weise formatierte Webseite umwandelt.

Eine solche Seite kann außerdem Verweise auf eingebettete Dateien wie Bilder oder Multimedia enthalten, die der Browser auf dieselbe Art anfordert wie das HTML-Dokument selbst und an der passenden Stelle auf der Seite platziert. Ein weiteres wichtiges Element von Webseiten sind die Hyperlinks, anklickbare Verknüpfungen zu anderen Dokumenten. Wenn Sie einen Hyperlink aktivieren, wird die entsprechende Datei angefordert und in den Browser geladen.

Damit das Web funktionieren kann, wirken einige wesentliche Konzepte zusammen:

- Das Anwendungsprotokoll HTTP, über das Dokumente beim Server angefordert und von diesem ausgeliefert werden. Die aktuelle Version des Protokolls, HTTP 1.1, wird in RFC 2616 beschrieben; in Kapitel 14, »Server für Webanwendungen«, erhalten Sie genauere Informationen darüber.
- Ein spezielles Format für Dokumentadressen, das als *Uniform Resource Locator* (URL) bezeichnet wird und dessen Definition sich in RFC 1738 befindet. Die URL wird beispielsweise in die Adresszeile des Browsers eingegeben; sie sieht zum Beispiel so aus: *http://www.rheinwerk-verlag.de/*.
- Die Seitenbeschreibungssprache HTML, in der Hypertext-Dokumente für das WWW geschrieben werden. Neuere HTML-Versionen werden nicht mehr in RFCs definiert; eine genaue Beschreibung von HTML finden Sie in Kapitel 18, »Webseitenerstellung mit (X)HTML und CSS«.

4.7 Übungsaufgaben

Im Folgenden ist jeweils genau eine Antwort richtig.

1. Was ist keine Aufgabe eines Netzwerks?
   - Kommunikation zwischen seinen Benutzern
   - gemeinsame Stromversorgung mehrerer Rechner
   - Austausch von Daten
   - verteilte Anwendungen

2. Welches der folgenden Merkmale gehört nicht zwangsläufig zur paketvermittelten Datenübertragung?
   - Absender- und Empfängeradresse in jedem Paket
   - die Unterteilung der Daten in kleinere Einheiten
   - ein Bestätigungsverfahren, das die Datenauslieferung garantiert
   - die Fähigkeit zur Weiterleitung der Datenpakete über verschiedene Wege
3. Welcher bis heute bedeutende Internetdienst wurde 1972 erfunden?
   - CGI
   - World Wide Web
   - Newsgroups
   - E-Mail
4. Wie werden die Standards des Internets dokumentiert?
   - in Patentschriften
   - in IEEE-Drafts
   - in Diplom- und Doktorarbeiten
   - in öffentlich verfügbaren RFC-Dokumenten
5. Welche Geräte wurden als erste zur Datenfernübertragung verwendet?
   - Funkgeräte
   - Akustikkoppler
   - Telegraphen
   - Modems
6. Was war die entscheidende Neuerung am World Wide Web?
   - die Einführung von Hypertext
   - wissenschaftliche Internetanwendung
   - die Verwendung eines textbasierten Kommunikationsprotokolls
   - die Anwendung von Hypertext über ein Netzwerk
7. Wie heißt das Protokoll, das für die WWW-Kommunikation verwendet wird?
   - LWP
   - WWWP
   - HTTP
   - HTML
8. Welche OSI-Schicht ist Nummer 3?
   - Bit-Übertragungsschicht
   - Vermittlungsschicht

- Sicherungsschicht
- Transportschicht

9. Welche Nummer hat die OSI-Darstellungsschicht?
   - 6
   - 5
   - 7
   - 4
10. Was ist eine Aufgabe der OSI-Sicherungsschicht?
    - Datenstromverschlüsselung
    - Erzeugung von Datenpaketen
    - Steuerung des Zugriffs auf das Übertragungsmedium
    - Routing
11. Welches ist keine Schicht im Internetschichtenmodell (DoD- oder DDN-Modell)?
    - Netzzugangsschicht
    - Internetschicht
    - Sitzungsschicht
    - Anwendungsschicht
12. Welcher OSI-Schicht entspricht die Internetschicht des Internetschichtenmodells in etwa?
    - Sicherungsschicht
    - Vermittlungsschicht
    - Transportschicht
    - Sitzungsschicht
13. Welches der folgenden Protokolle arbeitet auf der Transportschicht des Internetschichtenmodells?
    - FTP
    - TCP
    - IP
    - ARP
14. Welche Netzwerkart hat die größte Reichweite?
    - MAN
    - WAN
    - GAN
    - LAN

15. Bei welcher Netzwerktopologie sind alle Stationen mit einem zentralen Verteiler verbunden?
    - Baum
    - Stern
    - Ring
    - Bus
16. Welche der folgenden Aussagen über Server ist zutreffend?
    - Ein Server ist ein spezieller, sehr teurer Computer.
    - Ein Server ist ein Programm, das eine bestimmte Dienstleistung bereitstellt.
    - Ein Server muss stets in einen 19-Zoll-Schrank montiert werden.
    - Ein Server ist ein Programm, das eine Benutzeroberfläche für Netzwerkdienste bereitstellt.
17. Welche der folgenden Aussagen über Clients ist zutreffend?
    - Ein Client ist ein Programm, das eine bestimmte Dienstleistung bereitstellt.
    - Client ist lediglich eine andere Bezeichnung für einen Desktop-PC.
    - Ein Client ist ein Programm, das eine Benutzeroberfläche für Netzwerkdienste bereitstellt.
    - Client ist lediglich eine andere Bezeichnung für einen Browser.
18. Wie nennt man es, wenn die Aufgaben eines Webservers auf mehrere physikalische Rechner verteilt werden?
    - Web Caching
    - Proxy Service
    - Load Balancing
    - Round Robin
19. Welcher der folgenden Server ist kein klassischer Netzwerkserver?
    - Mailserver
    - X Window-Server
    - File-Server
    - Print-Server
20. Welcher IEEE-802-Standard beschreibt drahtlose Netzwerke?
    - IEEE 802.3
    - IEEE 802.5
    - IEEE 802.11
    - IEEE 802.15

21. Wodurch wird eine bestimmte Ethernet-Karte eindeutig gekennzeichnet?
    - durch die MAC-Adresse
    - durch die IP-Adresse
    - durch den Domainnamen
    - Die Ethernet-Karte selbst besitzt kein eindeutiges Identifikationsmerkmal.
22. Was bedeutet der "Promiscuous Mode" bei einer Ethernet-Karte?
    - Sie erhält mehrere IP-Adressen.
    - Sie dient als DHCP-Provider.
    - Sie empfängt alle Datenpakete aus ihrem Netzsegment.
    - Sie wird IP-Multicast-fähig.
23. Was geschieht beim CSMA/CD-Verfahren, wenn eine Datenkollision auftritt?
    - Aufgrund der Funktionsweise kann es in CSMA/CD-Netzen nicht zu Kollisionen kommen.
    - Der Verteiler (Hub oder Switch) regelt, welches Gerät als Erstes erneut senden darf.
    - Der Verteiler (Hub oder Switch) versendet nacheinander Kopien der kollidierten Pakete.
    - Jedes an der Kollision beteiligte Gerät sendet nach einer individuellen Zufallswartezeit erneut.
24. Wie setzt sich die Bezeichnung 10 Base 2 zusammen?
    - 10 mm dickes Kabel mit maximal 2 MBit/s
    - maximal 10 MBit/s, maximal 200 m langes Netzsegment
    - maximal 10 MBit/s, mindestens 2 m Abstand zwischen zwei Stationen
    - höchstens zehn Stationen pro maximal 200 m langem Netzsegment
25. Welcher Steckertyp wird für Twisted-Pair-Ethernet, aber auch etwa für ISDN verwendet?
    - RG-58
    - RJ-45
    - RJ-11
    - Cinch
26. Welche UTP-Kabel-Kategorie wird mindestens für Fast Ethernet benötigt?
    - 3
    - 4
    - 5
    - 6

27. Was ist der entscheidende Unterschied zwischen einem Hub und einem Switch?
    - Für Fast Ethernet ist auf jeden Fall ein Switch notwendig.
    - Nur ein Switch kann Ethernet-Schnittstellen unterschiedlicher Geschwindigkeiten miteinander verbinden.
    - Ein Switch besitzt mehr Ports als ein Hub.
    - Ein Switch verhindert Datenkollisionen durch direkte Verbindungen zwischen den Stationen.
28. Welche maximale Datenübertragungsrate unterstützt Fast Ethernet?
    - 10 MBit/s
    - 52 MBit/s
    - 100 MBit/s
    - 480 MBit/s
29. Zu welchem Zweck wurde das im WLAN-Bereich übliche Frequency-Hopping-Verfahren ursprünglich entwickelt?
    - Mobiltelefonie
    - Satellitenkommunikation
    - Flugnavigation
    - Torpedofernsteuerung
30. Welches Netzzugangsverfahren wird für WLAN verwendet?
    - CSMA/CD
    - Token Passing
    - CSMA/CA
    - PPP
31. Welchen Sicherheitsstandard für WLAN sollte man mindestens verwenden?
    - WEP
    - WPA2
    - WPA1
    - SSL
32. Welche technische Besonderheit sorgt für die hohen DSL-Übertragungsraten über normale Telefonleitungen?
    - Datenkomprimierung
    - Kanalbündelung
    - hochfrequente Signale
    - DSL verwendet nie normale Telefonleitungen, sondern Lichtwellenleiter.

33. Was ist der entscheidende Unterschied zwischen ADSL und SDSL?
    - SDSL bietet ein anderes Abrechnungsmodell, das für Geschäftskunden interessanter ist.
    - SDSL ist schneller als ADSL.
    - ADSL kann gewöhnliche Telefonleitungen verwenden, SDSL nicht.
    - SDSL bietet identische Datenübertragungsraten in beide Richtungen, ADSL ist im Upload langsamer als im Download.
34. Welche der folgenden Angaben ist keine gültige IPv4-Adresse?
    - 197.17.8.21
    - 212.211.210.209
    - 31.310.30.13
    - 0.7.8.9
35. Mit welchem Bit-Muster beginnt eine IP-Adresse der historischen Klasse C?
    - 0
    - 10
    - 110
    - 1110
36. In welchem Bereich liegt das erste Byte einer IP-Adresse der Klasse B?
    - 64 bis 127
    - 128 bis 191
    - 192 bis 223
    - 128 bis 159
37. Welche der folgenden IP-Adressen ist keine private, frei verfügbare Adresse gemäß RFC 1918?
    - 192.168.27.11
    - 172.21.47.11
    - 10.0.8.15
    - 172.47.11.12
38. Zu welchem der folgenden Netze gehört die IP-Adresse 196.17.8.92 nicht?
    - 196.17.0.0/16
    - 196.17.0.0/17
    - 196.17.8.0/26
    - 196.17.8.0/25

39. Das Netz 156.19.0.0/16 wird per CIDR in vier gleich große Netze unterteilt. Welches der folgenden ist eines der neuen Teilnetze?
    - 156.19.0.0/20
    - 156.19.16.0/18
    - 156.19.64.0/18
    - 156.19.128.0/19
40. Wie viele Hosts kann jedes Teilnetz maximal enthalten, wenn ein IPv4-Netz mit der Teilnetzmaske 255.255.255.0 in vier Teile unterteilt wird?
    - 62
    - 128
    - 127
    - 64
41. Was ist der Vorteil von VLSM gegenüber CIDR?
    - Die Teilnetzmaske kann auch zwischen ganzen Bytes der Adressen verlaufen.
    - Mehrere Netze lassen sich zu einem größeren Netz zusammenfassen (Supernetting).
    - Ein Netz lässt sich in Teilnetze unterschiedlicher Größe unterteilen.
    - Es gibt keinen Vorteil; CIDR ist flexibler als VLSM.
42. Wie lang ist der IPv4-Header mindestens?
    - 16 Byte
    - 20 Byte
    - 24 Byte
    - 32 Byte
43. Welche Bedeutung besitzt der TTL-Wert (Time To Live) im IP-Datagramm?
    - Er gibt die Uhrzeit an, wann das Datagramm erzeugt wurde.
    - Er gibt die Uhrzeit an, bis zu der das Datagramm bestehen wird.
    - Er zählt die Sekunden, die das Datagramm bereits existiert.
    - Er zählt die Hops, die das Datagramm erlebt, bis 0 herunter.
44. Zu welchem Problem können unterschiedliche MTUs verschiedener Netzwerkschnittstellen bei IP-Datagrammen führen?
    - Die Datagramme können nicht weitergeleitet werden.
    - Die Datagramme werden fragmentiert.
    - Die Datagramme müssen erneut gesendet werden.
    - Die Datagramme werden langsamer transportiert.

45. Was ist die genaue Definition eines Default Gateways?
    - der Router ins nächstgelegene Netz
    - der Router ins Internet
    - der Router für Verbindungen in alle Netze, für die kein separater Router existiert
    - der Router in einem Netz, in dem kein weiterer Router existiert
46. Welche Adresse ist vom Netzwerk 156.81.0.0/19 aus nur über einen Router zu erreichen?
    - 156.81.9.18
    - 156.81.18.9
    - 156.81.81.9
    - 156.81.0.9
47. Das Netzwerk 152.17.0.0/17 ist über drei Router mit anderen Netzen verbunden: r1 für das Netzwerk 152.17.128.0/17, r2 für 152.18.0.0/16 und r3 für alle anderen Netze. Welcher Router wird für eine Verbindung zur Adresse 152.18.210.22 verwendet?
    - r1
    - r2
    - r3
    - Keiner; die Adresse befindet sich im aktuellen Teilnetz.
48. Was wird im Zusammenhang mit IP-Routing als autonomes System bezeichnet?
    - ein Rechner, der nicht an ein Netzwerk angeschlossen ist
    - ein lokales Netzwerk ohne Verbindung zu anderen Teilnetzen
    - die Gesamtheit aller Netzwerke eines Betreibers
    - ein allein stehender Router
49. Zu welchem Zweck wird ein internes Routing-Protokoll eingesetzt?
    - für das Routing innerhalb eines Teilnetzes
    - für das Routing innerhalb eines autonomen Systems
    - für das Routing innerhalb eines Netzes ohne Außenverbindung
    - für das Routing zwischen mehreren virtuellen Netzwerkschnittstellen auf einem einzelnen Rechner
50. Welches der folgenden Routing-Protokolle ist ein externes Routing-Protokoll?
    - RIP
    - OSPF
    - BGP
    - keines der genannten

51. Welcher TCP/IP-Dienst ermöglicht die automatische Vergabe von IP-Adressen?
    - BOOTP
    - NAT
    - DHCP
    - ARP
52. Wie lang ist eine IPv6-Adresse?
    - 64 Bit
    - 128 Bit
    - 32 Bit
    - variabel
53. Welches Transportprotokoll verwendet das Verbindungstestprogramm Ping?
    - UDP
    - ICMP
    - TCP
    - keins, sondern RawIP
54. Welche Flags sind (nacheinander) bei den drei Datenpaketen gesetzt, die den Drei-Wege-Handshake bei TCP bilden?
    - SYN, ACK, SYN
    - SYN, SYN, ACK
    - SYN, SYN/ACK, ACK
    - SYN, ACK, SYN/ACK
55. Welche TCP-Portnummern sind »Well-known Ports« – festgelegte Portnummern für Serverdienste?
    - 0 bis 255
    - 0 bis 1023
    - variiert je nach Betriebssystem
    - 0 bis 32767
56. Was ist der Vorteil von UDP gegenüber TCP als Transportprotokoll?
    - schnellere Übertragung auf Kosten der Zuverlässigkeit
    - schnellere Übertragung durch höhere Übertragungsraten
    - sichere Übertragung durch Verschlüsselung
    - Möglichkeit der Übertragung in Nicht-IP-Netze
57. Welchen TCP-Port verwendet ein Webserver standardmäßig?
    - 21
    - 53

- 80
- 110

58. Welcher Serverdienst verwendet standardmäßig den TCP-Port 23?
    - echo
    - ftp
    - smtp
    - telnet
59. Aus welcher Datei können auf einem Unix-System Adressauflösungen von Hostnamen gelesen werden?
    - /var/hosts.txt
    - /etc/services.txt
    - /etc/hosts
    - /var/addresses
60. Welche der folgenden Aussagen über DNS ist zutreffend?
    - Das System besteht aus einer Textdatei, die bei Änderungen auf jeden Nameserver kopiert wird.
    - Ein DNS-Administrator kann Subdomains seiner Zone entweder selbst verwalten oder als untergeordnete Zonen delegieren.
    - Ein Webserver muss den Hostnamen www erhalten, damit er funktioniert.
    - Jeder Internetteilnehmer benötigt einen eigenen Nameserver.
61. Welche der folgenden Generic Top-Level-Domains gibt es nicht?
    - .com
    - .net
    - .doc
    - .info
62. Welche Länder-Top-Level-Domain verwendet Großbritannien?
    - .bn
    - .uk
    - mehrere: .en für England, .sc für Schottland, .wa für Wales und .ni für Nordirland
    - .bi für die gesamten Britischen Inseln
63. Welchen Vorteil besitzt SSH gegenüber Telnet?
    - sichere Datenübertragung durch Verschlüsselung
    - SSH verwendet TCP, Telnet dagegen UDP.
    - Telnet-Server funktionieren nur unter Unix, SSH-Server auch unter Windows.
    - Die Texteingabe ist bei SSH komfortabler.

64. Welchen der folgenden FTP-Befehle gibt es nicht?
    - `binary`
    - `get`
    - `decimal`
    - `put`
65. Für welche Dateien braucht bei der FTP-Übertragung nicht der Binärmodus verwendet zu werden?
    - Bilddateien
    - ausführbare Programme
    - HTML-Dokumente
    - Videodateien
66. Wie wird einem SMTP-Server mitgeteilt, dass der eigentliche E-Mail-Text beendet ist?
    - durch eine Leerzeile
    - durch den Befehl `END`
    - durch ein EOF (entspricht der Tastenkombination Strg + D)
    - durch einen einzelnen Punkt in einer Zeile
67. Welcher MIME-Type wird für ein JPEG-Bild verwendet?
    - application/x-jpeg
    - image/jpeg
    - image/jpg
    - application/image-jpg
68. Welcher RFC-822-Header gibt den MIME-Type an?
    - Content-type
    - Content-transfer-encoding
    - Mime-type
    - File-type
69. In welcher E-Mail-Codierung wird »Köln« als »K=FCln« wiedergegeben etc.?
    - 7bit
    - quoted-printable
    - base64
    - binary
70. Was ist der Vorteil von IMAP gegenüber POP3?
    - IMAP-Server bieten mehr Speicherplatz pro Postfach.
    - Auf einem IMAP-Server kann ein Benutzer mehrere E-Mail-Adressen haben.

- Auf IMAP-Servern werden die Daten länger aufbewahrt.
- Auf einem IMAP-Server kann ein Benutzer Ordner zur E-Mail-Verwaltung anlegen.

71. Welche Header-Zeile ist eine NNTP-Ergänzung zu den klassischen RFC-822-Headern?
    - Subject
    - From
    - Message-ID
    - Content-type

Kapitel 9
Grundlagen der Programmierung

There are 10 kinds of people: Those who understand binary notation and those who do not.
– Anonym

Ein Computer ist immer nur so nützlich wie die verfügbare Software. Heutzutage gibt es vorgefertigte Programme für beinahe jeden Verwendungszweck; viele von ihnen lernen Sie in den folgenden Kapiteln dieses Buches kennen. Dennoch gibt es – neben der Tatsache, dass Ihre Ausbildung es vielleicht verlangt – eine Reihe guter Gründe, Software selbst zu entwickeln:

- Trotz der immensen Fülle an Standardsoftware ist manchmal nicht genau das passende Programm verfügbar.
- Manche Software ist so teuer, dass sie das Budget von Privatpersonen oder auch kleineren Unternehmen bei Weitem übersteigt.
- Open-Source-Software löst das Preisproblem in vielen Fällen, aber gerade Open-Source-Software lässt sich oft mithilfe eingebauter Programmierschnittstellen noch besser an die eigenen Bedürfnisse anpassen.
- Es gibt kaum eine verlässlichere Möglichkeit, die Funktionsweise eines Computers zu verstehen, als ihn zu programmieren.

Um Software entwickeln zu können, benötigen Sie Programmierkenntnisse. Je nach Art und Einsatzgebiet von Programmen sind die verschiedenen Programmiersprachen unterschiedlich gut geeignet. Beispielsweise sind einige Sprachen besonders auf Geschwindigkeit optimiert, andere dagegen sind benutzerfreundlicher und leichter zu erlernen, wieder andere sind nur für spezielle Arten von Programmen geeignet oder funktionieren nur innerhalb eines bestimmten Anwendungsprogramms.

In diesem Buch beschäftigen sich mehrere Kapitel oder Teile von ihnen mit verschiedenen Aspekten der Programmierung:

- In Kapitel 1, »Einführung«, finden Sie einen kurzen Abriss über die Geschichte der Programmiersprachen.
- In Kapitel 2, »Mathematische und technische Grundlagen«, wird ein virtueller Prozessor erläutert, der über eine einfache Maschinensprache (genauer gesagt, eine Assembler-Sprache) mit wenigen Instruktionen verfügt.

- Kapitel 6, »Windows«, bietet einen Einstieg in die Programmierung mit der Windows PowerShell.
- Kapitel 7, »Linux«, enthält einen kurzen Abschnitt über Shell-Skripte.
- Das vorliegende Kapitel bietet einen allgemeinen Einstieg in die Programmierung. Anhand dreier verschiedener gängiger Sprachen werden die wichtigsten Komponenten von Computerprogrammen vorgestellt.
- In Kapitel 10, »Konzepte der Programmierung«, werden zahlreiche fortgeschrittene Programmiertechniken behandelt. Unter anderem werden Sie in die Programmierung grafischer Oberflächen eingeführt und lernen einige Grundlagen der System- und Netzwerkprogrammierung kennen.
- Kapitel 11, »Mobile Development«, führt in die Entwicklung von Mobile Apps ein; hier werden die Programmiersprache Swift für iOS und die Java-Entwicklung mit Googles SDK für Android behandelt.
- Kapitel 12, »Software-Engineering«, geht den wichtigen Schritt von der einzelnen Programmdatei zum Softwareprojekt. Sie lernen verschiedene nützliche Techniken kennen, um größere Programme zu planen und den Überblick in ihnen zu behalten.
- In Kapitel 13, »Datenbanken«, wird am Ende kurz auf die Verwendung von Schnittstellen zur Programmierung datenbankgestützter Anwendungen eingegangen.
- In Kapitel 16, »XML«, erhalten Sie einen kurzen Überblick über wichtige Schnittstellen zur XML-Programmierung.
- In Kapitel 19, »Webserveranwendungen«, wird die Programmiersprache PHP 5 vorgestellt.
- In Kapitel 20, »JavaScript und Ajax«, wird die Skriptsprache JavaScript eingeführt, mit der sich Webseiten im Browser »zum Leben erwecken« lassen. Sie lernen die Grundlagen der Sprache, das dynamische Nachladen von Inhalten mit Ajax und die komfortable JavaScript-Bibliothek jQuery kennen.

Das vorliegende Kapitel dient als grundlegendes Tutorial für fortgeschrittene Computeranwender, die bisher noch nicht programmiert haben. Es ist aber auch nützlich, wenn Sie grundsätzlich Programmierkenntnisse haben, aber eine oder mehrere der vorgestellten Sprachen noch nicht kennen. Im Einzelnen lernen Sie in diesem Kapitel zwei Compiler- und eine Skriptsprache kennen. Eine der Sprachen ist rein imperativ, eine objektorientiert, und eine ist eine Multiparadigmen-Sprache, das heißt, sie bietet Aspekte beider – und weiterer – Programmiersprachenarten (siehe »Entwicklung der Programmiersprachen« in Kapitel 1, »Einführung«). Im Einzelnen handelt es sich um folgende Sprachen:

- C – imperative Compilersprache
- Java – objektorientierte Compilersprache
- Python – Multiparadigmen-Skriptsprache

Warum Python?

In früheren Auflagen dieses Buches wurden anstelle von Python die beiden Sprachen Perl und Ruby behandelt. Da Perl etwas in die Jahre gekommen ist und Ruby nicht annähernd die Verbreitung von Python genießt, habe ich sie in der vorliegenden Auflage ersetzt. Die Abschnitte zu Perl und Ruby finden Sie bei Interesse im Web unter *buecher.lingoworld.de/fachinfo/perl* beziehungsweise *buecher.lingoworld.de/fachinfo/ruby*; sie wurden allerdings nicht an den jeweils neuesten Stand dieser Sprachen angepasst.

Der neue Abschnitt zur Programmiersprache Python ist bei Weitem der längste in diesem Kapitel, da sich viele Ansätze der Programmierung anhand dieser Sprache besonders gut erklären lassen. Zudem verfügt Python über die sogenannte *interaktive Shell*, die eingegebene Anweisungen sofort ausführt und das Ergebnis ausgibt – ein ideales Hilfsmittel, um das Programmieren zu erlernen.

9.1 Die Programmiersprache C

In gewisser Weise ist es ein wenig gewagt, einen Programmierkurs mit der Sprache *C* zu beginnen: Da diese Sprache sehr große Freiheiten bezüglich der Strukturierung von Programmen erlaubt, besteht die Gefahr, sich von Anfang an einen »schlampigen« Programmierstil anzugewöhnen. Andererseits ist C die älteste Programmiersprache, die noch heute von vielen Entwicklern genutzt wird. Außerdem hat die Syntax von C eine Vielzahl neuerer Sprachen stark beeinflusst – die Mehrheit aller in diesem Buch erwähnten Sprachen benutzt die grundlegenden Konstrukte von C.

Die Programmiersprache C wurde ab 1971 von Dennis Ritchie und Brian Kernighan entwickelt, um das Betriebssystem Unix neu zu implementieren. Aus diesem Grund sind Unix und C untrennbar miteinander verbunden; dennoch sind C-Compiler für fast jedes Betriebssystem verfügbar. Seit 1983 wurde eine Neufassung von C als ANSI- und später auch ISO-Standard entwickelt, die nach ihrem endgültigen Veröffentlichungsjahr C90 heißt. 1999 wurde schließlich die aktuellste Version C99 eingeführt, die ein paar weitere Freiheiten erlaubt.

Wie bereits erwähnt wurde, ist C eine Compilersprache. Ein C-Programm wird also zuerst vollständig in die Maschinensprache des jeweiligen Rechners (mit ein paar Betriebssystem-Bibliotheksaufrufen) übersetzt und dann ausgeführt. Bevor Sie mit dem Programmieren in C beginnen können, müssen Sie sich deshalb einen C-Compiler besorgen. Wenn Sie Linux, OS X oder eine andere Unix-Variante einsetzen, ist in der Regel bereits der GNU-C-Compiler GCC auf Ihrem System installiert oder zumindest auf dem Installationsdatenträger oder im Web verfügbar.

Wenn Sie dagegen Windows verwenden, stehen im Internet verschiedene Compiler zum kostenlosen Download bereit. Daneben existieren zahlreiche kommerzielle Angebote, in der Regel im Rahmen komplexer Entwicklungsumgebungen. Um die Beispiele in diesem Ab-

schnitt ohne Änderungen nachvollziehen zu können, sollten Sie sich eine Windows-Version des GCC beschaffen.

Besonders empfehlenswert ist in diesem Zusammenhang der *CygWin-Compiler*, da er auch gleich eine vollständige Unix-Arbeitsumgebung für Windows mitbringt, inklusive *bash* und der wichtigsten Unix-Systemprogramme. Herunterladen können Sie diese Software unter *www.cygwin.com*. Falls Sie unter Windows einen anderen Compiler einsetzen, funktionieren zwar alle Beispiele in diesem Abschnitt, aber die Compileraufrufe selbst können sich unterscheiden.

9.1.1 Das erste Beispiel

Am einfachsten erlernen Sie eine Programmiersprache, indem Sie möglichst viele Beispielprogramme ausprobieren, nachvollziehen und anschließend modifizieren. Daher beginnt dieser Abschnitt sofort mit dem ersten Beispiel, das anschließend genau erläutert wird. Öffnen Sie Ihren bevorzugten Texteditor, geben Sie den folgenden Code ein, und speichern Sie ihn unter dem Dateinamen *hallo.c*:

```
#include <stdio.h>
#include <stdlib.h>

int main() {
   char name[20];
   puts("Hallo Welt!");
   printf("Ihr Name, bitte: ");
   gets(name);
   printf("Hallo %s!\n", name);
   return EXIT_SUCCESS;
}
```

Wechseln Sie aus dem Editor in die Konsole, gehen Sie sich in das Verzeichnis, in dem Sie die Datei *hallo.c* gespeichert haben, und geben Sie Folgendes ein:

```
$ gcc hallo.c
```

Wenn Sie nicht den GCC verwenden, müssen Sie in der Bedienungsanleitung Ihres Compilers nachschlagen, wie der Befehl für die Kompilierung lautet.

Falls Sie das Listing korrekt abgetippt haben, wird der Prompt einfach kommentarlos wieder angezeigt. Andernfalls liefert der Compiler eine oder mehrere Fehlermeldungen, bequemerweise mit Angabe der jeweiligen Zeilennummer.

Falls Sie eine Unix-Version verwenden, sollten Sie besser die folgende Variante des Befehls eingeben:

```
$ gcc -o hallo hallo.c
```


Die Option `-o Dateiname` legt einen verbindlichen Dateinamen für das fertig kompilierte Programm fest; ohne diese Angabe trägt das Programm auf Unix-Rechnern je nach konkretem Binärformat einen Namen wie *a.out*. Unter Windows heißt das Resultat automatisch *hallo.exe*.

Unter Unix besteht der nächste Schritt darin, das Programm ausführbar zu machen:

```
$ chmod +x hallo
```

Geben Sie nun unter Unix `./hallo` ein; unter Windows genügt die Eingabe `hallo`. Der Grund für diesen Unterschied wurde in Kapitel 6, »Windows«, und Kapitel 7, »Linux«, erwähnt – unter Windows ist das aktuelle Verzeichnis `.` standardmäßig im Suchpfad enthalten, unter Unix nicht. Das Programm wird ausgeführt und erzeugt folgende Ausgabe:

```
Hallo Welt!
Ihr Name, bitte: Sascha
Hallo Sascha!
```

Es handelt sich bei diesem Programm um eine erweiterte Fassung des klassischen »Hello-World«-Beispiels. Es ist Tradition, das Erlernen einer Programmiersprache mit einem Programm zu beginnen, das diese Begrüßung ausgibt. Unter *http://www.roesler-ac.de/wolfram/hello.htm* finden Sie übrigens eine Website mit »Hello-World«-Programmen in über 400 Programmiersprachen.[1]

Im Folgenden wird das erste Programmierbeispiel Zeile für Zeile erläutert:

- `#include <stdio.h>`
  Diese Zeile ist keine richtige C-Anweisung, sondern eine Präprozessor-Direktive. Der Präprozessor ist ein Bestandteil des Compilers, der vor der eigentlichen Kompilierung verschiedene organisatorische Aufgaben erledigt. An dieser Stelle lädt er die Header-Datei *stdio.h*, die die Deklarationen der wichtigsten Funktionen für die Ein- und Ausgabe bereitstellt (Standard Input/Output).
- `#include <stdlib.h>`
  Diese zweite `#include`-Direktive importiert die Header-Datei *stdlib.h*. Sie enthält wichtige Funktionen zur Laufzeit- und Speicherkontrolle.
- `int main()`
  In dieser Zeile wird eine Funktion definiert. *Funktionen* sind benannte Codeblöcke, die über ihre Namen aufgerufen werden können. Die spezielle Funktion `main()` übernimmt in einem C-Programm die Aufgabe eines Hauptprogramms: Sie wird beim Start des Programms automatisch vom Betriebssystem aufgerufen.

1 Noch beeindruckender ist die Website *http://99-bottles-of-beer.net/*, die zurzeit 1.500 verschiedene Implementierungen zur Ausgabe des Saufliedes »99 Bottles of Beer« enthält.

Der Datentyp beziehungsweise Rückgabewert der Funktion `main()` sollte `int` (ganzzahlig) sein, um dem System einen Wert zurückgeben zu können, der Erfolg oder Fehler anzeigt. Die beiden Klammern hinter dem Funktionsnamen sind Platzhalter für mögliche Parametervariablen. Der Rumpf der Funktion, also die eigentlichen Anweisungen, steht in geschweiften Klammern.

- `char name[20];`
  Diese Zeile deklariert eine Variable mit der Bezeichnung `name`. Eine Variable ist ein benannter Speicherplatz. Wenn Sie den Namen der Variablen in einem Ausdruck (zum Beispiel in einer Berechnung) verwenden, wird automatisch ihr aktueller Wert eingefügt.

  Die Variable `name` hat den Datentyp `char[]`. Es handelt sich dabei um einen Verbund einzelner Zeichen, der in C als Ersatz für einen String-Datentyp (eine Zeichenkette) verwendet wird. Der Wert `[20]` in den eckigen Klammern bedeutet, dass die Textlänge maximal 20 Zeichen betragen darf.

  Diese Anweisung wird durch ein Semikolon (`;`) abgeschlossen. In C muss jede Anweisung mit einem Semikolon enden.
- `puts("Hallo Welt!");`
  Die Funktion `puts()` hat die Aufgabe, den angegebenen Text, gefolgt von einem Zeilenumbruch, auszugeben. Text in Anführungszeichen ist ein sogenanntes *Zeichenketten-* oder *String-Literal*, das heißt Text, der »wörtlich gemeint« ist: Er wird unverändert wiedergegeben.
- `printf("Ihr Name, bitte: ");`
  Die Anweisung `printf()` dient der Ausgabe von Text oder einer Formatierung für verschiedene Ausdrücke. Im vorliegenden Fall wird auch wieder nur einfacher Text ausgegeben, allerdings ohne abschließenden Zeilenumbruch, damit die folgende Eingabe in derselben Zeile stattfindet.
- `gets(name);`
  Mithilfe von `gets()` wird eine Zeichenkette von der Standardeingabe gelesen und in der als Argument angegebenen Variablen `name` gespeichert. Die Standardeingabe (`stdin`) ist für gewöhnlich die Tastatur, es sei denn, Sie leiten die Eingabe um, wie in den beiden vorangegangen Kapiteln 6, »Windows«, und 7, »Linux«, für die jeweilige Systemplattform beschrieben.
- `printf("Hallo %s!\n", name);`
  In dieser Anweisung wird der Befehl `printf()` zum ersten Mal für seinen eigentlichen Verwendungszweck eingesetzt: Die Zeichenfolge `%s` ist ein Platzhalter für einen String-Ausdruck. Das `\n` steht für einen Zeilenumbruch. Es gibt eine Reihe solcher speziellen Zeichenfolgen, die als *Escape-Sequenzen* bezeichnet werden. Der durch `%s` ersetzte Ausdruck wird durch ein Komma von der Formatangabe getrennt. In diesem Fall ist der Ausdruck die Variable `name` – der Benutzer wird also mit seinem zuvor eingegebenen Namen begrüßt.

- `return EXIT_SUCCESS;`
  Die Anweisung `return` beendet die Ausführung einer Funktion und gibt gegebenenfalls den Wert des angegebenen Ausdrucks an die aufrufende Stelle zurück. Wenn die Funktion `main()` den Wert `EXIT_SUCCESS` zurückliefert (auf den meisten Plattformen besitzt diese `stdlib.h`-Konstante den Wert 0), signalisiert sie dem Betriebssystem damit, dass alles in Ordnung ist. Um ein Programm mit einem Fehlerzustand zu beenden, wird dagegen die Konstante `EXIT_FAILURE` verwendet, die in der Regel den Wert 1 hat.

Syntax- und Laufzeitfehler

Bei Fehlern in Computerprogrammen unterscheidet man zwischen *Syntaxfehlern*, die bereits bei der Übersetzung abgefangen werden, und *Laufzeitfehlern*, die erst bei der Ausführung auftreten. Beispielsweise ist eine Anweisung wie

```
str = "Dies ist ein Text;
```

syntaktisch falsch (das Anführungszeichen, das den String abschließen müsste, fehlt), und der Compiler fängt sie ab. Dividieren Sie dagegen zum Beispiel im Verlauf Ihres Programms durch eine Variable, deren Wert zufälligerweise 0 ist, bricht die Programmausführung mit einer Fehlermeldung ab. Um Laufzeitfehler zu verhindern, müssen Sie die Werte, mit denen Sie arbeiten, stets gründlich überprüfen – insbesondere Benutzereingaben, denn diese können sogar Auswirkungen auf die Sicherheit Ihrer Softwareumgebung haben.

9.1.2 Elemente der Sprache C

Im letzten Abschnitt wurden bereits einige Merkmale der Programmiersprache C angesprochen. In diesem Abschnitt werden nun die wichtigsten Elemente von C systematisch behandelt.

Die grundlegende Syntax

Ein C-Programm besteht grundsätzlich aus einer Abfolge von *Anweisungen*. Eine Anweisung entspricht einem einzelnen Verarbeitungsschritt, den Ihr Programm durchführen soll. Jede Anweisung steht in einer eigenen Zeile und endet mit einem Semikolon. Falls Ihnen eine Zeile zu lang erscheint, dürfen Sie an einer sinnvollen Stelle einen Backslash (\) einfügen und in der nächsten Zeile weiterschreiben. Dies darf allerdings nicht innerhalb der Anführungszeichen eines String-Literals geschehen.

Es gibt verschiedene Typen von Anweisungen. Die wichtigsten sind Funktionsaufrufe, Deklarationen, Wertzuweisungen und Kontrollstrukturen. Diese Anweisungsarten weisen folgende Eigenschaften auf:

- *Funktionsaufrufe* bestehen aus dem Namen der aufgerufenen Funktion und den zugehörigen Argumenten. Es kann sich sowohl um eingebaute als auch um selbst definierte Funktionen handeln. Beispiel:

```
printf("hallo");
```

- *Deklarationen* sind Variablen- oder Funktionsvereinbarungen. Beide Arten der Deklaration bestehen aus einem Datentyp und einem selbst gewählten Bezeichner (dem Namen des Elements). Variablen können auf Wunsch schon bei der Deklaration einen Wert erhalten. Funktionen besitzen optional beliebig viele Parameter, die als Variablen mit Datentypangabe in die Klammern hinter den Funktionsnamen geschrieben werden. Der Funktionsrumpf steht in geschweiften Klammern und besteht aus beliebig vielen Anweisungen. Beispiele:

```
int wert;           /* Variablendeklaration */
float zahl = 2.75;  /* Deklaration mit Wertzuweisung */
int summe (int a, int b)
{...}               /* Funktionsdefinition */
```

- *Wertzuweisungen* dienen dazu, einer Variablen einen Wert zuzuordnen. Eine Zuweisung hat die Form `variable = ausdruck`. Der Ausdruck wird zunächst ausgewertet, anschließend wird sein Wert in der Variablen gespeichert. Beispiele:

```
wert = 7;
zahl = 5 / 2;
```

- *Kontrollstrukturen* sind eine Sammelbezeichnung für Anweisungen, die der Flusskontrolle des Programms dienen, dazu gehören beispielsweise Fallunterscheidungen und Schleifen. Beispiel:

```
if (a < 0) {
   printf("a ist negativ");
}                   /* Fallunterscheidung */
```

Neben den Anweisungen kann ein C-Programm *Kommentare* enthalten. Ein Kommentar steht zwischen den Zeichenfolgen /* und */ und kann beliebig viele Zeilen umfassen. Der Compiler ignoriert Kommentare; sie dienen dazu, Ihnen die Orientierung im Programmcode zu erleichtern. Kommentare dürfen nicht ineinander verschachtelt werden, da das erste Auftreten von */ den Kommentar bereits aufhebt.

Seit dem 1999 veröffentlichten Standard C99 dürfen auch einzeilige Kommentare verwendet werden, die ursprünglich in C++ eingeführt wurden und auch in Java und anderen Programmiersprachen bekannt sind. Diese beginnen mit den beiden Zeichen // und reichen bis zum Ende der aktuellen Zeile.

Leere Zeilen im Programmcode werden ignoriert, auch vor Anweisungen und zwischen den einzelnen Elementen einer Programmzeile dürfen beliebig viele Leerzeichen stehen. Der Ausdruck `a + b` ist äquivalent zu `a+b`. Allerdings dürfen Sie innerhalb von Bezeichnern keine Leerzeichen einfügen; auch einige Operatoren bestehen aus mehreren Zeichen, die nicht voneinander getrennt werden dürfen (beispielsweise `<=` oder `&&`).

Gängige Formatierungsregeln

Bei der Verwendung von Leerzeichen im Code sollten Sie vor allem konsistent sein. Weitverbreitete Praxis sind folgende Regeln:

- je ein Leerzeichen vor und hinter einen Operanden setzen, zum Beispiel `a = b * c + d`
- Zwischen Funktionsnamen und der öffnenden Klammer dahinter steht kein Leerzeichen, also etwa `printf(...)`.
- Bei Kontrollstruktur-Anweisungen wie `if` steht dagegen eine Klammer zwischen dem Schlüsselwort und der geklammerten Bedingung, beispielsweise `if (...)`.

Bezeichner für Variablen und Funktionen dürfen aus Buchstaben, Ziffern und _ (Unterstrich) bestehen. Sie dürfen allerdings nicht mit einer Ziffer beginnen. Der ANSI-C-Standard, an den sich im Prinzip alle aktuellen C-Versionen halten, schreibt vor, dass mindestens die ersten 31 Zeichen der Bezeichner ausgewertet werden müssen. Sollten zwei Bezeichner erst beim 32. Zeichen voneinander abweichen, halten manche Compiler sie für ein und denselben. Bei Bezeichnern wird zwischen Groß- und Kleinschreibung unterschieden.

Variablen

Wie bereits erwähnt, ist eine *Variable* ein benannter Speicherplatz. Anders als in der Mathematik besitzt eine Variable in einer Programmiersprache jederzeit einen definierten Wert. Es handelt sich also zur Laufzeit des Programms nicht um einen Platzhalter.

Variablen müssen zu Beginn jeder Funktion deklariert werden. Dies geschieht durch die Angabe des *Datentyps* und des *Bezeichners*; optional kann ein Anfangswert zugewiesen werden. Das folgende Beispiel deklariert die beiden Variablen `a` und `b`, wobei nur `b` ein Wert zugewiesen wird:

```
int a;
double b = 2.5;
```

`a` wird als `int` (Integer oder Ganzzahl) deklariert, dient also der zukünftigen Speicherung einer ganzen Zahl. `b` erhält den Datentyp `double`, der zur Ablage doppelt genauer Fließkommazahlen verwendet wird.

Tabelle 9.1 zeigt eine Übersicht über die wichtigsten einfachen Datentypen und ihre Bedeutung.

Datentyp	Bedeutung	Erläuterung
`int`	Integer (Ganzzahl)	eine ganze Zahl mit der Wortbreite des Prozessors: auf den meisten Rechnern 32 oder 64 Bit
`short`	kurzer Integer	Integer mit geringerer Speichergröße (oft 16 Bit)
`long`	langer Integer	Integer mit der größten Bit-Zahl (mindestens 32 Bit)
`char`	einzelnes Byte	8-Bit-Integer; wird oft zur Darstellung von ASCII-Zeichen verwendet – daher der Name `char` (für *character*).
`float`	Fließkommawert	Speichert Fließkommazahlen mit einfacher Genauigkeit (meist 32 Bit).
`double`	Fließkommawert	doppelt genauer Fließkommawert (in der Regel 64 Bit)

Tabelle 9.1 Die elementaren C-Datentypen

Die ganzzahligen Datentypen `int`, `short`, `long` und `char` speichern einen Integer-Wert je nach Bedarf mit oder ohne Vorzeichen. Sie können der Deklaration `signed` voranstellen, um Werte mit Vorzeichen zu speichern, oder `unsigned` für Werte ohne Vorzeichen. Dies bedeutet beispielsweise für einen 32-Bit-Integer, dass bei `signed` Werte zwischen –2.147.483.648 und +2.147.483.647 möglich sind, während `unsigned` die Werte 0 bis 4.294.967.295 zulässt. Eine Erläuterung dieser Werte finden Sie in Kapitel 2, »Mathematische und technische Grundlagen«.

Standardmäßig sind alle Integer-Werte `signed`, ein Sonderfall ist lediglich `char`: Da dieser Typ in der Regel zur Darstellung einzelner ASCII-Zeichen eingesetzt wird, ist er zumindest in diesem Zusammenhang `unsigned`. Für die Zeichen modernerer Zeichensätze wie Unicode, die mehr Speicher benötigen als 8 Bit, könnten Sie einfach `unsigned int` benutzen, empfehlenswerter ist jedoch die Verwendung des speziellen Typs `wchar_t`, wofür Sie allerdings mithilfe von `#include` die Header-Datei *stddef.h* einbinden müssen. Die verschiedenen Zeichensätze werden in Kapitel 17, »Weitere Datei- und Datenformate«, besprochen.

Auffällig ist, dass es in C keinen separaten Datentyp für boolesche Wahrheitswerte gibt. Folgerichtig gelten Ausdrücke mit dem Wert 0 als falsch und alle anderen als wahr.

Ein weiteres Merkmal von Variablen ist ihr *Gültigkeitsbereich* (*Scope*), der festlegt, in welchen Programmteilen eine Variable definiert bleibt. Grundsätzlich werden zwei verschiedene Arten von Variablen unterschieden:

- *Lokale Variablen*, auch *automatische Variablen* genannt, gelten nur innerhalb der Funktion, in der sie deklariert wurden. Dies gilt sowohl für Variablen, die zu Beginn des Funktionsrumpfes definiert werden, als auch für Parametervariablen, die in den Klammern hinter dem Funktionsnamen aufgeführt werden.

- *Globale Variablen* werden zu Beginn des Programmcodes (hinter eventuellen Präprozessor-Direktiven) deklariert und gelten im gesamten Programm. Falls Sie allerdings innerhalb einer Funktion eine gleichnamige Variable neu deklarieren, wird dort nur diese lokale Variable verwendet.

Eine besondere Form der lokalen Variablen sind die *statischen Variablen*: Wenn Sie in einer Funktion eine Variablendeklaration mit dem Schlüsselwort `static` einleiten, gilt die Variable zwar nur innerhalb dieser Funktion, behält aber ihren Wert bis zum nächsten Aufruf der Funktion bei.

Ausdrücke und Operationen

Zu den wichtigsten Fähigkeiten von Programmiersprachen gehört die Auswertung beziehungsweise Berechnung von *Ausdrücken*. An jeder Stelle, an der ein bestimmter Wert erwartet wird, kann stattdessen ein komplexer Ausdruck stehen, der zunächst ausgewertet und anschließend als Wert eingesetzt wird. Voraussetzung ist allerdings, dass der Ausdruck einen passenden Datentyp besitzt.

Die einfachsten Bestandteile von Ausdrücken sind die *Literale*. Es handelt sich dabei um Werte, die nicht weiter berechnet oder ersetzt werden müssen. C unterscheidet die folgenden vier Arten von Literalen:

- *Integer-Literale* dienen der Darstellung ganzzahliger Werte. Normalerweise werden sie dezimal notiert; dazu wird einfach die entsprechende Zahl mit optionalem Vorzeichen geschrieben. Wenn Sie einem Integer-Wert eine `0` voranstellen, wird er als Oktalzahl interpretiert: `033` bedeutet demzufolge nicht 33, sondern 27. Ein vorangestelltes `0x` kennzeichnet dagegen eine Hexadezimalzahl: `0x33` steht also für den (dezimalen) Wert 51.
- *Fließkommaliterale* repräsentieren Fließkommawerte. Beachten Sie, dass in C und anderen Programmiersprachen nicht das Komma als Dezimaltrennzeichen verwendet wird, sondern ein Punkt. Alternativ können Sie Fließkommaliterale in wissenschaftlicher Schreibweise (Exponentialschreibweise) angeben: `3.5E+5` steht beispielsweise für $3{,}5 \times 10^5$ (350.000); `4.78E-4` hat dagegen den Wert $4{,}78 \times 10^{-4}$ (0,000478).
- *Zeichen-Literale* enthalten ein einzelnes Zeichen aus einem Zeichensatz, mit dem der Compiler umgehen kann. Ein Zeichen-Literal muss in einfachen Anführungszeichen stehen, beispielsweise `'a'`.
- *String-Literale* enthalten Zeichenketten, das heißt beliebig lange Textblöcke. Sie müssen in doppelten Anführungszeichen stehen, etwa `"hallo"`. Ein Sonderfall ist die leere Zeichenkette, die durch zwei unmittelbar aufeinanderfolgende Anführungszeichen dargestellt wird: `""`. Sie wird bei Fallunterscheidungen wie der Wert 0 als falsch betrachtet.

Ein weiterer Bestandteil von Ausdrücken sind Variablen, die bei der Auswertung jeweils durch ihren aktuellen Wert ersetzt werden:

```
a = 5;
b = a + 7;          /* b hat nun den Wert 12 */
```

Mitunter besitzt eine Variable, die Sie in einem Ausdruck verwenden möchten, den falschen Datentyp. C konvertiert den Datentyp immer dann automatisch, wenn keine Gefahr besteht, dass dabei Werte verloren gehen oder verfälscht werden. Beispielsweise wird `int` ohne Weiteres akzeptiert, wo eigentlich ein Fließkommatyp erwartet wird. In Fällen, in denen diese Gefahr besteht, müssen Sie die Typumwandlung dagegen explizit anordnen. Dies geschieht durch das sogenannte *Typecasting*. Dabei wird der gewünschte Datentyp in Klammern vor die umzuwandelnde Variable oder auch vor ein Literal geschrieben:

```
double a = 4.7;
printf("%d", (int)a);
                    /* Ausgabe: 4 */
```

Sie können innerhalb eines Ausdrucks sogar eine Funktion aufrufen, vorausgesetzt, sie liefert einen Wert mit dem passenden Datentyp zurück:

```
a = sin(b);         /* a enthält den Sinus von b */
```

Neben all diesen Elementen, die jeweils einen einzelnen Wert ergeben, können Ausdrücke auch *Operatoren* enthalten. Diese dienen dazu, verschiedene Werte arithmetisch oder logisch miteinander zu verknüpfen. Beachten Sie, dass nicht jeder Operator zu jedem Datentyp passt.

Die *arithmetischen Operatoren* sind + (Addition), – (Subtraktion), * (Multiplikation), / (Division) und % (Modulo; der Rest einer ganzzahligen Division).

Die nächste Gruppe bilden die *logischen Operatoren*. Sie dienen dazu, Werte nach logischen Kriterien miteinander zu verknüpfen:

- Das *logische Und* (geschrieben `&&`) erzeugt den Wert 0, wenn mindestens einer der verknüpften Ausdrücke 0 (logisch falsch) ist, andernfalls erhält der Gesamtausdruck einen von 0 verschiedenen Wert und gilt damit als wahr.
- Das *logische Oder* wird als `||` notiert und erhält einen von 0 verschiedenen Wert, sobald mindestens einer der verknüpften Ausdrücke von 0 verschieden ist.
- Das *logische Nicht* wird durch ein `!` dargestellt, das dem zu negierenden Ausdruck vorangestellt wird. Der Ausdruck erhält dadurch den Wert 0, wenn er zuvor einen anderen Wert hatte, und 1, wenn sein ursprünglicher Wert 0 war.

Ähnlich wie die logischen Operatoren, aber auf einer anderen Ebene, arbeiten die *Bit-Operatoren*: Sie manipulieren den Wert ihrer Operanden bitweise, betrachten also jedes einzelne Bit. Im Einzelnen sind die folgenden Bit-Operatoren definiert:

- Das *bitweise Und* (geschrieben `&`) setzt im Ergebnis diejenigen Bits auf den Wert 1, die in beiden Operanden 1 sind, alle anderen auf 0. Beispiel: `94 & 73` führt zu dem Ergebnis 72. Erläutern lässt sich dieses Ergebnis nur anhand der binären Darstellung:

```
  0101 1110
& 0100 1001
-----------
  0100 1000
```

- Das *bitweise Oder* (`|`) setzt alle Bits auf 1, die in mindestens einem der Operanden den Wert 1 haben. `94 | 73` ergibt demzufolge 95:

```
  0101 1110
| 0100 1001
-----------
  0101 1111
```

- Das *bitweise exklusive Oder* (`^`) setzt nur diejenigen Bits auf 1, die in genau einem Operanden den Wert 1 haben, alle anderen dagegen auf 0. `94 ^ 73` liefert das Ergebnis 23:

```
  0101 1110
^ 0100 1001
-----------
  0001 0111
```

- Die *Bit-Verschiebung nach links* (`<<`) verschiebt die Bits des ersten Operanden um die Anzahl von Stellen nach links, die der zweite Operand angibt. Beispiel: `73 << 2` ergibt 292, entspricht also einer Multiplikation mit 2^2 (4).
- Die *Bit-Verschiebung nach rechts* (`>>`) verschiebt die Bits des ersten Operanden dagegen um die angegebene Anzahl von Stellen nach rechts. Beispiel: `94 >> 3` führt zu dem Ergebnis 11, da die letzten drei Binärstellen wegfallen.
- Die *bitweise Negation* oder das *Bit-Komplement* (eine vorangestellte Tilde ~) setzt alle Bits mit dem Wert 1 auf 0 und umgekehrt. Das Ergebnis ist abhängig von der Bit-Breite des entsprechenden Werts. Beispiel: `~73` ergibt als `unsigned` 8-Bit-Integer den Wert 182.

Für die Ablaufsteuerung von Programmen sind die *Vergleichsoperatoren* von besonderer Bedeutung: Sie vergleichen Ausdrücke miteinander und liefern je nach Ergebnis 0 oder einen wahren Wert. Es sind folgende Vergleichsoperatoren definiert:

- `==` ist der Gleichheitsoperator; das Ergebnis ist wahr, wenn die beiden verglichenen Ausdrücke gleich sind.
- `!=` überprüft die Ungleichheit von Ausdrücken, ist also wahr, wenn sie verschieden sind.
- `<` ist wahr, wenn der linke Operand kleiner ist als der rechte.
- `>` ist wahr, wenn der linke Operand größer als der rechte ist.

- <= ist wahr, wenn der linke Operand kleiner oder gleich dem rechten ist. Diese Operation ist die Negierung von >.
- >= ist wahr, wenn der linke Operand größer oder gleich dem rechten ist. Dies ist die Negierung von <.

Zu guter Letzt gibt es noch den *Wertzuweisungsoperator* =, der dem Operanden auf der linken Seite den Wert des Ausdrucks auf der rechten Seite zuweist. Bei dem linken Operanden handelt es sich in der Regel um eine Variable. Allgemein werden Elemente, die auf der linken Seite einer Wertzuweisung stehen können, als *LVALUE* bezeichnet.

Sehr häufig kommt es vor, dass eine Wertzuweisung den ursprünglichen Wert des LVALUE ändert, sodass der LVALUE selbst in dem Ausdruck auf der rechten Seite auftaucht. Für diese speziellen Fälle wurden einige Abkürzungen definiert; die wichtigsten sind in Tabelle 9.2 aufgeführt.

Langfassung	Kurzfassung	Erläuterung
`a = a + 5;`	`a += 5;`	LVALUE um den angegebenen Wert erhöhen

`a = a + 1;`	`a += 1;`
	`a++;`
	`++a;`
		LVALUE um 1 erhöhen (beachten Sie dabei die Erläuterung im Anschluss an die Tabelle)

`a = a - 5;`	`a -= 5;`	LVALUE um den angegebenen Wert vermindern

`a = a - 1;`	`a -= 1;`
	`a--;`
	`--a;`
		LVALUE um 1 vermindern

`a = a * 5;`	`a *= 5;`	LVALUE mit dem angegebenen Wert multiplizieren
`a = a / 5;`	`a /= 5;`	LVALUE durch den angegebenen Wert dividieren

Tabelle 9.2 Die wichtigsten Abkürzungen für kombinierte Wertzuweisungen

Neben den in der Tabelle angegebenen Abkürzungen gibt es auch für die logischen Operatoren und die Bit-Operatoren entsprechende Schreibweisen.

Eine Sonderstellung nehmen die Operatoren ein, die einen LVALUE um 1 erhöhen oder vermindern: Sie können ++ oder -- entweder vor oder hinter den LVALUE schreiben. Wenn Sie dies als einzelne Anweisung hinschreiben, besteht zwischen diesen Varianten kein Unterschied. Werden sie dagegen im Rahmen eines komplexen Ausdrucks verwendet, dann ist der Unterschied folgender:

- Das vorangestellte ++ wird *Prä-Inkrement* genannt. Es erhöht den LVALUE um 1 und verwendet den neuen Wert innerhalb des Ausdrucks:


```
a = 1;
b = ++a;            /* a hat den Wert 2, b auch. */
```

Entsprechend heißt ein vorangestelltes -- *Prä-Dekrement*. Der LVALUE wird um 1 vermindert, bevor er in einem Ausdruck verwendet wird.

- Ein nachgestelltes ++ wird als *Post-Inkrement* bezeichnet. Ein LVALUE mit Post-Inkrement wird in einem Ausdruck mit seinem alten Wert verwendet und erst danach um 1 erhöht:

```
a = 2;
b = a++;            /* a hat den Wert 3, b bleibt 2 */
```

Das nachgestellte -- heißt *Post-Dekrement*. Der alte Wert des LVALUE wird im Ausdruck verwendet, bevor es um 1 vermindert wird.

Ein besonderer Operator ist der *Fallunterscheidungsoperator*: Die Schreibweise `Ausdruck1 ? Ausdruck2 : Ausdruck3` hat `Ausdruck2` als Ergebnis, wenn `Ausdruck1` wahr ist, ansonsten ist der Wert `Ausdruck3`. Da er als einziger Operator drei Operanden hat, wird er auch als *ternärer* (dreigliedriger) Operator bezeichnet; entsprechend heißen die meisten Operatoren *binär* (zwei Operanden, zum Beispiel alle arithmetischen Operationen) beziehungsweise *unär* (ein Operand, etwa Vorzeichen). Hier zwei Beispiele:

```
a = 2;
b = (a == 1 ? 3 : 5);
/* a ist nicht 1, also erhält b den Wert 5 */

a = 1;
b = (a == 1 ? 3 : 5);
/* a ist 1, also erhält b den Wert 3 */
```

Bei der Arbeit mit Operatoren ist zu beachten, dass sie mit unterschiedlicher Priorität ausgewertet werden. Die folgende Liste stellt die Rangfolge der Operatoren in absteigender Reihenfolge dar. Die weiter oben stehenden Operatoren binden also stärker und werden zuerst aufgelöst:

- !, ~, ++, --, + (Vorzeichen), – (Vorzeichen)
- *, /, %
- + und - (arithmetische Operatoren)
- << und >>
- <, <=, >, >=
- == und !=
- & (bitweises Und)
- ^ (bitweises Exklusiv-Oder)
- | (bitweises Oder)

- && (logisches Und)
- || (logisches Oder)
- ?: (Operator für Fallunterscheidungen)
- =, +=, -= etc.

Sie können die Rangfolge der Operatoren durch die Verwendung von Klammern verändern. Beispielsweise besitzt der Ausdruck `3 * 4 + 7` den Wert 19, während `3 * (4 + 7)` das Ergebnis 33 liefert. Beachten Sie, dass zu diesem Zweck – anders als in der Mathematik – immer nur runde Klammern verwendet werden dürfen, egal, wie tief sie verschachtelt werden!

Kontrollstrukturen

Eine der wesentlichen Aufgaben von Computerprogrammen besteht darin, den Programmablauf in Abhängigkeit von bestimmten Bedingungen zu steuern. Dazu definiert C eine Reihe sogenannter *Kontrollstrukturen*, die man grob in Fallunterscheidungen und Schleifen unterteilen kann. Eine *Fallunterscheidung* überprüft die Gültigkeit einer Bedingung und führt abhängig davon bestimmte Anweisungen aus; eine *Schleife* sorgt dagegen dafür, dass bestimmte Anweisungsfolgen mehrmals hintereinander ausgeführt werden.

Die einfachste und wichtigste Kontrollstruktur ist die Fallunterscheidung mit `if()`. Der Ausdruck, der hinter dem Schlüsselwort `if` in Klammern steht, wird ausgewertet. Wenn er wahr (nicht 0) ist, wird die auf das `if` folgende Anweisung ausgeführt. Das folgende Beispiel überprüft, ob die Variable `a` größer als 100 ist, und gibt in diesem Fall »Herzlichen Glückwunsch!« aus:

```
if (a > 100)
   printf("Herzlichen Glückwunsch!\n");
```

Mitunter hängen mehrere Anweisungen von einem einzelnen `if()` ab. In diesem Fall müssen Sie hinter der Bedingungsprüfung einen *Anweisungsblock* schreiben, also eine Sequenz von Anweisungen in geschweiften Klammern. Das folgende Beispiel überprüft, ob die Variable `b` kleiner als 0 ist. In diesem Fall setzt sie `b` auf 100 und gibt eine entsprechende Meldung aus:

```
if (b < 0) {
   b = 100;
   printf("b auf 100 gesetzt.\n");
}
```

Die öffnende geschweifte Klammer schreiben manche Programmierer lieber in die nächste Zeile. Beide Varianten sind üblich und zulässig, Sie sollten sich allerdings konsequent an eine davon halten. In diesem Buch wird die Klammer stets in dieselbe Zeile gesetzt, allein schon aus Platzgründen.

Verwenden Sie stets geschweifte Klammern!

Letzten Endes lohnt es sich übrigens, auch bei `if`-Abfragen, von denen nur eine einzige Anweisung abhängt, geschweifte Klammern zu verwenden. Erstens kann Ihnen so nicht der Fehler passieren, dass Sie die Klammern vergessen, wenn später weitere Anweisungen dazukommen. Zweitens gibt es andere Programmiersprachen wie etwa Perl, bei denen die Klammern zwingend vorgeschrieben sind.

Für die Formulierung des Bedingungsausdrucks bieten sich einige Abkürzungen an, die mit der logischen Interpretation von 0 und anderen Werten zu tun haben. Wollen Sie beispielsweise Anweisungen ausführen, wenn die Variable `a` den Wert 0 hat, können Sie entweder die ausführliche Bedingung `a == 0` schreiben oder die Abkürzung `!a` verwenden: Die Negation von `a` ist genau dann wahr, wenn `a` gleich 0 ist. Sollen dagegen Anweisungen ausgeführt werden, wenn `a` nicht 0 ist, genügt sogar ein einfaches `a` als Bedingung. Diese Nachlässigkeit bei der Überprüfung von Datentypen macht C zu einer sogenannten *schwach typisierten Sprache*: Variablen besitzen festgelegte Datentypen, diese werden aber bei Bedarf sehr großzügig ineinander konvertiert.

Es kommt sehr häufig vor, dass auch bei Nichtzutreffen einer Bedingung spezielle Anweisungen ausgeführt werden sollen. Zu diesem Zweck besteht die Möglichkeit, hinter einer `if`-Abfrage einen `else`-Teil zu platzieren. Die Anweisung oder der Block hinter dem `else` wird genau dann ausgeführt, wenn die Bedingung der `if`-Abfrage nicht zutrifft. Das folgende Beispiel gibt »a ist positiv.« aus, wenn `a` größer als 0 ist, ansonsten wird »a ist nicht positiv.« zurückgegeben:

```
if (a > 0)
   printf("a ist positiv.\n");
else
   printf("a ist nicht positiv.\n");
```

Auch hinter dem `else` kann alternativ ein Block von Anweisungen in geschweiften Klammern folgen. Sie können hinter dem `else` sogar wieder ein weiteres `if` unterbringen, falls eine weitere Bedingung überprüft werden soll, wenn die ursprüngliche Bedingung nicht erfüllt ist. Die folgende verschachtelte Abfrage erweitert das vorangegangene Beispiel so, dass auch die Fälle 0 und negativer Wert unterschieden werden:

```
if (a > 0)
   printf("a ist positiv.\n");
else if (a < 0)
   printf("a ist negativ.\n");
else
   printf("a ist 0.\n");
```

Das folgende kleine Beispielprogramm verwendet verschachtelte `if-else`-Abfragen, um aus einer eingegebenen Punktzahl in einer Prüfung die zugehörige Note nach dem IHK-Notenschlüssel zu berechnen:

```
#include <stdio.h>

int main() {
   int punkte;
   int note;
   printf("Ihre Punktzahl, bitte: ");
   scanf("%d", &punkte);
   if (punkte < 30)
      note = 6;
   else if (punkte < 50)
      note = 5;
   else if (punkte < 67)
      note = 4;
   else if (punkte < 81)
      note = 3;
   else if (punkte < 92)
      note = 2;
   else
      note = 1;

   printf("Sie haben die Note %d erreicht.\n", note);
   return 0;
}
```

Die Funktion `scanf()` dient übrigens dazu, Daten verschiedener Formate einzulesen – im Gegensatz zu der zuvor verwendeten Funktion `gets()`, die nur zum Einlesen von Strings verwendet wird. Die Formatangabe `%d` steht für einen Integer (die Abkürzung `d` bedeutet *decimal*). Wichtig ist die Angabe des `&`-Zeichens vor dem Variablennamen – damit wird die Adresse und nicht der Wert angegeben, denn `scanf()` muss wissen, wo der eingelesene Wert gespeichert werden soll. Näheres zu diesem Thema finden Sie im Unterabschnitt »Zeiger und Arrays«.

In anderen Fällen kommt es vor, dass Sie eine Variable nacheinander mit verschiedenen festen Werten vergleichen müssen, nicht mit Wertebereichen wie im vorangegangenen Beispiel. Für diesen Verwendungszweck bietet C die spezielle Struktur `switch/case` an. Das Argument von `switch` muss ein LVALUE sein, das nacheinander mit einer Liste von Werten verglichen wird, die hinter dem Schlüsselwort `case` stehen. Die einzelnen `case`-Unterscheidungen stellen dabei Einstiegspunkte in den `switch`-Codeblock dar. Wenn das LVALUE

einem der Werte in der Liste entspricht, wird von dieser Stelle an der gesamte Block ausgeführt. Da dieses Verhalten oft unerwünscht ist, wird der Block vor jedem neuen `case` meist mithilfe von `break` verlassen.

Das folgende Beispiel ermittelt aus einer numerischen Note die entsprechende Zensur in Textform:

```
switch (note) {
   case 6:
      printf("ungenügend\n");
      break;
   case 5:
      printf("mangelhaft\n");
      break;
   case 4:
      printf("ausreichend\n");
      break;
   case 3:
      printf("befriedigend\n");
      break;
   case 2:
      printf("gut\n");
      break;
   case 1:
      printf("sehr gut\n");
      break;
   default:
      printf("unzulässige Eingabe\n");
}
```

Hinter der optionalen Markierung `default` können Anweisungen stehen, die ausgeführt werden, wenn der geprüfte LVALUE keinen der konkreten Werte hat. Dies eignet sich insbesondere, um Fehleingaben abzufangen.

Eine grundlegend andere Art von Kontrollstrukturen sind *Schleifen*. Sie sorgen dafür, dass ein bestimmter Codeblock mehrmals ausgeführt wird, entweder abhängig von einer Bedingung oder mit einer definierten Anzahl von Durchläufen.

Die einfachste Form der Schleife ist die `while()`-Schleife. In den Klammern hinter dem Schlüsselwort `while` wird genau wie bei `if()` eine Bedingung geprüft. Trifft sie zu, wird der *Schleifenrumpf* (die Anweisung oder der Block hinter dem `while`) ausgeführt. Nach der Ausführung wird die Bedingung erneut überprüft. Solange sie zutrifft, wird der Schleifenrumpf immer wieder ausgeführt. Das folgende Beispiel überprüft vor jedem Durchlauf, ob die Variable `i` noch kleiner als 10 ist, und erhöht sie innerhalb des Schleifenrumpfes jeweils um 1:

```
i = 0;
while (i < 10) {
   printf("%d\t", i);
   i++;
}
```

`i` ist der bevorzugte Name für Schleifenzählervariablen. Diese Tradition stammt aus der Mathematik, wo `i` oft als Zähler bei Summenformeln oder Ähnlichem eingesetzt wird (Abkürzung für »index«). Wenn mehrere Schleifen ineinander verschachtelt werden, heißen deren Zählervariablen `j`, `k`, `l` und so fort.

Die Ausgabe dieses kurzen Beispiels (`\t` steht für einen Tabulator) sieht folgendermaßen aus:

```
0   1   2   3   4   5   6   7   8   9
```

Da die Überprüfung der Bedingung vor dem jeweiligen Durchlauf erfolgt, findet der Abbruch statt, sobald `i` nicht mehr kleiner als 10 ist. Eine solche Schleifenkonstruktion wird als *kopfgesteuerte* Schleife bezeichnet.

Eine andere Art der Schleife überprüft die Bedingung erst nach dem jeweiligen Durchlauf und heißt deshalb *fußgesteuert*. In C wird sie durch die Schreibweise `do ... while()` realisiert. Diese Art der Schleife ist nützlich, wenn die zu überprüfende Bedingung sich erst aus dem Durchlauf selbst ergibt, beispielsweise bei der Prüfung von Benutzereingaben. Das vorangegangene Beispiel sieht als fußgesteuerte `do-while`-Schleife so aus:

```
i = 0;
do {
   printf("%d\t", i);
   i++;
} while (i < 10);
```

Interessanterweise stellt sich die Ausgabe dieser Schleife etwas anders dar:

```
0   1   2   3   4   5   6   7   8   9   10
```

Da die Bedingung erst nach der Ausgabe geprüft wird, erfolgt der Abbruch erst einen Durchlauf später. Anders als die kopfgesteuerte Schleife wird die fußgesteuerte immer *mindestens einmal* ausgeführt. Beachten Sie, dass hinter dem `while()` in diesem Fall ein Semikolon stehen muss.

Eine alternative Schreibweise für Schleifen ist die `for`-Schleife. Sie wird bevorzugt in Fällen eingesetzt, in denen eine festgelegte Anzahl von Durchläufen erwünscht ist. Die allgemeine Syntax dieser Schleife ist folgende:

```
for (Initialisierung; Wertüberprüfung; Wertänderung)
   Anweisung;
```


Die Initialisierung wird genau einmal vor dem Beginn der Schleife ausgeführt. Die Wertüberprüfung findet vor jedem Durchlauf statt. Wenn sie einen wahren Wert ergibt, wird der Schleifenrumpf ein weiteres Mal ausgeführt. Nach jedem Durchlauf findet die Wertänderung statt. Beispiel:

```
for (i = 0; i < 10; i++) {
   printf("%d\t", i);
}
```

Dies erzeugt exakt dieselbe Ausgabe wie das vorangegangene kopfgesteuerte `while`-Beispiel; der Code ist sogar absolut äquivalent. Jede `for`-Schleife lässt sich auf diese Weise durch eine `while`-Schleife ersetzen. Es handelt sich lediglich um eine spezielle Formulierung, die für determinierte Schleifen (Schleifen mit festgelegter Anzahl von Durchläufen) besser geeignet ist.

Funktionen

Wie bereits erwähnt, besteht ein C-Programm aus beliebig vielen Funktionen, die Sie innerhalb Ihres Programms von jeder Stelle aus aufrufen können. Die wichtigste Funktion ist `main()`, weil sie die Aufgabe des Hauptprogramms übernimmt.

Eine Funktion kann jeden der eingangs für Variablen genannten Datentypen innehaben. Es wird erwartet, dass eine Funktion mit einem bestimmten Datentyp mithilfe von `return` einen Wert dieses Typs an die aufrufende Stelle zurückgibt. Eine Funktion, die »nur« bestimmte Anweisungen ausführen, aber keinen Wert zurückgeben soll, kann den speziellen Datentyp `void` haben.

Die wichtigste Aufgabe von Funktionen ist die Strukturierung des Programms. Es lohnt sich, häufig benötigte Anweisungsfolgen in separate Funktionen zu schreiben und bei Bedarf aufzurufen. Eine solche *Modularisierung* des Codes macht das Programm übersichtlicher, weil Sie sich in jeder Funktion auf eine einzelne Aufgabe konzentrieren können. Auf diese Weise lassen sich mehrere Abstraktionsebenen in ein Programm einführen: Grundlegende Bausteine können einmal implementiert und zu immer komplexeren Einheiten zusammengesetzt werden.

Eine Funktion kann nicht nur einen Rückgabewert haben, sondern auch einen oder mehrere Eingabewerte, die in Form von Parametervariablen in die Klammern hinter dem Funktionsnamen geschrieben werden. Eine Funktion mit Parametern erwartet die Übergabe entsprechend vieler Werte mit dem korrekten Datentyp. Aus Sicht der aufrufenden Stelle werden diese Werte als *Argumente der Funktion* bezeichnet, innerhalb der Funktion können sie wie normale lokale Variablen verwendet werden.

Das folgende Beispiel zeigt eine Funktion namens `verdoppeln()`, die einen Wert vom Datentyp `int` entgegennimmt und das Doppelte dieses Werts zurückgibt:

```
int verdoppeln(int wert) {
   return 2 * wert;
}
```

Diese Funktion kann von einer beliebigen Programmstelle aus innerhalb eines Ausdrucks aufgerufen werden, wo ein Integer-Wert zulässig ist. Im folgenden Beispiel wird `verdoppeln()` aus einer `printf()`-Anweisung heraus aufgerufen, um das Doppelte der Variablen `b` auszugeben:

```
printf("%d\n", verdoppeln(b));
```

Eine Funktion vom Datentyp `void` wird dagegen als einzelne Anweisung aufgerufen. Das folgende Beispiel definiert eine Funktion namens `begruessen()`, die einen Gruß für den angegebenen Namen ausgibt:

```
void begruessen(char[] name) {
   printf("Hallo, %s!\n", name);
}
```

Ein Aufruf dieser Funktion sieht etwa folgendermaßen aus:

```
begruessen("Klaus");
```

Die Ausgabe sieht natürlich so aus:

```
Hallo, Klaus!
```

Übrigens kann auch die Funktion `main()` so geschrieben werden, dass sie Argumente entgegennimmt. Dies dient der Annahme von Kommandozeilenparametern. Die standardisierte Syntax für die Parameter von `main()` lautet folgendermaßen:

```
int main(int argc, char *argv[])
```

Die Variable `argc` enthält dabei die Anzahl der übergebenen Argumente, während das Array `argv[]` die einzelnen Argumentwerte als Strings aufnimmt. `argv[0]` enthält dabei kein echtes Argument, sondern den Namen des aufgerufenen Programms. Arrays werden im folgenden Abschnitt näher erläutert.

Zeiger und Arrays

Der wichtigste Grund, warum C als schwierig zu erlernen und zu benutzen gilt, ist die Tatsache, dass in dieser Sprache mit *Zeigern* operiert werden kann. Ein Zeiger ist eine spezielle Variable, deren Wert eine Speicheradresse ist. Im Grunde handelt es sich dabei also um eine Art indirekte Variable: eine »normale« Variable ist ein benannter Speicherplatz, in dem unmittelbar ein konkreter Wert abgelegt wird; ein Zeiger verweist dagegen auf den Ort, an dem sich der konkrete Wert befindet.

Zeiger sind unter anderem wichtig, damit Funktionen einander den Speicherort bestimmter Werte mitteilen können, um diese Werte gemeinsam zu manipulieren. Ein Zeiger verweist jeweils auf einen Speicherplatz mit einem bestimmten Datentyp. Er unterscheidet sich von einer Variablen dieses Datentyps durch ein vorangestelltes *:

```
int a;              /* normale int-Variable */
int *b;             /* Zeiger auf int */
```

Der Wert, der einer Zeigervariablen zugewiesen wird, ist normalerweise die Adresse einer anderen Variablen. Diese wird durch den Dereferenzierungsoperator, ein vorangestelltes &, ermittelt. Im folgenden Beispiel wird der Zeigervariablen `a` die Adresse von `b` als Wert zugewiesen:

```
int b = 9;
int *a = &b;        /* a zeigt auf b */
```

Wenn Sie daraufhin versuchen würden, den Wert von `a` selbst auszugeben, wäre das Ergebnis die unvorhersagbare und völlig sinnfreie Nummer einer Speicheradresse. Wenn Sie dagegen den Wert von `*a` ausgeben, erhalten Sie den Inhalt von `b`.

Die interessante Frage ist natürlich, wozu man so etwas überhaupt benötigt. Ein gutes Beispiel ist eine Funktion, die den tatsächlichen Wert einer Variablen ändert, die ihr als Argument übergeben wird. Ein solcher Funktionsaufruf wird als *Call by Reference* bezeichnet, im Gegensatz zur einfachen Wertübergabe, die auch *Call by Value* heißt. Die folgenden beiden Funktionen demonstrieren diesen Unterschied:

```
void doppel1(int a) {
   a *= 2;
}

void doppel2(int *a) {
   *a *= 2;
}
```

Wenn die erste Funktion mit einer Variablen als Argument aufgerufen wird, ändert diese Variable selbst ihren Wert nicht:

```
b = 3;
doppel1(b);         /* Wert von b: 3 */
```

Die andere Funktion wird dagegen mit der Adresse einer Variablen aufgerufen und manipuliert unmittelbar den Inhalt dieser Speicherstelle:

```
b = 3;
doppel2(&b);        /* Wert von b: 6 */
```

Nahe Verwandte der Zeiger sind die *Arrays*. Es handelt sich dabei um Variablen, die mehrere durch einen numerischen Index ansprechbare Werte besitzen. Realisiert werden Arrays durch hintereinanderliegende Speicherstellen, in denen die einzelnen Werte abgelegt werden. Jedes Array lässt sich alternativ durch einen Zeiger auf die Speicherstelle des ersten Elements beschreiben. Die weiteren Elemente können angesprochen werden, indem zu dieser Adresse die Anzahl der Bytes addiert wird, die ein einzelnes Element einnimmt.

Ein Array wird deklariert, indem hinter dem Variablennamen die gewünschte Anzahl von Elementen in eckigen Klammern angegeben wird:

```
int a[10];          /* 10 int-Werte */
```

Die zehn Elemente des Arrays `a[]` werden als `a[0]` bis `a[9]` angesprochen. Alternativ können Sie die Zeiger-Schreibweise wählen: Die Elemente heißen dann `*a` bis `*(a + 9)`.

Sie können einem Array bei der Deklaration auch Anfangswerte zuweisen und dabei die Anzahl der Elemente weglassen, weil sie implizit feststeht:

```
int test[] = {1, 2, 3, 4, 5};
```

Das folgende Beispiel definiert ein Array mit zehn Werten vom Datentyp `int`, die nacheinander vom Benutzer eingegeben werden. Anschließend gibt das Programm das gesamte Array sowie den kleinsten und den größten enthaltenen Wert aus:

```
#include <stdio.h>

int main() {
   int werte[10];
   int ein;
   int i, min, max;
   printf("Bitte zehn Werte zwischen 1 und 100!\n");
   for (i = 0; i < 10; i++) {
      printf("%d. Wert: ", i + 1);
      scanf("%d", &ein);
      werte[i] = ein;
   }
   /* max und min auf das Anfangselement setzen: */
   min = werte[0];
   max = werte[0];
   printf("Ihre Werte: ");
   for (i = 0; i < 10; i++) {
      printf("%d ", werte[i]);
      if (werte[i] > max)
         max = werte[i];
      if (werte[i] < min)
```



```
         min = werte[i];
   }
   printf("\n");
   printf("Kleinster Wert: %d\n", min);
   printf("Größter Wert: %d\n", max);
   return 0;
}
```

Eine der wichtigsten Aufgaben von Arrays besteht darin, den nicht vorhandenen String-Datentyp zu ersetzen. Anstelle eines Strings verwendet C ein Array von `char`-Werten, dessen Ende durch das Zeichen `\0` (ASCII-Code 0) gekennzeichnet wird. Das Byte für diese Endmarkierung müssen Sie bei der Deklaration des `char`-Arrays mit einplanen: Ein `char[10]` ist ein String mit maximal neun nutzbaren Zeichen.

Strukturen

Mitunter ist es nützlich, mehrere Werte verschiedener Datentypen »unter einem gemeinsamen Dach« zu verwalten. Zu diesem Zweck stellt C einen speziellen komplexen Datentyp namens `struct` bereit. In einer *Struktur* können sich beliebig viele Variablen verschiedener Datentypen befinden, was besonders nützlich ist, um komplexe Datenstrukturen zwischen Funktionen hin- und herzureichen.

Das folgende Beispiel definiert eine Struktur namens `person`, die verschiedene Daten über Personen verwaltet:

```
struct person {
   char vorname[20];
   char nachname[30];
   int alter;
};
```

Beachten Sie, dass eine `struct`-Definition mit einem Semikolon enden muss, anders als andere Blöcke in geschweiften Klammern.

Eine Variable dieses Datentyps wird folgendermaßen deklariert:

```
struct person klaus;
```

Wenn Sie die einzelnen Elemente innerhalb einer Strukturvariablen ansprechen möchten, wird dafür die Form `variable.element` verwendet. Hier sehen Sie beispielsweise, wie die soeben definierte Variable `klaus` mit Inhalt versehen wird:

```
klaus.vorname = "Klaus";
klaus.nachname = "Schmitz";
klaus.alter = 42;
```

Oftmals werden Zeiger auf Strukturen als Funktionsargumente eingesetzt. Für die relativ unhandliche Schreibweise `(*strukturvariable).element`, die Sie verwenden müssten, um aus der Funktion heraus auf die Elemente einer Strukturvariablen zuzugreifen, wird die Kurzfassung `strukturvariable->element` definiert. Die folgende Funktion kann beispielsweise aufgerufen werden, um die angegebene Person ein Jahr älter zu machen:

```
void geburtstag(struct person *wer) {
   wer->alter++;
}
```

Der Aufruf dieser Funktion erfolgt beispielsweise so:

```
geburtstag(&klaus);
```

9.1.3 Die C-Standardbibliothek

Wie Sie möglicherweise bemerkt haben, stehen viele Funktionen, die man von einer Programmiersprache erwartet, im bisher besprochenen Sprachkern von C nicht zur Verfügung. Vor allem die Ein- und Ausgabefunktionen sind nicht darin enthalten, weil die Ein- und Ausgabe sich je nach verwendeter Rechnerplattform erheblich unterscheidet. Diese Funktionen werden stattdessen in externen Dateien zur Verfügung gestellt. Es handelt sich dabei um vorkompilierte Bibliotheksdateien, die vom Compiler mit dem eigenen Programmcode verknüpft werden. Die Schnittstellen der Bibliothek sind in sogenannten *Header-Dateien* definiert, die über die Präprozessor-Direktive `#include` eingebunden werden.

Die Laufzeitbibliothek der Programmiersprache C ist je nach Hardwareplattform und Betriebssystem unterschiedlich aufgebaut. Allerdings definiert der ANSI-Standard der Sprache eine Reihe vorgeschriebener Bibliotheksfunktionen, die von jeder beliebigen C-Implementierung unterstützt werden. Die Gesamtheit dieser standardisierten Funktionen wird als *C-Standardbibliothek* bezeichnet. Da so gut wie alle Betriebssysteme zu großen Teilen in C geschrieben sind, wird ihr Verhalten in erheblichem Maße von dieser Standardbibliothek beeinflusst.

Die Standardbibliothek besteht aus einer Reihe thematisch gegliederter Header-Dateien. Drei der wichtigsten werden im vorliegenden Abschnitt kurz vorgestellt.

Ein- und Ausgabe: stdio.h

In dieser wichtigsten aller Bibliotheksdateien, *stdio.h*, sind sämtliche Ein- und Ausgabefunktionen der Programmiersprache C zusammengefasst. Viele dieser Funktionen betreffen die Standardeingabe und Standardausgabe, also die Eingabe über die Tastatur und die Ausgabe auf der Konsole – falls sie nicht auf Dateien umgeleitet wurden. Andere Funktionen beschäftigen sich mit dem Öffnen, Lesen oder Schreiben von Dateien.

- `puts(String-Ausdruck)`
  Die Funktion `puts()` schreibt den Wert des angegebenen String-Ausdrucks auf die Standardausgabe, gefolgt von einem Zeilenumbruch.
- `printf(Format, Wert1, Wert2, ...)`
  Auch diese Funktion dient der Ausgabe von Inhalten auf die Konsole. Das erste Argument ist ein String mit Formatplatzhaltern, die für die anschließend aufgelisteten Werte stehen. Die wichtigsten Formatplatzhalter sind `%s` für einen String, `%d` für einen Integer und `%f` für Fließkommawerte.
- `sprintf(String-Variable, Format, Wert1, Wert2, ...)`
  `sprintf()` funktioniert genauso wie `printf()` – allerdings wird der formatierte String nicht ausgegeben, sondern in der als erstes Argument übergebenen String-Variablen gespeichert.
- `scanf(Format, Adresse)`
  `scanf()` dient der Eingabe eines Wertes über die Standardeingabe (meist Tastatur); der eingegebene Wert wird unter der angegebenen Speicheradresse abgelegt. Die Adresse wird in der Regel durch Dereferenzierung einer Variablen (vorangestelltes `&`) angegeben, um die Eingabe in der entsprechenden Variablen zu speichern. Die Formatangabe besteht normalerweise nur aus einem einzelnen Formatplatzhalter (siehe `printf()`).
- `gets(Variable)`
  Mithilfe von `gets()` wird ein String von der Standardeingabe gelesen und in der angegebenen Variablen gespeichert.
- `getchar()`
  Diese Funktion liest ein einzelnes Zeichen von der Standardeingabe. Beachten Sie, dass die Eingabe mit den Mitteln der Standardbibliothek dennoch immer zeilenorientiert verläuft: Sie können zwar in einer Schleife einzelne Zeichen einlesen, erhalten aber erst bei einem Zeilenende (wenn der Benutzer ↵ drückt) ein Ergebnis. Echte zeichenorientierte Eingabe ist eine Angelegenheit plattformabhängiger Bibliotheken.
- `fopen(Dateiname, Modus)`
  Diese Funktion öffnet eine Datei auf einem Datenträger. Damit Sie auf diese Datei zugreifen können, wird das Funktionsergebnis von `fopen()` einer Variablen vom Typ `FILE` zugewiesen – der Wert ist ein eindeutiger Integer, der als *Dateideskriptor* oder *Dateihandle* bezeichnet wird. Der Dateiname kann ein beliebiger Pfad im lokalen Dateisystem sein. Beachten Sie unter Windows lediglich, dass das Pfadtrennzeichen `\` in einem C-String verdoppelt werden muss, weil es normalerweise Escape-Sequenzen wie `\n` einleitet. Der Modus kann unter anderem eines der Zeichen `"r"` (lesen), `"w"` (schreiben) oder `"a"` (anfügen) sein. Beispiel:
-
```
fh = fopen ("test.txt", "r");
                    /* test.txt zum Lesen öffnen */
```
- `fclose(Dateideskriptor)` schließt die angegebene Datei.

- `fputs(Deskriptor, String-Ausdruck)` schreibt den Wert des übergebenen String-Ausdrucks mit anschließendem Zeilenumbruch in die gewünschte Datei.
- `fprintf(Deskriptor, Format, Werte)` besitzt dieselbe Syntax wie `printf()`, schreibt aber in die angegebene Datei.
- `fscanf(Deskriptor, Format, Variable)` funktioniert wie `scanf()`, liest aber aus der angegebenen Datei.
- `fgets(Variable, Zeichenzahl, Deskriptor)` liest einen String aus der angegebenen Datei mit der entsprechenden maximalen Zeichenzahl oder bis zum ersten Zeilenumbruch.
- `fflush(Deskriptor)` leert den Puffer eines Eingabedatenstroms. Dies ist manchmal vor Eingabeoperationen wie `fscanf()` erforderlich, damit diese nicht »automatisch« aus der vorherigen Eingabe bedient werden. Für die Standardeingabe können Sie übrigens `fflush(stdin)` schreiben.

String-Funktionen: string.h

Die Header-Datei *string.h* enthält verschiedene Funktionen zur String-Manipulation und -Analyse. Zu den wichtigsten gehören folgende:

- `strcmp(String1, String2)` vergleicht die beiden angegebenen Strings miteinander. Das Ergebnis ist 0, wenn sie gleich sind, negativ, wenn `String1` alphabetisch vor `String2` kommt, und positiv, wenn es umgekehrt ist. Die Variante `stricmp()` unterscheidet nicht zwischen Groß- und Kleinschreibung.
- `strncmp(String1, String2, n)` und `strnicmp(String1, String2, n)` vergleichen nur die ersten `n` Zeichen der beiden Strings, wiederum mit beziehungsweise ohne Rücksicht auf Groß- und Kleinschreibung.
- `strcpy(String1, String2)` kopiert den Wert von `String2` an die Adresse von `String1`. Dies ist die einzige korrekte Methode, um einer String-Variablen den Wert einer anderen zuzuweisen!
- `strcat(String1, String2)` hängt den Wert von `String2` an das Ende von `String1` an.

Überprüfen Sie stets alle String-Längen!

Sie müssen selbst vor jeder String-Operation die Länge der beteiligten Zeichenketten überprüfen, insbesondere bei Eingaben von Benutzern oder aus anderen unsicheren Quellen: Da diese Funktionen keinen integrierten Schutz vor Überläufen bieten, sind sie eines der bevorzugten Angriffsziele für Cracker.

Datum und Uhrzeit: time.h

Die Header-Datei *time.h* definiert verschiedene Funktionen für die Arbeit mit Datum und Uhrzeit:

- `time(NULL)` fragt die aktuelle Systemzeit ab und liefert sie als Wert vom Typ `time_t` zurück. Als Argument in den Klammern wird eigentlich ein Zeiger auf `time_t` erwartet; da das Ergebnis aber bereits die Zeit enthält, wird in der Regel der spezielle Wert `NULL` (Zeiger auf gar nichts!) übergeben.
- `localtime(*Zeitangabe)` wandelt die Rückgabe von `time()` – die Sekunden seit EPOCH – in eine vorformatierte Ortszeit um. Das Argument ist ein Zeiger auf `time_t`, der Rückgabewert eine komplexe Struktur namens `struct tm`. Oft wird `localtime()` nur als »Zwischenwert« für `strftime()` verwendet.
- `strftime(String, Zeichenzahl, Format, *localtime-Wert)` formatiert die Zeitangabe nach der Vorschrift des angegebenen Formats und speichert das Ergebnis in der String-Variablen ab, die als erstes Argument vorliegt. Die Formatangaben entsprechen dem in Kapitel 7, »Linux«, besprochenen Unix-Befehl `date`. Das folgende Beispiel liest das aktuelle Datum und gibt es formatiert aus:

```
time_t jetzt;
char zeit[20];
...
jetzt = time(NULL);
strftime(zeit, 19, "%d.%m.%Y, %H:%M",
     localtime(&jetzt));
printf("Heute ist der %s.\n", zeit);
```

  Die Ausgabe lautet beispielsweise folgendermaßen:

```
Heute ist der 01.05.2015, 14:47.
```

- `difftime (Zeitangabe1, Zeitangabe2)` gibt die Differenz zwischen zwei `time_t`-Zeitangaben in Sekunden an.

Der Präprozessor

Formal hat der *Präprozessor* zwar nichts mit der Standardbibliothek zu tun, wird aber trotzdem hier kurz angeschnitten, weil er unter anderem für das Einbinden der Header-Dateien mithilfe von `#include` zuständig ist. Viele C-Programme bestehen aus mehr Präprozessor-Direktiven als aus gewöhnlichen Anweisungen, weil der Präprozessor die Definition bestimmter Abkürzungen ermöglicht.

Die wichtigste Präprozessor-Direktive haben Sie bereits kennengelernt: `#include` bindet eine Header-Datei ein, die Schnittstellendefinition einer Bibliothekskomponente. Sie können auch eigene häufig genutzte Funktionen in selbst geschriebene Header-Dateien auslagern, müssen dabei aber Folgendes beachten: `#include <Datei>` sucht in den standardisierten Include-Verzeichnissen Ihres Compilers oder Betriebssystems nach der angegebenen Header-Datei. Wenn Sie auf eine Datei im Verzeichnis des C-Programms selbst verweisen möchten, wird stattdessen die Schreibweise `#include "Datei"` verwendet.

Eigene Header-Dateien schreiben

Wenn Sie eigene Header-Dateien schreiben möchten, um Programmfunktionalität auszulagern, beachten Sie Folgendes: In der Header-Datei mit der Endung *.h* werden Funktionen nur deklariert, es werden also nur ihre Köpfe hineingeschrieben und mit einem Semikolon abgeschlossen. Die Implementierung erfolgt dagegen in einer ansonsten gleichnamigen Datei mit der Endung *.c*.

Angenommen, in einer Datei namens *func.h* steht folgende Deklaration:

```
int is_prime(int number);
```

Die Funktion soll überprüfen, ob das Argument `number` eine Primzahl ist oder nicht, und entsprechend 1 für wahr beziehungsweise 0 für falsch zurückgeben. Eine mathematisch primitive Implementierung könnte so aussehen und in der Implementierungsdatei *func.c* stehen:

```
int is_prime(int number) {
  if (number == 0 || number == 1) {
    return 0;
  }
  if (number == 2) {
    return 1;
  }
  for (int i = 2; i <= number / 2; i++) {
    if (number % i == 0) {
      return 0;
    }
  }
  return 1;
}
```

In der Programmdatei, beispielsweise *main.c*, können Sie dann Ihre Header-Datei einbinden und die Funktion aufrufen. Hier eine sehr kurze Beispielimplementierung, die überprüft, ob ein vom Benutzer eingegebener Integer eine Primzahl ist oder nicht:

```
#include <stdio.h>
#include "func.h"

int main(int argc, char* argv[]) {
  int input;
  printf("Ganze Zahl: ");
  scanf("%d", &input);
  if (is_prime(input)) {
    printf("%d ist eine Primzahl.\n", input);
  } else {
    printf("%d ist keine Primzahl.\n", input);
  }
```



```
  return 0;
}
```

Beim Kompilieren müssen Sie die beiden *.c*-Dateien angeben. Beispiel:

```
$ gcc -o main main.c func.c
```

Eine weitere wichtige Funktion des Präprozessors ist die Definition symbolischer Konstanten mithilfe der Direktive `#define`. Diese werden vor allem verwendet, um konstante Werte tief im Inneren des Programms zu vermeiden, wo sie sich später nur schwer auffinden und ändern lassen. Angenommen, Sie möchten in Ihrem Programm den Umrechnungsfaktor von DM nach € verwenden, dann können Sie ihn folgendermaßen als symbolische Konstante festlegen:

```
#define DM 1.95583
```

In Ihrem Programm wird nun jedes Vorkommen von DM noch vor der eigentlichen Kompilierung durch 1.95583 ersetzt – außer innerhalb der Anführungszeichen von String-Literalen. Beachten Sie, dass am Ende einer Konstantendefinition kein Semikolon stehen darf.

Diese Fähigkeit des Präprozessors wird auch oft zur bedingten Kompilierung eingesetzt: Die Direktive `#ifdef` fragt ab, ob die angegebene symbolische Konstante existiert, und kompiliert nur in diesem Fall alle Zeilen bis zum Auftreten von `#endif`. Dies wird zum Beispiel zur Unterscheidung verschiedener Rechnerplattformen verwendet. Im folgenden Beispiel wird eine zusätzliche Anweisung mitkompiliert, falls eine symbolische Konstante namens `DEBUG` definiert ist:

```
#ifdef DEBUG
printf("Debug-Modus aktiviert.\n");
#endif
```

Die gegenteilige Variante `#ifndef` (nicht definiert) wird häufig verwendet, um zu überprüfen, ob eine bestimmte Header-Datei bereits irgendwo in den Programmdateien eines Projekts eingebunden wurde. Dazu wird der gesamte Deklarationsblock von `#ifndef` und `#endif` umschlossen, und die Konstante wird darin – oft ohne konkreten Wert – definiert. Eine solche Version der eben beschriebenen Header-Datei *func.h* sähe wie folgt aus:

```
#ifndef FUNCTIONS

#define FUNCTIONS

int is_prime(int number);

#endif
```

9.2 Java

Die Programmiersprache Java wurde 1995 von dem bekannten Server- und Workstation-Hersteller *Sun Microsystems* vorgestellt. Zu den wichtigsten Entwicklern des Projekts gehören *James Gosling* und *Bill Joy*. Java hat vor allen Dingen die Besonderheit, dass der kompilierte Programmcode auf verschiedenen Rechnern und Betriebssystemen ausgeführt werden kann. Für diese Systeme ist lediglich Java-Unterstützung in Form einer *virtuellen Maschine* (JVM – Java Virtual Machine) erforderlich. Virtuelle Java-Maschinen sind für zahlreiche verschiedene Plattformen verfügbar, unter anderem für Windows, Linux, OS X und alle anderen Unix-Varianten. Dieser Ansatz wurde von Sun als *Write once, run everywhere* bezeichnet.

Im Jahr 2009 wurde Sun vom Datenbankkonzern *Oracle* übernommen. Die Bedeutung dieses Ereignisses für die Zukunft von Java war eine Zeit lang unklar, aber da auch Oracle schon vor der Übernahme von Sun massiv in Java-Technologien investiert hat, erscheint diese Zukunft auch weiterhin gewährleistet.

Anfangs wurde Java vor allen Dingen eingesetzt, um sogenannte *Applets* zu schreiben. Dabei handelt es sich um kleine Java-Programme, die in einem Webbrowser ausgeführt werden, der eine JVM enthält. Da für Multimedia-Angebote im Web inzwischen erheblich bessere Lösungen verfügbar sind, hat sich der Schwerpunkt der Java-Verwendung auf andere Bereiche verlagert: Java-Anwendungen werden im professionellen Serverbereich eingesetzt, weil sie aufgrund ihrer Plattformunabhängigkeit mit den verschiedensten Systemen kooperieren können; auch zur Entwicklung von Desktop-Anwendungen wie Office- oder Grafikprogrammen wird Java gern genutzt.

Wenn Sie Java-Programme schreiben möchten, brauchen Sie das *Java Software Development Kit* (*Java SDK* oder kurz *JDK*), das Sie unter *http://www.oracle.com/technetwork/java/index.html* für mehrere Plattformen herunterladen können. Falls Ihr System dort nicht zu finden ist, hilft in der Regel eine Suchmaschine wie etwa Google weiter.

Das Java SDK wird in drei verschiedenen Varianten angeboten:

- Die *Standard Edition (Java SE)*, auf die sich dieser Abschnitt bezieht, ist für sämtliche Desktop-Anwendungen geeignet. Die aktuelle Version ist 8 Update 51.
- Die *Enterprise Edition* (*Java EE*) enthält Unterstützung für die Entwicklung verteilter, datenbankgestützter Serveranwendungen; es handelt sich um eine Sammlung von Standards, die kompatible Server bereitstellen müssen. Einzelne Aspekte von Java EE, nämlich Datenbankzugriffe und XML-Verarbeitung, werden in Kapitel 13, »Datenbanken«, beziehungsweise Kapitel 16, »XML«, vorgestellt.
- Die *Micro Edition* (*Java ME*) dient der Entwicklung von Anwendungen für mobile Geräte wie Smartphones oder Tablets sowie für Embedded Systems. Insbesondere Android-Apps werden vorwiegend in Java geschrieben, allerdings in der Regel mit Googles eigenem Android SDK (Näheres dazu erfahren Sie in Kapitel 11, »Mobile Development«).

Die Installation des JDK SE ist sehr einfach: Sie laden einfach ein für Ihre Plattform geeignetes Archiv oder ausführbares Programm herunter, das sich in den meisten Fällen per Doppelklick installieren lässt. Das einzige kleine Problem besteht darin, dass Sie zwei Umgebungsvariablen setzen beziehungsweise anpassen müssen, um mit Java arbeiten zu können. Wie Systemvariablen unter verschiedenen Betriebssystemen manipuliert werden, haben Sie bereits in Kapitel 6, »Windows«, und Kapitel 7, »Linux«, erfahren.

Um den Java-Compiler und die JVM von überall her aufrufen zu können, müssen Sie das Unterverzeichnis *bin* Ihrer Java-Installation zur Umgebungsvariablen `PATH` hinzufügen. Außerdem müssen Sie eine weitere Systemvariable namens `CLASSPATH` einrichten, die auf die Java-Klassenbibliothek verweist – in der Regel das Verzeichnis *lib* innerhalb der Java-Installation. Zu `CLASSPATH` werden, durch Semikolon getrennt, auch andere Verzeichnisse (oder Archivdateien wie *.zip* oder *.jar*) hinzugefügt, in denen weitere Java-Klassen enthalten sind. Es empfiehlt sich, auch `.` als Synonym für das aktuelle Arbeitsverzeichnis hinzuzufügen, damit auch Ihre eigenen Java-Programme überall gefunden werden.

Der wichtigste Unterschied zwischen Java und C besteht darin, dass Java *objektorientiert* ist, während C zu den *prozeduralen* oder *imperativen* Programmiersprachen zählt (zur Systematik siehe Kapitel 1, »Einführung«). Das *objektorientierte Programmieren* (*OOP*) ist ein modernerer Ansatz der Softwareentwicklung. Während in imperativen Sprachen wie C zunächst eine Datenstruktur entwickelt wird, die mit den Programmfunktionen nicht näher verbunden ist, bilden Funktionen und Datenstrukturen bei der OOP eine untrennbare Einheit.

Ein Java-Programm ist immer eine *Klassendefinition*. Eine *Klasse* ist das übergeordnete Bauelement der Objektorientierung. Sie besteht aus einer Reihe von Variablen, die als *Eigenschaften* oder *Attribute* bezeichnet werden, und aus mehreren Funktionen, die *Methoden* genannt werden. Durch die *Kapselung* der Eigenschaften und Methoden zu Klassen lassen sich die verschiedenen Bestandteile der realen Welt oder der zu verwaltenden Daten realitätsnäher nachbilden: Beispielsweise müsste die imperative Nachbildung eines Autos durch externe Funktionen zum »Fahren« (Änderung von Daten wie Tankfüllung oder Kilometerstand) gebracht werden. Ein »objektorientiertes Auto« enthält dagegen einfach eine entsprechende Methode, die aufgerufen wird, sodass das Auto »selbst fährt«.

Die Klassendefinition selbst ist lediglich eine Vorlage für die Erzeugung konkreter *Objekte*. Es können beliebig viele Objekte einer Klasse erzeugt werden; jedes dieser Objekte besitzt alle Eigenschaften, die in der Klasse definiert wurden. Objekte werden auch als *Instanzen* einer Klasse bezeichnet.

Einer der wichtigsten Vorteile der Kapselung besteht darin, dass Detaildaten immer nur an einer Stelle verwaltet und verändert werden, was die Häufigkeit von Fehlern erheblich verringert. Außerhalb einer bestimmten Klasse dürfen die Werte von Daten, die Sie »nichts angehen«, nicht manipuliert werden. Um bei dem Auto-Beispiel zu bleiben, sollte beispielsweise der Kilometerstand nur durch die offizielle Methode »Fahren« geändert werden und nicht durch eine direkte Wertzuweisung.

Seit ihrer Entwicklung in den 70er-Jahren des letzten Jahrhunderts wurde die OOP in vielen verschiedenen Programmiersprachen realisiert. Eine der ersten von ihnen war *Smalltalk*, die das Konzept erstmalig bekannt machte. Die verbreitetsten objektorientierten Sprachen sind heute C++, eine objektorientierte Erweiterung von C, Java und C# (»C sharp«) von Microsoft. Da C++ nicht konsequent objektorientiert ist, um die Kompatibilität mit C aufrechtzuerhalten, wird an dieser Stelle die Sprache Java besprochen, die sich zwar an C++ anlehnt, aber einen Großteil des imperativen Erbes von C weglässt. C# ist laut Angabe von Microsoft ebenfalls eine Weiterentwicklung von C++, erinnert aber in vielen Aspekten eher an Java.

Dies ist nur der Anfang

Dieser Abschnitt ist nur eine kurze Einführung in die Sprachgrundlagen von Java. Was diese Sprache besonders leistungsfähig macht, sind die Unmengen von mitgelieferten und auch separat erhältlichen Klassenbibliotheken. In verschiedenen späteren Kapiteln lernen Sie einige von ihnen kennen, beispielsweise zur Grafikprogrammierung, für Datenbankzugriffe oder auch zur XML-Verarbeitung.

9.2.1 Grundlegende Elemente der Sprache Java

Um einen Einstieg in die Java-Programmierung zu finden, sehen Sie hier gleich das erste Beispielprogramm. Es handelt sich dabei um die Java-Entsprechung des ersten C-Beispielprogramms, was die Unterschiede deutlich macht. Geben Sie das Programm in Ihren bevorzugten Texteditor ein, und speichern Sie es unter *Hallo.java*. Eine Java-Quellcode-Datei muss so heißen wie die Klasse, die darin definiert wird, und die Endung *.java* aufweisen. Dabei müssen Sie die Groß- und Kleinschreibung des Klassennamens genau übernehmen; üblicherweise beginnen Klassennamen mit einem Großbuchstaben. Hier das Listing:

```
import java.io.*;

public class Hallo {
   public static void main(String args[]) {
      BufferedReader eingabe = new BufferedReader(
           new InputStreamReader (System.in)
      );
      String name = "";
      System.out.println("Hallo Welt!");
      System.out.print("Ihr Name, bitte: ");
      try {
         name = eingabe.readLine();
      } catch(IOException e) {}
      System.out.println("Hallo " + name + "!");
   }
}
```


Wenn Sie zuvor Ihre `path`- und `CLASSPATH`-Einstellungen korrekt vorgenommen haben, können Sie das Programm nun folgendermaßen kompilieren, sofern Sie sich im entsprechenden Verzeichnis befinden:

```
$ javac Hallo.java
```

Falls Sie keine Fehlermeldung erhalten, finden Sie im aktuellen Verzeichnis die fertig kompilierte Datei *Hallo.class* vor. Sie können das Programm anschließend folgendermaßen starten:

```
$ java Hallo
```

Das ausführbare Programm `java` aktiviert die JVM. Als Kommandozeilenargument wird der Name einer kompilierten Java-Klasse ohne Dateiendung angegeben.

Wie Sie sehen, ist dieses Programm erheblich komplexer als die C-Version, obwohl es dieselbe Aufgabe erfüllt. Im Folgenden wird das Programm zeilenweise erläutert:

- `import java.io.*;`
  Mithilfe der Anweisung `import` werden bestimmte Teile der Java-Klassenbibliothek importiert. Der Vorgang ist vergleichbar mit dem Einbinden von Header-Dateien mithilfe von `#include`. Allerdings ist `import` eine normale Java-Anweisung; einen Präprozessor gibt es nicht. Außerdem unterscheiden sich die importierten Klassen formal nicht von Ihren eigenen Programmen. Eine Aufteilung in Header- und Programmdatei ist in Java ebenfalls nicht vorgesehen.

  `java.io.*` bezeichnet alle Klassen, die im Verzeichnis `io` der grundlegenden Klassenbibliothek liegen. Es handelt sich um eine Sammlung von Klassen für die Ein- und Ausgabe. Solche Bestandteile der Klassenbibliothek werden als *Packages* bezeichnet.

  Anstatt alle Klassen (*) eines Namensraums zu importieren, können Sie auch einzelne Klassen importieren – im obigen Fall würde die Anweisung `import java.io.*` durch die folgenden drei Zeilen ersetzt:

```
import java.io.InputStreamReader;
import java.io.FileReader;
import java.io.IOException;
```

Namensräume in Java

Die durch Punkte getrennten Namensräume und Unternamensräume werden übrigens durch die Verzeichnisstruktur des Quellcodes vorgegeben. Die Konvention für eigene Namensräume besagt, dass der umgekehrte eigene Domainname die Wurzel bildet, darunter folgen der Projektname und ganz unten die verschiedenen Teilaspekte des Projekts. Angenommen, der Domainname lautet *lingoworld.de*, das Projekt heißt *javatutorial* und der aktuelle Bereich *examples*. Dann würden sich die entsprechenden Klassen im Unterverzeichnis *de/lingoworld/javatutorial/examples* des Quellcode-Verzeichnisses befinden, und der Namensraum hieße `de.lingoworld.javatutorial.examples`.

- `public class Hallo`
  Mit dem Schlüsselwort `class` wird eine Klassendefinition eingeleitet. Mithilfe von `public` wird die definierte Klasse für sämtliche anderen Klassen, also für alle Programme, zugänglich.
- `public static void main(String args[])`
  Wie in C steht `main()` für das Hauptprogramm. Allerdings gibt es in einem Java-Programm einige wichtige formale Unterschiede: Da die JVM diese Methode von außen aufrufen muss, wird das Schlüsselwort `public` benötigt. Alle Bestandteile einer Klasse, die nicht `public` sind, stehen außerhalb der Klasse selbst nicht zur Verfügung. `static` wird benötigt, weil von dieser Klasse kein konkretes Objekt abgeleitet wird. Eine Klasse ist eigentlich nur eine Art Bauanleitung für Objekte; ihre `static`-Bestandteile können aber ohne Existenz eines konkreten Objekts verwendet werden. In der Regel hat `main()` in Java den Datentyp `void`. Das Array `args[]` nimmt Kommandozeilenargumente entgegen.
- `BufferedReader eingabe = new BufferedReader(`
  `new InputStreamReader(System.in));`
  Diese verschachtelte Operation sorgt für ein Objekt namens `eingabe`, das zeilenweise von der Standardeingabe lesen kann. Im Kern von Java ist leider keine Möglichkeit gegeben, ganze Zeilen einzulesen. Deshalb wird die Standardeingabe – repräsentiert durch `System.in` – in ein Objekt vom Typ `InputStreamReader` (Lesevorrichtung für kontinuierliche Datenströme) verpackt, das wiederum von einem `BufferedReader` (zuständig für die Zwischenspeicherung von Eingabedaten) umhüllt wird. Solche verschachtelten Objektkonstruktionen machen Java relativ kompliziert, aber durch die große Auswahl spezialisierter Klassen auch sehr flexibel.
- `String name = "";`
  Im Gegensatz zu C stellt Java einen echten String-Datentyp zur Verfügung, der als Klasse realisiert ist. Diese Anweisung erzeugt eine String-Variable mit der Bezeichnung `name`; ihr wird als Anfangswert der leere String zugewiesen, weil es beim späteren Eingabeversuch passieren könnte, dass sie gar keinen Wert erhält.
- `System.out.println("Hallo Welt!");`
  Die Methode `println()` der Standardausgabe (`System.out`) gibt einen übergebenen String mit anschließendem Zeilenumbruch aus.
- `System.out.print("Ihr Name, bitte: ");`
  Die Methode `print()` gibt dagegen einen String ohne Zeilenumbruch aus.
- `try {...}`
  In einen `try`-Block werden Anweisungen immer dann geschrieben, wenn sie einen möglichen Fehler produzieren könnten. Laufzeitfehler werden in Java als sogenannte *Ausnahmen* (Exceptions) betrachtet, die mithilfe von `try/catch` (siehe unten) abgefangen und sinnvoll behandelt werden können.

- `name = eingabe.readLine();`
  Die Methode `readLine()` der Klasse `BufferedReader` liest eine Zeile aus einem Eingabestrom, in diesem Fall von der Standardeingabe.
- `catch(IOException e) {}`
  Mithilfe von `catch()` kann eine Ausnahme abgefangen werden, die innerhalb des vorangegangenen `try`-Blocks ausgelöst wurde. In diesem Fall handelt es sich um eine `IOException`, also einen Ein-/Ausgabefehler. In den geschweiften Klammern kann Code für spezielle Maßnahmen zur Fehlerbehandlung stehen. Die wichtigste Aufgabe von `try/catch` besteht darin, den sofortigen Programmabbruch bei einem Fehler zu verhindern.
- `System.out.println("Hallo " + name + "!");`
  Das Besondere an dieser Ausgabeanweisung ist die Verkettung mehrerer Strings durch den Operator `+`.

Unterschiede zu C

Erfreulicherweise müssen viele grundlegende Konzepte von Java nicht mehr erläutert werden, weil sie mit C übereinstimmen. Beispielsweise sind die einfachen Datentypen, Ausdrücke, Operatoren und Kontrollstrukturen nahezu identisch, deshalb werden an dieser Stelle nur die wesentlichen Unterschiede aufgezählt. Hier ist nicht die Tatsache gemeint, dass Java objektorientiert ist, sondern es geht nur um die Unterschiede bei vergleichbaren Elementen.

- Im Gegensatz zu C haben die Integer-Datentypen in Java eine festgelegte Bit-Breite: `int` ist 32 Bit groß, `short` 16 Bit, `long` 64 Bit.
- Java definiert einen separaten Datentyp für 8-Bit-Integer namens `byte`; `char` wird dagegen nur zur Darstellung eines einzelnen Zeichens verwendet und besitzt eine Breite von 16 Bit, um die wichtigsten Unicode-Zeichen darzustellen.
- Java kennt einen Datentyp für boolesche Wahrheitswerte: `boolean`. Er kann nur die vorgegebenen Wahrheitswerte `true` oder `false` annehmen. Sämtliche Bedingungsprüfungen für Kontrollstrukturen müssen in Java einen `boolean`-Wert haben. Wenn Sie also beispielsweise prüfen möchten, ob die Variable `a` den Wert 0 hat, müssen Sie den Vergleich `a == 0` explizit hinschreiben! Die typische C-Kurzschreibweise `!a` ist nicht zulässig. Aus diesem Grund wird Java als *stark typisierte Sprache* bezeichnet, weil sehr streng über die Einhaltung der korrekten Datentypen gewacht wird.
- Die Syntax für die Deklaration eines Arrays mit einer festgelegten Anzahl von Elementen unterscheidet sich von C: Es erfolgt ein Aufruf der Objekterzeugungsanweisung `new`, gefolgt vom Datentyp und der Elementanzahl:

  ```
  int werte[] = new int[20];
  ```

- Variablen können in Java an einer beliebigen Stelle deklariert werden. Beachten Sie allerdings, dass eine Variable, die innerhalb des Anweisungsblocks einer Kontrollstruktur

deklariert wird, nur in diesem Block gilt. Beispielsweise produziert der folgende Code einen Fehler:

```
for (int i = 0; i < 10; i++) {
   ...
}
System.out.println(i);
                    // i existiert hier nicht mehr!
```

- Der einzige wichtige Operator, den Java zusätzlich zu C definiert, ist der String-Verkettungsoperator + (siehe entsprechendes Beispiel zuvor). Er ist zwar praktischer als die umständliche C-Funktion `strcat()`, aber dafür sorgt er mitunter für Verwirrung, weil er mit dem arithmetischen + verwechselt werden kann. Beispielsweise gibt die folgende Anweisung `Summe: 35` aus:

  ```
  System.out.println("Summe: " + 3 + 5);
  ```

- Andere Operationen mit Strings werden über Methoden der Klasse `String` realisiert. Jeder String-Ausdruck ist automatisch eine Instanz dieser Klasse, auch ohne formale objektorientierte Instanzerzeugung.

  Sie können zum Beispiel zwei Strings miteinander vergleichen, indem Sie `string1.equals(string2)` aufrufen; `string1` und `string2` können dabei Variablen, Literale oder Ausdrücke mit dem Datentyp `String` sein. Eine Variante ist `string1.equalsIgnoreCase(string2)` für einen Vergleich ohne Berücksichtigung von Groß- und Kleinschreibung. Das Ergebnis ist jeweils `true`, wenn die Strings gleich sind, oder `false`, wenn sie verschieden sind.

  Einen allgemeineren Vergleich von Strings – analog zur C-Funktion `strcmp()` – bietet die Methode `string1.compareTo(string2)`. Wie dort ist das Ergebnis kleiner als 0, wenn `string1` im Zeichensatz vor `string2` steht, 0, wenn die beiden Strings gleich sind, und größer als 0, wenn `string1` nach `string2` kommt.

  Weitere interessante String-Methoden sind folgende:

  – `string1.charAt(pos)` gibt das Zeichen an der Position `pos` zurück (Positionen beginnen bei 0). Zum Beispiel gibt `"Köln".charAt(1)` das `'ö'` zurück. Beachten Sie, dass der Datentyp des Rückgabewerts `char` und nicht `String` ist.

  – `string1.substring(anfang, ende)` liefert die Zeichen von der Position `anfang` bis ausschließlich `ende` zurück. Beispielsweise ergibt `"Köln".substring(1, 3)` den String `"öl"`.

  – `string1.indexOf(ch)` gibt die erste Position in `string1` zurück, an der das Zeichen `ch` (Typ `char`) vorkommt, oder `-1`, wenn `ch` gar nicht vorkommt. Eine alternative Form sucht nach dem Vorkommen eines Teil-Strings: `string1.indexOf(string2)`. `lastIndexOf()` gibt dagegen die Position des letzten Vorkommens des gesuchten Zeichens oder Teil-Strings zurück.

  – `string1.length()` liefert die Länge des Strings in Zeichen.

- Im Unterschied zu C verwendet Java keine Zeiger. Dies entfernt eine der wichtigsten Quellen für Fehler und Sicherheitsprobleme aus der Sprache. Für einen Call by Reference werden stattdessen Objektreferenzen verwendet (mehr darüber erfahren Sie in den nächsten Abschnitten).
- In Java gibt es zusätzliche Arten von Kommentaren. Der mit C++ eingeführte einzeilige Kommentar beginnt mit // und reicht bis zum Ende der Zeile. Daneben existiert der spezielle JavaDoc-Kommentar, der mit /** beginnt und mit */ endet. Das mit dem JDK gelieferte Programm `javadoc` kann aus diesen Kommentaren automatisch eine Programmdokumentation generieren.

9.2.2 Objektorientierte Programmierung mit Java

Sämtliche Klassen der Java-Klassenbibliothek und alle, die Sie selbst definieren, stammen direkt oder indirekt von der Klasse `Object` ab. `Object` gehört zum Kern-Package `java.lang`, das nicht mithilfe von `import` eingebunden werden muss. Wie der Name schon sagt, stellt `lang` den Sprachkern von Java zur Verfügung, darunter wichtige Klassen wie `String`, `Math` (mathematische Konstanten und Funktionen) oder `System` (wichtige Betriebssystemschnittstellen).

Objektorientierte Programmierung lässt sich am besten anhand eines Beispiels veranschaulichen. Da das eingangs erwähnte Auto-Beispiel in fast jedem Buch über OOP verwendet wird, soll hier zur Abwechslung ein anderes zum Einsatz kommen: Es werden die verschiedenen Arten von Personen in einer Ausbildungsumgebung modelliert. Die grundlegende Klasse, von der alle anderen abgeleitet werden, heißt `Person` und definiert diejenigen Eigenschaften, die alle beteiligten Personen gemeinsam haben:

```
public class Person {
   // Eigenschaften:
   private String name;
   private String vorname;
   private int alter;

   // Konstruktor:
   public Person(String n, String v, int a) {
      this.name = n;
      this.vorname = v;
      this.alter = a;
   }

   // Methoden:
   public void geburtstag() {
      this.alter++;
   }
```

```
   public String getName() {
      return this.vorname + " " + this.name;
   }

   public int getAlter() {
      return this.alter;
   }
}
```

Speichern Sie diese Klassendefinition zunächst unter *Person.java*. Sie lässt sich ohne Weiteres kompilieren, aber natürlich nicht ausführen, da sie wegen der fehlenden `main()`-Methode kein ausführbares Programm ist.

Wie Sie möglicherweise bemerkt haben, handelt es sich bei diesem Beispiel um die Java-Entsprechung der C-Struktur, in der ebenfalls Personendaten abgelegt wurden. In Java gibt es `struct` übrigens nicht, weil es sich dabei aus der Sicht der OOP lediglich um den Sonderfall einer Klasse ohne Methoden handelt.

Die einzelnen Bestandteile der Klassendefinition werden nun kurz erläutert:

- Als Erstes werden die verschiedenen Eigenschaften der Klasse deklariert. Sie haben die Geheimhaltungsstufe `private`, sind also außerhalb eines Objekts dieser Klasse nicht sichtbar. Innerhalb der Methoden der Klasse können die Eigenschaften mit einem vorangestellten `this.` angesprochen werden; Sie können es aber auch weglassen, solange die Bezeichner eindeutig sind. `this` repräsentiert während der Ausführung der Konstruktoren und Methoden einer Klasse die aktuelle Instanz selbst.
- Der *Konstruktor* ist eine spezielle Methode, die immer dann aufgerufen wird, wenn eine Instanz der Klasse erzeugt wird. Konstruktoren tragen stets den Namen der Klasse und haben keinen Datentyp. Sie werden typischerweise verwendet, um das neu erzeugte Objekt zu initialisieren, etwa um den Eigenschaften Anfangswerte zuzuweisen. Falls Sie keine Konstruktoren definieren, besitzt die Klasse automatisch einen *Standardkonstruktor*, der nichts Besonderes tut.
- Die drei Methoden dieser Klasse besitzen alle die Geheimhaltungsstufe `public`, damit sie von außen für Instanzen der Klasse aufgerufen werden können. Diese Methoden sind die offiziellen Schnittstellen, über die die Werte der Eigenschaften gelesen oder geändert werden können. Über dieses erlaubte Maß hinaus besteht keine weitere Möglichkeit dazu.

Um die Klasse `Person` ausprobieren zu können, wird das folgende kleine Programm verwendet:

```
public class PersonenTest {
   public static void main(String args[]) {
      Person klaus = new Person("Schmitz", "Klaus", 42);
      System.out.println("Person: " + klaus.getName());
```



```
      klaus.geburtstag();
      System.out.println("Neues Alter: " +
           klaus.getAlter());
   }
}
```

Eine *Instanz* ist formal eine Variable, deren Datentyp die entsprechende Klasse ist. Der Konstruktor der Klasse wird mithilfe von `new` aufgerufen. Die Methoden werden durch einen `.` (Punkt) vom Instanznamen (in diesem Beispiel: `klaus`) getrennt.

Überladen von Konstruktoren und Methoden

Mitunter ist es nützlich, ein Objekt mithilfe verschiedener Eingabewerte zu erzeugen. Daher besteht die Möglichkeit, mehrere Konstruktoren zu definieren. Auch bestimmte Methoden könnten ihre Funktionalität auf verschiedene Art und Weise zur Verfügung stellen, die ebenfalls von unterschiedlichen Parametern abhängt. Die mehrfache Definition eines Konstruktors oder einer Methode mit verschiedenen Parametern wird als *Überladen* bezeichnet.

Innerhalb der Klasse `Person` könnten Sie beispielsweise einen alternativen Konstruktor definieren, der aufgerufen wird, wenn der Vorname unbekannt ist:

```
public Person(String n, int a) {
   this.name = n;
   this.alter = a;
   this.vorname = "";
}
```

Da bereits ein anderer Konstruktor definiert ist, können Sie ihn innerhalb des neuen Konstruktors aufrufen. Dadurch lässt sich der zweite Konstruktor erheblich kürzer fassen:

```
public Person(String n, int a) {
   this(n, "", a);
}
```

Je nachdem, wie Sie eine Instanz der Klasse `Person` erzeugen, wird einer der beiden Konstruktoren aufgerufen:

```
Person klaus = new Person("Schmitz", "Klaus", 42);
     // ruft den ersten Konstruktor auf
Person meyer = new Person("Meyer", 32);
     // ruft den zweiten Konstruktor auf
```

Das Überladen von Methoden funktioniert genauso. Beispielsweise könnte eine weitere Version der Methode `geburtstag()` existieren, die das Alter explizit einstellt:

```
public void geburtstag(int a) {
   this.alter = a;
}
```

Wenn Sie einfach `geburtstag()` aufrufen, wird die Person wie gehabt ein Jahr älter; ein Aufruf wie `geburtstag(33)` ruft dagegen die neue Methode auf und setzt das Alter auf den angegebenen Wert. Ein standardkonformerer Name für eine Methode, die explizit den Wert des Attributs `alter` setzt, wäre allerdings `setAlter()`.

Vererbung

Eines der wichtigsten Merkmale der OOP besteht darin, dass Sie speziellere Klassen von allgemeineren ableiten können. Dies wird als *Vererbung* bezeichnet. Die allgemeine übergeordnete Klasse heißt *Elternklasse* (Parent Class), während die speziellere untergeordnete Klasse *Kindklasse* (Child Class) oder *abgeleitete Klasse* genannt wird. In der abgeleiteten Klasse müssen nur diejenigen Eigenschaften und Methoden definiert werden, die in der Elternklasse noch nicht vorhanden waren oder geändert wurden.

In Java wird die Vererbung durch das Schlüsselwort `extends` gekennzeichnet. Beachten Sie, dass die abgeleitete Klasse nur diejenigen Bestandteile der Elternklasse verwenden kann, deren Geheimhaltungsstufe nicht `private` ist. Da es nicht empfehlenswert ist, allzu viele Komponenten einer Klasse als `public` zu definieren, bietet Java die spezielle Geheimhaltungsstufe `protected` an. Eigenschaften und Methoden, die `protected` sind, werden nicht nach außen veröffentlicht, können aber in Kindklassen eingesetzt werden.

Bevor Sie die folgenden Klassen von `Person` ableiten können, sollten Sie dort jedes Vorkommen von `private` durch `protected` ersetzen und die Datei neu kompilieren.

Die folgenden beiden Klassen `Lehrer` und `Schueler` müssen in gleichnamigen Dateien gespeichert werden. Sie können dann kompiliert werden:

```
public class Lehrer extends Person {
   // zusätzliche Eigenschaft:
   private String fach;

   // Konstruktor:
   public Lehrer(String n, String v, int a, String f) {
      super(n, v, a);
      this.fach = f;
   }

   // Neue Methode:
   public String getFach() {
      return this.fach;
   }
```



```
}

public class Schueler extends Person {
   // zusätzliche Eigenschaft:
   private int klasse;

   // Konstruktor:
   public Schueler(String n, String v, int a, int k) {
      super(n, v, a);
      this.klasse = k;
   }

   // Neue Methoden:
   public int getKlasse() {
      return this.klasse;
   }

   public void versetzung() {
      this.klasse++;
   }
}
```

Die einzige erklärungsbedürftige Besonderheit in den abgeleiteten Klassen dürfte das Schlüsselwort `super` sein. Es ruft explizit den durch die Auswahl der Argumente spezifizierten Konstruktor der Elternklasse auf.

In einem Programm können diese beiden Klassen beispielsweise folgendermaßen verwendet werden:

```
Lehrer welsch = new Lehrer("Welsch", "Jo", 64, "Mathe");
System.out.println(welsch.getName() + " unterrichtet " +
     welsch.getFach());

Schueler tim = new Schueler("Witt", "Tim", 16, 11);
tim.versetzung();
System.out.println(tim.getName()
     + " versetzt in Klasse " + tim.getKlasse());
```

Die Ausgabe dieser beiden Beispiele sollte so aussehen:

```
Jo Welsch unterrichtet Mathe
Tim Witt versetzt in Klasse 12
```

Interfaces

Anders als in C++ und anderen Sprachen ist in Java keine *Mehrfachvererbung* erlaubt; eine Klasse kann also immer nur von genau einer anderen abgeleitet werden. Mitunter kann dies sehr lästig sein: Zwei verschiedene Klassen, die ansonsten nichts miteinander zu tun haben, könnten einen gewissen gemeinsamen Aspekt aufweisen und von einer anderen Stelle aus unter diesem Aspekt betrachtet werden. Beispielsweise ist ein Buch ein völlig anderes Objekt als eine Suppenschüssel. Beide könnten aber als Artikel im gleichen Supermarkt verkauft werden und als solche gemeinsame Eigenschaften wie eine Artikelnummer oder einen Preis aufweisen.

Um Objekte beliebiger Klassen unter einem bestimmten Gesichtspunkt als die gleiche Art von Objekt betrachten zu können, verwendet Java das Verfahren der *Interfaces*. Ein Interface ähnelt einer Klassendefinition, enthält aber lediglich Methodendeklarationen, die nicht implementiert werden, also keine Anweisungen enthalten. Eine Klasse, die ein Objekt dieser Art sein soll, muss alle im Interface deklarierten Methoden implementieren.

Das folgende Beispiel definiert ein Interface namens `Artikel`, das verschiedene Methoden deklariert:

```
public interface Artikel {
   public int getArtikelNummer();
   public int getPreis();
}
```

Die Klasse `Buch` implementiert das Interface `Artikel` und definiert daher die Methoden `getArtikelNummer()` und `getPreis()`:

```
public class Buch implements Artikel {
   private int artikelNummer;
   private int preis;
   ...

   public int getArtikelNummer() {
      return artikelNummer;
   }

   public int getPreis() {
      return preis;
   }
}
```

Der Hauptnutzen dieser Interface-Implementierung besteht darin, dass eine Methode, die mit verschiedenen Artikeln arbeitet, diese alle als Daten vom gleichen Typ ansprechen kann, nämlich `Artikel`.

In der Java-Klassenbibliothek sind Unmengen von Interfaces enthalten, die Sie in Ihren eigenen Programmen implementieren können. Bekannte Beispiele dafür sind die Interfaces `Serializable` oder `Runnable`. `Serializable` wird für Klassen verwendet, deren Datenbestand sich als sequenzieller Datenstrom darstellen (serialisieren) lässt, während `Runnable` von Programmen implementiert wird, die als Thread laufen sollen (siehe Kapitel 10, »Konzepte der Programmierung«).

Wenn Sie ein Interface implementieren, müssen Sie auch jede enthaltene Methode implementieren, selbst wenn Sie glauben, sie nicht zu benötigen. Eine nicht implementierte Methode führt bereits beim Kompilieren zu einer Fehlermeldung wie dieser:

```
error: Class is not abstract and does not override abstract method run() in Runnable
```

Eine abstrakte Klasse ist übrigens eine Klasse, die nicht instanziiert werden darf. Sie dient als allgemeine Vorlage für konkretere Klassen, die durch Vererbung die grundlegende Funktionalität der abstrakten Klasse übernehmen und erweitern. Da erst die Instanz einer Klasse die Methoden eines Interface benötigt, brauchen Sie sie in einer abstrakten Klasse nicht unbedingt zu implementieren.

Wenn Sie seine abstrakte Klasse schreiben möchten, müssen Sie das Schlüsselwort `abstract` zur Deklaration hinzufügen:

```
public abstract class AbstractExample {
  // ...
}
```

Anders als in Interfaces können Sie in abstrakten Klassen Methoden implementieren, die dann von abgeleiteten Klassen geerbt werden – oder bei Bedarf natürlich auch überschrieben werden können.

9.2.3 Weitere Java-Elemente

In diesem Unterabschnitt werden noch einige Spezialitäten der Java-Klassenbibliothek und der Spracharchitektur vorgestellt, die Ihnen bei der Arbeit mit der Sprache von Nutzen sein können: Collections stellen eine überlegende Alternative zu Arrays dar, Enums sind spezielle Klassen, in denen symbolische Konstanten gesammelt werden, und zum Schluss werden Dateizugriffe erläutert.

Collections

Zusätzlich zu einfachen Arrays gibt es in der Java-Klassenbibliothek eine Reihe von Klassen, die das *Java Collections Framework* (JCF) bilden. Die Klassen befinden sich im Package `java.util`. Collections bieten leistungsfähigere Möglichkeiten zur Verarbeitung von Datensammlungen als Arrays. Beispielsweise wird zwischen Listen (mit festgelegter Reihenfolge), Mengen (ohne Duplikate, aber ohne bestimmte Reihenfolge) und Maps (Schlüssel-Wert-

Zuordnungen) unterschieden, und es ist sehr einfach, Schleifen zu schreiben, die jedes Element einer Collection nacheinander bearbeiten.

Der Datentyp der Elemente in einer Collection wird mithilfe einer speziellen Syntax angegeben:

```
Collection-Klasse<Datentyp>
```

Genau genommen, enthält eine Collection keine Werte, sondern Referenzen – das ist zu beachten, weil Sie keine primitiven Datentypen wie `int` verwenden können, sondern nur echte Objekte; für ganze Zahlen wäre dies die Klasse `Integer`. Wenn Sie also beispielsweise eine Liste mit ganzen Zahlen erzeugen wollen, wird dies so geschrieben:

```
List<Integer>
```

`List<...>` selbst ist ein Interface, und es gibt verschiedene Implementierungen. Die gängigste ist die `ArrayList<...>` mit allen wichtigen Eigenschaften eines Arrays. Um die Integer-Liste also konkret als `ArrayList` zu definieren, können Sie Folgendes schreiben:

```
List<Integer> intList = new ArrayList<>();
```

Wie Sie sehen, können Sie den enthaltenen Datentyp (hier `Integer`) beim Aufruf von `new` weglassen, da er bereits durch die Deklaration feststeht, und dies ist auch weitverbreitete Praxis.

Einige wichtige Methoden von Listen und teilweise auch von anderen Collections sind folgende:

- `liste.add(element)` fügt ein Element hinzu.
- `liste.addAll(collection)` fügt alle Elemente der Collection zur bestehenden Liste hinzu.
- `liste.remove(index|element)` entfernt ein Element; als Argument können Sie entweder den numerischen Index oder das zu entfernende Objekt angeben.
- `liste.removeAll(collection)` entfernt alle Elemente der Collection aus der Liste.
- `liste.get(index)` liefert das Element an der angegebenen Indexposition zurück.
- `liste.set(index, element)` ersetzt das Element am angegebenen Index durch das neue Element.
- `liste.contains(element)` gibt `true` zurück, wenn die Liste das Element enthält, andernfalls `false`.
- `liste.containsAll(collection)` gibt `true` zurück, wenn die Liste alle Elemente der Collection enthält, andernfalls `false`.
- `liste.size()` gibt die Anzahl der Elemente zurück.

Beachten Sie, dass nicht jede Implementierung der Collection-Interfaces alle diese Methoden enthält. Genaueres können Sie in der Java-Dokumentation zu Collections nachlesen; das

Interface `List` wird beispielsweise unter *https://docs.oracle.com/javase/7/docs/api/java/util/List.html* beschrieben.

Um über die Elemente einer Liste zu iterieren, jedes Element also in einer Schleife zu verarbeiten, wird folgende Schreibweise verwendet:

```
for (Typ element:liste) {
  // element enthält pro Durchlauf das jeweils nächste Element
}
```

Bei der eben definierten `intList` sieht dies beispielsweise wie folgt aus:

```
for (int i:intList) {
  // i ist das jeweilige Element
}
```


Wie Sie sehen, ist `int` als Typ für die Schleifenvariable erlaubt, obwohl die Liste die objektorientierte Variante `Integer` enthält.

Das folgende Beispiel fügt die Kommandozeilenargumente zu einer Liste hinzu, sofern es sich um Integer handelt, und gibt anschließend die Anzahl der Elemente, die Summe und den Mittelwert aus:

```
import java.util.List;
import java.util.ArrayList;

public class ListTest {
  public static void main(String[] args) {
    if (args.length < 1) {
      System.out.println("Verwendung: java ListTest integer [...]");
      System.exit(1);
    }
    List<Integer> intList = new ArrayList<>();
    for (int i = 0; i < args.length; i++) {
      try {
        int value = Integer.parseInt(args[i]);
        intList.add(value);
        System.out.println("- Element hinzugefügt: " + value);
      } catch(NumberFormatException e) {
        System.out.println("! '" + args[i] + "' ist kein gültiger Integer.");
      }
    }
    int sum = 0;
    for (int i:intList) {
      sum += i;
```

```
    }
    double average = (double)sum / intList.size();
    System.out.println();
    System.out.println("Anzahl Elemente: " + intList.size());
    System.out.println("Summe:           " + sum);
    System.out.println("Mittelwert:      " + average);
  }
}
```

Kompilieren Sie *ListTest.java*, und führen Sie das Programm aus, etwa so:

```
$ java ListTest 6 8 9 2 3 hallo 7
- Element hinzugefügt: 6
- Element hinzugefügt: 8
- Element hinzugefügt: 9
- Element hinzugefügt: 2
- Element hinzugefügt: 3
! 'hallo' ist kein gültiger Integer.
- Element hinzugefügt: 7

Anzahl Elemente: 6
Summe:           35
Mittelwert:      5.833333333333333
```

Da Kommandozeilenargumente Strings sind, müssen sie zunächst in Integer umgewandelt werden. Dazu wird die statische Methode `Integer.parseInt(...)` verwendet, die eine `NumberFormatException` auslöst, wenn der Wert nicht umgewandelt werden kann. Diese Exception wird abgefangen, und eine Warnmeldung wird ausgegeben.

Bei der Berechnung des Mittelwertes ist es wichtig, mindestens einen der beiden Integer-Werte in einen Fließkommawert umzuwandeln, da ansonsten auch das Ergebnis ganzzahlig wäre:

```
double average = (double)sum / intList.size();
```

Wie bereits erwähnt, enthält eine Menge (`Set`) jedes Element nur einmal. Auch `Set` ist ein Interface; eine mögliche Implementierung ist die Klasse `HashSet`, die intern eine Schlüssel-Wert-Liste (`HashMap`) verwendet, um die Einmaligkeit der Elemente zu garantieren. Das folgende kleine Beispiel demonstriert, dass Elemente nicht doppelt vorkommen:

```
import java.util.Set;
import java.util.HashSet;

public class SetTest {
```



```
  public static void main(String[] args) {
    Set<String> stringSet = new HashSet<>();
    stringSet.add("Hallo");
    stringSet.add("Welt");
    stringSet.add("Welt");
    for (String str:stringSet) {
      System.out.println(str);
    }
  }
}
```

Wenn Sie *SetTest.java* kompilieren und ausführen, erhalten Sie wie erwartet folgende Ausgabe:

```
$ java SetTest
Hallo
Welt
```

Es ist übrigens auch möglich, Collection-ähnliche Klassen mit Datentyp-Platzhaltern zu schreiben. Dazu verwenden Sie für die Datentypen symbolische Bezeichner wie `T`. Die Platzhalter werden als *Generics* bezeichnet; sie wurden in Version 5 von Java eingeführt.

Das folgende Beispiel implementiert eine Klasse `Pair`, die zwei Elemente desselben Datentyps enthält, sowie eine Methode `equal()`, die `true` zurückgibt, wenn die beiden Elemente identisch sind, und ansonsten `false`. Dies könnte beispielsweise die Grundlage für ein Memory-Spiel sein, die die beiden aufgedeckten Karten vergleicht:

```
public class Pair<T> {
  private T value1;
  private T value2;

  public Pair(T _value1, T _value2) {
    value1 = _value1;
    value2 = _value2;
  }

  public boolean equal() {
    if (value1.equals(value2)) {
      return true;
    }
    return false;
  }
}
```

Die folgende Beispielklasse definiert zwei Paare verschiedenen Datentyps und vergleicht ihre Elemente:

```
public class PairTest {
  public static void printEqual(Pair pair, String title) {
    if (pair.equal()) {
      System.out.println("Gleiches Paar in " + title);
    } else {
      System.out.println("Ungleiches Paar in " + title);
    }
  }

  public static void main(String[] args) {
    Pair<Integer> pair1 = new Pair<>(4, 5);
    PairTest.printEqual(pair1, "pair1");
    Pair<String> pair2 = new Pair<>("Hallo", "Hallo");
    PairTest.printEqual(pair2, "pair2");
  }
}
```

Führen Sie *PairTest.java* nach dem Kompilieren aus; Sie erhalten die folgende naheliegende Ausgabe:

```
$ java PairTest
Ungleiches Paar in pair1
Gleiches Paar in pair2
```

Enums

Eine ganz andere Art von Aufzählung ist das `enum`-Konstrukt: Es handelt sich um eine Sammlung garantiert voneinander verschiedener symbolischer Konstanten. Ein `enum` wird wie eine Klasse in einer eigenen Datei definiert, die den Namen des `enum` trägt. Im einfachsten Fall ist nur die Liste von Konstanten enthalten, wie im folgenden Beispiel, das die vier grundlegenden Himmelsrichtungen enthält:

```
public enum Direction {
  NORTH,
  EAST,
  SOUTH,
  WEST
}
```

Andere Klassen können mit `enum-Name.Konstantenname` auf die Konstanten zugreifen, also etwa `Direction.NORTH` für den Norden. Das folgende kleine Beispielprogramm verwendet die Himmelsrichtungen, um einige interessante Orte aus George R. R. Martins Fantasy-Reihe

»A Song of Ice And Fire« (die Grundlage der beliebten Fernsehserie »Game of Thrones«) zu lokalisieren, und stellt die `HashMap` als weitere Collection vor:

```
import java.util.Map;
import java.util.HashMap;

public class WesterosTest {
  public static void main(String[] args) {
    Map<String, Direction> locations = new HashMap<>();
    locations.put("The Wall", Direction.NORTH);
    locations.put("Vale of Arryn", Direction.EAST);
    locations.put("Dorne", Direction.SOUTH);
    locations.put("Iron Islands", Direction.WEST);
    for (Map.Entry<String, Direction> entry:locations.entrySet()) {
      System.out.printf("%s is in the %s.\n", entry.getKey(), entry.getValue());
    }
  }
}
```

Das Interface `Map` und seine Implementierung `HashMap` benötigen zwei Datentypen, einen für die Schlüssel und einen für die Werte. Während die Schlüssel alle verschieden sein müssen, können beliebig viele identische Werte vorhanden sein – ideal also, um die Orte als Schlüssel und ihre Himmelsrichtungen als Werte zu verwenden. Die Methode `map.put(Schlüssel, Wert)` fügt einen Wert für den angegebenen Schlüssel hinzu oder ändert diesen, falls der Schlüssel bereits existiert. Mit `map.get(Schlüssel)` können Sie einen einzelnen Wert auslesen.

In diesem Beispiel sollen allerdings alle Schlüssel und Werte ausgegeben werden. Die Schleifenvariable hat dafür den Typ `Map.Entry<>` mit denselben Datentypen für Schlüssel und Wert wie die `Map` selbst; die `Map`-Methode `entrySet()` liefert diese Einträge. Aus dem `Entry` können Sie mit `getKey()` und `getValue()` den Schlüssel beziehungsweise den Wert extrahieren. Zur Ausgabe kommt hier übrigens die Methode `System.out.printf()` zum Einsatz; sie hat dieselbe Syntax wie die gleichnamige C-Funktion.

Speichern Sie *Directions.java* und *WesterosTest.java* im selben Verzeichnis. Anschließend können Sie die letztere Datei kompilieren und ausführen; die Ausgabe sieht etwa so aus:

```
$ java WesterosTest
Iron Islands is in the WEST.
Dorne is in the SOUTH.
Vale of Arryn is in the EAST.
The Wall is in the NORTH.
```

Wie Sie sehen, entspricht die Reihenfolge der Ausgabe nicht derjenigen, in der Sie die Schlüssel und Werte hinzugefügt haben – Sortierung ist nicht die Aufgabe einer `Map`, sodass sie die

Reihenfolge eher für möglichst schnelles Auslesen optimiert. Wenn Ihnen die Reihenfolge wichtig ist, können Sie anstelle einer `HashMap` eine `SortedMap` verwenden.

Ein `enum` kann übrigens nicht nur passive Konstanten enthalten, sondern auch Attribute und Methoden für diese Konstanten. Anders als gewöhnliche Klassen kennt dieses Konstrukt allerdings keine Vererbung, und Sie dürfen für den Konstruktor und andere Methoden nicht das Schlüsselwort `public` verwenden. Schauen Sie sich als Beispiel das folgende `enum` an, das auch Zwischenhimmelsrichtungen enthält und auf Wunsch deren Gradzahlen auf dem Kompass liefert:

```
public enum FineDirection {
  NORTH (0),
  NORTHEAST (45),
  EAST (90),
  SOUTHEAST (135),
  SOUTH (180),
  SOUTHWEST (225),
  WEST (270),
  NORTHWEST (315);

  private final Integer degrees;

  FineDirection(Integer _degrees) {
    degrees = _degrees;
  }

  int degrees() {
    return degrees;
  }
}
```

Ein Zugriff auf eine der Konstanten ruft den Konstruktor auf und übergibt ihm die Argumente, die hinter der jeweiligen Konstanten in Klammern stehen. Anschließend können Sie die Methoden aufrufen. `FineDirection.SOUTHWEST.degrees()` liefert beispielsweise die Gradzahl für Südwesten (225) zurück.

Sie können übrigens auch über die Konstanten eines `enum` iterieren, indem Sie mit dessen Methode `values()` die Liste der Konstanten auslesen. Das folgende Beispielprogramm gibt alle acht Hauptwindrichtungen und ihre Gradzahlen aus:

```
public class EnumTest {
  public static void main(String[] args) {
    for (FineDirection dir:FineDirection.values()) {
      System.out.printf("%s has %d degrees.\n", dir, dir.degrees());
```



```
    }
  }
}
```

Die Ausgabe des Programms sieht natürlich so aus:

```
NORTH has 0 degrees.
NORTHEAST has 45 degrees.
EAST has 90 degrees.
SOUTHEAST has 135 degrees.
SOUTH has 180 degrees.
SOUTHWEST has 225 degrees.
WEST has 270 degrees.
NORTHWEST has 315 degrees.
```


Dateizugriffe

In Java stehen, wie bereits erwähnt, verschiedene Klassen für die unterschiedlichsten Aspekte der Ein- und Ausgabe zur Verfügung. Dies betrifft auch den Umgang mit Dateien. Hier sehen Sie nur ein kurzes Beispiel. Das folgende Programm liest sämtliche Zeilen aus der als Kommandozeilenargument angegebenen Textdatei und gibt sie in umgekehrter Reihenfolge aus.

```
import java.io.*;
import java.util.*;

public class FileTurner {
   public static void main (String args[]) {
      try {
         turnFile (args[0]);
      }
      catch (FileNotFoundException e) {
         System.out.println("Datei nicht gefunden!");
      }
      catch (IOException e) {
         System.out.println("Dateifehler!");
      }
   }

   static void turnFile (String filename) throws
        FileNotFoundException, IOException {
      BufferedReader reader = new BufferedReader
           (new FileReader (filename));
      String zeile = "";
```

```
        Stack zeilen = new Stack();
        while ((zeile = reader.readLine()) != null) {
           zeilen.push (zeile);
        }
        while (!zeilen.empty()) {
           zeile = (String)zeilen.pop();
           System.out.println(zeile);
        }
     }
}
```

Das Programm demonstriert allerdings nicht nur, wie Sie in Java aus einer Textdatei lesen können, sondern zeigt noch einige andere interessante Aspekte der Java-Programmierung:

- Innerhalb der Methode `turnFile()` werden verschiedene Methoden aufgerufen, die Ausnahmen auslösen können. Damit nicht jede dieser Anweisungen innerhalb eines `try`/`catch`-Blocks stehen muss, werden sämtliche möglichen Ausnahmen mithilfe einer `throws`-Klausel an die aufrufende Stelle delegiert. Aus diesem Grund steht der Aufruf von `turnFile()` von `main()` innerhalb eines `try`-Blocks; die anschließenden `catch`-Blöcke fangen die entsprechenden Ausnahmen ab: `java.io.FileNotFoundException` (Datei nicht gefunden) und `java.io.IOException` (Schreib-/Lesefehler).
- Für das Umkehren der Zeilen aus der Textdatei wird die spezielle Datenstruktur `java.util.Stack` verwendet, mit deren Hilfe in Java die im nächsten Kapitel besprochene Stack-Funktionalität implementiert wird: Die Methode `push()` fügt ein beliebiges Objekt (eine Instanz der allgemeinsten Klasse `java.lang.Object`) am Ende des Stacks hinzu, `pop()` entnimmt das letzte Objekt und gibt es zurück. Mithilfe von Typecasting kann es in ein Objekt seiner ursprünglichen Klasse zurückverwandelt werden: im vorliegenden Fall `(String)zeilen.pop()`.
- Zum Lesen ganzer Zeilen aus der Textdatei wird der `java.io.FileReader`, der dem Lesen aus einer Datei dient, von einem `java.io.BufferedReader` umhüllt, dessen Methode `readLine()` jeweils eine Zeile aus dem zugrunde liegenden Eingabestrom liest. Die Bedingung der `while()`-Schleife macht es sich übrigens zunutze, dass `readLine()` am Dateiende `null` zurückgibt: Die Wertzuweisung `zeile = reader.readLine()` wird dazu unmittelbar mit `null` verglichen.

9.3 Python

Zeit für die dritte Programmiersprache in diesem Kapitel: *Python*. Anders als bei C und Java handelt es sich um eine *interpretierte Sprache*, das heißt Programme werden im Quellcode

ausgeliefert und zur Laufzeit von einem speziellen Programm, dem *Python-Interpreter*, übersetzt. Hinter den Kulissen findet, wie bei den meisten modernen Skriptsprachen, eine sogenannte *Just-in-Time-Kompilierung* statt – der Code wird keineswegs Zeile für Zeile während der Ausführung übersetzt, sondern in der Regel viel schneller, und die übersetzte Form wird im Arbeitsspeicher oder sogar auf einem Datenträger zwischengespeichert.

Python ist eine sogenannte *Multiparadigmen-Sprache*. Es handelt sich also nicht um eine rein imperative, objektorientierte oder funktionale Sprache, sondern Python bietet Aspekte dieser und anderer Programmiersprachen.

Die erste Version der Sprache wurde seit 1989 von dem niederländischen Programmierer *Guido van Rossum* entwickelt; die fertige Version 1.0 erschien 1994. Der Name bezieht sich nicht auf das gleichnamige Tier, sondern auf die britische Comedy-Gruppe *Monty Python*, von der van Rossum ein großer Fan ist. In der offiziellen Python-Dokumentation wimmelt es daher von Bezügen auf Monty-Python-Filme und -Sketche (der vorliegende Abschnitt hält an dieser Tradition fest). Inzwischen arbeiten zahlreiche Programmierer an der Weiterentwicklung von Python, aber Guido van Rossum hat als »Benevolent Dictator for Life« (BDFL) noch immer das letzte Wort.

Im Jahr 2000 wurde Python 2.0 vorgestellt; die wichtigsten neuen Features waren Unicode-Unterstützung und eine voll ausgestattete Garbage Collection. Python 3.0, veröffentlicht im Jahr 2008, ist eine ausführliche Neuimplementierung der Sprache und nicht zu Version 2 abwärtskompatibel. Allerdings wurden zahlreiche Features seitdem für die Verwendung mit 2.x rückportiert. Da es noch immer diverse weitverbreitete Bibliotheken gibt, die nicht mit Python 3 kompatibel sind, und da Python in vielen Softwareprojekten eine wichtige Rolle als eingebettete Skriptsprache spielt, wird auch Python 2 noch immer weitergepflegt, zumindest mit Sicherheitsupdates und Bugfixes. Die aktuellen Releases sind 3.4.3 und 2.7.9. Dieser Abschnitt behandelt ausschließlich Python 3.

Auf den meisten Linux-Distributionen ist Python bereits vorinstalliert oder kann mithilfe des Paketmanagers nachinstalliert werden. Oft verbirgt sich hinter dem Kommando `python` die Version 2.x, während Python 3 als `python3` aufgerufen wird. OS X enthält ebenfalls Python 2 ab Werk, während Python 3 nachinstalliert werden kann. Installer mit ausführlicher Anleitung für OS X, Windows und andere Betriebssysteme finden Sie auf der Python-Website, *www.python.org*.

Im Folgenden wird der Interpreter-Aufruf stets als `python3` geschrieben, um klarzustellen, dass Sie diese Version benötigen, um alle Beispiele ausprobieren zu können. Um ein Python-Skript auszuführen, geben Sie Folgendes ein:

```
$ python3 Dateiname
```

Die Dateiendung für Python-Skripte ist üblicherweise *.py*.

Wenn Sie den Interpreter ohne Angabe eines Dateinamens starten, erhalten Sie eine interaktive Shell. Hier zum Beispiel die Ausgabe unter OS X:

```
$ python3
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 23 2015, 02:52:03)
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Hier können Sie Python-Ausdrücke und Kommandos eingeben, die sofort ausgewertet werden, was eine sehr praktische Methode ist, Python auszuprobieren. In den folgenden Unterabschnitten wird die Python-Shell oft zum Einsatz kommen; die entsprechenden Beispiele werden durch ihren Prompt gekennzeichnet: >>>.

Sie können die Shell unter anderem als einfachen Taschenrechner verwenden:

```
>>> 3+4
7
```

Wenn Sie die Shell wieder beenden möchten, können Sie

```
>>> exit()
```

eingeben. Auf Unix-Systemen genügt es auch, Strg + D (End of File) zu drücken.

9.3.1 Das erste Beispiel

Wie üblich folgt hier zunächst das erweiterte »Hallo-Welt«-Programm. Geben Sie es in einem Editor Ihrer Wahl ein, und speichern Sie es unter *hello.py*:

```
print("Hallo Welt!")
name = input("Ihr Name bitte: ")
print("Hallo " + name + "!")
```

Anschließend können Sie das Skript wie folgt ausführen:

```
$ python3 hello.py
Hallo Welt!
Ihr Name bitte: Sascha
Hallo Sascha!
```

Wie Sie sehen, ist das Programm in Python viel kürzer als in den bisher behandelten Sprachen. Wie viele Skriptsprachen beschränkt sich Python auf das Wesentliche; beispielsweise ist keine formale Hauptprogramm-Deklaration erforderlich, um eine Folge von Anweisungen auszuführen.

Im Einzelnen bedeuten die drei Zeilen Folgendes:

- `print("Hallo Welt!")`

  Die Methode `print()` gibt einen String aus, gefolgt von einem Zeilenumbruch. Die Methode hat zusätzliche Parameter, die beispielsweise die Ersetzung des Zeilenumbruchs durch ein anderes Trennzeichen erlauben, wobei auch ein leerer String möglich ist, um mehrere Strings ohne Unterbrechung auszugeben.

- `name = input("Ihr Name bitte: ")`

  Diese Anweisung definiert eine Variable mit der Bezeichnung `name`. Der Methodenaufruf `input()` ermöglicht dem Benutzer die Eingabe einer Textzeile zur Laufzeit. Durch die Wertzuweisung wird die Eingabe, ohne den abschließenden Zeilenumbruch, in der Variablen `name` gespeichert.

- `print("Hallo " + name + "!")`

  Auch dieser Aufruf von `print()` dient der Ausgabe; der einzige Unterschied zur ersten Zeile des Skripts besteht darin, dass hier ein String aus zwei Literalen und dem Wert einer Variablen zusammengesetzt wird. Genau wie in Java werden Strings in Python mithilfe des Operators + verkettet.

9.3.2 Grundelemente von Python

Dieser Unterabschnitt betrachtet die grundlegenden Bestandteile von Python und konzentriert sich dabei besonders auf diejenigen Aspekte, die sich grundlegend von C und Java unterscheiden. In weiteren Unterabschnitten werden die Objektorientierung sowie einige ausgewählte Elemente der Python-Standardbibliothek behandelt.

Besonderheiten der Syntax

Anders als in Java und C enden Anweisungen in Python nicht mit einem Semikolon, sondern mit einem Zeilenumbruch:

```
>>> print("Hallo")
Hallo
>>> print("Welt")
Welt
```

Allerdings können Sie das Semikolon optional verwenden, um mehrere Anweisungen in dieselbe Zeile zu schreiben:

```
>>> print("Hallo"); print("Welt")
Hallo
Welt
```

Dies ist jedoch nur in Ausnahmefällen empfehlenswert, da es nicht zu lesbaren Programmen beiträgt.

Umgekehrt können Sie Anweisungen wie in C über mehrere Zeilen verteilen, wenn Sie vor dem jeweiligen Zeilenumbruch einen Backslash setzen:

```
>>> 2 * \
... 3 \
... * 4
24
```

Die Einrückung macht den Unterschied

Die wichtigste Besonderheit von Python ist, dass *allein die Einrückung* darüber bestimmt, in welchen Zusammenhang eine Anweisung gehört. Dies wurde von einer ebenfalls in den Niederlanden entwickelten Sprache namens ABC übernommen, an der Guido van Rossum mitarbeitete, bevor er Python entwickelte.

Betrachten Sie dazu das folgende Beispiel:

```
print("Allgemeine Anweisung") # Wird immer ausgeführt
if a > 0:
    print("a ist positiv.")   # Nur, wenn a > 0
    print("Noch eine Zeile.") # Nur, wenn a > 0
print("Ende.")               # Wird immer ausgeführt
```

Dieses Konzept ist gewöhnungsbedürftig und zwingt zu äußerster Disziplin beim Einrücken, aber es folgt dem allgemeinen Konzept von Python, möglichst sparsam und pragmatisch zu sein. Außerdem schadet Disziplin beim Einrücken ohnehin nicht, da sie Programme auch in Sprachen, in denen sie keine syntaktische Bedeutung hat, lesbarer macht.

Python bietet zwei verschiedene Arten von Kommentaren. Die Raute (#) leitet einen einzeiligen Kommentar ein, der bis zum nächsten Zeilenumbruch reicht. Beispiel:

```
print("Test")  # Anweisung und Kommentar
# Nur Kommentar
```

Ein mehrzeiliger Kommentar beginnt und endet jeweils mit drei doppelten Anführungszeichen. Dieser Kommentarstil wird vor allem zur Generierung von Programmdokumentationen aus dem Quellcode verwendet. Hier ein Beispiel:

```
"""Mehrzeiliger Kommentar
Wird üblicherweise zur Programmdokumentation verwendet.
"""
```

Literale

Wie in den anderen bereits behandelten Sprachen bestehen Ausdrücke auch in Python aus Literalen, Variablen, Operatoren und Funktions- beziehungsweise Methodenaufrufen.

Die verfügbaren *Literale* sind denjenigen in den anderen Sprachen ähnlich; es gibt nur wenige Besonderheiten. Jedes Literal, und natürlich auch der Wert jedes Ausdrucks, gehört in Python zu einer bestimmten Klasse; Sie können diese mithilfe der Funktion `type(ausdruck)` ermitteln. Beispiele:

```
>>> type(42)
<class 'int'>
>>> type(3.1415926)
<class 'float'>
>>> type("Camelot")
<class 'str'>
>>> type([1,2,3])
<class 'list'>
```


Numerische Literale sind weitgehend mit denjenigen in C identisch. Sie können ganze Zahlen dezimal angeben (Standard), also beispielsweise `-23` oder `42`. Hexadezimalzahlen werden wie in C durch vorangestelltes `0x` gekennzeichnet:

```
>>> 0xABCD
43981
```

Oktalzahlen werden seit Python 3.0 anders geschrieben als in C, nämlich mit vorangestelltem `0o` (Ziffer Null, kleiner Buchstabe »O«) anstelle einer einfachen Null:

```
>>> 0o27
23
```

Zusätzlich können Sie Dualzahlen als Literale schreiben, indem Sie ihnen die Zeichenfolge `0b` voranstellen:

```
>>> 0b101010
42
```

Fließkommaliterale können wie üblich mit Dezimalpunkt geschrieben werden, beispielsweise `0.23` oder `-1.03`, oder mithilfe der Exponentialschreibweise, etwa `4e4` für 4×10^4 (40000) oder `3e-3` für $3 * 10^{-3}$ (0.003).

Eine Besonderheit von Python ist, dass bereits in der Grundausstattung *komplexe Zahlen* enthalten sind. Eine komplexe Zahl besteht aus einem *Realteil*, also einer beliebigen reellen Zahl, und einem hinzuaddierten *Imaginärteil* (ein Vielfaches von i, definiert als eine Zahl, deren Quadrat –1 ist, also $i := i^2 = -1$). Da das Quadrat sowohl positiver als auch negativer Zahlen für reelle Zahlen positiv ist, werden Zahlen mit negativem Quadrat, die beispielsweise in der Physik eine Rolle spielen, als imaginär bezeichnet. Komplexe Zahlen werden wie folgt geschrieben:

```
>>> complex(3, 4)
(3+4j)
```

In den Klammern von `complex()` wird also zuerst der Real- und dann der Imaginärteil angegeben. Die interne Darstellung besteht, wie Sie sehen, aus Klammern, dem Realteil, einem Plus- oder Minuszeichen (je nachdem, ob der Imaginär-Multiplikator positiv oder negativ ist), dem Multiplikator selbst und dem Buchstaben j. Die eingebauten Attribute `real` und `imag` liefern den Imaginär- beziehungsweise Realteil zurück:

```
>>> complex(3, 5).real
3.0
>>> complex(4, -4).imag
-4.0
```

Ansonsten können Sie mit komplexen Zahlen gemäß den für sie geltenden mathematischen Regeln rechnen, auch gemischt mit anderen Zahlentypen. Beispiele:

```
>>> complex(1, 2) + complex(3, 4)
(4+6j)
>>> complex(3, 5) + 7
(10+5j)
```

Auch für Strings gibt es verschiedene Schreibweisen:

- doppelte Anführungszeichen, zum Beispiel `"The Holy Grail"`
- einfache Anführungszeichen, etwa `'Dead Parrot'`
- Sogenannte *Here Documents* (auf Deutsch manchmal Hier-Dokumente), die mit drei einfachen Anführungszeichen beginnen, sich über beliebig viele Zeilen erstrecken können und wieder mit drei einfachen Anführungszeichen enden. Der Name bezieht sich darauf, dass diese Strings »bis hierhin« reichen. Dies ist ein Beispiel aus der interaktiven Shell, das zeigt, wie die Zeilenumbrüche Teil des Strings werden:

  ```
  >>> '''Arthur
  ... Galahad
  ... Lancelot
  ... Robin'''
  'Arthur\nGalahad\nLancelot\nRobin'
  ```

Anders als in manchen anderen Skriptsprachen gibt es keinen funktionalen Unterschied zwischen einfachen und doppelten Anführungszeichen, und auch ein einzelnes Zeichen in einfachen Anführungszeichen ist kein Zeichen-Literal, sondern immer noch ein String. Sie können innerhalb eines Strings mit einer bestimmten Sorte von Anführungszeichen die jeweils andere Sorte ohne Escaping verwenden:


```
>>> "'We have Spam, bacon, sausage and Spam,' the waitress said."
"'We have Spam, bacon, sausage and Spam,' the waitress said."
>>> '"We have Spam, egg, Spam, Spam, bacon and Spam," she added.'
'"We have Spam, egg, Spam, Spam, bacon and Spam," she added.'
```

Möchten Sie dagegen dieselbe Sorte Anführungszeichen verschachteln, müssen Sie sie mithilfe eines Backslashs (\) escapen:

```
>>> '"\'Tis but a scratch," the Black Knight roared.'
'"\'Tis but a scratch," the Black Knight roared.'
>>> "\"It's just a flesh wound,\" he added."
'"It\'s just a flesh wound," he added.'
```


Besondere Literale sind `True` und `False`, die zur Klasse `bool` gehören, sowie `None`, die einzige Instanz der Klasse `NoneType`. Beachten Sie, dass alle diese Werte mit großen Anführungszeichen geschrieben werden müssen. `True` und `False` sind boolesche Wahrheitswerte, also zum Beispiel die Ergebnisse von Vergleichsoperationen:

```
>>> 2 < 3
True
>>> 2 > 3
False
```

`None` ist die Python-Variante von `null` in Java, also eine leere Referenz.

Variablen, die noch nicht definiert wurden, haben nicht den Wert `None`, sondern der Zugriff auf sie führt zu einer Fehlermeldung, wie das folgende Beispiel zeigt:

```
>>> print(x)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'x' is not defined
```

Neben den bisher besprochenen skalaren (einwertigen) Literalen gibt es noch verschiedene Arten von Listen und weitere zusammengesetzte Literale, etwa eine einfache Liste wie `[1, 2, 3]`. Diese werden später in ihrem eigenen Unterabschnitt behandelt.

Variablen

Variablen werden in Python durch einfache Wertzuweisung definiert. Beispiele:

```
>>> a = 1
>>> b = "Hallo"
>>> a
```

```
1
>>> b
'Hallo'
```

Der Typ ist dabei ausschließlich an den enthaltenen Wert gebunden, nicht an die Variable selbst. Sie können derselben Variablen daher nacheinander Werte verschiedener Typen zuweisen:

```
>>> x = 1
>>> type(x)
<class 'int'>
>>> x = "Test"
>>> type(x)
<class 'str'>
```

In Bezeichnern sind Buchstaben, Ziffern und Unterstriche erlaubt, und wie üblich dürfen sie nicht mit einer Ziffer beginnen. Es wird zwischen Groß- und Kleinschreibung unterschieden, wie der folgende Versuch durch die Fehlermeldung zeigt:

```
>>> test = 3
>>> test
3
>>> Test
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'Test' is not defined
```

Regeln für Bezeichner in Python

Es gibt einige Konventionen für Bezeichner in Python:

- Variablen und Funktionen beginnen mit einem Kleinbuchstaben: Wenn sie aus mehreren Wörtern bestehen, werden diese durch Unterstriche getrennt. Beispiele: `value`, `number_of_elements`
- Klassennamen werden großgeschrieben (mit Ausnahme von Datentypen und vielen Basisklassen der Standardbibliothek), und mehrere Wörter werden durch Binnenmajuskeln voneinander getrennt (CamelCase). Beispiele: `Book`, `BookPublisher`
- Elemente, die als privat gelten, also nicht Teil der öffentlichen Schnittstelle eines Programms sind, sollten mit einem einzelnen Unterstrich beginnen. Beispiele: `_internal_value`, `_InternalClass`
- Viele interne Elemente von Python selbst beginnen und enden mit je zwei Unterstrichen, etwa die Methode `__init__()` – der Konstruktor einer Klasse – oder die Konstante `__name__` (Name des aktuellen Moduls oder Ausführungskontextes)

Variablen sind grundsätzlich Referenzen, das heißt, sie verweisen auf Objekte, die sich irgendwo im Speicher befinden. Wo es sich im Speicher befindet (beziehungsweise ob zwei identisch aussehende Objekte dasselbe Objekt sind), können Sie mithilfe der Identitätsfunktion `id()` herausfinden:

```
>>> id(42)
4297372000
>>> id("Hallo")
4338199216
```

Es wird zwischen *unveränderlichen Objekten* (*immutable objects*) und *veränderlichen Objekten* (*mutable objects*) unterschieden. Veränderliche Objekte behalten ihren Speicherort auch dann, wenn ihr Wert verändert wird, während eine Wertänderung einer Variablen mit einem unveränderlichen Objekt nach der Wertänderung auf eine andere Speicherstelle verweist.

Skalare Datentypen sind in der Regel unveränderlich, wie das folgende Beispiel zeigt – der Identitätswert verändert sich:

```
>>> a = 23
>>> id(a)
4297371392
>>> a += 1   # a um 1 erhöhen
>>> id(a)
4297371424
```

Zusammengesetzte Datentypen wie Listen oder Objekte komplexer Klassen sind dagegen meist veränderlich; sie behalten ihre Identität auch über eine Wertänderung hinaus:

```
>>> a = [1, 2, 3]  # Einfache Liste
>>> id(a)
4338176968
>>> a.append(4)    # Weiteres Element hinzufügen
>>> a
[1, 2, 3, 4]
>>> id(a)
4338176968
```

Wenn eine Variable auf ein veränderliches Objekt verweist und Sie einer weiteren Variablen den Wert der ersten zuweisen, dann verweisen beide auf dasselbe Objekt, und ihr Wert verändert sich entsprechend. Hier ein Beispiel mit zwei Variablen, die auf dieselbe Liste verweisen:

```
>>> ref1 = ['ham', 'eggs', 'bacon']
>>> ref2 = ref1
>>> ref2.append('Spam')   # Element an ref2 anhängen
```

```
>>> ref1
['ham', 'eggs', 'bacon', 'Spam']
```

Wie Sie sehen, verändert das Hinzufügen des Elements zu `ref2` auch den Wert von `ref1`, da sie auf dasselbe Objekt verweisen (was Sie bei Bedarf wieder mithilfe von `id()` überprüfen können).

Bei unveränderlichen Objekten kann es durchaus vorkommen, dass verschiedene Variablen auf dasselbe interne Objekt verweisen, wenn derselbe Wert gespeichert wird, da dieses Vorgehen speicherschonend ist. Dies muss jedoch nicht der Fall sein. In jedem Fall modifiziert die Änderung der einen Variablen aber nicht den Wert der anderen. Sehen Sie sich dazu folgendes Beispiel an:

```
>>> var1 = 23
>>> var2 = var1
>>> id(var1)
4297371392
>>> id(var2)
4297371392
>>> var2 += 1    # um 1 erhöhen
>>> var2
24
>>> var1
23
>>> id(var2)
4297371424
```

Wie Sie sehen, hatten `var1` und `var2` in meiner Sitzung dieselbe Identität, solange sie denselben Wert speicherten. Sobald `var2` verändert wird, hat dieser eine neue Identität und einen neuen Wert, während `var1` den ursprünglichen Wert behält.

Operatoren

Die meisten Operatoren für einfache Datentypen sind mit denjenigen in C und Java identisch. Beispielsweise stehen für numerische Datentypen die arithmetischen Operatoren `+` (Addition), `-` (Subtraktion), `*` (Multiplikation), `/` (Division) und `%` (Modulo, also der Rest der ganzzahligen Division) zur Verfügung.

Der normale Divisionsoperator `/` gibt unabhängig vom Datentyp der Operanden immer eine Fließkommazahl (Klasse `float`) zurück, wie das folgende Beispiel zeigt:

```
>>> 6 / 3
2.0
>>> type(6 / 3)
<class 'float'>
```


Die spezielle Variante `//` gibt dagegen eine Ganzzahl (`int`) zurück, solange beide Operanden ebenfalls `int` sind:

```
>>> 6 // 3
2
>>> 7 // 3
2
>>> type(7 // 3)
<class 'int'>
```

Ist mindestens einer der Operanden ein `float`, gibt allerdings auch `//` ein Ergebnis vom Typ `float` zurück:

```
>>> 6.5 // 3
2.0
>>> type(6.5 // 3)
<class 'float'>
```

Eine weitere Ergänzung gegenüber C und Java ist der Potenzoperator `**`, wobei `a ** b` für a^b steht. Hier einige Beispiele:

```
>>> 2 ** 10
1024
>>> 3 ** 3
27
>>> 9 ** 0.5
3.0
```

Auch die Bit-Operatoren sind mit denjenigen in den anderen Programmiersprachen identisch. Eine detaillierte Erläuterung mitsamt Binärdarstellungen finden Sie in Abschnitt 9.1 zur Programmiersprache C; hier sehen Sie nur je ein kommentiertes Beispiel für jeden dieser Operatoren:

```
>>> 3 & 7   # bitweise Und
3
>>> 3 | 8   # bitweise Oder
11
>>> 3 ^ 7   # bitweise Exklusiv Oder
4
>>> ~7      # Bit-Umkehrung
-8
>>> 17 << 2 # Bit-Verschiebung nach links
68
>>> 17 >> 2 # Bit-Verschiebung nach rechts
4
```

Alle arithmetischen und Bit-Operatoren können mit einem Gleichheitszeichen kombiniert werden, um als spezielle Wertzuweisungsoperatoren eine Variable zu modifizieren. Hier nur zwei Beispiele:

```
>>> a = 7
>>> a += 6          # Wert von a um 6 erhöhen
>>> a
13
>>> a <<= 2         # Wert von a um 2 Bit nach links verschieben
>>> a
52
```

Vergleichsoperatoren gibt es in Python natürlich ebenfalls, und auch sie sind denjenigen in den anderen Programmiersprachen ähnlich: `<` (kleiner als), `>` (größer als), `<=` (kleiner oder gleich), `>=` (größer oder gleich) und `==` (gleich) sind identisch; für `!=` (ungleich) existiert `<>` als gleichberechtigte alternative Schreibweise. Der Rückgabewert jeder Vergleichsoperation ist `True`, wenn sie zutrifft, und andernfalls `False`.

Bei numerischen Werten verhalten sich diese Operatoren wie erwartet; dabei wird bei Bedarf automatisch zwischen Fließkomma- und Ganzzahlen konvertiert:

```
>>> 23 == 23
True
>>> 23 < 23
False
>>> 42 == 42.0
True
```

Eine weitergehende Konvertierung, etwa zwischen Strings und numerischen Werten, erfolgt nicht, wie das folgende Beispiel zeigt:

```
>>> 3 == "3"
False
```

Vergleiche, die eine Ordnung der Elemente erfordern, ergeben sogar eine Fehlermeldung, wenn die Typen inkompatibel sind:

```
>>> 3 < "3"
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unorderable types: int() < str()
```

Ansonsten können Vergleichsoperatoren auch für Strings, Listen und andere Typen verwendet werden, solange die verglichenen Objekte denselben Typ haben. Bei Strings ist die Ordnung die Position der enthaltenen Zeichen im Zeichensatz, während bei Listen sowohl

die Anzahl der Elemente als auch deren interne Ordnung betrachtet werden. Hier einige Beispiele:

```
>>> "hallo" == "hello"
False
>>> "hallo" < "hello"
True
>>> "comp" < "computer"
True
>>> [1, 2, 3] == [1, 2, 3]
True
>>> [1, 2, 3] == [1, 2, 3, 4]
False
>>> [1, 2, 3] < [1, 2, 3, 4]
True
>>> ["a", "b", "c"] < ["b", "c", "d"]
True
```

Bei gerichteten Vergleichen von Listen müssen alle Elemente beider Listen denselben (oder einen konvertierbaren) Typ haben, ansonsten führt der Versuch wiederum zu Fehlermeldungen:

```
>>> [1, 2, 3] == [1.0, 2.0, 3.0]    # konvertierbar
True
>>> [1, 2, 3] < ["a", "b", "c"]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unorderable types: int() < str()
>>> [1, "a"] < ["a", 1]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unorderable types: int() < str()
```

Eine Besonderheit von Python ist, dass die logischen Operatoren nicht durch Sonderzeichen, sondern mithilfe der Wörter `and` (logisches Und), `or` (logisches Oder) beziehungsweise `not` (logische Verneinung) ausgedrückt werden. Ansonsten funktionieren sie wie in den anderen bisher behandelten Sprachen.

Ein Python-spezifischer Operator ist der *Identitätsoperator* `is`, der nur dann `True` ergibt, wenn dasselbe Objekt referenziert wird. Hier ein Beispiel dafür, wie er funktioniert:

```
>>> var1 = [1, 2, 3]
>>> var2 = var1
>>> var1 is var2
```

```
True
>>> [1, 2, 3] is [1, 2, 3]
False
```

Da `var1` und `var2` auf dasselbe veränderliche Objekt verweisen, sind sie identisch, sodass `var1 is var2` den Wert `True` zurückliefert. Die beiden identisch aussehenden veränderlichen Literale mit dem Wert `[1, 2, 3]` sind dagegen nicht dasselbe Objekt, und `is` liefert `False` zurück. Aus Bequemlichkeitsgründen ist auch der Umkehroperator `is not` definiert; `a is not b` ist dabei die Umkehrung von `a is b` und eine alternative Schreibweise für `not(a is b)`[2].

Schließlich gibt es noch den Element-Operator `in`, der `True` zurückgibt, wenn der linke Operand im rechten enthalten ist. Er funktioniert sowohl zum Testen von Teil-Strings in anderen Strings als auch für skalare Werte als Elemente in Listen und anderen zusammengesetzten Typen. Hier sehen Sie ein paar Beispiele:

```
>>> "al" in "Hallo"
True
>>> "ha" in "Hallo"
False
>>> "Hallo" in "Hallo"
True
>>> 3 in [1, 2, 3, 4]
True
>>> 3 in ["1", "2", "3", "4"]
False
```

Wie Sie sehen, wird bei Strings zwischen Groß- und Kleinschreibung unterschieden: `"ha" in "Hallo"` ergibt `False`. Werden zwei identische Strings verglichen, gilt der linke als Teil-String des rechten; `"Hallo" in "Hallo"` ergibt `True`. Und wie üblich werden Ziffern nicht automatisch in Strings konvertiert oder umgekehrt, sodass der String `"3"` nicht in der Liste `[1, 2, 3, 4]` enthalten ist. Bei Listen kann der Operator übrigens nur zum Prüfen auf einzelne Elemente verwendet werden; zum Testen auf Teilmengen kommen `<` beziehungsweise `<=` zum Einsatz. Genau wie beim Identitätsoperator ist auch hier eine Umkehrung definiert; sie heißt `not in` und liefert jeweils das Gegenteil von `in` zurück.

Die Rangfolge der Operatoren ist in Python wie folgt (je weiter oben ein Operator steht, desto stärker bindet er und desto früher wird er ausgewertet):

- Exponent (**)
- Bit-Komplement (~), Plus als Vorzeichen (+) und Minus als Vorzeichen (-)
- Multiplikation (*), Division (/), ganzzahlige Division (//) und Modulo (%)

2 Tatsächlich können Sie sogar `not a is b` schreiben, also die Klammern weglassen, da die logischen Operatoren in Python die niedrigste Priorität haben.

- Addition (+) und Subtraktion (-)
- Bit-Verschiebung nach links (<<) und nach rechts (>>)
- bitweise Und (&)
- bitweise Oder (|) und Exklusiv-Oder (^)
- kleiner als (<), kleiner oder gleich (<=), größer als (>) und größer oder gleich (>=)
- gleich (==) und ungleich (!= oder <>)
- Zuweisungs- und Modifikationsoperatoren (=, +=, -=, *=, /= etc.)
- Identität (`is`) und Nicht-Identität (`is not`)
- Element (`in`) und nicht Element (`not in`)
- logisches Und (`and`), logisches Oder (`or`) und logische Verneinung (`not`)

Wie üblich kann die Rangfolge mithilfe von Klammern verändert werden. Beispiel:

```
>>> 23 + 42 * 2
107
>>> (23 + 42) * 2
130
```

Listen, Mengen und andere Typen mit mehreren Elementen

Neben den bisher hauptsächlich behandelten skalaren Datentypen bietet Python einige sehr praktische Typen mit mehreren Elementen. Einige der wichtigsten dieser Typen werden in diesem Unterabschnitt behandelt; im nächsten erfahren Sie dann, wie Sie diese Objekte elementweise durchlaufen (iterieren) können.

Viele Sammlungstypen sind immer verfügbar

Anders als die Collections in Java gehören die meisten mehrgliedrigen Datentypen in Python zum Sprachkern. Es muss also nichts importiert werden, um sie zu benutzen.

Die einfache *Liste* ist der gängigste dieser Datentypen. Es handelt sich um eine index-sortierte Sammlung beliebig vieler Elemente beliebigen Typs. *Index-sortiert* bedeutet, dass die Ordnung nicht in der Größe der Elemente selbst begründet ist, sondern in der Reihenfolge, in der sie sich in der Liste befinden. Eine Liste wird durch ein Paar eckiger Klammern gekennzeichnet, und die Elemente darin werden durch Kommata voneinander getrennt. Das folgende Beispiel erzeugt eine Liste verschiedener ganzer Zahlen:

```
>>> list1 = [7, 4, 9, 16, 1, 7]
>>> list1
[7, 4, 9, 16, 1, 7]
```

Der Zugriff auf einzelne Elemente erfolgt durch den Indexoperator, der ebenfalls durch eckige Klammern dargestellt wird. Der Index des ersten Elements ist dabei 0. Hier zwei Beispiele:

```
>>> list1[0]
7
>>> list1[4]
1
```

Mit negativen Indexwerten lässt sich übrigens vom Ende her gerechnet auf eine Liste zugreifen; der Indexwert `-1` entspricht dabei dem letzten Element:

```
>>> satz = ["Das", "ist", "ja", "wohl", "das", "Letzte"]
>>> satz[-1]
'Letzte'
>>> satz[-3]
'wohl'
```

Sie können den Indexoperator auch benutzen, um dem entsprechenden Element einer Liste einen anderen Wert zuzuweisen. Beispiel:

```
>>> peoples_front_of_judea = ["Reg", "Stan", "Judith", "Brian"]
>>> peoples_front_of_judea
['Reg', 'Stan', 'Judith', 'Brian']
>>> peoples_front_of_judea[1] = "Loretta"
>>> peoples_front_of_judea
['Reg', 'Loretta', 'Judith', 'Brian']
```

Indizes können auch verwendet werden, um Teillisten zurückzuliefern; in diesem Fall werden sie als *Slice-Operator* (»Scheibe« oder »Ausschnitt«) bezeichnet. Die Schreibweise dafür ist `[Startindex:Endindex]`, wobei der `Endindex` dem Index des ersten Elements entspricht, das nicht mehr in der Teilliste enthalten sein soll. Das folgende Beispiel beginnt bei Index 1, also dem zweiten Element der Liste, und schließt das fünfte Element aus, sodass die Indizes 1 bis 3 enthalten sind, die den Elementen 2 bis 4 entsprechen:

```
>>> list1
[7, 4, 9, 16, 1, 7]
>>> list1[1:4]
[4, 9, 16]
```

Wenn zwei aufeinanderfolgende Indizes angegeben werden, ist das Ergebnis nicht etwa ein einzelnes Element, sondern eine Liste mit einem Element – beachten Sie die eckigen Klammern um das erste Ergebnis in diesem vergleichenden Beispiel:

```
>>> list1[2:3]   # Teilliste mit einem Element
[9]
>>> list1[2]     # einzelnes Element
9
```


Sie können noch ein drittes Argument hinter einen weiteren Doppelpunkt setzen; es gibt eine Schrittweite an. Sehen Sie sich dazu ein Beispiel mit einer Liste aufeinanderfolgender Elemente an:

```
>>> list2 = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> list2[2:8:2]
[2, 4, 6]
```

Teillisten sind Kopien der ursprünglichen Liste anstelle von Referenzen auf diese, das heißt, wenn sie Variablen zugewiesen werden, können sie unabhängig von der ursprünglichen Liste manipuliert werden. Die spezielle Schreibweise `[:]`, ganz ohne numerische Werte, erzeugt eine Kopie der gesamten Liste. Das folgende Beispiel erzeugt auf diese Weise eine Kopie von `list2`, hängt mithilfe der Methode `append()` ein weiteres Element an und zeigt anschließend, dass die beiden Listen verschieden sind:

```
>>> list3 = list2[:]
>>> list3.append(10)
>>> list3
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
>>> list2
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Auch der Slice-Operator kann zur Wertzuweisung an den entsprechenden Ausschnitt der Liste verwendet werden; der zugewiesene Wert muss in diesem Fall ebenfalls eine Liste sein. Bei einem einfachen Slice ohne Schrittweite können Sie unabhängig von der Größe des Ausschnitts beliebig viele Elemente einfügen. Hier ein Beispiel, das ein einzelnes Element durch drei neue ersetzt:

```
>>> liste = [0, 1, 2, 3]
>>> liste[1:2] = ["Neues Element 1", "Neues Element 2", "Neues Element 3"]
>>> liste
[0, 'Neues Element 1', 'Neues Element 2', 'Neues Element 3', 2, 3]
```

Wenn Sie dem Ausschnitt eine leere Liste zuweisen, können Sie auf diese Weise Elemente aus der Liste löschen:

```
>>> liste = ["Bleibt da", "kann weg", "kann auch weg", "bleibt auch da"]
>>> liste[1:3] = []
>>> liste
['Bleibt da', 'bleibt auch da']
```

Umgekehrt können Sie Elemente einfügen, ohne eines zu löschen, indem Sie denselben Index für das Start- und das ausschließende Endelement angeben:

```
>>> liste = ["vor den neuen", "nach den neuen"]
>>> liste[1:1] = ["neues Element 1", "neues Element 2"]
>>> liste
['vor den neuen', 'neues Element 1', 'neues Element 2', 'nach den neuen']
```

Wenn Sie Slices mit Schrittweite zur Wertzuweisung verwenden, dann muss die Anzahl der Elemente in der ersetzenden Liste mit der Anzahl der ersetzten Elemente übereinstimmen. Dies kann zum Beispiel so aussehen:

```
>>> liste = [0, 9, 2, 9, 4, 9, 6, 9]
>>> liste[1:8:2] = [1, 3, 5, 7]
>>> liste
[0, 1, 2, 3, 4, 5, 6, 7]
```

Wenn Sie nicht auf die korrekte Anzahl von Elementen achten, erhalten Sie eine Fehlermeldung:

```
>>> liste = ["a", "b", "c", "d", "e", "f", "g"]
>>> liste[1:7:2] = ["x", "x"]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: attempt to assign sequence of size 2 to extended slice of size 3
```

Die Operanden + und * haben besondere Bedeutungen für Listen: `Liste + Liste` hängt die beiden Listen aneinander, und `Liste * Integer` erzeugt eine neue Liste, die die enthaltenen Elemente so oft wiederholt, wie angegeben. In beiden Fällen werden Kopien der ursprünglichen Liste(n) erzeugt, falls Sie nicht die Modifikationsoperatoren += beziehungsweise *= verwenden. Beispiele:

```
>>> list_a = [1, 2, 3]
>>> list_b = [4, 5, 6]
>>> list_a + list_b
[1, 2, 3, 4, 5, 6]
>>> list_a * 2
[1, 2, 3, 1, 2, 3]
>>> list_a += list_b
>>> list_a
[1, 2, 3, 4, 5, 6]
>>> list_a *= 2
>>> list_a
[1, 2, 3, 4, 5, 6, 1, 2, 3, 4, 5, 6]
```

Ein anderer Datentyp mit mehreren Elementen ist das *Tupel* (englisch *tuple*). Es kann als unveränderliche Liste betrachtet werden, ist aber eher eine Art Datensatz mit einer festgelegten

Anzahl von Elementen. Ein Tupel-Literal wird als durch Kommata getrennte Liste in runde Klammern geschrieben, beispielsweise wie folgt:

```
>>> tuple1 = (1, 4, 9, 16, 25)
>>> tuple1
(1, 4, 9, 16, 25)
```

Soll ein Tupel nur ein Element haben, müssen Sie trotzdem ein Komma hinter dem Element einfügen, denn ein einzelner skalarer Wert in runden Klammern wird nicht als Tupel interpretiert:

```
>>> tuple2 = (1,)
>>> type(tuple2)
<class 'tuple'>
>>> test_tuple = (1)
>>> type(test_tuple)
<class 'int'>
```

Alternativ können Sie die eingebaute Funktion `tuple()` verwenden, um eine Standardliste in ein Tupel umzuwandeln:

```
>>> list = ["Brian", "Reg", "Loretta"]
>>> tuple(list)
('Brian', 'Reg', 'Loretta')
>>> tuple([4,5,6])
(4, 5, 6)
```

Auf Elemente eines Tupels wird wie bei Listen mit dem Index- oder Slice-Operator zugegriffen. Hier einige Beispiele:

```
>>> pfoj = ("Reg", "Loretta", "Judith", "Brian")
>>> pfoj[3]
'Brian'
>>> pfoj[2:4]
('Judith', 'Brian')
>>> pfoj[0:4:2]
('Reg', 'Judith')
```

Da ein Tupel eine unveränderliche Liste ist, können diese Operatoren natürlich nicht zur Wertzuweisung verwendet werden; der Versuch erzeugt eine Fehlermeldung:

```
>>> pfoj[1] = "Stan"
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
```

Die Operatoren + und * beziehungsweise += und *= können allerdings genau wie bei Listen verwendet werden. Betrachten Sie dazu die folgenden Beispiele:

```
>>> round_table = ("Arthur", "Lancelot")
>>> round_table * 2                    # modifizierte Kopie
('Arthur', 'Lancelot', 'Arthur', 'Lancelot')
>>> round_table
('Arthur', 'Lancelot')
>>> round_table + ("Galahad", "Robin")   # modifizierte Kopie
('Arthur', 'Lancelot', 'Galahad', 'Robin')
>>> round_table
('Arthur', 'Lancelot')
>>> round_table += ("Galahad", "Robin")  # modifiziertes Original
>>> round_table
('Arthur', 'Lancelot', 'Galahad', 'Robin')
>>> round_table *= 2                   # modifiziertes Original
>>> round_table
('Arthur', 'Lancelot', 'Galahad', 'Robin', 'Arthur', 'Lancelot', 'Galahad', 'Robin')
```

Ein anderer Typ mit mehreren Elementen ist die Menge (Klassenname `set`). Sie enthält eine ungeordnete Sammlung unterschiedlicher Elemente, das heißt, der Zugriff kann nicht mithilfe von Indexoperationen erfolgen. Mengen werden hauptsächlich verwendet, um zu prüfen, ob bestimmte Werte darin enthalten sind oder nicht, sowie für Mengenoperationen wie Vereinigungs-, Schnitt- oder Teilmenge.

Ein Mengen-Literal wird als kommaseparierte Liste von Werten in geschweiften Klammern geschrieben. Hier ein Beispiel:

```
>>> primes = {2, 3, 5, 7, 11, 13, 17, 19}
```

Diese Menge enthält alle Primzahlen unter 20, sie kann also verwendet werden, um zu überprüfen, ob eine Zahl eine solche Primzahl ist oder nicht. Dazu kommt der bereits besprochene Operator `in` zum Einsatz (beziehungsweise `not in`, um zu prüfen, ob eine Zahl keine Primzahl kleiner als 20 ist):

```
>>> primes = {2, 3, 5, 7, 11, 13, 17, 19}
>>> 11 in primes
True
>>> 15 in primes
False
>>> 9 not in primes
True
>>> 7 not in primes
False
```


Falls Sie bei der Wertzuweisung dasselbe Element mehrfach aufführen, wird es nur einmal in die Menge übernommen:

```
>>> test_set = {1, 1, 2, 2, 3, 3}
>>> test_set
{1, 2, 3}
```

Es wurde bereits erwähnt, dass die Elemente einer Menge ungeordnet sind; das bedeutet, dass die folgende Prüfung für Mengen `True` ergibt:

```
>>> {1, 2} == {2, 1}
True
```

Für geordnete Sammlungen wie Listen oder Tupel gilt dies selbstverständlich nicht, wie dieses Beispiel mit Listen zeigt:

```
>>> [1, 2] == [2, 1]
False
```

Beachten Sie, dass leere Mengen nicht als `{}` geschrieben werden dürfen; dies ist der Datentyp Dictionary, der im Anschluss behandelt wird. Die korrekte Schreibweise für leere Mengen ist `set()`. Die eingebaute Funktion `set()` kann auch verwendet werden, um die Elemente anderer Sammlungen in eine Menge zu übernehmen; betrachten Sie dazu das folgende Beispiel mit einem Tupel:

```
>>> tuple1 = (1, 2, 3, 4, 1)
>>> set1 = set(tuple1)
>>> set1
{1, 2, 3, 4}
```

Das im Tupel doppelt vorhandene Element 1 wird wie gehabt aus der Menge entfernt.

Wenn Sie prüfen möchten, ob eine Menge Teilmenge einer anderen ist, wenden Sie den Operator < an; entsprechend wird > für eine Obermenge eingesetzt. Beispiele:

```
>>> set1 = {1, 2, 3, 4}
>>> {1, 2} < set1
True
>>> {5} < set1
False
>>> set1 > {3}
True
>>> set1 > {5}
False
```

Da < und > für echte Teil- beziehungsweise Obermengen verwendet werden, ist eine Menge nicht Teil- oder Obermenge ihrer selbst. Die Operatoren <= und >= ergeben jedoch `True`, denn sie stehen für »Teilmenge oder gleich« beziehungsweise »Obermenge oder gleich«.

Weitere interessante Mengenoperationen sind schließlich Vereinigungs-, Schnitt- und Differenzmenge. Die Vereinigungsmenge wird mithilfe des Operators | gebildet, der bei numerischen Werten für bitweise Oder steht; sie enthält jedes Element beider ursprünglicher Mengen. Hier ein Beispiel:

```
>>> set1 = {1, 2, 3, 4}
>>> set2 = {4, 5, 6, 7}
>>> set1 | set2
{1, 2, 3, 4, 5, 6, 7}
```

Die Schnittmenge, dargestellt durch den Bitweise-Und-Operator &, enthält nur diejenigen Elemente, die in beiden Mengen vorkommen:

```
>>> set_a = {1, 2, 3, 4, 5}
>>> set_b = {4, 5, 6, 7, 8}
>>> set_a & set_b
{4, 5}
```

Die Differenz- oder Restmengenoperation schließlich entfernt aus der linken Menge alle diejenigen Elemente, die auch in der rechten Menge vorkommen; der zuständige Operand ist das Minuszeichen (-):

```
>>> original_set = {1, 2, 3, 4, 5}
>>> removal_set = {3, 4, 6, 7}
>>> original_set - removal_set
{1, 2, 5}
```

Alle drei Operanden können auch in der Wertzuweisungsvariante mit nachgestelltem Gleichheitszeichen verwendet werden, um die linke Menge selbst gemäß der gewünschten Operation zu verändern. Beispiele:

```
>>> test_set = {1, 2, 3, 4, 5, 6, 7, 8, 9}
>>> test_set
{1, 2, 3, 4, 5, 6, 7, 8, 9}
>>> test_set |= {9, 10, 11}             # Vereinigungsmenge
>>> test_set
{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11}
>>> test_set &= {1, 3, 5, 7, 8, 12, 14} # Schnittmenge
>>> test_set
{8, 1, 3, 5, 7}
```



```
>>> test_set -= {8, 1, 2}               # Differenzmenge
>>> test_set
{3, 5, 7}
```

Die unveränderliche Variante der Menge heißt `frozenset`. Für diese gibt es keine eigene Literal-Schreibweise; sie wird stets mithilfe des Funktionsaufrufs `frozenset()` gebildet, wobei Sie eine konventionelle Menge, eine Liste oder ein Tupel als Argument verwenden können. Der einzige Unterschied zur normalen Menge besteht darin, dass Sie bei einer unveränderlichen Menge keine Elemente hinzufügen oder entfernen können – dafür sind bei gewöhnlichen Mengen die Methoden `add()` und `remove()` zuständig. Die Modifikation per Vereinigungs-, Schnitt- oder Differenzmengenoperator (|=, &= beziehungsweise -=) funktioniert dagegen auch beim `frozenset`. Hier ein kurzes Beispiel, das zunächst eine Menge erstellt, dann ein `frozenset` mit dessen Elementen erzeugt und anschließend versucht, zu beiden mittels `add()` ein Element hinzuzufügen:

```
>>> set1 = {1, 2, 3}
>>> set2 = frozenset(set1)
>>> set1.add(4)
>>> set1
{1, 2, 3, 4}
>>> set2.add(4)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: 'frozenset' object has no attribute 'add'
>>> set2
frozenset({1, 2, 3})
```

Ein besonderer Sammlungsdatentyp ist das *Dictionary*. Es handelt sich um eine beliebig lange Liste von Schlüssel-Wert-Paaren, wobei die Schlüssel eindeutig sein müssen, die Werte jedoch nicht. In manchen anderen Programmiersprachen wird das Konzept als Hash oder als assoziatives Array bezeichnet.

Das Dictionary-Literal steht wie eine Menge in geschweiften Klammern; Schlüssel und Wert werden durch Doppelpunkte voneinander getrennt und die einzelnen Paare durch Kommata.

Anders als bei Listen muss ein Schlüssel kein numerischer Index sein, sondern kann jeder unveränderliche Datentyp sein. Am häufigsten werden Strings als Schlüssel verwendet. Das folgende Beispiel verwendet die Abkürzungen der Wochentage als Schlüssel und die ausgeschriebenen Varianten als Werte:

```
>>> {"Mo":"Montag", "Di":"Dienstag", "Mi":"Mittwoch",
... "Do":"Donnerstag", "Fr":"Freitag", "Sa":"Samstag",
... "So":"Sonntag"}
{'So': 'Sonntag', 'Do': 'Donnerstag', 'Mi': 'Mittwoch', 'Fr': 'Freitag',
'Mo': 'Montag', 'Sa': 'Samstag', 'Di': 'Dienstag'}
```

Wie Sie sehen, hat das Ergebnis keineswegs die Reihenfolge, die Sie ursprünglich angegeben haben (Python legt sie möglichst speicheroptimierend ab). Bei einem Dictionary geht es eben nicht um eine bestimmte Anordnung, sondern um die Beziehung zwischen den Schlüsseln und Werten.

Um anhand des Schlüssels auf ein Element zuzugreifen, wird der Schlüssel als Index verwendet:

```
>>> wochentage = {"Mo": "Montag", "Di": "Dienstag",
... "Mi": "Mittwoch", "Do": "Donnerstag", "Fr": "Freitag",
... "Sa": "Samstag", "So": "Sonntag"}
>>> wochentage["Sa"]
'Samstag'
```

Wenn Sie versuchen, auf ein nicht vorhandenes Element zuzugreifen, erhalten Sie eine Fehlermeldung, wie in diesem Beispiel:

```
>>> wochentage["X"]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'X'
```

Um diesen Fehler zu vermeiden, können Sie vor dem Zugriffsversuch über den Index den Operator `in` verwenden, um zu testen, ob ein bestimmter Schlüssel vorhanden ist. Beispiele:

```
>>> "Sa" in wochentage
True
>>> "X" in wochentage
False
```

Mit einem einfachen `in` können Sie allerdings nur das Vorhandensein von Schlüsseln überprüfen; die Suche nach einem bestimmten Wert hat nicht das gewünschte Ergebnis:

```
>>> "Samstag" in wochentage
False
```

Um nach Werten zu suchen, können Sie aber die Methode `values()` eines Dictionarys aufrufen; sie liefert eine spezielle Liste vom Typ `dict_values` zurück, in der Sie wiederum mittels `in` suchen können:


```
>>> "Samstag" in wochentage.values()
True
>>> "Pythontag" in wochentage.values()
False
>>> wochentage.values()
dict_values(['Freitag', 'Mittwoch', 'Sonntag',
'Dienstag', 'Donnerstag', 'Samstag', 'Montag'])
```

Genau wie bei Listen kann der Indexoperator übrigens auch beim Dictionary zum Hinzufügen oder Ändern von Werten verwendet werden. Beispiele:

```
>>> dict = {}                  # leeres Dictionary
>>> dict["banana"] = "yellow"  # Neu-Wertzuweisung
>>> dict
{'banana': 'yellow'}
>>> dict["apple"] = "red"
>>> dict
{'banana': 'yellow', 'apple': 'red'}
>>> dict["apple"] = "green"    # Wertänderung
>>> dict
{'banana': 'yellow', 'apple': 'green'}
```

Strings sind eine Art Mischtyp; ihr Inhalt kann als einzelner Wert oder als Abfolge einzelner Zeichen betrachtet werden. Deshalb können Sie auf Strings auch mit dem Index- oder Slice-Operator zugreifen, allerdings nur lesend. Hier einige Beispiele:

```
>>> text = "Dies ist ein Test-String."
>>> text[5]
'i'
>>> text[5:8]
'ist'
>>> text[0:10:2]
'De s '
>>> text[-1]
'.'
```

Die Funktion `range()` stellt keine richtige Liste bereit, sondern einen Wertebereich, auf den Sie jedoch ebenfalls mit dem Indexoperator zugreifen können. Es können ein bis drei Argumente übergeben werden:

- `range(n)` umfasst den Bereich von 0 bis `n`-1.
- `range(m, n)` umfasst den Bereich von `m` bis `n`-1.
- `range(m, n, step)` umfasst den Bereich von `m` bis `n`-1 mit der Schrittweite `step` beziehungsweise von `m` bis `n`+1 bei negativer Schrittweite.

Mit `list(range(...))` können Sie alle im Bereich enthaltenen Zahlen in eine Liste übernehmen, wodurch sich die Elemente am einfachsten untersuchen lassen. Hier einige Beispiele:

```
>>> list(range(10))         # 0 bis 10-1
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> list(range(5, 10))      # 5 bis 10-1
[5, 6, 7, 8, 9]
>>> list(range(2, 20, 2))   # 2 bis 20-1, Schrittweite 2
[2, 4, 6, 8, 10, 12, 14, 16, 18]
>>> list(range(20, 2, -2))  # 20 bis 2+1, Schrittweite -2
[20, 18, 16, 14, 12, 10, 8, 6, 4]
```

Ein praktisches Hilfsmittel für alle Aufzählungstypen ist die Funktion `len()`, die die Anzahl der Elemente zurückgibt. Hier zwei Beispiele, eines für eine Liste und eines für einen String, bei dem entsprechend die Anzahl der Zeichen ermittelt wird:

```
>>> len([1, 2, 3, 4])
4
>>> len("Hello world")
11
```

Kontrollstrukturen

Um den Programmablauf zu steuern, besitzt Python wie jede Programmiersprache *Kontrollstrukturen*. Diese lassen sich wie üblich in Fallentscheidungen und Schleifen unterteilen – eine Fallentscheidung verzweigt nur einmal aufgrund des Wahrheitswertes eines Ausdrucks, während eine Schleife unter Umständen mehrmals durchlaufen wird.

Die einfachste *Fallentscheidung* hat folgende Struktur:

```
if Ausdruck:
    Anweisung
    ...
# Normaler Programmablauf geht hier weiter
```

Wenn der überprüfte Ausdruck `True` ist oder als `True` interpretiert wird, werden die abhängigen Anweisungen ausgeführt, andernfalls geht der Programmablauf direkt in derjenigen Zeile weiter, die wieder die Einrückung der `if`-Anweisung aufweist. Die Zahl `0`, der leere String `""`, leere Aufzählungstypen wie die leere Liste `[]` oder das leere Dictionary `{}` sowie `None` werden als `False` interpretiert, alle anderen Nicht-Boolean-Ausdrücke als `True`.

Wie bereits erwähnt, bestimmt die Einrückung, welche Zeilen von der Fallentscheidung abhängen und welche nicht. Dabei ist es wichtig, dass die Einrückung aller betroffenen Zeilen identisch ist. Empfehlenswert sind vier Leerzeichen pro Stufe; die Hauptsache ist aber, dass Sie sich konsequent an die einmal gewählte Form halten.

Das folgende Beispiel überprüft, ob die Variable `favorite_color` den Wert `"red"` hat, und gibt in diesem Fall die Meldung `"You may pass!"` aus:

```
>>> favorite_color = "red"
>>> if favorite_color == "red":
...     print("You may pass!")
...
You may pass!
```

Da `favorite_color` zuvor der Wert `"red"` zugewiesen wurde, erfolgt die erwartete Ausgabe aus der abhängigen Codezeile. Wenn Sie dasselbe Beispiel erneut mit einem anderen Wert für `favorite_color` ausführen, erfolgt keine Ausgabe:

```
>>> favorite_color = "blue"
>>> if favorite_color == "red":
...     print("You may pass!")
...
>>>
```


Besonderheit der interaktiven Shell

Beachten Sie, dass Sie den `if`-Block in der interaktiven Python-Shell mit einer Leerzeile abschließen müssen, sodass er sofort ausgeführt wird. In einem gespeicherten Skript können Sie dagegen mit einer nicht mehr eingerückten Zeile fortfahren, bei der der Programmablauf unabhängig von der Fallentscheidung in jedem Fall weitergeht.

Wie in den meisten Programmiersprachen kann die `if`-Fallentscheidung auch in Python einen `else`-Zweig haben, dessen abhängige Anweisungen ausgeführt werden, wenn der überprüfte Ausdruck nicht `True` ist. Sehen Sie sich dazu dieses erweiterte Beispiel an:

```
>>> favorite_color = "red"
>>> if favorite_color == "red":
...     print("You may pass!")
... else:
...     print("You shall not pass!")
...
You may pass!
```

Wie im ersten Beispiel erfolgt die Ausgabe des `if`-Zweigs, aber die Gegenprobe zeigt, dass für eine andere Lieblingsfarbe der `else`-Zweig ausgeführt wird:

```
>>> favorite_color = "blue"
>>> if favorite_color == "red":
...     print("You may pass!")
... else:
```

```
...     print("You shall not pass!")
...
You shall not pass!
```

Für den Fall, dass das erste `if` nicht zutrifft, können Sie einen weiteren Ausdruck überprüfen; das Schlüsselwort dafür heißt `elif` (Abkürzung für »else if«). Trifft der darin überprüfte Ausdruck zu, werden die zugehörigen Anweisungen ausgeführt, andernfalls geht es bei einem eventuell vorhandenen `else`-Zweig, beim nächsten `elif` oder im normalen Programmablauf weiter. Das folgende Beispiel überprüft eine weitere potenzielle Lieblingsfarbe und gibt einen eigenen Kommentar dazu aus:

```
>>> favorite_color = "pink"
>>> if favorite_color == "red":
...     print("You may pass!")
... elif favorite_color == "pink":
...     print("Close, but not close enough -- you failed!")
... else:
...     print("You shall not pass!")
...
Close, but not close enough -- you failed!
```

Eine spezielle Schreibweise von `if` und `else` – ohne `elif` – ist der *konditionale Ausdruck* (Englisch *conditional expression*), der am ehesten dem aus C bekannten ternären Fallentscheidungsoperator `Ausdruck ? Dann-Wert : Sonst-Wert` entspricht. Das Format ist wie folgt:

```
Dann-Wert if Ausdruck else Sonst-Wert
```

Der `else`-Teil darf dabei nicht weggelassen werden. Hier zwei einfache Beispiele, je eines für den Dann-Fall und eines für den Sonst-Fall:

```
>>> a = 5
>>> "Ja, 5" if a == 5 else "Nein, keine 5"
'Ja, 5'
>>> "Ja, 4" if a == 4 else "Nein, keine 4"
'Nein, keine 4'
```

Sie können diese Art von Ausdrücken einer Variablen zuweisen; beachten Sie aber, dass der konditionale Ausdruck in diesem Fall in Klammern stehen muss:

```
>>> monster = "Jabberwocky"
>>> what_to_do = ("Run!" if monster == "Jabberwocky" else "Never mind")
>>> what_to_do
'Run!'
>>> monster = "Killer Rabbit"
```



```
>>> what_to_do = ("Run!" if monster == "Jabberwocky" else "Never mind")
>>> what_to_do
'Never mind'
```

Auch in zusammengesetzten Ausdrücken können Sie dieses Format verwenden, ebenfalls in Klammern:

```
>>> holy_grail = True
>>> "You have found" + (" the Holy Grail" if holy_grail else " nothing special")
'You have found the Holy Grail'
>>> holy_grail = False
>>> "You have found" + (" the Holy Grail" if holy_grail else " nothing special")
'You have found nothing special'
```


Die einfachste *Schleife* ist die `while`-Schleife, deren Syntax folgendermaßen aussieht:

```
while Ausdruck:
    Anweisung
    ...
# Normaler Programmablauf
```

Die verschachtelten Anweisungen werden immer wieder ausgeführt, solange der überprüfte Ausdruck `True` ergibt. Hier ein interaktives Beispiel, das mittels `input()` Benutzereingaben entgegennimmt und dies wiederholt, bis die gewünschte Eingabe erfolgt ist:

```
>>> passwort = "geheim"
>>> eingabe = ""
>>> while eingabe != passwort:
...     eingabe = input("Passwort bitte: ")
...
Passwort bitte: pass
Passwort bitte: test
Passwort bitte: geheim
>>>
```

Natürlich können Sie die `while`-Schleife beispielsweise auch zum Zählen verwenden, wenn Sie die Zählervariable innerhalb des Schleifenrumpfs, also in den abhängigen Anweisungen, manipulieren (allerdings gibt es wesentlich elegantere Konstrukte zum Zählen, wie Sie bald sehen werden). Das folgende Beispiel listet die Zahlen von 0 bis 4 auf:

```
>>> i = 0
>>> while i < 5:
...     print(i)
...     i += 1
...
```

```
0
1
2
3
4
```

else-Blöcke in while-Schleifen

Eine Python-Besonderheit ist die Tatsache, dass `while`-Schleifen einen `else`-Block haben können. Dieser wird ausgeführt, falls der geprüfte Ausdruck von Anfang an `False` war, sodass der Schleifenrumpf niemals ausgeführt wird. Beispiel:

```
>>> a = 10
>>> while a < 10:
...     print(a)
...     a += 1
... else:
...     print("a war nie kleiner als 10.")
...
a war nie kleiner als 10.
```

Die `for`-Schleife wird in Python verwendet, um über die Elemente eines Objekts zu *iterieren*, das bedeutet, diese Elemente der Reihe nach durchzugehen. Deshalb wird `for` auch als *Iterator* bezeichnet. Die Syntax ist wie folgt:

```
for Variable in Aufzählung:
    Anweisung  # Variable hat nacheinander die Werte der Elemente in der Aufzählung
    ...
# Normaler Programmablauf
```

Betrachten Sie als erstes Beispiel die Iteration über die folgende Liste (es handelt sich um die römisch-dekadenten Snacks, die Brian in Monty Pythons »Life of Brian« im Amphitheater verkauft):

```
>>> snacks = [
... "Wolf Nipple Chips",
... "Dromedary Pretzels",
... "Jaguar Ear Lobes",
... "Tuscany Fried Bat",
... "Otter Noses"]
>>> for snack in snacks:
...     print("Get your " + snack + " while supplies last!")
...
Get your Wolf Nipple Chips while supplies last!
```



```
Get your Dromedary Pretzels while supplies last!
Get your Jaguar Ear Lobes while supplies last!
Get your Tuscany Fried Bat while supplies last!
Get your Otter Noses while supplies last!
```

Mit einer `for`-Schleife können Sie über jeden Aufzählungstyp iterieren, außerdem über einige andere Typen – beispielsweise die einzelnen Zeichen eines Strings:

```
>>> for letter in "letters":
...     print("Das Wort enthält den Buchstaben " + letter)
...
Das Wort enthält den Buchstaben l
Das Wort enthält den Buchstaben e
Das Wort enthält den Buchstaben t
Das Wort enthält den Buchstaben t
Das Wort enthält den Buchstaben e
Das Wort enthält den Buchstaben r
Das Wort enthält den Buchstaben s
```

Bei einem Dictionary wird über die Schlüssel iteriert; der Zugriff auf die Werte erfolgt wie üblich durch den Indexoperator. Beispiel:

```
>>> countries = {"de": "Deutschland", "at": "Österreich", "fr": "Frankreich"}
>>> for tld in countries:
...     print(countries[tld] + " hat die Top-Level-Domain " + tld)
...
Frankreich hat die Top-Level-Domain fr
Österreich hat die Top-Level-Domain at
Deutschland hat die Top-Level-Domain de
```

Die `for`-Schleife kann auch innerhalb eines Ausdrucks in eckigen Klammern verwendet werden, um eine neue Liste zu erzeugen; ein solches Konstrukt wird als *List Comprehension* bezeichnet. Die Syntax ist wie folgt:

```
neue_liste = [ausdruck for element in original_liste]
```

Die Liste `neue_liste` wird mit den Elementen befüllt, die aus der Iteration über `original_liste` befüllt werden. Hier ein Beispiel, das die Quadrate der Zahlen von 1 bis 10 berechnet und sie in einer neuen Liste speichert:

```
>>> zahlen = list(range(1, 11))
>>> quadrate = [zahl ** 2 for zahl in zahlen]
>>> quadrate
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

Die Konstruktion kann auch zusammen mit `if` verwendet werden, um Listen nach bestimmten Kriterien zu filtern. Das Ganze sieht schematisch so aus:

```
gefilterte_liste = [ausdruck for element in original_liste if ausdruck]
```

Die neue Liste `gefilterte_liste` enthält danach den berechneten Ausdruck für alle diejenigen Elemente von `original_liste`, auf die der überprüfte Ausdruck zutrifft. Das folgende Beispiel filtert nur die geraden Zahlen aus einer fortlaufenden Liste natürlicher Zahlen heraus:

```
>>> zahlen = list(range(1, 11))
>>> gerade = [zahl for zahl in zahlen if zahl % 2 == 0]
>>> gerade
[2, 4, 6, 8, 10]
```

Ein- und Ausgabe auf der Konsole

Ein weiteres Hilfsmittel zum Bilden von Ausdrücken und Anweisungen sind die verschiedenen aufrufbaren Funktionen und Methoden. Als Beispiel werden hier einige beschrieben, die der Ein- und Ausgabe dienen.

Eine *Funktion* ist alleinstehend, also nicht auf ein Objekt bezogen. Die Funktion `print()` dient beispielsweise dazu, Text auf der Konsole auszugeben:

```
>>> print("Hallo Python")
Hallo Python
```

Eine *Methode* ist dagegen eine Funktion innerhalb eines Objekts, die in dessen Klasse definiert ist. Sie wird durch einen Punkt getrennt hinter das Objekt geschrieben. Ein Beispiel ist die Methode `append()` einer Liste; sie dient dazu, Elemente anzuhängen. Beispiel:

```
>>> list = [1, 2, 3, 4, 5]
>>> list
[1, 2, 3, 4, 5]
>>> list.append(6)
>>> list
[1, 2, 3, 4, 5, 6]
```

Zum Schreiben von Programmen werden stets Funktionen zur Ein- und Ausgabe benötigt. Soweit die Konsole betroffen ist, sind diese im Sprachkern von Python enthalten (einige Dateizugriffsfunktionen sind ebenfalls eingebaute Funktionen, aber nicht alle). Es handelt sich im Wesentlichen um die Funktionen `print()` zur Ausgabe und `input()` zur Eingabe.

`print()` nimmt im einfachsten Fall einen String oder ein als String interpretierbares Objekt entgegen und gibt dessen Wert, gefolgt von einem Zeilenumbruch, auf der Konsole aus:


```
>>> print("Hello Python")
Hello Python
>>> print(4 + 3)
7
>>> print(1 == 2)
False
```

Sie können beliebig viele durch Komma getrennte Ausdrücke ausgeben lassen, wobei diese standardmäßig durch ein Leerzeichen und nicht durch einen Zeilenumbruch voneinander getrennt werden. Beispiele:

```
>>> print("Hello", "World,", "hello", "Python!")
Hello World, hello Python!
>>> print(3, "+", 4, "=", 3 + 4)
3 + 4 = 7
```

Sie können das Leerzeichen zwischen den einzelnen durch Komma getrennten Argumenten sowie den Zeilenumbruch am Ende durch andere Zeichen oder Zeichenfolgen ersetzen. Dazu werden *benannte Argumente* eingesetzt, eine Besonderheit von Python: nach einer definierten oder auch beliebigen Anzahl nicht benannter Argumente können solche in der Form `Schlüsselwort = Wert` folgen, die meistens für Konfigurationsaufgaben verwendet werden.

Zwei der möglichen benannten Argumente für `print()` sind `sep` (kurz für Separator) als Trenn-String zwischen den einzelnen Standardargumenten und `end` für den String am Ende. Das folgende Beispiel verwendet ein Komma und ein Leerzeichen als Separator und einen Punkt, gefolgt von einem Zeilenumbruch, als Endmarkierung:

```
>>> print("Arthur", "Galahad", "Lancelot", "Robin", sep = ", ", end = ".\n")
Arthur, Galahad, Lancelot, Robin.
```

Für Benutzereingaben kommt die Funktion `input()` zum Einsatz. Sie wartet, bis der Benutzer zur Laufzeit des Programms eine Eingabe vorgenommen und diese mit ↵ abgeschlossen hat. Anschließend gibt sie den Inhalt der Benutzereingabe (ohne den abschließenden Zeilenumbruch) als Wert zurück. Dadurch kann dieser Wert beispielsweise einer Variablen zugewiesen werden. Das folgende Beispiel nimmt eine Eingabe entgegen und speichert sie in der Variablen `eingabe`:

```
>>> eingabe = input()
hallo
>>> eingabe
'hallo'
```

Optional können Sie der Funktion ein Argument übergeben, dessen String-Wert als Eingabeaufforderung ausgegeben wird:

```
>>> name = input("Ihr Name? ")
Ihr Name? Sascha
>>> name
'Sascha'
```

Mit Dateien arbeiten

Für den Zugriff auf Dateien werden diese zunächst mit `open()` geöffnet; das Ergebnis ist im Erfolgsfall ein Objekt, dessen Methoden etwa zum Lesen und Schreiben von Daten in der Datei verwendet werden. Die wichtigsten Parameter von `open()` sind der Dateiname als erstes, unbenanntes Argument sowie der Modus (Lesen, Schreiben etc.) als zweites Argument oder – empfehlenswerter – als benanntes Argument `mode`.

Das folgende Beispiel öffnet die Datei *test.txt* im aktuellen Verzeichnis zum Lesen (`mode = "r"`; dies ist eigentlich Standard und könnte daher weggelassen werden) und weist das erhaltene Objekt der Variablen `file` zu; *test.txt* muss dazu natürlich existieren:

```
>>> file = open("test.txt", mode = "r")
```

Um den gesamten Inhalt der Datei auszulesen, wird die Methode `read()` des Objekts `file` aufgerufen:

```
>>> content = file.read()
>>> content
'Dies ist eine Textdatei\nSie hat drei Zeilen\nDie letzte endet nicht mit
einem Zeilenumbruch\n'
```

Wie Sie sehen, fügt `read()` am Ende des ausgelesenen Textes einen Zeilenumbruch ein, obwohl die Originaldatei in der letzten Zeile keinen enthält, wie der Inhalt vermuten lässt.

Wenn Sie ein zweites Mal `read()` auf dieselbe Datei aufrufen, erhalten Sie einen leeren String als Ergebnis:

```
>>> file.read()
''
```

Das liegt daran, dass intern ein *Dateizeiger* (englisch *file cursor*) verwendet wird, der die aktuelle Position markiert. Mithilfe der Methode `seek(Position)` können Sie den Zeiger auf eine andere Position setzen. Hier ein Beispiel, das zum dritten Zeichen der Datei (Position 2) springt und von dort erneut liest:

```
>>> file.seek(2)
2
>>> file.read()
'es ist eine Textdatei\nSie hat drei Zeilen\nDie letzte endet nicht mit
einem Zeilenumbruch\n'
```


Das File-Objekt stellt (zumindest für Textdateien) auch einen Iterator bereit, mit dem Sie die Datei zeilenweise auslesen können. Das folgende Beispiel kehrt zum Anfang der Datei zurück, liest sie in einer `for`-Schleife zeilenweise ein und gibt die Zeilen aus; die String-Methode `strip()` entfernt dabei den jeweiligen Zeilenumbruch am Ende:

```
>>> for line in file:
...     print(line.strip())
...
Dies ist eine Textdatei
Sie hat drei Zeilen
Die letzte endet nicht mit einem Zeilenumbruch
```

Neben `mode = "r"` gibt es diverse weitere Modi, um die Datei zum Lesen, zum Schreiben oder für diverse Mischungen daraus mit verschiedenen Optionen zu öffnen. Mit `mode = "w"` (write) öffnen Sie die Datei nur zum Schreiben; falls sie existiert, wird sie überschrieben, ansonsten neu angelegt. Auch `mode = "a"` (append) öffnet die Datei zum Schreiben und legt sie gegebenenfalls neu an; falls sie jedoch existiert, wird der Dateizeiger an ihr Ende gesetzt, sodass Sie weiteren Inhalt hinzufügen können.

Um in eine Datei zu schreiben, wird die Methode `write()` des File-Objekts verwendet. In diesem Fall müssen Sie sich selbst um die abschließenden Zeilenumbrüche kümmern, falls Sie welche benötigen. Alternativ können Sie die Funktion `print()` verwenden, wenn Sie darin den benannten Parameter `file` auf Ihr Dateiobjekt setzen.

Das folgende Beispiel öffnet die Datei *ausgabe.txt* zum Schreiben und schreibt mit beiden Verfahren je eine Zeile hinein. Anschließend wird die Datei mit `close()` geschlossen, dann erneut zum Lesen geöffnet und ihr Inhalt eingelesen:

```
>>> file = open("ausgabe.txt", "w")
>>> file.write("Erste Zeile, mit write() geschrieben.\n")
38
>>> print("Zweite Zeile, mit print() geschrieben.", file = file)
>>> file.close()
>>> file = open("ausgabe.txt", "r")
>>> file.read()
'Erste Zeile, mit write() geschrieben.\nZweite Zeile, mit print() geschrieben.\n'
```

Die speziellen Modi `"r+"`, `"w+"` und `"a+"` öffnen eine Datei zum kombinierten Lesen und Schreiben. `"r+"` führt zu einer Fehlermeldung, falls die Datei nicht existiert, während `"w+"` und `"a+"` sie in diesem Fall neu anlegen. `"w+"` ersetzt eine eventuell vorhandene Datei; `"r+"` und `"a+"` lassen sie bestehen. `"a+"` schreibt stets ans Ende der vorhandenen Datei – unabhängig von der Position des Dateizeigers, an der gelesen wird. `"r+"` und `"w+"` setzen den Dateizeiger dagegen zunächst an den Anfang und beachten dessen Position sowohl beim Lesen als auch beim Schreiben.

Hier ein Beispiel, das "w+" verwendet und in einer Datei schreibt, liest und den Zeiger positioniert, um die (durch ihre Position gekennzeichneten) ungeraden Ziffern durch Leerzeichen zu ersetzen:

```
>>> file = open("even.txt", "w+")
>>> file.write("01234567890")
11
>>> file.seek(0)
0
>>> file.read()
'01234567890'
>>> for i in [1, 3, 5, 7, 9]:
...     file.seek(i)
...     file.write(" ")
...
1
1
3
1
5
1
7
1
9
1
>>> file.seek(0)
0
>>> file.read()
'0 2 4 6 8 0'
```

Die vielen Ausgabezeilen der Schleife sind die jeweiligen Positionen von `seek()` und die Anzahl der von jedem `write()` geschriebenen Zeichen (1). Es handelt sich nicht um Ausgaben, die in einem regulären Python-Skript auf der Konsole erscheinen würden, sondern um die übliche Anzeige von Funktionsrückgabewerten innerhalb der interaktiven Shell.

Strings formatieren

Besonders interessant für die Ausgabe ist die String-Formatierungsmethode `format()` der String-Klasse (interner Name `str`). Die Methode ähnelt der C-Funktion `sprintf()`, ist jedoch noch praktischer und vielseitiger als diese. Das allgemeine Format lautet:

Format-String`.format(`*Wert1*`, `*Wert2*`, ...)`

Der Format-String enthält Platzhalter, die durch die Werte ersetzt werden. Der einfachste Platzhalter ist `{}`; er wird gemäß seiner Position im String durch das entsprechende Argument in seiner Standard-String-Darstellung ersetzt. Hier ein einfaches Beispiel mit zwei derartigen Platzhaltern:

```
>>> "{}, {}!".format("Hello", "world")
'Hello, world!'
```

Wenn Sie die Reihenfolge bestimmen möchten, in der die Argumente eingesetzt werden, können Sie deren bei 0 beginnenden Index in die geschweiften Klammern schreiben. Dadurch können Sie auch dasselbe Argument mehrfach verwenden. Beispiel:

```
>>> "It's a {1}, {1} {0}!".format("world", "mad")
"It's a mad, mad world!"
```

Noch praktischer sind mitunter benannte Argumente, die beliebige Namen tragen können und deren Namen dann anstelle der numerischen Indizes in die geschweiften Klammern gesetzt werden:

```
>>> "{dividend} / {divisor} = {quotient}".format(
... dividend = 20,
... divisor = 5,
... quotient = 20 // 5)
'20 / 5 = 4'
```

Hinter einem Doppelpunkt können Sie innerhalb der geschweiften Klammern eine `printf()`-Formatangabe eintragen – und zwar unabhängig davon, ob davor ein numerischer Index, ein benanntes Argument oder gar nichts steht. Diese Formatangaben bestehen aus mehreren Elementen, von denen nur das letzte Pflicht ist:

1. Ein Füllzeichen – wenn der zu formatierende Wert kürzer ist als die angegebene Stellenzahl, wird mit dem hier angegebenen Zeichen aufgefüllt, ansonsten mit Leerzeichen.
2. die Mindestanzahl der Stellen als ganze Zahl
3. Ein Ausrichtungszeichen: `<` für explizit linksbündig, `>` für explizit rechtsbündig `=` für rechtsbündige numerische Werte mit Vorzeichen zu Beginn des verfügbaren Platzes oder `^` für zentriert. Wenn Sie das Zeichen weglassen, werden Zahlen rechtsbündig und Strings linksbündig ausgerichtet.
4. Ein Vorzeichen: Wenn Sie hier ein `+` einsetzen, werden auch positive Zahlen explizit durch ihr Vorzeichen (+) gekennzeichnet, andernfalls nur negative.
5. Die Anzahl der Nachkommastellen hinter einem Punkt – hier wird bei Bedarf gerundet, oder es werden Nullen hinzugefügt.
6. das gewünschte Format, zum Beispiel `s` für einen String, `d` für eine ganze Zahl oder `f` für eine Fließkommazahl

Hier drei Beispiele, die einige denkbare Fälle demonstrieren, nämlich Währungsformatierung (zwei Nachkommastellen), linksbündige String-Tabelle sowie rechtsbündige String-Tabelle mit speziellem Füllzeichen:

```
>>> "{:.2f} EUR, inkl. 19% Mwst.: {:.2f} EUR".format(1000, 1000 / 1.19 * 0.19)
'1000.00 EUR, inkl. 19% Mwst.: 159.66 EUR'
>>> "{0:15s} {1:15s} {2:15s}".format("iOS", "Android", "Windows Phone")
'iOS             Android         Windows Phone  '
>>> "{0:->15s}{1:->15s}{2:->15s}".format("iOS", "Android", "Windows Phone")
'------------iOS--------Android--Windows Phone'
```

9.3.3 Objektorientierung in Python

Das Erstellen von Klassen, Attributen und Methoden funktioniert in Python etwas anders als in C; im Wesentlichen ist es weniger Schreibarbeit, da viele Details automatisch ablaufen. Beispielsweise werden Attribute nicht deklariert, da Python ohnehin keine fest typisierte Sprache ist, sondern sie entstehen durch Wertzuweisung innerhalb der Methoden beziehungsweise des Konstruktors.

Einführungsbeispiel

Am einfachsten lassen sich einige Grundlagen an einem Beispiel zeigen. Speichern Sie das folgende Listing unter dem Namen *buch.py*, und führen Sie es wie folgt auf der Konsole aus:

```
$ python3 buch.py
```

Das Beispiel definiert eine Klasse namens `Buch` mit diversen Eigenschaften, einem Konstruktor und einer speziellen Methode, erzeugt zwei Instanzen davon und gibt diese aus. Hier zunächst das vollständige Listing:

```
class Buch:
    # Konstruktor
    def __init__(self, autor, titel, jahr):
        self.autor = autor
        self.titel = titel
        self.jahr = jahr
    # String-Darstellung
    def __str__(self):
        return "{autor}: '{titel}' ({jahr})".format(
            autor = self.autor,
            titel = self.titel,
            jahr = self.jahr
        )
```



```
# Hauptprogramm

# Instanzen von Buch erzeugen
buch1 = Buch("Douglas Adams", "The Hitchhiker's Guide to the Galaxy", 1979)
buch2 = Buch("George R. R. Martin", "A Game of Thrones", 1996)

# Ausgabe
print(buch1)
print(buch2)
```

Wenn Sie das Skript ausführen, lautet die Ausgabe wie folgt:

```
$ python3 buch.py
Douglas Adams: 'The Hitchhiker's Guide to the Galaxy' (1979)
George R. R. Martin: 'A Game of Thrones' (1996)
```

Die Klassendefinition beginnt mit dem Schlüsselwort `class` (Geheimhaltungsstufen wie in Java gibt es in Python grundsätzlich nicht); hinter dem Klassennamen steht – wie in Blockstrukturen üblich – ein Doppelpunkt, und der Klassenrumpf wird darunter eingerückt:

```
class Buch:
    # Klassendefinition
```

Der Konstruktor ist in Python eine Methode mit dem speziellen Namen `__init__()` – vor und hinter dem Namen stehen je zwei Unterstriche. Methoden werden mit dem Schlüsselwort `def` gekennzeichnet; außerhalb von Klassen können Sie damit auch alleinstehende Funktionen definieren. Jede Methode erhält mindestens einen Parameter, der gemäß Konvention stets den Bezeichner `self` erhält, aber theoretisch jeden Namen haben kann. Das automatisch übergebene Argument ist dabei die aktuelle Instanz der Klasse, für die die Methode aufgerufen wird, beziehungsweise im Fall des Konstruktors die neu erzeugte Instanz.

Der Konstruktor der Klasse `Buch` sieht wie folgt aus:

```
def __init__(self, autor, titel, jahr):
    self.autor = autor
    self.titel = titel
    self.jahr = jahr
```

Die Attribute werden durch die Schreibweise `self.Attributname` gekennzeichnet und entstehen automatisch durch Wertzuweisung. Wie hier gezeigt, werden sie häufig durch die Werte von Parametern mit denselben Namen ohne vorangestelltes `self` initialisiert.

Auch für Attribute und Methoden gibt es in Python keine Geheimhaltungsstufen; Sie können mittels `objekt.attribut` von außen auf jedes beliebige Attribut zugreifen und mit `objekt.methode(...)` jede Methode aufrufen. Sie können sogar neue Attribute eines einzelnen Objekts erzeugen, indem Sie ihnen einen Wert zuweisen.

Allerdings gibt es die Konvention, dass Methoden oder Attribute, die mit einem einzelnen Unterstrich beginnen, als privat betrachtet werden sollten – das heißt nicht, dass sie nicht von außen zugänglich sind, aber als Benutzer der Klasse sollten Sie damit rechnen, dass sie in einer späteren Version der Klasse jederzeit entfernt oder geändert werden können.

Die einzige Methode der Beispielklasse, `__str__()`, hat folgenden Inhalt:

```
def __str__(self):
    return "{autor}: '{titel}' ({jahr})".format(
        autor = self.autor,
        titel = self.titel,
        jahr = self.jahr
    )
```

Es handelt sich wie beim Konstruktor um eine spezielle Methode: Sie wird automatisch aufgerufen, wenn das Objekt in einem String-Kontext verwendet wird, im Verwendungsbeispiel geschieht dies durch den Aufruf als Argument von `print()`. Explizit können Sie auch `Instanz.__str__()` schreiben oder die lesefreundlichere Variante `str(Instanz)` verwenden, um den Rückgabewert dieser Funktion zu erhalten.

Die Methode `__str__()` gibt hier lediglich eine speziell formatierte String-Darstellung der jeweiligen Instanz zurück – wie in den anderen bisher vorgestellten Programmiersprachen mithilfe des Schlüsselworts `return`.

Eine Instanz wird durch den Aufruf von `Klassenname(Argument, ...)` erzeugt, wie etwa in der ersten Instanziierung aus dem Beispielskript:

```
buch1 = Buch("Douglas Adams", "The Hitchhiker's Guide to the Galaxy", 1979)
```

Üblicherweise wird das Ergebnis, also die neue Instanz, in einer Variablen gespeichert. Wie Sie sehen, wird das `self`-Argument nicht explizit übergeben, sondern nur alle nachfolgenden Argumente.

Zum Schluss wird die Methode `__str__()` implizit aufgerufen, da `print()` stets die String-Darstellung seiner Argumente ausgibt:

```
print(buch1)
```

Die String-Darstellung greift auf die `__str__()`-Methode zurück, falls diese vorhanden ist. Andernfalls ist der String wesentlich weniger informativ. Sehen Sie sich dazu das folgende

kurze Beispiel aus der interaktiven Shell an, in der eine Klasse definiert und instanziiert wird; anschließend wird die neue Instanz mit `print()` ausgegeben:

```
>>> class Test:
...     pass
...
>>> test = Test()
>>> print(test)
<__main__.Test object at 0x10073ac18>
```

Die spezielle Anweisung `pass` tut übrigens nichts; sie dient wie in diesem Beispiel als spezieller Platzhalter für einen Anweisungsblock, in dem kein Code ausgeführt werden soll.

Methoden und Funktionen definieren

Im Einführungsbeispiel wurden bereits der Konstruktor und eine weitere Methode definiert. In diesem Unterabschnitt erhalten Sie einige weitere Informationen über das Implementieren von Methoden und alleinstehenden Funktionen.

Wie bereits erwähnt, wird eine Methode beziehungsweise Funktion mit dem Schlüsselwort `def` eingeleitet. Dahinter folgt der Name der Methode, dann runde Klammern mit optionalen Parametern und schließlich ein Doppelpunkt. Darunter wird der Funktionsrumpf eingerückt. Schematisch sieht das Ganze wie folgt aus:

```
def Bezeichner(Parameter, ...):
    Anweisung
    ...
```

Um eine Methode sinnvoll nutzen zu können, muss stets mindestens ein Parameter vorhanden sein, und Sie sollten diesen ersten Parameter stets `self` nennen, da ihm automatisch eine Referenz auf die aktuelle Instanz selbst übergeben wird. Für alleinstehende Funktionen gilt dies nicht; diese können auch ganz ohne Parameter sinnvoll sein.

Da Python eine interpretierte Sprache ist, die traditionell während der Ausführung übersetzt wurde, muss eine Funktion definiert werden, bevor sie aufgerufen werden kann. Ihre Definition muss also über dem ersten Aufruf im Skript stehen. Auch der Import einer anderen Datei, in der die Funktion definiert wird, muss entsprechend vorher stattfinden (Importe werden später behandelt).

Sie können Parametern optional einen Standardwert im Format `Parametername = Wert` zuweisen. Dabei müssen alle Parameter ohne Standardwerte links von denjenigen mit Standardwert stehen. Wenn Sie die Funktion aufrufen, können Sie die Argumente mit Standardwert weglassen. Hier ein einfaches Beispiel mit einer alleinstehenden Funktion in der interaktiven Shell:

```
>>> def funktion(a, b = 1):
...     print("a = {:d}, b = {:d}".format(a, b))
...
>>> funktion(3, 4)
a = 3, b = 4
>>> funktion(7)
a = 7, b = 1
```

Wenn Sie versuchen, ein Argument ohne Standardwert beim Aufruf wegzulassen, erhalten Sie eine Fehlermeldung:

```
>>> funktion()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: funktion() missing 1 required positional argument: 'a'
```

Wie Sie sehen, wird das fehlende Argument in der Fehlermeldung als *Positionsargument* (*positional argument*) bezeichnet. Das liegt daran, dass alle Argumente ohne Standardwert ausschließlich durch ihre Position in der Definition und im Aufruf gekennzeichnet werden. Argumente mit Standardwert sind dagegen automatisch *Schlüsselwortargumente* (*keyword arguments*), das heißt, sie können beim Aufruf in beliebiger Reihenfolge in der Form `Name = Wert` angegeben und aufgrund ihres Standardwerts auch weggelassen werden. Hier ein Beispiel mit einer Funktion, die nur Schlüsselwortargumente besitzt, und diversen gültigen Aufrufen:

```
>>> def funktion2(a = 1, b = 2):
...     print("a = {:d}, b = {:d}".format(a, b))
...
>>> funktion2(3, 4)
a = 3, b = 4
>>> funktion2(3, b = 7)
a = 3, b = 7
>>> funktion2(b = 12)
a = 1, b = 12
>>> funktion2(a = 2, b = 1)
a = 2, b = 1
>>> funktion2(b = 17, a = 18)
a = 18, b = 17
```

Wenn Sie möchten, können Sie Funktionen auch so schreiben, dass sie beliebig viele Positions- und beliebig benannte Schlüsselwortargumente annehmen. In diesem Fall sind die Positionsargumente eine Liste und die Schlüsselwortargumente ein Dictionary; traditionell werden die entsprechenden Parameter als `args` beziehungsweise `kwargs` bezeichnet. Damit sie als Liste und Dictionary behandelt werden, müssen Sie ihnen in der Funktionsdefinition

ein beziehungsweise zwei Sternchen voranstellen, also `*args` und `**kwargs`. Im Fall einer Methode steht der Parameter `self` vor beiden.

Das folgende Beispiel, eine alleinstehende Funktion, gibt die Positions- und Schlüsselwortparameter geordnet aus:

```
>>> def any_argument(*args, **kwargs):
...     print("Positional arguments:")
...     for arg in args:
...         print("- {}".format(arg))
...     print()
...     print("Keyword arguments:")
...     for key in kwargs:
...         print("- {}: {}".format(key, kwargs[key]))
...
>>> any_argument("Brian", "Reg", "Loretta", "Judith",
... team = "People's Front of Judea", place = "Judea")
Positional arguments:
- Brian
- Reg
- Loretta
- Judith

Keyword arguments:
- place: Judea
- team: People's Front of Judea
```

Natürlich können Sie die Funktion auch mit nur einer der beiden Argumentsorten aufrufen. Beispiele:

```
>>> any_argument("Arthur", "Galahad", "Lancelot", "Robin")
Positional arguments:
- Arthur
- Galahad
- Lancelot
- Robin

Keyword arguments:
>>> any_argument(team = "Round Table", place = "Camelot")
Positional arguments:

Keyword arguments:
- place: Camelot
- team: Round Table
```

Wenn eine Funktion oder Methode einen Wert zurückgeben soll, geschieht dies wie in den anderen hier behandelten Programmiersprachen mithilfe des Schlüsselworts `return`. Dabei verlässt diese Anweisung die Funktion unmittelbar; im Fall eines `return`, das von einem `if` abhängt, können Sie sich also beispielsweise das `else` sparen und einfach ein weiteres, ausgerücktes `return` hinzufügen. Allerdings kann ein explizites `else` die Angelegenheit klarer machen. Hier als Beispiel eine Methode, die mithilfe der Modulo-Operation eine übergebene Zahl prüft und `True` zurückgibt, wenn diese gerade ist, und ansonsten `False`:

```
>>> def gerade(zahl):
...     if zahl % 2 == 0:
...         return True
...     return False
...
>>> gerade(8)
True
>>> gerade(9)
False
```

Lambda-Funktionen

Eine spezielle Form der Funktion ist die *Lambda-Funktion*, deren Konzept aus der funktionalen Programmierung stammt. Es handelt sich um anonyme Funktionen, die beispielsweise einer Variablen zugewiesen, als Rückgabewert einer regulären Funktion verwendet oder auch als Argumente an andere Funktionen übergeben werden können.

Die allgemeine Syntax einer Lambda-Funktion sieht so aus:

```
lambda argument1, argument2, ...: ausdruck
```

Dies definiert eine Funktion, die die angegebenen Argumente entgegennimmt, den Ausdruck auswertet und dessen Wert zurückgibt. Das folgende einfache Beispiel speichert eine Lambda-Funktion in der Variablen `quadrat`. Die Funktion nimmt ein Argument entgegen und gibt dessen Quadrat zurück:

```
>>> quadrat = lambda x: x ** 2
```

Da die Lambda-Funktion in einer Variablen gespeichert ist, können Sie sie in diesem Fall mithilfe dieser Variablen aufrufen:

```
>>> quadrat(7)
49
```

Hier ein weiteres Beispiel – eine reguläre Funktion, die je nach Argument eine unterschiedliche Lambda-Funktion zurückgibt, nämlich eine Berechnungsfunktion, die auf einem übergebenen Operator basiert:


```
>>> def operation(operator):
...     if operator == '+':
...         return lambda a, b: a + b
...     if operator == '-':
...         return lambda a, b: a - b
...     if operator == '*':
...         return lambda a, b: a * b
...     if operator == '/':
...         return lambda a, b: a / b
...     raise ValueError("'{}' is not a valid operator.".format(operator))
...
```


Wie Sie sehen, wird je nach Rechenzeichen eine Lambda-Funktion erzeugt und zurückgegeben, die die entsprechende Operation ausführt. Sie können das Ergebnis der Funktion in einer Variablen speichern, um die entsprechende Operation dauerhaft zur Verfügung zu haben:

```
>>> plus = operation("+")
>>> plus(3, 5)
8
```

Am Ende der Funktion wird für ungültige Rechenzeichen eine Ausnahme ausgelöst (siehe den Unterabschnitt »Ausnahmebehandlung« für Details). Das Ganze sieht beispielsweise so aus, wenn Sie `operation()` mit einem falschen Zeichen aufrufen:

```
>>> unknown = operation("?")
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<stdin>", line 10, in operation
ValueError: '?' is not a valid operator.
```

Lambda-Funktionen können auch verwendet werden, um Listen zu sortieren, zu filtern oder neue Listen aus Berechnungen zu erzeugen. Während sich Filterung und Umrechnung leichter mithilfe von List Comprehensions erledigen lassen, wird für komplexe Sortierungen in jedem Fall eine Funktion benötigt, entweder eine benannte externe oder eben eine Lambda-Funktion.

Betrachten Sie etwa die folgende Liste von Tupeln, die jeweils den Namen und das Alter von Personen angeben:

```
>>> people = [('Klaus', 28),
...     ('Anna', 39),
...     ('Zacharias', 23),
...     ('Theresa', 25)
... ]
```

Angenommen, Sie möchten diese Liste nach dem Alter der Personen sortieren, also dem Element jedes Tupels mit dem Index 1. Dazu wird jedes Element der Liste an eine Lambda-Funktion übergeben, die dann das entsprechende Element als Sortierschlüssel extrahiert. Dies sieht so aus:

```
>>> sorted(people, key = lambda person: person[1])
[('Zacharias', 23), ('Theresa', 25), ('Klaus', 28), ('Anna', 39)]
```

Die Funktion `sorted()` gibt allgemein eine sortierte Kopie eines Aufzählungsobjekts zurück. Zusätzlich gibt es auch eine Methode namens `sort()`, die die enthaltenen Elemente in der Liste selbst sortiert. Beispiel:

```
>>> list = [5, 1, 4, 2, 3]
>>> list
[5, 1, 4, 2, 3]
>>> list.sort()
>>> list
[1, 2, 3, 4, 5]
```

Magische Methoden

Methoden wie der Konstruktor `__init__()` oder der String-Umwandler `__str__()` werden als *magische Methoden* (englisch *magic methods*) bezeichnet, da sie in bestimmten Zusammenhängen automatisch aufgerufen werden. Neben Typumwandlungen können Sie mithilfe magischer Methoden beispielsweise auch Operatoren für Ihre eigenen Klassen implementieren, etwa arithmetische Operationen, Vergleichsoperationen oder den Indexoperator.

Das Thema würde hier zu weit führen, aber hier als kleines Beispiel eine Implementierung des Additionsoperators `+` für eine Klasse, die einen einfachen numerischen Wert namens `value` kapselt:

```
>>> class TestNumber:
...     def __init__(self, value):
...         self.value = value
...     def __add__(self, other):
...         return TestNumber(self.value + other.value)
...
>>> t1 = TestNumber(23)
>>> t2 = TestNumber(42)
>>> sum = t1 + t2
>>> sum
<__main__.TestNumber object at 0x10063ae10>
>>> sum.value
65
```


Näheres zu diesem interessanten Thema erfahren Sie in der Python-Online-Dokumentation im Kapitel »Data Model« unter *https://docs.python.org/3/reference/datamodel.html*.

Vererbung

Wie im Abschnitt zu Java bereits ausgeführt wurde, ist die Vererbung ein wichtiges Hilfsmittel der Objektorientierung. Sie ermöglicht es, aus allgemeineren Klassen speziellere abzuleiten, ohne die gemeinsamen Grundlagen mehrfach implementieren zu müssen.

In Python wird die Vererbung durchgeführt, indem der Name der gewünschten Elternklasse in der `class`-Anweisung in Klammern hinter dem Klassennamen angegeben wird; schematisch sieht dies also wie folgt aus:

```
class AbgeleiteteKlasse(Elternklasse):
    # Klassendefinition
```

Das folgende einfache Beispiel aus der interaktiven Shell definiert zunächst eine Klasse namens `Parent` und leitet dann eine Klasse namens `Child` davon ab. Eine der beiden Methoden wird in `Child` überschrieben; zur Kontrolle geben alle Methoden aus, zu welcher Klasse sie gehören:

```
>>> class Parent:
...     def override_me(self):
...         print("override_me() in Parent")
...     def dont_override(self):
...         print("dont_override() in Parent")
...
>>> class Child(Parent):
...     def override_me(self):
...         print("override_me() in Child")
...
```

Wenn Sie nun Instanzen beider Klassen erzeugen und die Methoden aufrufen, sehen Sie, dass die Methode `override_me()` der abgeleiteten Klasse wie erwartet die gleichnamige Methode der Elternklasse überlagert:

```
>>> child = Child()
>>> child.override_me()
override_me() in Child
>>> child.dont_override()
dont_override() in Parent
>>> parent = Parent()
>>> parent.override_me()
```

```
override_me() in Parent
>>> parent.dont_override()
dont_override() in Parent
```

Unter Umständen kann es sinnvoll sein, den Konstruktor oder die gleichnamige Methode der Elternklasse aufzurufen. Dafür stellt Python 3 die Funktion `super()` bereit, die die aktuelle Instanz stets als Instanz der übergeordneten Klasse betrachtet und so deren Elemente zugänglich macht. Die folgende von `Child` abgeleitete Klasse `Grandchild` zeigt, wie es funktioniert – ihre Methode `override_me()` ruft zunächst die gleichnamige Methode von `Child` auf, bevor sie ihre eigene Funktionalität hinzufügt:

```
>>> class Grandchild(Child):
...     def override_me(self):
...         super().override_me()
...         print("override_me() in Grandchild")
...
```

Instanziierung und Methodenaufruf erzeugen daraufhin die folgende Ausgabe:

```
>>> grandchild = Grandchild()
>>> grandchild.override_me()
override_me() in Child
override_me() in Grandchild
```

Der Konstruktor der Elternklasse wird entsprechend als `super().__init__(...)` aufgerufen; ein Beispiel dazu gibt es später.

Eine Besonderheit von Python gegenüber Java und vielen anderen objektorientierten Sprachen ist die Möglichkeit der *Mehrfachvererbung* (*multiple inheritance*); das bedeutet, dass eine Klasse von mehreren anderen Klassen abgeleitet werden kann. Das Syntaxschema ist wie folgt:

```
class Klassenname(Elternklasse1, Elternklasse2, ...):
    # Code der abgeleiteten Klasse
```

Im folgenden kleinen Beispiel wird eine Klasse namens `Kid` von den beiden Klassen `Mother` und `Father` abgeleitet; die Kindklasse selbst hat keinen eigenen Inhalt (daher die leere Operation `pass`). Jede der beiden Elternklassen vererbt ihr eine Methode:

```
>>> class Mother:
...     def mother_method(self):
...         print("Mother's method")
...
>>> class Father:
...     def father_method(self):
```



```
...         print("Father's method")
...
>>> class Kid(Mother, Father):
...     pass
...
>>> kid = Kid()
>>> kid.mother_method()
Mother's method
>>> kid.father_method()
Father's method
```

Interessant wird es natürlich, wenn mehrere Elternklassen gleichnamige Methoden besitzen. In diesem Fall wird die Methode aus der ersten Klasse in die Liste der übergeordneten Klassen übernommen, wie das folgende Beispiel zeigt:

```
>>> class Mom:
...     def parent_method(self):
...         print("Mom's method")
...
>>> class Dad:
...     def parent_method(self):
...         print("Dad's method")
...
>>> class LittleOne(Mom, Dad):
...     pass
...
>>> little_one = LittleOne()
>>> little_one.parent_method()
Mom's method
```

Ausnahmebehandlung

Ähnlich wie Java kann auch Python Ausnahmen (Exceptions) auslösen beziehungsweise abfangen. Das Abfangen funktioniert dabei schematisch mithilfe einer `try/except`-Konstruktion:

```
try:
    # Anweisungen, die eine Ausnahme auslösen können
except Ausnahmetyp:
    # Code für den Ausnahmefall
```

Das folgende Beispiel fängt einen `ZeroDivisionError` ab, also den Fehler, der auftritt, wenn man versucht, durch 0 zu dividieren:

```
>>> dividend = 10
>>> divisor = 0
>>> try:
...     result = dividend / divisor
... except ZeroDivisionError:
...     print("Divisor darf nicht 0 sein!")
...
Divisor darf nicht 0 sein!
```

Falls ein anderer Fehler auftritt als der mit `except` abgefangene, führt er wie gehabt zu einer Standardfehlermeldung. Beispiel:

```
>>> dividend = "KeineZahl"
>>> divisor = "AuchKeineZahl"
>>> try:
...     result = dividend / divisor
... except ZeroDivisionError:
...     print("Divisor darf nicht 0 sein!")
...
Traceback (most recent call last):
  File "<stdin>", line 2, in <module>
TypeError: unsupported operand type(s) for /: 'str' and 'str'
```

Sie können aber beliebig viele `except`-Blöcke verwenden, um verschiedene Fehlertypen abzufangen. Das obige Beispiel lässt sich so wie folgt erweitern:

```
>>> dividend = "KeineZahl"
>>> divisor = "AuchKeineZahl"
>>> try:
...     result = dividend / divisor
... except ZeroDivisionError:
...     print("Divisor darf nicht 0 sein!")
... except TypeError:
...     print("Operanden müssen Zahlen sein!")
...
Operanden müssen Zahlen sein!
```

Um in eigenem Code eine Ausnahme auszulösen, wird das Schlüsselwort `raise` verwendet; dabei können Sie entweder vorhandene Ausnahmeklassen verwenden oder Ihre eigenen von diesen ableiten, um verschiedene Fehlerzustände besser unterscheiden zu können. Optional können Sie dabei einen eigenen Fehlermeldungstext übergeben.

Das folgende Beispiel leitet die Ausnahme `NoIntError` von `TypeError` ab (ohne spezifische Anweisungen) und definiert anschließend die Funktion `double_value()`, die das Doppelte eines Integers zurückgibt oder einen `NoIntError` auslöst, wenn das Argument kein Integer ist:


```
>>> class NoIntError(TypeError):
...     pass
...
>>> def double_value(int_number):
...     if type(int_number) is not int:
...         raise NoIntError("Argument must be an integer.")
...     return int_number * 2
...
>>> double_value(8)
16
>>> double_value(9.2)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<stdin>", line 3, in double_value
__main__.NoIntError: Argument must be an integer.
```


Ein ausführliches Beispiel

Nachdem die diversen Grundlagen der Objektorientierung beschrieben wurden, folgt hier ein etwas umfangreicheres Beispiel. Es bildet Bücher und E-Books als Python-Klassen ab, mitsamt Helferklassen für Autoren und Genres. Hier zunächst einmal das komplette Listing, das Sie unter *library.py* speichern können; Ausgabe und Erläuterungen neuer Konzepte folgen danach:

```
class Author:
    """Represents a book's author

    Attributes:
      firstname (str): the author's first name
      lastname (str): the author's last name
    """
    def __init__(self, firstname, lastname):
        """Constructor for Author instances

        Args:
          firstname (str): the new author's first name
          lastname (str): the new author's last name
        """
        self.firstname = firstname
        self.lastname = lastname
    def __str__(self):
        """Represents an Author instance in string context
        """
        return "{} {}".format(self.firstname, self.lastname)
```

```
class Genre:
    """Represents a book's genre

    Genres can be nested using their parent attribute

    Attributes:
      title (str): the genre's name
      parent (Genre): the parent genre, or None for top-level genres
    """
    def __init__(self, title, parent = None):
        """Constructor for Genre instances

        Args:
          title (str): the new genre's name
          parent (Genre, optional): the parent genre, if any
        """
        self.title = title
        self.parent = parent
    def __str__(self):
        """Represents a Genre instance in string context
        """
        result = self.title
        if self.parent is not None:
            result = "{} > {}".format(self.parent.__str__(), result)
        return result

class Book:
    """Represents a book

    All attributes except for the title are optional and default to None

    Attributes:
      title (str): the book's title
      authors (Author or list of Author objects): the book's author(s)
      genre (Genre): the book's genre
      year (int): the book's publication year
    """
    def __init__(self, title, authors = None, genre = None, year = None):
        """Constructor for Book instances

        Args:
          title (str): the new book's title
```



```
          authors (Author or list of Author objects, optional):
            the new book's author(s)
          genre (Genre, optional): the new book's genre
          year (int, optional): the new book's publication year
        """
        self.title = title
        self.authors = authors
        self.genre = genre
        self.year = year
    def has_author(self, author):
        """Checks whether a specific author is among the book's authors

        Args:
          author (Author): the author to search for
        """
        result = False
        if hasattr(self.authors, "__getitem__") and author in self.authors:
            result = True
        elif self.authors is author:
            result = True
        return result
    def in_genre(self, genre):
        """Checks whether the book belongs to a specific genre/subgenre

        Args:
          genre (Genre): the genre to search for
        """
        result = False
        current_genre = self.genre
        while result == False and current_genre is not None:
            if current_genre is genre:
                result = True
            current_genre = current_genre.parent
        return result
    def __str__(self):
        """Represents a Book instance in string context
        """
        result = ""
        if self.authors is not None:
            if hasattr(self.authors, "__getitem__"):
                author_strings = [author.__str__() for author in self.authors]
                result += ", ".join(author_strings)
            else:
```

```
                result += self.authors.__str__()
            result += ": "
        result += "'{}'".format(self.title)
        if self.year is not None:
            result += ", {}".format(self.year)
        if self.genre is not None:
            result += " ({})".format(self.genre)
        return result

class Ebook(Book):
    """Represents an ebook as a specialization of a book

    See parent class for details; only additional features are described here

    Attributes:
      tech (str): the ebook's technology or file format,
        e.g. "PDF", "mobi", or "Kindle"
    """
    def __init__(self, title, authors = None, genre = None, year = None,
        tech = None):
        """Constructor for Ebook instances

        See parent constructor for details; only additional
        features are described here

        Args:
          tech (str, optional): the new ebook's technology or file format
        """
        super().__init__(title, authors, genre, year)
        self.tech = tech
    def __str__(self):
        """Represents an Ebook instance in string context
        """
        result = super().__str__()
        if self.tech is not None:
            result += " [{}]".format(self.tech)
        return result

class Library:
    """Represents a collection of books

    Provides a method to filter the list, e.g. by genre
```



```
    Attributes:
      books (list): the list of books/ebooks in the collection
    """
    def __init__(self, books = []):
        """Constructor for Library instances

        Args:
          books (list, optional): the initial list of books
        """
        self.books = books
    def add(self, book):
        """Adds a book to the collection

        Args:
          book (Book): the book to add
        """
        self.books.append(book)
    def remove(self, book):
        """Removes a book from the collection

        Args:
          book (Book): the books to remove
        """
        try:
            self.books.remove(book)
        except ValueError:
            pass
    def filter(self, field, value):
        """Returns a list of books that match a filter

        Args:
          field (str): field to filter for, one of "author", "genre", or "year"
          value (mixed): value to match the field against
        """
        if field == "author":
            return [book for book in self.books if book.has_author(value)]
        if field == "genre":
            return [book for book in self.books if book.in_genre(value)]
        if field == "year":
            return [book for book in self.books if book.year == value]
        raise ValueError("{} is not a valid field.".format(field))

if __name__ == "__main__":
```

```
author_martin = Author("George R. R.", "Martin")
author_tuttle = Author("Lisa", "Tuttle")
author_weir = Author("Andy", "Weir")
genre_fiction = Genre("Fiction")
genre_scifi = Genre("Science Fiction", genre_fiction)
genre_hardscifi = Genre("Hard Science Fiction", genre_scifi)
genre_fantasy = Genre("Fantasy", genre_fiction)
book_windhaven = Book("Windhaven", authors = [author_martin, author_tuttle],
  genre = genre_scifi, year = 1981)
book_clash_of_kings = Book("A Clash of Kings", authors = author_martin,
  genre = genre_fantasy, year = 1998)
ebook_silverbough = Ebook("The Silver Bough", authors = author_tuttle,
  genre = genre_fantasy, year = 2012, tech = "Kindle")
book_martian = Book("The Martian", authors = author_weir,
  genre = genre_hardscifi, year = 2014)
library = Library([book_windhaven, book_clash_of_kings, ebook_silverbough])
library.add(book_martian)
print("All books:")
for book in library.books:
    print("- {}".format(book))
print()
print("Books in the Science Fiction genre:")
for book in library.filter("genre", genre_scifi):
    print("- {}".format(book))
print()
print("Books co-written by Lisa Tuttle:")
for book in library.filter("author", author_tuttle):
    print("- {}".format(book))
```

Führen Sie das Skript folgendermaßen aus:

```
$ python3 library.py
```

Sie erhalten folgende Ausgabe:

```
All books:
- George R. R. Martin, Lisa Tuttle: 'Windhaven', 1981 (Fiction > Science Fiction)
- George R. R. Martin: 'A Clash of Kings', 1998 (Fiction > Fantasy)
- Lisa Tuttle: 'The Silver Bough', 2012 (Fiction > Fantasy) [Kindle]
- Andy Weir: 'The Martian', 2014 (Fiction > Science Fiction > Hard Science Fiction)

Books in the Science Fiction genre:
- George R. R. Martin, Lisa Tuttle: 'Windhaven', 1981 (Fiction > Science Fiction)
- Andy Weir: 'The Martian', 2014 (Fiction > Science Fiction > Hard Science Fiction)
```



```
Books co-written by Lisa Tuttle:
- George R. R. Martin, Lisa Tuttle: 'Windhaven', 1981 (Fiction > Science Fiction)
- Lisa Tuttle: 'The Silver Bough', 2012 (Fiction > Fantasy) [Kindle]
```

Wie Sie sehen, enthält das Listing zahlreiche Python-Doc-Kommentare, die mit drei Anführungszeichen beginnen und wieder enden. Über das Konsolen-Tool `pydoc` können Sie sich wie folgt die daraus generierte Dokumentation anzeigen lassen, wenn Sie sich im Verzeichnis der Datei *library.py* befinden:

```
$ pydoc library
```

Beachten Sie, dass die Dateiendung *.py* in diesem Fall nicht mit angegeben werden darf. Die Dokumentation wird seitenweise mithilfe von `less` (Unix) oder `more` (Windows) angezeigt; Sie können darin mit ↑ und ↓ zeilenweise blättern, mit der Leertaste eine ganze Seite vorwärts blättern und die Hilfe mit Q beenden.

Der hier verwendete konkrete Stil für die Dokumentation wurde von Google eingeführt und ist inzwischen weitverbreitet. Dabei wird der Dokumentationsblock unter der jeweiligen Klasse oder Methode eingerückt und hat folgende Eigenschaften:

```
"""Kurzbeschreibung (eine Zeile)

Längere Beschreibung (optional, darf mehrzeilig sein)

Attributes [Klasse] beziehungsweise Args [Methode]:
  Bezeichner (Typ ...): Beschreibung; kann in die nächste Zeile übergehen
    und wird dort weiter eingerückt
"""
```

In dem Skript werden insgesamt fünf Klassen und ein Hauptprogramm definiert. Die Klassen sind:

- `Author` (Darstellung eines Autors mit Vor- und Nachnamen)
- `Genre` (Darstellung eines Genres mit einem Titel und einem optionalen übergeordneten Genre für Hierarchien)
- `Book` (Darstellung eines Buches mit einem Titel, einem oder mehreren Autoren, Genre und Erscheinungsjahr)
- `Ebook` (abgeleitet von `Book`; enthält zusätzlich eine Angabe zur Technologie beziehungsweise dem Dateiformat)
- `Library` (Darstellung einer Büchersammlung)

Es ist übrigens kein Problem, beliebig viele Klassen innerhalb derselben Datei zu definieren; eine Regel bezüglich der Übereinstimmung von Datei- und Klassennamen wie in Java gibt es in Python nicht.

Das Hauptprogramm wird in Abhängigkeit von folgender `if`-Fallentscheidung ausgeführt:

```
if __name__ == "__main__":
    # Anweisungen des Hauptprogramms
```

Die Systemkonstante `__name__` enthält den Namen des aktuellen Moduls, also des Ausführungskontextes. Der Name des Hauptprogramms ist dabei `"__main__"`, sodass dieser Code nur ausgeführt wird, wenn das Skript direkt aufgerufen wird. Importieren Sie es dagegen, um seine Klassenbibliothek zu nutzen, wird der Codeblock ignoriert. So können Sie auf praktische Weise Testcode für Ihre Klassen einfügen, der bei der Nutzung der Klassen nicht stört.

Jede der hier definierten Klassen hat einen Konstruktor, und alle bis auf `Library` haben eine `__str__()`-Methode. Beides wurde bereits im Einführungsbeispiel zur Objektorientierung erläutert. Die erste Besonderheit finden Sie in der `__str__()`-Methode der Klasse `Genre`, die hier nochmals ohne Doc-Kommentar wiedergegeben wird:

```
def __str__(self):
    result = self.title
    if self.parent is not None:
        result = "{} > {}".format(self.parent.__str__(), result)
    return result
```

Wie Sie sehen, wird zunächst der Titel des aktuellen Genres als Ergebnis festgelegt. Anschließend wird geprüft, ob die Eigenschaft `parent` gesetzt ist, und falls das so ist, wird das Ergebnis von deren `__str__()`-Methode vor das bisherige Ergebnis gesetzt. Da es sich bei `parent` in diesem Fall wieder um eine Instanz von `Genre` handelt, kann gegebenenfalls auch dessen `parent` hinzugefügt werden etc.. Auf diese Weise wird eine komplette Kette hierarchisch geordneter Genre-Bezeichnungen zusammengefügt.

Die Klasse `Book` besitzt außer dem Konstruktor und `__str__()` noch zwei weitere Methoden, die jeweils eine Prüfung von Attributen vornehmen: `has_author(author)` geht die Liste der Autoren durch und gibt `True` zurück, falls der gesuchte Autor darin gefunden wird, und andernfalls `False`. `in_genre(genre)` prüft dagegen, ob das Buch dem gesuchten Genre oder einem von dessen untergeordneten Genres angehört, und gibt ebenfalls ein Boolean-Ergebnis zurück.

Da die Eigenschaft `authors` der Klasse `Book` sowohl ein einzelnes `Author`-Objekt als auch eine Liste solcher Objekte sein kann, muss dies zunächst überprüft werden. Der entsprechende Code – der auch in `__str__()` wieder ähnlich zum Einsatz kommt – sieht so aus:


```
if hasattr(self.authors, "__getitem__") and author in self.authors:
    result = True
elif self.authors is author:
    result = True
```

Die Funktion `hasattr(Objekt, String)` überprüft, ob das Objekt ein Attribut mit dem angegebenen Namen besitzt, wobei auch eine Methode eine spezielle Art von Attribut ist. Da `__getitem__()` der interne Name des Indexoperators ist, kann man davon ausgehen, dass es sich bei einem Objekt mit dieser Methode um einen Listentyp oder zumindest um ein Objekt handelt, dessen Inhalt sich mit `in` durchsuchen lässt.

Die Methode `in_genre()` beginnt beim Genre des Buches selbst und durchsucht so lange die Kette von Eltern-Genres, bis das gesuchte gefunden wird oder kein weiteres Genre mehr vorhanden ist (`parent` mit dem Wert `None`, also ein Top-Level-Genre):

```
result = False
    current_genre = self.genre
    while result == False and current_genre is not None:
        if current_genre is genre:
            result = True
        current_genre = current_genre.parent
```

Die Klasse `Ebook` ist von `Book` abgeleitet; die einzige Besonderheit ist hier der Aufruf des Konstruktors der übergeordneten Klasse, um die Eigenschaften zu initialisieren, die `Book` und `Ebook` gemeinsam haben:

```
super().__init__(title, authors, genre, year)
```

Die letzte Klasse in der Datei, `Library`, stellt eine Büchersammlung dar. Neben dem Konstruktor besitzt sie die drei Methoden `add()` zum Hinzufügen eines Buches, `remove()` zum Entfernen und `filter()` zum Filtern der Sammlung nach diversen Kriterien. `add()` und `remove()` sind recht kurz und simpel, außer dass `remove()` stillschweigend eine Exception ignoriert, die beim Entfernen eines nicht vorhandenen Buches auftreten würde.

Die Methode `filter()` erwartet einen der Feldnamen `"author"`, `"genre"` oder `"year"` sowie einen Wert, nach dem gesucht werden soll. Jeder Filtervorgang verwendet eine List Comprehension zum Filtern der internen Bücherliste. Die Filterung nach Autor oder Genre ruft für jedes Buch dessen Methode `has_author()` beziehungsweise `is_genre()` auf, während für das Jahr ein einfacher Vergleich genügt. Für ungültige Feldnamen wird eine Ausnahme ausgelöst.

Das Hauptprogramm definiert einige Genres, Autoren, Bücher und eine Bibliothek, in der Letztere gesammelt werden. Danach werden sowohl die gesamte Bücherliste als auch zwei gefilterte Listen ausgegeben.

Wenn Sie die Klassen des Beispiels für eigene Experimente benutzen möchten, können Sie sie in ein anderes Python-Skript oder in die interaktive Shell importieren. Dazu kommt das Schlüsselwort `import` zum Einsatz, das auf verschiedene Arten verwendet werden kann. Schematisch gesehen, gibt es die drei folgenden Möglichkeiten:

- `import Modulname` – alle Klassen des Moduls unter einem eigenen Namensraum importieren
- `from Modulname import *` – alle Klassen des Moduls in den Namensraum des aktuellen Moduls importieren
- `from Modulname import Klassenname` – eine bestimmte Klasse aus dem Modul in den Namensraum des aktuellen Moduls importieren

Hier sind einige Erläuterungen nötig. Ein *Modul* ist ein Ausführungskontext; das Standardmodul ist jeweils das Hauptprogramm, dessen Name – wie bereits erwähnt – `"__main__"` lautet. Wenn Sie eine Datei importieren, stellt diese ein eigenes Modul dar; der Modulname ist der Dateiname ohne die Endung *.py*.

Ein einfaches `import` ohne `from` stellt das importierte Modul als eigenen Namensraum zur Verfügung; in diesem Fall müssen Sie den entsprechenden Klassen den Modulnamen und einen Punkt voranstellen. Angenommen, Sie befinden sich in dem Verzeichnis, in dem die Datei *library.py* liegt, und importieren diese wie folgt in die interaktive Shell:

```
>>> import library
```

Dann müssen Sie beispielsweise Folgendes eingeben, um ein neues Genre zu definieren:

```
>>> superheroes = library.Genre("Superheroes")
```

Importieren Sie *library.py* dagegen in den Namensraum des aktuellen Moduls, dann funktioniert das Definieren eines Genres wie folgt:

```
>>> from library import *
>>> horror = Genre("Horror")
```

Der Vorteil eines eigenen Namensraums ist, dass die Klassennamen aus dem importierten Modul anderen Bezeichnern nicht in die Quere kommen. Der Nachteil ist natürlich, dass Sie mehr schreiben müssen und die Skripte unter Umständen schlechter lesbar sind.

Falls sich die zu importierende Datei nicht im aktuellen Verzeichnis befindet, müssen Sie den Pfad angeben. Dabei wird der / zwischen Verzeichnisnamen durch einen Punkt (`.`) ersetzt. Angenommen, *library.py* befindet sich im Unterverzeichnis *lib* des aktuellen Verzeichnisses. Dann würde der Import der Klasse `Author` in den aktuellen Namensraum so aussehen:

```
>>> from lib.library import Author
```


9.3.4 Die Python-Standardbibliothek

Im letzten Unterabschnitt des Python-Tutorials werden einige interessante Bestandteile der Standardbibliothek vorgestellt. Module der offiziellen Python-Klassenbibliothek werden genau auf dieselbe Weise importiert wie Ihre eigenen Module. Python durchsucht das Bibliotheksverzeichnis dabei automatisch, egal, in welchem Verzeichnis Sie sich befinden.

In späteren Kapiteln lernen Sie weitere Module der Python-Klassenbibliothek kennen, die zu den verschiedenen dort behandelten Themen passen.

Kommandzeilenargumente

Das Modul `sys` stellt eine Schnittstelle zu Funktionen des Betriebssystems bereit, genauer gesagt: Funktionen der Shell. Die folgende Anweisung importiert das Element `argv` aus diesem Modul in den aktuellen Namensraum; es dient zur Verarbeitung von Kommandozeilenargumenten:

```
>>> from sys import argv
```

Hier ein kleines Beispielskript, das die Verwendung von Kommandozeilenargumenten demonstriert:

```
from sys import argv

print("{} arguments:".format(len(argv)))
print()

for index, arg in enumerate(argv):
    print("{}. {}".format(index, arg))
```

Die Funktion `enumerate(Liste)` ermöglicht es, wie gezeigt, auf die Indizes von Listen zuzugreifen. Speichern Sie das Skript als *args.py*, und führen Sie es mit einigen Argumenten auf der Konsole aus:

```
$ python3 args.py Test 1 2 3
5 arguments:

0. args.py
1. Test
2. 1
3. 2
4. 3
```

Wie Sie sehen, ist `argv[0]` stets der Name des Skripts selbst; die eigentlichen Argumente beginnen bei Index 1.

Verzeichnisse auslesen

Die Klassen zum Lesen von Verzeichnisinhalten befinden sich im Modul `os` und seinen Untermodulen; dieses Modul ermöglicht den Zugriff auf die hardwarenäheren Teile des Systems. Wenn Sie einfach nur den Inhalt des aktuellen Verzeichnisses auslesen möchten, funktioniert dies wie folgt:

```
>>> from os import listdir
>>> for file in listdir("."):
...     print(file)
...
__pycache__
args.py
ausgabe.txt
author.py
author.pyc
[etc.]
```

Der Byte-Code-Cache

Dateien mit der Endung *.pyc* sind übrigens kompilierter Python-Byte-Code, der bei der ersten Ausführung nach jeder Änderung erzeugt wird und künftig die Ausführung beschleunigt. Auch das Unterverzeichnis *__pycache__* dient dem Caching von Byte-Code.

Es ist oft von Interesse, ob es sich bei Verzeichniseinträgen um normale Dateien oder um Unterverzeichnisse handelt. Entsprechende Prüfungsmethoden sind im Untermodul `os.path` enthalten; sie heißen `isfile()` für Dateien und `isdir()` für Verzeichnisse. Hier ein Beispiel, das Unterverzeichnissen ein `d`, Dateien ein `f` und anderen Einträgen (zum Beispiel mit `mkfifo` erzeugte Named Pipes) ein Fragezeichen voranstellt. Um den Code wiederverwenden zu können, ist er als Funktion definiert, der ein Verzeichnispfad übergeben wird:

```
>>> from os import listdir
>>> from os.path import isfile, isdir
>>> def ls(path):
...     for entry in listdir(path):
...         if isdir(entry):
...             print("d {}".format(entry))
...         elif isfile(entry):
...             print("f {}".format(entry))
...         else:
...             print("? {}".format(entry))
...
>>> ls(".")
d __pycache__
? __test
f args.py
[etc.]
```


Die Funktion `walk(Pfad)` ermöglicht das rekursive Auslesen eines Verzeichnisses und der gesamten Unterverzeichnisstruktur. Der Iterator liefert in jedem Durchgang ein Tupel mit dem relativen Pfadnamen des jeweiligen Verzeichnisses, einer Liste der Unterverzeichnisse und einer Liste der Dateien. Das folgende Beispiel gibt diese Daten aus:

```
>>> from os import walk
>>> for (dir, subdirs, files) in walk("testdir"):
...     print("Dir: {}, subdirectories: {}, files: {}".format(dir, subdirs, files))
...
Dir: testdir, subdirectories: ['othersub', 'sub'], files: ['file1.txt', 'file2.txt']
Dir: testdir/othersub, subdirectories: ['subsub'], files: [
'file1.txt', 'file2.txt', 'file3.txt']
Dir: testdir/othersub/subsub, subdirectories: [], files: ['file1.txt']
Dir: testdir/sub, subdirectories: [], files: ['file1.txt']
```

Reguläre Ausdrücke

In Kapitel 7, »Linux«, wurde das Shell-Kommando `grep` vorgestellt, das Dateien und andere Textquellen nach komplexen Suchmustern durchsucht, die als *reguläre Ausdrücke* (*regular expressions*) bezeichnet werden. Python besitzt ein Modul, das die Suche in Strings und andere nützliche Operationen mit regulären Ausdrücken unterstützt; Sie können es wie folgt importieren:

```
>>> import re
```

Danach stehen Ihnen diverse Operationen für reguläre Ausdrücke zur Verfügung. Die einfachste ist `re.search(regex, string)` – sie sucht im gesamten String nach einem Treffer für den regulären Ausdruck `regex`. Wird einer gefunden, dann erhalten Sie ein Match-Objekt zurück, andernfalls `None`. Im einfachsten Fall können Sie also ein `if re.search()`-Konstrukt verwenden, da das Match-Objekt als `True` gilt und `None` als `False`. Aber Sie können das Match-Objekt auch genauer analysieren. Das folgende Beispiel sucht in einem String nach einem Vokal und speichert das Match-Objekt in einer Variablen:

```
>>> first_vowel = re.search("[aeiou]", "Hello")
>>> first_vowel
<_sre.SRE_Match object; span=(1, 2), match='e'>
```

Das Konstrukt `[aeiou]` in einem regulären Ausdruck bedeutet, dass eines der in eckigen Klammern angegebenen Zeichen erwartet wird. Das Match-Objekt hat einige nützliche

Methoden zur Untersuchung des Treffers. Die Methode `group()` gibt den Teil-String zurück, auf den der reguläre Ausdruck passt:

```
>>> first_vowel.group()
'e'
```

Mit `span()` erhalten Sie ein Tupel, das die Anfangsposition und die erste nicht mehr dazugehörige Position des Treffers enthält:

```
>>> first_vowel.span()
(1, 2)
```

Die Funktion `re.finditer(regex, String)` stellt einen Iterator bereit, der alle Treffer als Match-Objekte zurückliefert. Hier ein Beispiel, das nacheinander alle Vokale in dem String `"Hello World"` findet:

```
>>> for match in re.finditer("[aeiou]", "Hello World"):
...     print("{} found at position {}.".format(match.group(), match.span()[0]))
...
e found at position 1.
o found at position 4.
o found at position 7.
```

Weitere Beispiele für die Verwendung regulärer Ausdrücke finden Sie in Kapitel 10, »Konzepte der Programmierung«.

Datum und Uhrzeit

Die wichtigsten Methoden für Datum und Uhrzeit finden Sie in den Modulen `datetime` und `time`. Sie können zum Beispiel alle Klassen des Moduls `datetime` importieren, um sie ohne Namensraum-Präfix zu benutzen:

```
>>> from datetime import *
```

Eine einfache Klasse aus dem Modul `datetime` heißt wiederum `datetime`; ein Objekt dieser Klasse speichert eine Datums- und Uhrzeitangabe. Wenn Sie einen konkreten Zeitpunkt speichern möchten, lautet die Konstruktor-Syntax wie folgt:

```
datetime(year, month, day, hour, minute, second, microsecond)
```

Jahr, Monat und Tag sind dabei Pflichtfelder, alle anderen sind optional und haben den Standardwert 0. Das folgende Beispiel speichert ein konkretes Datum und eine konkrete Uhrzeit ohne Mikrosekunden in einer Variablen:

```
>>> dt = datetime(2015, 4, 18, 13, 37, 25)
```


Die String-Darstellung des gespeicherten Zeitpunkts ist ein ISO-Zeitstempel:

```
>>> print(dt)
2015-04-18 13:37:25
```

Die einzelnen Bestandteile der Zeitangabe lassen sich über öffentliche Attribute abrufen:

```
>>> dt.year        # Jahr
2015
>>> dt.month       # Monat
4
>>> dt.day         # Tag im Monat
18
>>> dt.hour        # Stunde
13
>>> dt.minute      # Minute
37
>>> dt.second      # Sekunde
25
>>> dt.microsecond # Mikrosekunde
0
```

Der Wochentag für die Zeitangabe kann über eine der Methoden `weekday()` oder `isoweekday()` ermittelt werden; es handelt sich um einen numerischen Wert, der bei `weekday()` mit 0 für Montag und bei `isoweekday()` mit 1 für Montag beginnt. Der 18. April 2015 war ein Samstag, sodass die folgenden Ergebnisse herauskommen:

```
>>> dt.weekday()
5
>>> dt.isoweekday()
6
```

Mit `datetime.today()` oder `datetime.now()` können Sie die aktuelle Systemzeit als Datumsobjekt speichern; der Unterschied ist, dass `now()` im Gegensatz zu `today()` eine optionale Zeitzonenangabe enthalten kann. Beispiel:

```
>>> print(datetime.today())
2015-05-14 12:49:21.125654
```

Die Methode `strftime()` wandelt die Zeitangabe gemäß einer Formatierungsvorschrift in einen String um. Die Syntax der Formatierungsvorschrift entspricht der gleichnamigen Funktion, die bereits im Abschnitt zur Programmiersprache C erwähnt wurde. Das folgende Beispiel formatiert die aktuelle Systemzeit in deutscher Schreibweise und wandelt dabei auch den Wochentag in Text um:

```
>>> wochentage = ["Montag", "Dienstag", "Mittwoch", "Donnerstag", "Freitag",
"Samstag", "Sonntag"]
>>> dt = datetime.today()
>>> print("{}, {}".format(wochentage[dt.weekday()], dt.strftime("%d.%m.%Y, %H:%M")))
Donnerstag, 14.05.2015, 13:44
```

Interessant ist auch die Klasse `timedelta`. Sie definiert eine Zeitspanne, die zu einer Zeitangabe addiert oder von dieser abgezogen werden kann. Der Konstruktor kann beliebig viele der folgenden benannten Argumente entgegennehmen (wenn Sie keines verwenden, ist die Differenz 0 und das `timedelta`-Objekt recht nutzlos):

```
timedelta(
    days = n,
    seconds = n,
    microseconds = n,
    milliseconds = n,
    minutes = n,
    hours = n,
    weeks = n
)
```

Hier ein Beispiel, das drei Wochen und zwei Tage zum aktuellen Zeitpunkt hinzuaddiert beziehungsweise von diesem abzieht:

```
>>> dt = datetime.today()
>>> delta = timedelta(weeks = 3, days = 2)
>>> dt
datetime.datetime(2015, 5, 14, 13, 54, 27, 443589)
>>> dt + delta
datetime.datetime(2015, 6, 6, 13, 54, 27, 443589)
>>> dt - delta
datetime.datetime(2015, 4, 21, 13, 54, 27, 443589)
```

9.4 Übungsaufgaben

1. Welche der folgenden Strings sind gültige C-Bezeichner?
   - `variable1`
   - `1variable`
   - `_variable`
   - `binärzahl`

2. Finden Sie den Fehler im folgenden C-Programm, möglichst ohne es abzutippen und zu kompilieren:

```
#include <stdio.h>
int ergebnis() {
  return a + b;
}

int main(int argc, char* argv[]) {
  int a = 17;
  int b = 29;
  printf("%d + %d = %d", a, b, ergebnis());
  return 0;
}
```

3. Wie prüfen Sie in C, ob die `int`-Variable a größer als 0 und kleiner als 10 ist?
   - `0 < a < 10`
   - `a > 0 & a < 10`
   - `0 < a && < 10`
   - `a > 0 && a < 10`
4. Welchen Wert hat die Variable `ergebnis` nach Ausführung der folgenden C-Operationen?

```
int a = 7;
int b = 9;
int ergebnis = ++a + b++;
```

5. Der folgende C-Codeausschnitt soll von 10 bis 1 herunterzählen. Tut er dies? Falls nicht, was tut er stattdessen?

```
for (int i = 10; i > 1; i++) {
  printf("%d\n", i);
}
```

6. Schreiben Sie ein kleines C-Programm mit folgender Spezifikation: Der User soll drei ganze Zahlen eingeben, und das Programm berechnet die Summe und den Mittelwert (Summe geteilt durch Anzahl) dieser Zahlen und gibt diese Werte aus.
7. Das folgende kleine Java-Programm soll die beiden Zahlen 23 und 42 addieren und das Ergebnis ausgeben. Tut es dies? Falls nicht, was gibt es stattdessen aus?

```
public class Rechentest {
  public static void main(String[] args) {
    int a = 23;
    int b = 42;
    System.out.println("23 + 42 = " + a + b);
  }
}
```

8. Wie prüfen Sie in Java, ob die beiden Strings `str1` und `str2` denselben Inhalt haben?
   - `str1 == str2`
   - `str1 = str2`
   - `str1.equals(str2)`
   - `str1 - str2 == 0`
9. Finden Sie alle Fehler in der folgenden Java-Klassendefinition:

```
public class TestKlasse() {
  private int wert = "Hallo Welt";

  // Konstruktor
  public void TestKlasse(int _wert) {
    wert = _wert;
  }

  public int getWert() {
    return _wert;
  }

  public int setWert(int _wert) {
    wert = _wert;
  }
}
```

   Zur Kontrolle: Es sind fünf Fehler, der Compiler beschwert sich aber nur über vier von ihnen.
10. Schreiben Sie eine Java-Klasse namens `Kontakt`, die die String-Attribute `vorname`, `nachname`, `email` und `phone`, Getter- und Setter-Methoden für diese Attribute sowie einen Konstruktor mit einem Argument für jedes Attribut enthält. Schreiben Sie anschließend ein Beispielprogramm, das mehrere Elemente vom Typ `Kontakt` zu einem `Set<Kontakt>` hinzufügt und anschließend die Liste aller Kontakte ausgibt.

    Jeder Kontakt soll dabei in einer eigenen Zeile stehen und folgendes Format haben (Beispiel):

```
Müller, Anna; E-Mail: anna.mueller@example.com; Telefon: 0161/12345678
```

11. Welche der folgenden (in C gültigen) Ausdrücke sind auch in Python erlaubt?
    - `a & b`
    - `a && b`
    - `a == 1 ? b : c`
    - `a++`
    - `b -= 3`

12. Was enthält die Python-Liste `list` nach Ausführung der folgenden Anweisungen (versuchen Sie, es vorherzusagen, bevor Sie es ausprobieren)?

```
list = [1, 2, 3, 4]
list[2:2] = [5, 6, 7]
```

13. Schreiben Sie in Python eine List Comprehension, die aus den Zahlen von 1 bis 100 alle diejenigen auswählt, die weder durch 2 noch durch 3 teilbar sind.
14. Schreiben Sie ein Python-Skript, das die Zahlen von 1 bis 100 untereinander rechtsbündig in eine Datei namens *zahlen.txt* schreibt.
15. Was gibt das folgende Python-Skript aus?

```
class A:
  def ausgabe(self):
    print("Ich bin A.")

class B(A):
  def ausgabe(self):
    print("Ich bin B.")

class C(B, A):
  pass

c = C()
c.ausgabe()
```

16. Schreiben Sie eine Python-Klasse namens `Tier` mit den Eigenschaften `name` und `beine`, die den Namen der Tierart beziehungsweise die Anzahl ihrer Beine enthält. Erstellen Sie dann im Hauptprogramm eine Liste von Tieren mit unterschiedlicher Beinanzahl, und geben Sie diese, sortiert nach der Anzahl der Beine, aus.

Kapitel 13
Datenbanken

Sammle erst die Fakten, dann kannst du sie verdrehen, wie es dir passt.
– Mark Twain

Zu den wichtigsten Rechneranwendungen gehören die Speicherung, Verwaltung und Manipulation beliebiger Informationen. *Datenbanken* erfüllen diesen Verwendungszweck am besten. In diesem Kapitel werden zunächst die verschiedenen Arten von Datenbanken vorgestellt. Anschließend werden die Installation und die Anwendung der beliebten Open-Source-Datenbank MySQL behandelt. Danach erhalten Sie einen Überblick über die bedeutendsten Optionen und Funktionen der in den meisten Datenbanksystemen verwendeten Abfragesprache SQL. Als Beispiel für die Datenbankprogrammierung wird schließlich JDBC vorgestellt, eine allgemeine Schnittstelle für den Zugriff auf Datenbanken aus Java-Programmen.

Zunächst ist es wichtig, genau zu definieren, was eine Datenbank eigentlich ist. Der Begriff bezeichnet nämlich zwei verschiedene Dinge: zum einen die Datensammlung selbst, zum anderen das Programm, das diese Daten verwaltet. Bei den Daten handelt es sich um eine nach bestimmten Regeln strukturierte Ansammlung von Informationen zu verschiedenen Themengruppen, beispielsweise Kundendaten von Unternehmen, Diagnose- und Behandlungsinformationen von Ärzten oder die private CD-Sammlung eines Musikfans.

Das Anwendungsprogramm, mit dem diese Daten verwaltet werden können, enthält mehr oder weniger mächtige Funktionen zum Suchen, Sortieren, Filtern und formatierten Ausgeben dieser Daten. Ein solches Programm wird als *Database Management System* (DBMS), also als *Datenbankverwaltungssystem*, bezeichnet.

Die Daten, die von einem DBMS verwaltet werden, lassen sich nach verschiedenen Kriterien unterscheiden – übrigens ein beliebtes Thema für IHK-Fragen, insbesondere im EDV-Bereich kaufmännischer Berufe.

Jede Information, die in einer Datenbank gespeichert wird, ist Stammdatum oder Bewegungsdatum und gleichzeitig Rechendatum oder Ordnungsdatum. Die folgenden Definitionen können Ihnen helfen, diese Begriffe zu unterscheiden:

- *Stammdaten* sind unveränderliche (oder zumindest selten veränderte) Informationen, die dauerhafte Auskunft über die Objekte oder Sachverhalte geben, die in der Datenbank gespeichert sind. Beispiele sind etwa der Name einer Person oder die Bestellnummer eines Artikels.

- *Bewegungsdaten* sind dagegen Informationen, die sich ständig im Fluss befinden und die Dynamik von Geschäftsabläufen und anderen Prozessen abbilden. Der Saldo auf einem Konto oder die Körpertemperatur eines Patienten sind gute Beispiele dafür.
- *Rechendaten* sind Daten, die entweder selbst als Glied in einer Berechnung stehen oder aber als Berechnungsgrundlage dienen. Dazu gehören etwa Preise, Zinssätze oder gefahrene Kilometer.
- *Ordnungsdaten* dienen dagegen der Einteilung, Klassifizierung und Filterung von Informationen. Zu den Ordnungsdaten zählen etwa Namen, Postleitzahlen oder Kfz-Kennzeichen.

Wichtig ist, wie bereits erwähnt, dass jedes Datum stets zwei der zuvor genannten Eigenschaften aufweist. Die folgende Liste zeigt Beispiele für jede der vier möglichen Kombinationen:

- Stammdatum und Rechendatum sind beispielsweise der Preis einer Ware, das Grundgehalt eines Mitarbeiters oder der effektive Jahreszins eines Kredits.
- Stammdatum und Ordnungsdatum sind etwa der Name eines Kunden, die Bestellnummer eines Artikels oder der Titel eines Buches.
- Bewegungsdaten und Rechendaten sind zum Beispiel die Anzahl der abgeleisteten Überstunden eines Mitarbeiters, die Anzahl der gefahrenen Kilometer oder der aktuelle Wechselkurs für eine Fremdwährung.
- Bewegungsdaten und Ordnungsdaten sind unter anderem das aktuelle Kalenderdatum, die Anzahl der Resturlaubstage eines Mitarbeiters oder die auf Lager befindliche Stückzahl eines Artikels.

13.1 Die verschiedenen Datenbanktypen

Da Daten zu vielen verschiedenen Zwecken gespeichert werden müssen, existieren unterschiedliche Arten von Datenbanken. Das wichtigste Modell ist die relationale Datenbank, zu der auch das in diesem Kapitel ausführlich vorgestellte MySQL gehört. In diesem Abschnitt werden die verschiedenen Datenbanktypen vorgestellt. Dies sind im Wesentlichen folgende:

- *Einzeltabellendatenbanken* dienen der einfachen Verwaltung von Daten eines bestimmten Typs, beispielsweise Anschriften oder Briefmarkensammlungen.
- *Relationale Datenbanken* bieten die Möglichkeit, mehrere Einzeltabellen miteinander zu verknüpfen, um die Daten konsistent zu halten: Informationen, die an verschiedenen Stellen vorkommen, müssen nur einmal aufgeführt werden.
- *Objektorientierte Datenbanken* arbeiten auf der Grundlage von Klassen und Objekten wie die objektorientierten Programmiersprachen C++ oder Java. Im Gegensatz zu relationalen Datenbanken sind sie in der Lage, sehr komplexe, nichtlineare Beziehungen zwischen den gespeicherten Informationen abzubilden.

- *Volltextdatenbanken* sind nicht unbedingt ein eigener Datenbanktyp. Speziell geht es hier um die Implementierung einer effektiven Volltextsuche in vorhandenen Textarchiven. Inzwischen sind auch relationale und objektorientierte Datenbanken mit Volltext-Suchfunktionen ausgestattet.
- *XML-Datenbanken* speichern die Informationen in Form von XML-Dokumenten ab. XML ist eine Metasprache für die Definition von Dokumentstrukturen und wird in Kapitel 16, »XML«, ausführlich behandelt.
- *Bild- und Multimedia-Datenbanken* sind meist erweiterte relationale Datenbanken, die die Verwaltung von Mediadaten wie Bildern, Sounddateien oder Digitalvideos realisieren. In der Regel ermöglichen sie die Suche nach Media-Dateien auf den diversen Datenträgern sowie deren Katalogisierung nach verschiedenen Kriterien. Bekannte Beispiele sind Canto Cumulus, das häufig von DTP- oder Presse-Profis eingesetzt wird, oder das kostengünstige Programm ThumbsPlus, das eher für kleine Büros oder Privatleute geeignet ist, die Ordnung in ihr Mediadaten-Chaos bringen möchten. Inzwischen verfügen die Desktops der meisten Betriebssysteme allerdings auch selbst über Bildvorschaufunktionen und zusätzliche Informationen zu Mediendateien.
- *NoSQL-Datenbanken* bilden einen recht neuen Ansatz in der Datenbanktechnik. Es handelt sich um Datenbanken, die auf die traditionelle Datenbankabfrage SQL und meist auch auf den relationalen Ansatz verzichten. Stattdessen speichern viele von ihnen die Daten als Dokumente mit beliebigen, frei definierbaren Metadatenfeldern, über die sie gesucht und gegebenenfalls auch verknüpft werden können. Ein bekanntes Beispiel ist CouchDB; am Ende dieses Kapitels erhalten Sie einen kurzen Überblick über die Möglichkeiten dieser Datenbank.

Neben diesen Grundtypen gibt es auch konkrete Datenbanksoftware, die verschiedene Misch- oder Übergangsformen bildet. Beispielsweise verwenden XML-Datenbanken oft keine reinen XML-Dokumente, sondern legen diese in einer relationalen Grundstruktur ab. Umgekehrt bieten die meisten modernen relationalen Datenbanksysteme spezielle Funktionen für Daten im XML-Format.

In den folgenden Abschnitten werden nur die Einzeltabelle, die relationale Datenbank und die objektorientierte Datenbank näher vorgestellt. Volltext- und Media-Datenbanken sind zu speziell und zu unterschiedlich, um sie allgemein zu beschreiben, und die Behandlung von XML-Datenbanken ohne XML-Kenntnisse ergibt keinen Sinn.

13.1.1 Einzeltabellendatenbanken

Die einfachste Datenbankart verwendet nur eine einzige Tabelle zur Abspeicherung aller Informationen. Die Einzeltabelle ist das Grundprinzip aller einfachen Adressverwaltungs- oder CD-Sammlungsprogramme. Die meisten Eigenschaften von Einzeltabellendatenbanken gelten auch für die professionelleren und erheblich vielseitigeren relationalen Datenbanken,

schließlich handelt es sich bei Letzteren um eine Sammlung miteinander verknüpfter Einzeltabellen.

In den Spalten einer Datenbanktabelle stehen die verschiedenen Informationskategorien. Die einzelnen Zellen werden *Datenfelder* genannt und sind die kleinste Informationseinheit der Datenbank: Sie enthalten je eine Einzelinformation über ein Element der Datenbank. Eine ganze Zeile ist die Kombination aller Informationen über ein Element und wird *Datensatz* (*Record*) genannt. Übrigens wird das jeweilige Element, über das ein Datensatz Informationen enthält, als *Entität* (*Entity*) bezeichnet.

In Tabelle 13.1 sehen Sie ein Beispiel für eine Datenbanktabelle, die verschiedene Informationen über die Mitarbeiter eines Unternehmens enthält.

Name	Vorname	Eintrittsdatum	Abteilung	Grundgehalt
Becker	Wolfgang	01.05.1987	Einkauf	3.800 €
Huber	Angelika	01.12.1996	Geschäftsleitung	5.900 €
Juarez	Manolo	01.10.1999	Verkauf	4.200 €
Klein	Franziska	01.09.2006	Auszubildende	635 €

Tabelle 13.1 Beispiel für eine einfache Einzeltabellendatenbank

Von einer Einzeltabellendatenbank dürfen Sie nicht den komplexen Funktionsumfang eines relationalen DBMS erwarten, aber von den folgenden Grundfunktionen sollten Sie dennoch ausgehen können:

- Sortieren der Tabelle auf- und absteigend nach einer beliebigen Kategorie (im Beispiel sind die Datensätze aufsteigend nach Namen sortiert). Eine absteigende Sortierung nach Gehältern würde beispielsweise die folgende Reihenfolge erzeugen: Huber, Juarez, Becker, Klein.
- Suchen nach einem beliebigen Feldinhalt und Ausgabe der relevanten Datensätze. Beispielsweise würde die Suche nach dem Eintrittsjahr 1996 Frau Huber zurückgeben.
- Einen Schritt weiter als die einfache Suche geht die Filterung der Tabelle. Dies bedeutet, dass nur noch diejenigen Zeilen angezeigt werden, die bestimmten Kriterien entsprechen. Beispielsweise enthielte eine Liste aller Mitarbeiter, die mehr als 4.000 € verdienen, nur noch Frau Huber und Herrn Juarez.

Darüber hinaus enthalten die meisten Einzeltabellendatenbanken je nach Verwendungszweck zahlreiche Formatierungsoptionen, um Eingabemasken oder ausdruckbare Berichte und Ähnliches zu erzeugen. Schließlich gibt es nicht allzu viele universelle Einzeltabellen-DBMS, sondern viel häufiger Programme, die für die Verwaltung einer bestimmten Datenart wie Adressen oder DVD-Sammlungen geeignet sind.

Die Grenzen der Einzeltabelle

Wenn Sie eine Weile mit einer Einzeltabellendatenbank arbeiten, werden Sie feststellen, dass vor allem eine Einschränkung besonders stört: Es gibt keine Möglichkeit, Inkonsistenzen auszugleichen. Wird beispielsweise die zuvor gezeigte Mitarbeitertabelle um fünf Kollegen erweitert, die alle in der Abteilung Verkauf arbeiten, dann gibt es keine Möglichkeit des Schutzes davor, dass der Name »Verkauf« in einem dieser fünf Datensätze falsch geschrieben wird. Daraus ergäbe sich natürlich eine vollkommen falsche Tabellenlogik mit Auswirkungen auf Such- und Filterfunktionen.

Noch schwieriger wird es, wenn Daten benötigt werden, die im Grunde gar nicht das Entity betreffen, das im Datensatz beschrieben wird, sondern zusätzliche Informationen über eines der anderen Felder enthalten. Beispielsweise könnten Sie es nützlich finden, neben der Abteilung auch deren Leiter zu erwähnen. Stellen Sie sich den Aufwand vor, wenn dieser Leiter wechselt, oder die Probleme, wenn Sie den Namen irgendwo falsch schreiben.

Einzeltabellen sind also nur für ganz einfache Anwendungszwecke geeignet: Eine kleine Adress- oder Briefmarkensammlungsverwaltung geht gerade noch, während sämtliche Unternehmensanwendungen nur mit relationalen Datenbanken vernünftig funktionieren. Diese ermöglichen es nämlich, dass Sie verschiedene Arten von Daten jeweils in eigenständigen Tabellen ablegen und sie anschließend über sogenannte *Schlüssel* miteinander verknüpfen.

13.1.2 Relationale Datenbanken

Genau wie Einzeltabellen verwenden auch relationale Datenbanken ein Tabellenmodell zum Ablegen von Daten. Eine relationale Datenbank besteht allerdings aus beliebig vielen Einzeltabellen, die auf vielfältige Weise miteinander verknüpft werden können. Dieser Aufbau sorgt dafür, dass die Daten in der Datenbank konsistent sind: Jede Information muss nur ein einziges Mal gespeichert werden, sodass es nicht zu Mehrdeutigkeiten kommen kann.

Technisch gesehen, basieren relationale Datenbanken auf der relationalen Algebra. Danach wird eine einzelne Tabelle als *Relation* bezeichnet. Jeder Datensatz ist ein Tupel (also geordnete Sammlung einer festgelegten Anzahl von Werten), in dem die Tabellenspalten die Attribute A1 bis An bilden. Das Relationenschema R = (A1, ..., An) legt Anzahl und Datentyp der Attribute fest. Eine Relation r(R) ist formal eine Relation mit dem Relationenschema R, besteht also aus Tupeln, für die gilt:

$$r(R) = r(A_1, ..., A_n)$$

Die Verknüpfungen zwischen den Spalten einer Tabelle und zwischen den einzelnen Tabellen werden als *Beziehungen* oder *Verknüpfungen* (englisch: *relationships*) bezeichnet. Auch der Begriff *Relationen* kommt dafür häufig zum Einsatz, obwohl er hier nichts mit Relationen im Sinne der relationalen Algebra zu tun hat. Eine Beziehung entsteht durch die Verwendung von Schlüsseln: Der *Primärschlüssel* des Datensatzes einer Tabelle wird als Wert in ein

Feld einer anderen Tabelle eingetragen. Der Primärschlüssel ist ein spezielles Datenfeld oder eine Kombination der Werte mehrerer Felder, die innerhalb der Tabelle einen einmaligen Wert besitzen und den Datensatz somit eindeutig kennzeichnen. Der Primärschlüssel einer Tabelle, auf den in einer anderen Tabelle verwiesen wird, heißt dort *Fremdschlüssel*.

Der Primärschlüssel ist übrigens ein Sonderfall eines sogenannten *Indexes*. Indizes ermöglichen den schnellen Zugriff auf bestimmte Tabelleninhalte, indem sie die Informationen einer Datenbankspalte separat in geordneter Form abspeichern. Um einen Datensatz anhand eines indizierten Feldes zu finden, muss nicht die gesamte Datenbanktabelle sequenziell durchsucht werden.

Es gibt drei Arten von Beziehungen, die je nach Art der gespeicherten Information nützlich sind:

- Eine *1:1-Beziehung* (oft, erneut abweichend von der relationalen Algebra, auch *1:1-Relation* genannt) verknüpft einen Datensatz einer Tabelle mit genau einem Datensatz einer anderen Tabelle. Dies ist immer dann nützlich, wenn bestimmte Informationsaspekte über ein Entity nicht so oft benötigt werden wie andere Aspekte. Die seltener oder in einem anderen Zusammenhang verwendeten Informationen lassen sich auf diese Weise einfach ausblenden. Beispielsweise könnte eine Tabelle Informationen über angebotene Artikel wie Bezeichnung, Preis und Mehrwertsteuersatz enthalten, eine andere dagegen den Lagerbestand.
- Eine *1:n-Beziehung* oder *Eins-zu-viele-Beziehung* verbindet einen Datensatz einer Tabelle mit beliebig vielen Datensätzen einer anderen Tabelle. Dies ist der häufigste und nützlichste Verknüpfungstyp, weil er den größten Beitrag zur Vermeidung von Inkonsistenzen leistet: Detailinformationen über Werte, die in einer Spalte einer Tabelle beliebig oft vorkommen können, werden in einer separaten Tabelle erfasst. Die erste Tabelle enthält in dem entsprechenden Feld nur noch einen Verweis auf einen bestimmten Datensatz der zweiten Tabelle.
- Eine *m:n-Beziehung*, auch *Viele-zu-viele-Beziehung* genannt, kombiniert beliebig viele Vorkommen eines bestimmten Wertes mit beliebig vielen Vorkommen eines anderen. Stellen Sie sich beispielsweise eine Tabelle vor, die eine Liste lieferbarer Waren enthält, und eine weitere Tabelle, in der die Adressen von Kunden erfasst werden. Jeder Artikel kann von beliebig vielen Kunden gekauft werden, und jeder Kunde kann beliebig viele unterschiedliche Artikel kaufen. Das relationale Datenbankmodell verlangt allerdings, dass diese Art von Beziehung indirekt dargestellt wird: Eine dritte Tabelle listet die einzelnen Kaufaktionen auf; jeder Datensatz enthält dabei eine Verknüpfung zu einem bestimmten Kunden und eine weitere zu einem einzelnen Artikel. Jeder dieser beiden Fremdschlüssel für sich bildet eine 1:n-Beziehung; die m:n-Beziehung zwischen Kunden und Artikeln besteht nicht direkt.

Ein einfaches Beispiel

Das unter den m:n-Beziehungen angedeutete Beispiel soll im Folgenden konkret ausgeführt werden: Eine Tabelle enthält Daten über Käufer, die zweite Informationen über die Artikel, und die dritte Tabelle listet jeden einzelnen Kauf auf. Die Beziehung zwischen den drei Tabellen *ADRESSEN*, *KAEUFE* und *ARTIKEL* wird in Abbildung 13.1 dargestellt.

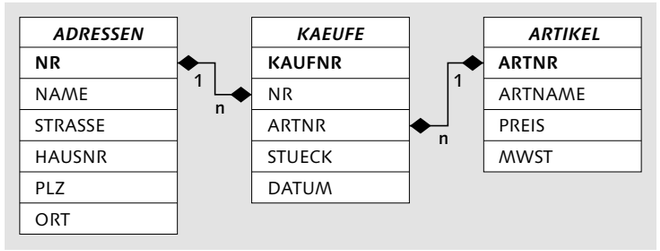


Abbildung 13.1 Relationen zwischen den Tabellen einer einfachen relationalen Datenbank

Das oberste, fett gesetzte Feld jeder Tabelle ist der Primärschlüssel. Die Tabelle *ADRESSEN* enthält die Kundendaten mit dem Primärschlüssel *NR*. *ARTIKEL* hat den Primärschlüssel *ARTNR* (Artikelnummer). Die Tabelle *KAEUFE* schließlich verwendet einen Primärschlüssel namens *KAUFNR*. Da keine andere Tabelle auf einzelne Käufe zugreift, benötigt die Tabelle momentan eigentlich keinen Primärschlüssel.

Die Beschriftungen an den kleinen Rauten, von denen die Beziehungen ausgehen, zeigen den Beziehungstyp an. Beide Beziehungen in der Datenbank sind 1:n-Beziehungen. Die m:n-Beziehungen zwischen den Tabellen *ADRESSEN* und *ARTIKEL* können natürlich nicht eingezeichnet werden, weil sie nur indirekt besteht.

Konkret kann die Tabelle *ADRESSEN* zum Beispiel mit den Werten aus Tabelle 13.2 gefüllt werden.

NR	NAME	STRASSE	HAUSNR	PLZ	ORT
1	Schmidt	Kleiner Weg	1	50678	Köln
2	Müller	Alte Str.	78	80543	München
3	Becker	Störtebekerweg	45	20567	Hamburg
4	Heinze	Grüne Allee	36	10345	Berlin

Tabelle 13.2 Die Datenbanktabelle ADRESSEN

Natürlich müssen Sie die Kunden nicht durchnummerieren, sondern können sich Ihr eigenes individuelles Schema für Kundennummern ausdenken. Wichtig ist lediglich, dass jede

dieser Nummern nur ein einziges Mal in der Tabelle vorkommt und dass jeder Kunde eine (eindeutige) Nummer bekommt.

Die Tabelle *ARTIKEL* enthält Informationen über die angebotenen Artikel (siehe Tabelle 13.3). Beachten Sie, dass das Feld *MWST* nicht etwa einen konkreten Wert enthält, sondern den Mehrwertsteuersatz, also den Wert 7 oder den Wert 19. Noch praktischer wären allerdings 1:1-Relationen zu einer weiteren Tabelle, die die konkreten Mehrwertsteuersätze enthält; schließlich können sich diese ändern. Die Preise werden aus im weiteren Verlauf näher erläuterten Gründen in Cent angegeben.

ARTNR	ARTNAME	PREIS	MWST
1	Cola	119	19
2	Vollmilch	79	7
3	Toastbrot	149	7
4	Zahnpasta	179	19

Tabelle 13.3 Die Datenbanktabelle ARTIKEL

Nachdem diese beiden Tabellen eingerichtet sind, können nun Käufe getätigt werden. In der Tabelle *KAEUFE* könnten etwa die folgenden Geschäftsvorfälle erfasst werden (siehe Tabelle 13.4):

KAUFNR	NR	ARTNR	STUECK	DATUM
1	3	3	2	2015-04-15
2	2	1	4	2015-04-16
3	2	2	1	2015-04-16
4	1	4	1	2015-04-18
5	1	3	1	2015-04-18

Tabelle 13.4 Die Datenbanktabelle KAEUFE

Die Einträge in der Tabelle *KAEUFE* sind in dieser Form für Menschen so gut wie unlesbar. Es geht auch gar nicht darum, sie in einer Form vorzuhalten, in der sie sofort lesbar sind, sondern darum, die Daten redundanzfrei und kompakt zu speichern. Für die lesbare Ausgabe beherrscht jedes relationale Datenbankverwaltungssystem (RDBMS, Relational Database Management System) sogenannte *Auswahlabfragen*, mit deren Hilfe Sie anhand der Relationen Daten aus verschiedenen Tabellen zusammenstellen können.

Tabelle 13.5 zeigt das Ergebnis einer solchen Auswahlabfrage. Hier werden die Käufe mit den eigentlichen Kunden- und Artikelnamen aufgeführt, alphabetisch nach Kunden und anschließend alphabetisch nach Artikelbezeichnungen sortiert. Damit Sie das Ergebnis nachvollziehen können, wird die Kaufnummer aus der Tabelle *KAEUFE* übernommen. Die letzte Spalte enthält den errechneten Gesamtpreis für jeden einzelnen Kauf (das Produkt aus Stückzahl und Einzelpreis).

KAUFNR	NAME	ARTNAME	STUECK	GESAMTPREIS
1	Becker	Toastbrot	2	298
2	Müller	Cola	4	476
3	Müller	Vollmilch	1	079
5	Schmidt	Toastbrot	1	149
4	Schmidt	Zahnpasta	1	179

Tabelle 13.5 Eine Auswahlabfrage, die die Käufe in lesbarer Form darstellt

Beachten Sie, dass die Ergebnisse von Auswahlabfragen normalerweise nicht abgespeichert werden. Schließlich basieren sie auf Daten, die sich durch die nachträgliche Änderung oder Ergänzung von Datensätzen in den zugrunde liegenden Tabellen jederzeit ändern können.

Die meisten relationalen Datenbanksysteme verwenden für Abfragen eine standardisierte Sprache namens *SQL* (Structured Query Language). Diese Abfragesprache wird in Abschnitt 13.3, »SQL-Abfragen«, vorgestellt.

Normalisierung

Das wichtigste Ziel beim Erstellen relationaler Datenbanken ist – wie bereits erwähnt – die Beseitigung von Redundanzen zur Vermeidung von Inkonsistenzen. Dieser Vorgang wird als *Normalisierung* der Datenbank bezeichnet. Insgesamt sind fünf, eigentlich sogar sechs sogenannte *Normalformen* definiert, die aufeinander aufbauen und schrittweise den Weg zu einem fertigen relationalen Datenbankmodell weisen:

- Die erste Normalform (1NF) verlangt, dass die Information in jedem Feld einer Datenbank *atomar* ist – also eine nicht weiter zerlegbare Einzelinformation. »Einzelinformation« hängt allerdings vom Anwendungszweck ab: Ein Zustelldienst müsste Anschriften beispielsweise in all ihre Einzelteile zerlegen, während eine Anschrift als reine Endkundeninformation durchaus als Gesamtwert in einem einzelnen Feld stehen könnte. Verboten sind gemäß der 1NF insbesondere auch listenartige Wiederholungen gleichartiger Informationen – etwa mehrere Telefonnummern einer Person.

- Die zweite Normalform (2NF) fordert zusätzlich, dass Datensätze nur direkte Informationen über ein und denselben Sachverhalt enthalten. Formal gesagt, dürfen alle Felder in einer Tabelle, die die zweite Normalform erfüllt, nur vom Primärschlüssel dieser Tabelle abhängen. Beispielsweise dürfte eine Person mit zwei Wohnsitzen nicht zweimal in eine Kundentabelle aufgenommen werden. In diesem Fall müssten die Wohnorte in eine separate Tabelle geschrieben werden; der Bezug auf die Kunden müsste in dieser Tabelle als Fremdschlüssel eingetragen werden.
- Die dritte Normalform (3NF) ist erfüllt, wenn alle Felder funktional unabhängig voneinander sind. Der Unterschied zur zweiten Normalform erscheint geringfügig. Eine Tabelle, die zwar die zweite, aber nicht die dritte Normalform erfüllt, belässt eine eindeutig zu einem bestimmten Feld gehörende Zusatzinformation innerhalb einer Tabelle, in der dieses Feld keinen Schlüssel bildet. Beispielsweise hat in der Personaltabelle einer Unternehmensdatenbank, in der in einem Feld die Abteilung eines Mitarbeiters steht, der Name des Abteilungsleiters nichts zu suchen, weil er nur von der Abteilung, aber nicht vom Mitarbeiter abhängt.
- Die Boyce-Codd-Normalform (BCNF), benannt nach Pionieren des relationalen Datenbankmodells, ist eine strengere Variante der dritten Normalform. Eine Tabelle, die sich in der dritten Normalform befindet, kann die BCNF verletzen, wenn der Primärschlüssel aus mehreren Feldern zusammengesetzt ist und der Wert irgendeines Feldes nicht vom gesamten Primärschlüssel, sondern nur von einem seiner Felder abhängt. Um die Boyce-Codd-Normalform zu erreichen, muss eine solche Tabelle in zwei Einzeltabellen aufgeteilt werden, die jeweils eines dieser Felder als Primärschlüssel aufweisen.
- Die vierte Normalform (4NF) betrifft sogenannte *mehrwertige Abhängigkeiten* (*multi-valued dependencies*). Eine mehrwertige Abhängigkeit liegt vor, wenn eine Beziehung zwischen verschiedenen Informationen nicht so in zwei Tabellen unterteilt werden kann, dass eine 1:n- oder die umgekehrte n:1-Relation entsteht.

  Angenommen, in einer zweispaltigen Tabelle werden durch einen Fremdschlüssel Personen referenziert und in der zweiten Spalte durch einen weiteren Fremdschlüssel die von diesen Personen verwendeten Betriebssysteme. Da eine Person mehrere Betriebssysteme einsetzen kann, kann sowohl jede Person als auch jedes Betriebssystem mehrmals vorkommen.

  Die vierte Normalform wird verletzt, sobald eine weitere Spalte hinzukommt, die beispielsweise die von diesen Personen beherrschten Programmiersprachen auflistet: In diesem Fall entstehen Redundanzen durch die mehrfache Nennung der Betriebssysteme und Programmiersprachen für denselben Benutzer. Eine andere Ausprägung wären beliebige, nicht zusammenhängende Paare von Betriebssystem und Programmiersprache. Wenn sich die Anzahl der von einer Person verwendeten Betriebssysteme und der Programmiersprachen unterscheidet, bleiben sogar Felder leer.

Tabelle 13.6 zeigt schematisch, wie das aussieht. Die Lösung besteht natürlich darin, eine solche Tabelle in zwei Einzeltabellen zu unterteilen, deren Primärschlüssel jeweils die Person ist.

Name	Betriebssystem	Programmiersprache
Schmitz	Windows XP	Java
Schmitz	Windows XP	Perl
Müller	OS X	C++
Müller	Windows 2000	C++
Becker	FreeBSD	Perl
Becker	FreeBSD	Java
Becker	Linux	Perl
Becker	Linux	Java

Tabelle 13.6 Eine Datenbanktabelle, die die vierte Normalform verletzt: Es treten Redundanzen auf, weil eine Person mehrere Betriebssysteme oder Programmiersprachen verwenden kann.

- Die fünfte Normalform (5NF) erfordert schließlich, dass innerhalb einer Tabelle nur triviale Join-Abhängigkeiten existieren dürfen. Eine Join-Abhängigkeit besteht in jeder Tabelle, die sich in mehrere Einzeltabellen aufteilen ließe, indem derselbe Schlüssel jeweils auf einzelne Spalten der Tabelle angewandt wird. Trivial ist eine Join-Abhängigkeit dann, wenn durch eine Verknüpfung zweier solcher Einzeltabellen keine Redundanz durch einen verdoppelten Datensatz vorkäme.

  Wenn Sie beispielsweise zwei Tabellen vereinigen, von denen die eine einzelne Personen und deren Wohnort und die andere dieselben Personen und deren Bundesland auflistet, würde auch die entstehende Join-Tabelle die fünfte Normalform erfüllen, da jede Person in genau einem Ort und genau einem Bundesland wohnt.

  Ein Join der Wohnorte-Tabelle und einer Tabelle, in der jede Zeile eine Person und eine ihrer Telefonnummern enthält, verletzt die fünfte Normalform dagegen: Da eine Person mehrere Telefonnummern besitzen kann, würde ihr Wohnort mehrfach genannt. Diese Informationen müssen also getrennt voneinander gespeichert bleiben.

Die Bezeichnung *Normalisierung* könnte als kontinuierlicher Prozess missverstanden werden, der auf eine bereits bestehende Datenbank angewendet wird. In Wirklichkeit müssen Sie sich bereits bei der Datenbankmodellierung Gedanken darüber machen, also bevor Sie eine Datenbank in der Praxis einrichten.

Transaktionen

Ein fortgeschrittenes Feature relationaler Datenbanksysteme ist die Unterstützung von *Transaktionen*. Es handelt sich um die Möglichkeit, ein »Paket« aus beliebig vielen Änderungen in der Datenbank insgesamt zu bestätigen (*Commit*) oder rückgängig zu machen (*Rollback*).

Stellen Sie sich zur Verdeutlichung ein Online-Einkaufssystem vor: Ein Kunde kann beliebig viele Artikel zum Warenkorb hinzufügen und wieder daraus entfernen. Zum Schluss kann er die ausgewählten Waren bestellen oder den gesamten Vorgang abbrechen. Transaktionen sind die ideale Lösung für solche Aufgaben: Sobald der Kunde auf das System zugreift, wird eine neue Transaktion gestartet. Daraufhin merkt sich das Datenbanksystem alle Bewegungen im Warenkorb und alle sonstigen Einstellungen. Kommt es zu einer Bestellung, wird der Commit der Transaktion durchgeführt; bei einem Abbruch erfolgt dagegen der Rollback.

Da die Transaktionsfähigkeit die Performance eines RDBMS beeinträchtigt, handhabt der in Abschnitt 13.2, »MySQL – ein konkretes RDBMS«, behandelte MySQL-Server sie auf pragmatische Weise: Er bietet unterschiedliche Tabellentypen an; die neueren InnoDB-Tabellen unterstützen Transaktionen, während die klassischen MyISAM-Tabellen schneller verarbeitet werden.

RDBMS-Arten

Da relationale Datenbankverwaltungssysteme das verbreitetste Datenbankmodell sind, gibt es eine riesige Menge von Produkten, die diesem Standard entsprechen. Man kann sie grob in folgende Gruppen unterteilen:

- *Desktop-Datenbanken* sind Datenbankanwendungen, die für die Datenverwaltung am Einzelplatz oder in kleinen Arbeitsgruppen geeignet sind. Sie bieten in der Regel eine grafische Benutzeroberfläche, die die Tabellen und Abfragen übersichtlich und einfach darstellt, und ermöglichen das einfache Erzeugen von Eingabemasken und ausdruckbaren Berichten. Bekannte Beispiele für Desktop-Datenbanksysteme sind das in der Microsoft-Office-Familie integrierte Programm Access, OpenOffice.org Base oder die aus dem Mac-Bereich stammende und inzwischen auch für Windows verfügbare Datenbank FileMaker.
- *Kommerzielle Datenbankserver* werden insbesondere für verteilte Unternehmensanwendungen eingesetzt. Es handelt sich um komplexe und sehr teure modular erweiterbare Systeme. Bekannte Beispiele sind Oracle, Microsoft SQL Server oder IBM DB2. Im erweiterten Sinn gehören auch Branchenlösungen oder Warenwirtschaftssysteme wie SAP ERP dazu, weil sie alle auf speziell angepassten Datenbanken basieren.
- *Freie Datenbankserver* sind eine günstige Alternative zu den kommerziellen Produkten. Die bekanntesten Open-Source-Datenbanksysteme sind das in Abschnitt 13.2, »MySQL – ein konkretes RDBMS«, vorgestellte MySQL sowie PostgreSQL.

13.1.3 Objektorientierte Datenbanken

Trotz der soeben besprochenen Normalisierung lassen sich gewisse komplexe Datenstrukturen nur unzureichend mithilfe einer relationalen Datenbank modellieren. Aus diesem Grund wurde das objektorientierte Datenbankmodell eingeführt, das eine völlig freie und beliebige Strukturierung der Daten ermöglicht. Eine objektorientierte Datenbank besteht aus Klassen und Objekten und ähnelt damit der objektorientierten Programmierung, die in Kapitel 9, »Grundlagen der Programmierung«, erläutert wird.

Ein gutes Beispiel für ein Gefüge, das sich durch relationale Modellierung nicht vernünftig darstellen lässt, wären Verbindungen und Entfernungen zwischen verschiedenen Orten, wie sie beispielsweise für ein Speditionsunternehmen benötigt werden. Tabelle 13.7 unternimmt dennoch den Versuch, diese Informationen nach relationalem Muster darzustellen.

Diese Tabelle ist aus verschiedenen Gründen ungeeignet. Erstens enthalten zwei Spalten Informationen der gleichen Art; es gibt kein Kriterium, das bestimmt, welche Stadt unter *VON_ORT* aufgeführt wird und welche unter *NACH_ORT*. Die Darstellung der Entfernung in beide Richtungen ist auch keine Alternative, weil es dadurch zu Redundanzen käme. Durch die Normalisierungsregeln für relationale Datenbanken lässt sich dieses Modell nicht weiter verbessern.

VON_ORT	NACH_ORT	ENTFERNUNG
Köln	Berlin	570
Köln	Hamburg	370
Köln	München	594
Hamburg	München	781
Hamburg	Berlin	286
München	Berlin	585

Tabelle 13.7 Eine für das relationale Datenbankdesign ungeeignete Tabelle

Übrigens ist es auch keine Lösung, die durchaus relational darstellbaren Entfernungen von einer bestimmten Stadt zu allen anderen zu verwenden. Dies verhindert nämlich die Aufnahme günstigerer Verbindungen in die Datenbank. Angenommen, eine Tabelle listet die Entfernungen von Köln zu den anderen Städten auf. Sicherlich würde niemand, der von Hamburg nach Berlin muss, den Umweg über Köln wählen.[1]

1 Obwohl ich allen Hamburgern versichern kann, dass es sich lohnen würde ;-).

Mithilfe der objektorientierten Modellierung ist die Darstellung dieser Entfernungen dagegen ein Leichtes. Hier lässt sich eine Klasse namens `Ort` einrichten, die einfach ein Array von Zielen enthält, zu denen Verbindungen bestehen.

Es gibt verschiedene Sprachen, in denen sich objektorientierte Datenbanken formulieren lassen. Einige Lösungen verwenden die Syntax objektorientierter Programmiersprachen wie C++, während andere Systeme ihre eigenen Sprachen benutzen. Einen allgemeinen Standard für objektorientierte Datenbankverwaltungssysteme (OODBMS), wie ihn SQL für relationale Datenbanken bildet, gibt es noch nicht. Dennoch arbeitet ein Gremium namens *Object Database Management Group* (ODMG) an einer solchen Standardisierung. Eine einigermaßen verbreitete Objektmodellierungssprache ist die von dieser Gruppe definierte Object Definition Language (ODL).

Eine übergeordnete Datenstruktur, am ehesten vergleichbar mit einer Tabelle in einer relationalen Datenbank, wird durch eine Klasse gebildet, deren Definition das Schlüsselwort `class` einleitet. Eine Informationskategorie, die in etwa einer Spalte in einer relationalen Datenbanktabelle entspricht, wird durch das Schlüsselwort `attribute`, die Angabe eines Datentyps und eine Bezeichnung eingerichtet. Das folgende Beispiel entspricht der zuvor dargestellten relationalen Tabelle *ADRESSEN*:

```
class Adressen {
   attribute long Nr;
   attribute string Name.;
   attribute string Strasse;
   attribute short Hausnr;
   attribute short Plz;
   attribute string Ort;
}
```

Die atomaren Datentypen `string`, `long` und `short` sollten sich von selbst erklären. Es existieren keine separaten Typen für ganzzahlige und für Fließkommazahlen. Eine objektorientierte Umsetzung der Tabelle *ARTIKEL* sieht folgendermaßen aus:

```
class Artikel {
   attribute long ArtNr;
   attribute string ArtName;

   attribute long Preis;
   attribute short MWSt;
}
```

Eine Beziehung wird in der ODL durch das Schlüsselwort `relationship` dargestellt. Da jeder Datensatz eine Instanz einer Klasse ist, müssen Sie nicht mit Schlüsseln arbeiten, sondern können ein Objekt dieser Klasse direkt referenzieren. Die Darstellung der Tabelle in einer OO-Datenbank lautet demnach:


```
class Kaeufe {
   attribute long KaufNr;
   relationship Adressen Kunde;
   relationship Artikel Ware;
   attribute short Stueck;
   attribute struct Datum {
      short Tag;
      short Monat;
      short Jahr;
   } KaufDatum;
}
```

Der Datentyp `struct` bietet die Möglichkeit, eine nichtatomare Information als verschachtelte Gruppe einzufügen. Dies garantiert im vorliegenden Fall die leicht handhabbare Darstellung eines Datums. Die Syntax für `struct` entspricht dabei dem in der Programmiersprache C üblichen Standard: Vor der öffnenden geschweiften Klammer steht der Datentypname der Struktur, hinter der schließenden wird ein konkretes Element dieses Typs deklariert.

Die ursprüngliche Aufgabe, die Entfernungstabelle darzustellen, lässt sich nun mithilfe der ODL-Syntax recht einfach lösen:

```
class Ort {
   attribute string Name.;
   struct Entfernung {
      relationship Ort Zielort;
      attribute short Kilometer;
   };
   array (struct Entfernung) Entfernungen;
}
```

Ein `array` ist – wie in Programmiersprachen – eine Liste beliebig vieler Elemente eines bestimmten Datentyps. In diesem Fall wird eine Entfernung als Struktur aus einer Verknüpfung mit einem Ort und der Kilometeranzahl gebildet. Die entsprechenden Entfernungen werden in einem Array dargestellt.

Um objektorientierte Datenstrukturen mit Werten zu füllen und um diese Werte später nach verschiedenen Kriterien zu lesen und auszuwerten, wird eine zweite Sprache benötigt. Die von der ODMG vorgeschlagene Version einer solchen objektorientierten Abfragesprache heißt *OQL* (Object Query Language). Sie verwendet weitgehend dieselben Befehle und die gleiche Syntax wie die in Abschnitt 13.3, »SQL-Abfragen«, vorgestellte relationale Abfragesprache SQL.

13.2 MySQL – ein konkretes RDBMS

In diesem Abschnitt werden die Installation, die Konfiguration und die ersten Schritte mit einem relationalen Datenbanksystem besprochen. Es handelt sich um die weitverbreitete, beliebte Datenbank MySQL. Dieses System habe ich ausgewählt, weil Sie es kostenlos herunterladen und leicht installieren können und weil es in Zusammenarbeit mit der Programmiersprache PHP die Grundlage vieler dynamischer Websites bildet. Die MySQL-Programmierung mit PHP wird in Kapitel 19, »Webserveranwendungen«, erläutert.

Der ursprüngliche Entwickler von MySQL, *Michael »Monty« Widenius*, entwickelt seit 2009 eine alternative Implementierung der Datenbank unter dem Namen *MariaDB*. Sie ist weitgehend zu MySQL kompatibel und kann unter *www.mariadb.com* heruntergeladen werden. Hauptsächlich geht es bei MariaDB um eine strengere Open-Source-Auslegung, als sie von Oracle zu erwarten ist, aber die alternative Datenbank bietet auch einige neue Features wie zusätzliche Storage-Engines und Geschwindigkeitsoptimierungen.

13.2.1 MySQL installieren und konfigurieren

In diesem Abschnitt wird die Installation von MySQL in der aktuellen stabilen Version 5.6 erläutert. Zusätzlich erfahren Sie, wie Sie die beiden nützlichen Zusatzprogramme MySQL Administrator und MySQL Query Browser installieren können.

Installation unter Unix

Überprüfen Sie zunächst, ob MySQL nicht bereits Bestandteil Ihrer Systemdistribution ist; in der Regel dürfte dies der Fall sein. Auf der Website *http://www.mysql.com* finden Sie für zahlreiche Unix-Varianten Binärpakete. Laden Sie die passenden herunter, entpacken Sie sie, und erstellen Sie der Bequemlichkeit halber einen Symlink von dem relativ langen Pfadnamen nach */usr/local/mysql*.

Für so gut wie alle Unix-Versionen ist natürlich die Quellcode-Distribution geeignet. Diese müssen Sie wie üblich als Erstes entpacken; anschließend können Sie in das neu erstellte Verzeichnis wechseln:

```
# tar -xzvf mysql-5.6.26.tar.gz
# cd mysql-5.6.26
```

Aus Sicherheitsgründen sollten Sie einen neuen Benutzer und eine neue Gruppe erstellen; der MySQL-Server wird dann unter dieser User- und Group-ID ausgeführt. Dazu können Sie auf einem Linux-System beispielsweise die folgenden Befehle verwenden:

```
# groupadd mysql
# useradd -g mysql mysql
```


Da dieser »Benutzer« sich niemals persönlich anmelden wird, können Sie sich als Passwort einen vollkommen unmöglichen Zeichensalat ausdenken, den Sie sich nicht einmal zu merken brauchen.

Nun wird die folgende Befehlsfolge zur Konfiguration und Kompilierung eingegeben:

```
# ./configure --prefix=/usr/local/mysql \
--with-mysqld-user=mysql
# make
# make install
```

Sie können `configure` vorher auch mit der Option `--help` aufrufen, um Informationen über weitere Einstellungen zu erhalten. Die hier verwendeten Optionen legen */usr/local/mysql* als Stammverzeichnis fest und sorgen dafür, dass der Server unter der soeben angelegten User-ID ausgeführt wird.

Nach dem Entpacken der Binärvariante beziehungsweise der Installation der Source-Version müssen Sie die sogenannten *MySQL Grant Tables* erstellen, die die Authentifizierungs- und Berechtigungsdaten enthalten. Dazu können Sie folgendes Skript im *bin*-Verzeichnis Ihrer MySQL-Installation ausführen:

```
# ./mysql_install_db --user=mysql
```

Als Nächstes sollten Sie einige Besitzrechte setzen: Die ausführbaren MySQL-Programme sollten auf *root* übertragen werden, das Datenverzeichnis auf den neu angelegten User *mysql*. Wenn Sie MySQL mit dem `PREFIX` `/usr/local/mysql` installiert haben und sich in diesem Verzeichnis befinden, können Sie dies folgendermaßen tun:

```
# chown -R root.
# chown -R mysql data
# chgrp -R mysql.
```

Nun können Sie den `mysql`-Server starten:

```
# /usr/local/mysql/bin/mysqld_safe --user=mysql &
```

Das angehängte `&` führt dazu, dass der Befehl im Hintergrund ausgeführt wird, sodass Sie das entsprechende Terminal für andere Aufgaben weiterverwenden können. Falls Sie MySQL beim Hochfahren des Systems automatisch starten möchten, können Sie sich an die ausführliche Anleitung halten, die in Kapitel 7, »Linux«, teils für den Apache-Webserver und teils für MySQL selbst gegeben wurde. Das Skript *support-files/mysql.server* im MySQL-Verzeichnis kann dabei als Startskript dienen.

Zusätzlich zum MySQL-Grundpaket, das nur mit dem Kommandozeilen-Client *mysql* ausgeliefert wird, empfiehlt sich die Installation grafischer Steuer-Tools. Empfehlenswert ist beispielsweise der webbasierte Client phpMyAdmin (Download und Installationsanleitung

unter *http://www.phpmyadmin.net*). Daneben gibt es zwei Programme bei *mysql.com*: MySQL Administrator ermöglicht die grafische Konfiguration des Servers, während der MySQL Query Browser vor allem dem Erstellen und Durchsuchen von Datenbanken mithilfe von SQL-Abfragen dient.

Für Linux können Sie Binärpakete der beiden Programme herunterladen. Alle anderen Unix-Versionen müssen mit der Quellcode-Distribution vorliebnehmen. Die Binärvarianten können Sie nach dem Entpacken sofort einsetzen; die Quellcode-Distributionen werden mit dem »Automake-Dreisatz« (`configure`; `make`; `make install`) kompiliert und installiert.

Installation unter Windows

Die Download-Seite für den MySQL-Server 5.6 bietet zwei verschiedene Pakete für Windows, je einmal für 32- und für 64-Bit-Systeme: Unter dem Link Windows MSI Installer finden Sie den MySQL-Installer. Windows ZIP Archive ist dagegen ein Paket, das Sie einfach ohne Installation entpacken können (am sinnvollsten nach *C:\MySQL*), was schneller geht, aber weniger Einstellungsmöglichkeiten bietet.

Die beste Wahl ist der Standard-Installer. Führen Sie die heruntergeladene MSI-Datei (zurzeit *mysql-5.6.26-win32.msi* oder *mysql-5.6.26-win64.msi*) per Doppelklick aus. Anschließend werden auf mehreren Dialogseiten Fragen gestellt, unter anderem nach Installationsumfang und Installationsverzeichnis. Näheres über diesen Installer enthält der Online-Artikel unter *http://dev.mysql.com/tech-resources/articles/4.1/installer.html* (wie die Nummer in der URL andeutet, wurde er für MySQL 4.1 geschrieben, die erste Version, die mit diesem Installer geliefert wurde).

Zum Schluss der eigentlichen Installation können Sie Configure MySQL Server now wählen, um den MySQL Instance Configuration Wizard aufzurufen. Alternativ finden Sie ihn nachträglich unter MySQL im Startmenü.

In diesem Dialog werden nacheinander folgende Dialogseiten angezeigt:

- Welcome: zeigt eine kurze Information über die MySQL-Konfiguration.
- Konfigurationsdetails: Wählen Sie Detailed Configuration (diese Variante wird hier beschrieben) oder Standard Configuration für weniger Optionen.
- Servertyp: Wählen Sie Developer Machine für einen Arbeitsrechner, auf dem MySQL neben vielen Anwendungsprogrammen ausgeführt wird. Für den Praxiseinsatz können Sie sich zwischen Server Machine (Installation auf einem allgemeinen Serverrechner) oder Dedicated MySQL Server Machine (exklusiver Serverrechner für MySQL) entscheiden.
- Tabellentypen: Normalerweise sollten Sie Multifunctional Database wählen, um einfach zwischen transaktionsorientierten InnoDB- und performanceoptimierten MyISAM-Tabellen wählen zu können. Transactional Database Only optimiert den MySQL-Server für den InnoDB-Support, während Non-Transactional Database ausschließlich MyISAM installiert.

- InnoDB Tablespace Settings. Hier wird das temporäre Verzeichnis für Transaktionsdaten ausgewählt. Stellen Sie sicher, dass Sie die Festplatte oder Partition mit dem meisten freien Platz auswählen; in der Regel geschieht dies automatisch.
- Anzahl der gleichzeitigen Clientverbindungen: Wählen Sie für einen Arbeitsplatzrechner Decision Support (DSS)/OLAP mit maximal 20 Verbindungen. Für einen praxistauglichen Server ist Online Transaction Processing (OLTP) geeignet. Unter Manual Setting • Concurrent Connections können Sie die Höchstzahl der Verbindungen manuell eingeben.
- Um den Datenbankserver über ein Netzwerk zu nutzen (Standard), müssen Sie Enable TCP/IP Networking aktivieren. Der Standard-TCP-Port 3306 ist in der Regel die richtige Wahl, es sei denn, Sie möchten mehrere MySQL-Instanzen auf demselben Rechner betreiben. Beachten Sie gegebenenfalls auch Ihre Firewall-Einstellungen.
- Als Zeichensatz ist in der Regel Standard Character Set (Latin1) zu empfehlen – zumindest für die meisten lateinisch geschriebenen europäischen Sprachen. Wenn Sie viel mit internationalen Sprachen arbeiten oder vorwiegend moderne (Web-)Software einsetzen, empfiehlt sich dagegen UTF8. Die dritte Möglichkeit ist die manuelle Auswahl eines Zeichensatzes.
- Damit der MySQL-Server automatisch gestartet wird, sollten Sie auf der nächsten Registerkarte Install As Windows Service wählen. Solange auf demselben Rechner nicht mehrere MySQL-Server-Instanzen ausgeführt werden, ist der vorgeschlagene Dienstname MySQL in Ordnung. Launch the MySQL Server automatically schaltet den üblichen automatischen Start für den Dienst ein.
- Unter Modify Security Settings sollten Sie zunächst ein gutes, das heißt ausreichend langes und nicht im Wörterbuch vorkommendes Passwort für den MySQL-Administrator *root* auswählen. Außerdem empfiehlt es sich, die Einstellung Enable root access from remote machines (Administratorzugriff von entfernten Rechnern aus) zu deaktivieren.

  Die Einrichtung eines anonymen, also benutzernamen- und passwortlosen Kontos mithilfe von Create An Anonymous Account ist übrigens nicht zu empfehlen, sofern Sie keine Anwendung benutzen, die es unbedingt benötigt.
- Nachdem Sie alles eingestellt haben, führt Execute die gewünschten Arbeitsschritte durch.

Für die Windows-Versionen von MySQL Administrator und MySQL Query Browser werden übrigens bequeme Binär-Installer angeboten. Die wenigen einfachen Installationsschritte sind im Prinzip selbsterklärend.

13.2.2 Erste Schritte mit dem mysql-Client

Nach der Installation können Sie Ihren MySQL-Datenbankserver am einfachsten über den Kommandozeilen-Client *mysql* ansprechen. Wenn Sie ein anonymes Konto eingerichtet haben, starten Sie ihn einfach durch die folgende Eingabe:

```
$ mysql
```

Andernfalls müssen Sie über die Option `-u <Username>` den Benutzernamen angeben und mithilfe von `-p` festlegen, dass das Passwort abgefragt werden soll. Als Beispiel hier der MySQL-Verwaltungsbenutzer *root*:

```
$ mysql -u root -p
Enter password: *********
```

Nachdem die Verbindung zum Datenbankserver hergestellt wurde, sehen Sie den Prompt des `mysql`-Clients:

```
mysql>
```

Nun können Sie spezifische Steuerbefehle wie `help` (Anzeigen der Hilfe) oder `quit` (Beenden) eintippen oder aber die im nächsten Abschnitt besprochenen SQL-Abfragen. SQL-Abfragen können beliebig viele Zeilen lang sein (einschließlich ↵); sie werden erst ausgeführt, wenn Sie sie mit einem Semikolon abschließen.

Bevor Sie mit einer Datenbank arbeiten können, müssen Sie diese auswählen. Dies geschieht mithilfe der Anweisung `use <Datenbank>`. Beispiel:

```
mysql> use kaufhaus
```

Im Auslieferungszustand enthält MySQL nur die Verwaltungsdatenbank *mysql* sowie die leere Demo-Datenbank *test*. Bevor Sie also ernsthafte Experimente unternehmen können, müssen Sie eine Datenbank anlegen. Dies geschieht mithilfe einer `CREATE DATABASE`-Abfrage, die im nächsten Abschnitt erläutert wird.

Praktisch ist außerdem die Möglichkeit, SQL-Anweisungen aus einer externen Textdatei zu importieren. Dazu dient die Anweisung `source <Pfad>`. Beispiel:

```
mysql> source test.sql
```

13.3 SQL-Abfragen

In diesem Abschnitt werden einige Einzelheiten der Datenbankabfrage SQL näher erläutert. So gut wie alle relationalen Datenbanksysteme verstehen irgendeine Version dieser Sprache. Die hier vorgestellten SQL-Funktionen und -Merkmale funktionieren allesamt unter MySQL und sind, falls nicht anders vermerkt, konform mit dem SQL99-Standard. Beachten Sie jedoch, dass SQL99 einige weitere Fähigkeiten besitzt, die von MySQL bisher noch immer nicht voll unterstützt werden. Um die Beispiele im vorliegenden Abschnitt unter anderen RDBMS wie PostgreSQL, Microsoft SQL Server oder Oracle auszuführen, sind gegebenenfalls Anpassungen erforderlich, die Sie der Dokumentation Ihres Datenbanksystems entnehmen müssen.

Die Bezeichnung *Abfrage* (*Query*) ist ein wenig irreführend, weil Sie mithilfe von Abfragen nicht nur die Inhalte von Datenbanktabellen lesen, sondern auch ändern können. SQL unterstützt im Wesentlichen vier Arten von Datenbankabfragen:

- Auswahlabfragen (*Select Queries*) liefern ausgesuchte Felder einer oder mehrerer Tabellen zurück; optional können Kriterien angegeben werden, nach denen die Datensätze gefiltert werden sollen.
- Einfügeabfragen (*Insert Queries*) fügen neue Datensätze in eine Tabelle ein.
- Änderungsabfragen (*Update Queries*) ändern die Werte bestimmter Felder nach bestimmten Regeln und Kriterien.
- Löschabfragen (*Delete Queries*) entfernen Datensätze, die bestimmte Bedingungen erfüllen.

Neben diesen grundlegenden Abfragetypen, die bereits bestehende Tabellen betreffen, bietet SQL Befehle zum Anlegen und Entfernen von Datenbanken und von Tabellen innerhalb dieser Datenbanken.

Da SQL-Datenbanken zumindest auf Unix-Systemen zwischen Groß- und Kleinschreibung bei Tabellennamen unterscheiden, sollten Sie die Schreibung Ihrer Tabellen- und Feldbezeichnungen stets konsistent halten. Bei den SQL-Anweisungen und -Funktionen selbst wird dagegen nicht zwischen Groß- und Kleinschreibung unterschieden; traditionell werden sie komplett in Großbuchstaben geschrieben. Übrigens ist es egal, in wie viele Zeilen Sie eine Abfrage unterteilen. Falls sie mehrzeilig ist, muss sie allerdings durch ein Semikolon abgeschlossen werden.[2]

13.3.1 Datenbanken und Tabellen erzeugen

Um über SQL eine ganz neue Datenbank anzulegen, wird der Befehl `CREATE DATABASE` verwendet. Beispielsweise erzeugt die folgende Abfrage eine neue Datenbank namens *supermarkt*, die anschließend die bereits besprochenen Tabellen *adressen*, *artikel* und *kaeufe* enthalten soll:

```
CREATE DATABASE supermarkt;
```

Eine Tabelle wird per SQL über die Funktion `CREATE TABLE` angelegt. In Klammern werden – durch Komma getrennt – die einzelnen Feldnamen, ihre Datentypen und Optionen aufgelistet. Indizes, mit Ausnahme des Primärschlüssels, werden nicht beim Erstellen des jeweiligen Feldes, sondern separat über das Schlüsselwort `INDEX` angelegt.

Die SQL-Abfrage, mit deren Hilfe die Tabelle *adressen* eingerichtet wird, sieht folgendermaßen aus:

2 Der Kommandozeilen-Client *mysql* benötigt das Semikolon immer, weil er sonst davon ausgeht, dass eine Abfrage nach dem Zeilenumbruch weitergeht.

```
CREATE TABLE adressen (
   adressnr INT AUTO_INCREMENT PRIMARY KEY,
   name VARCHAR(50) NOT NULL,
   strasse VARCHAR(50) NOT NULL,
   hausnr CHAR(10) NOT NULL,
   plz CHAR(5) NOT NULL,
   ort VARCHAR(40) NOT NULL,
   INDEX (name),
   INDEX (ort)
);
```

Die Tabelle *artikel* wird mithilfe der folgenden Abfrage erstellt:

```
CREATE TABLE artikel (
   artnr INT AUTO_INCREMENT PRIMARY KEY,
   artname VARCHAR (30),
   preis INT,
   mwst ENUM ('7', '19'),
   INDEX (artname)
);
```

Schließlich wird noch die Tabelle *kaeufe* benötigt, die durch die folgende Abfrage erstellt werden kann:

```
CREATE TABLE kaeufe (
   kaufnr INT AUTO_INCREMENT PRIMARY KEY,
   adressnr INT,
   artnr INT,
   stueck INT,
   datum DATE
);
```

Falls Sie eine Tabelle wieder löschen möchten, wird die Funktion `DROP TABLE` verwendet. Die folgende SQL-Anweisung entfernt beispielsweise die Tabelle `unwichtig`:

```
DROP TABLE unwichtig;
```

Analog dazu können Sie mithilfe von `DROP DATABASE` eine ganze Datenbank löschen.

Felddatentypen und -optionen

Für jede Tabellenspalte, die Sie über eine `CREATE TABLE`-Abfrage einrichten, müssen Sie die folgenden Informationen angeben:

- Einen selbst gewählten Feldnamen. Dieser Name darf Buchstaben, Ziffern und Unterstriche enthalten, aber nicht mit einer Ziffer beginnen.

- Einen Felddatentyp. Die diversen möglichen Datentypen werden im weiteren Verlauf des Kapitels aufgezählt.
- Weitere Optionen. Hier können Sie besondere Eigenschaften der Spalte angeben, beispielsweise `PRIMARY KEY` (Primärschlüssel), `AUTO_INCREMENT` (automatisches Durchnummerieren) oder `NOT NULL` (das Feld muss einen Wert besitzen).

Die verschiedenen in SQL definierten Datentypen sind folgende:

- Ganzzahlen verschiedener Wortbreite
  Je nach Bedarf können Sie sich einen der folgenden ganzzahligen Datentypen aussuchen. Die tatsächliche Wortbreite ist allerdings abhängig von der Implementierung. Die Angaben gelten für MySQL und können je nach konkretem Datenbanksystem abweichen:
  - `TINYINT` (8 Bit)
  - `SMALLINT` (16 Bit)
  - `MEDIUMINT` (24 Bit)
  - `INT` (32 Bit)
  - `BIGINT` (64 Bit)
- Fließkommazahlen verschiedener Genauigkeit
  SQL bietet zwei verschieden genaue Datentypen für Fließkommawerte an. In MySQL gelten die folgenden Wortbreiten:
  - `FLOAT` (4 Byte)
  - `DOUBLE` (8 Byte)
  - Ein zulässiges Synonym für `DOUBLE` ist `REAL`.
  - Der Datentyp `DECIMAL` definiert eine Festkommazahl. Die Gesamtzahl der Stellen sowie die Anzahl der Nachkommastellen werden durch Komma getrennt in Klammern geschrieben. Beispielsweise wäre `DECIMAL (6,2)` für Währungsbeträge in Supermärkten geeignet.
- Datums- und Uhrzeitwerte
  SQL bietet verschiedene Datentypen für die Angabe von Kalenderdaten und Uhrzeiten:
  - `DATE` ist ein Datum im Format `"2015 – 04 – 11"`. Der zulässige Bereich ist `"0001 – 01 – 01"` bis `"9999 – 12 – 31"`.
  - `TIME` enthält eine Uhrzeitangabe im Format `"18:59:37"`.
  - `DATETIME` kombiniert eine Datums- und eine Uhrzeitangabe in der Schreibweise `"2015 – 04 – 11 18:59:37"`.
  - `YEAR` enthält eine Jahreszahl zwischen 1900 und 2155.
  - `TIMESTAMP` ist ein spezielles Feld, das beim Erstellen oder Ändern des zugehörigen Datensatzes automatisch ausgefüllt wird; es ist damit ideal für die nützliche Information, wann der Datensatz zuletzt geändert wurde. Das Format entspricht `DATETIME`.

- Textdatentypen
  Für Textinformationen existieren verschiedene Datentypen, die hier aufgelistet werden:
  - `CHAR(n)` ist eine Zeichenkette mit einer festen Länge von `n` Zeichen. Der angegebene Wert darf höchstens 255 sein. »Feste Länge« bedeutet, dass auf jeden Fall die angegebene Anzahl von Zeichen gespeichert wird, selbst wenn der eigentliche Text kürzer sein sollte. Dies erhöht die Verarbeitungsgeschwindigkeit, aber auch den Speicherbedarf.
  - `VARCHAR(n)` gibt eine Zeichenkette variabler Länge mit bis zu `n` (maximal 65.535) Zeichen an. Ein `VARCHAR`-Feld belegt nur so viel Speicher, wie es tatsächlich Zeichen enthält. Dafür werden `VARCHAR`-Felder langsamer gefunden als `CHAR`-Felder.
  - `TINYTEXT` ist ein Synonym für `VARCHAR (255)`.
  - `TEXT` gibt Text variabler Länge mit bis zu 65.535 Zeichen an.
  - `MEDIUMTEXT` darf maximal über 16,7 Millionen Zeichen enthalten.
  - `LONGTEXT` darf sogar über 4 Milliarden Zeichen enthalten.
- Binärdaten
  Zum sicheren Abspeichern von Binärdaten wie Bildern, Audiodaten und sonstigen proprietären Datenformaten wird von SQL das BLOB-Format (**B**inary **L**arge **Ob**ject) angeboten. Es gibt folgende Ausprägungen von BLOBs unterschiedlicher Größe:
  - `TINYBLOB` (bis zu 255 Byte)
  - `BLOB` (bis zu 65.535 Byte)
  - `MEDIUMBLOB` (über 16,7 Millionen Byte)
  - `LONGBLOB` (über 4 Milliarden Byte)
- Aufzählungstypen
  Mitunter ist es effektiver, eine Liste vorgefertigter Werte anzugeben, als ein frei ausfüllbares Textfeld einzurichten. MySQL definiert zu diesem Zweck die beiden folgenden Aufzählungstypen:
  - `ENUM` ist eine Aufzählung von maximal 65.535 verschiedenen Zeichenketten. Intern wird der Wert eines Feldes als Nummer des jeweiligen Aufzählungselements gespeichert.
  - `SET` enthält dagegen eine Aufzählung von maximal 255 verschiedenen Zeichenketten. Der Wert eines Feldes in einer solchen Spalte kann aus beliebig vielen kommagetrennten Werten aus der Aufzählung bestehen. Dazu besetzt jeder mögliche Wert ein eigenes Bit (1, 2, 4, 8, 16 etc.), und die Wertemischung in einem Feld ist die Summe dieser Bits.

  Beide Arten von Listen werden hinter dem Datentyp in Klammern und durch Komma getrennt angegeben. Das folgende Beispiel zeigt, wie es funktioniert:

```
steuerklasse ENUM ('I', 'II', 'III', 'IV', 'V', 'VI')
```

Hinter der Angabe des Datentyps können unter anderem folgende Optionen für Felder angegeben werden:

- `BINARY` ist eine Option, die Textdatentypen in Binärtypen umwandelt, in denen das Abspeichern binärer Daten unabhängig von Zeichensätzen und Zeilenumbruchlogik sicher möglich ist.
- `UNSIGNED` sorgt dafür, dass der Wertebereich eines ganzzahligen Typs ohne Vorzeichen betrachtet wird. Beispielsweise besitzt ein `TINYINT` dadurch nicht mehr den Wertebereich –128 bis +127, sondern 0 bis 255.
- `ZEROFILL` füllt alle Felder bis zur angegebenen Maximallänge nach links mit Nullen auf. Die Option impliziert automatisch `UNSIGNED`.
- `NULL` oder `NOT NULL` legen fest, ob ein Feld leer sein darf (`NULL`) oder nicht (`NOT NULL`). Der Standard ist `NULL`.
- `DEFAULT` gibt einen Standardwert für jedes Feld einer Spalte vor, das keinen sonstigen Wert besitzt.
- `AUTO_INCREMENT` richtet eine Spalte so ein, dass diese Spalte bei der Erzeugung neuer Zeilen automatisch fortlaufende Werte erhält. Dies ist beispielsweise für Primärschlüssel gut geeignet.
- `PRIMARY KEY` richtet ein Feld als Primärschlüssel ein, und zwar nur genau eines pro Tabelle. Für zusammengesetzte Primärschlüssel muss stattdessen die Option `PRIMARY KEY(Feld1, Feld2, ...)` außerhalb der Felddefinitionen genutzt werden.

Wie bereits erwähnt, werden Indizes außer dem Primärschlüssel erst nach dem Erstellen der Spalten eingerichtet. Neben dem Schlüsselwort `INDEX`, das einen einfachen Index einleitet, werden die alternativen Angaben `UNIQUE` (ein bestimmter Feldwert darf nur einmal in der Tabelle vorkommen) und `FULLTEXT` für die Volltextsuche unterstützt.

13.3.2 Auswahlabfragen

Um Daten aus einer Datenbank zu lesen, wird die SQL-Anweisung `SELECT` verwendet. Schematisch sieht ein solcher Aufruf folgendermaßen aus:

```
SELECT feld1, feld2, ...
FROM tabelle1, tabelle2, ...
WHERE kriterium;
```

Diese Abfrage wählt die Felder `feld1`, `feld2` etc. derjenigen Datensätze aus den Tabellen `tabelle1`, `tabelle2` und folgenden aus, auf die die Kriterien zutreffen.

Anstelle der einzelnen Felder können Sie auch * schreiben, um alle Felder einer Tabelle auszuwählen. Die folgende Abfrage zeigt beispielsweise die gesamte Tabelle *adressen* an:

```
SELECT * FROM adressen;
```

Benötigen Sie dagegen nur die Namen und die Postleitzahlen der Kunden, wird die folgende Schreibweise verwendet:

```
SELECT name, plz FROM adressen;
```

Wenn Sie nur ein Feld auswählen, ist manchmal der Modifikator `DISTINCT` nützlich: Er zeigt doppelt vorkommende Feldinhalte nur jeweils einmal an. Die folgende Abfrage zeigt jede unterschiedliche Postleitzahl aus *adressen* genau einmal an:

```
SELECT DISTINCT plz FROM adressen;
```

Wenn Sie Werte aus mehreren Tabellen auswählen (bevorzugt über die Verknüpfung durch Joins; siehe den nächsten Abschnitt), müssen Sie denjenigen Spaltenbezeichnungen den Tabellennamen voranstellen, die in mehreren Tabellen identisch vorkommen. Beispielsweise müssten Sie `adressen.nr` und `kaeufe.nr` schreiben, wenn beide in derselben Abfrage vorkämen.

Um dies zu umgehen, empfiehlt es sich in der Praxis, allen Feldnamen jeder Tabelle ein Kürzel für die jeweilige Tabellenbezeichnung voranzustellen. Beispielsweise könnten die Felder der Tabelle *adressen* mit *ad_* beginnen, also etwa *ad_nr*, *ad_name* oder *ad_plz*.

Häufiger werden Auswahlabfragen verwendet, bei denen über die `WHERE`-Klausel Bedingungen angegeben werden. Die Bedingungen vergleichen in der Regel die Werte einzelner Felder mit bestimmten Ausdrücken oder miteinander. Beispielsweise liefert die folgende Abfrage den Namen, die Postleitzahl und den Ort aller Kunden aus der Tabelle *adressen*, die in Köln wohnen:

```
SELECT name, plz, ort
FROM adressen
WHERE ort="Köln";
```

Die folgende Abfrage wählt dagegen die vollständigen Daten aller Kunden mit Postleitzahlen aus, die mit einer 5 beginnen:

```
SELECT *
FROM adressen
WHERE plz LIKE "5%"
```

Die `LIKE`-Klausel vergleicht den Wert eines Feldes mit einem einfachen Muster, in dem ein `%` für beliebig viele Zeichen und ein `_` für genau ein Zeichen steht. Hier sehen Sie einige Beispiele für solche Muster:

- `name LIKE "a%"` liefert alle Personen, deren Name mit `a` anfängt.
- `name LIKE "%b%"` gibt alle Personen zurück, in deren Namen mindestens ein b vorkommt.
- `strasse LIKE "%weg"` liefert alle Straßenangaben, die auf »-weg« enden.

- `name LIKE "Me_er"` steht für alle Kunden, die Meier oder Meyer heißen.
- `name LIKE "M%r"` gibt alle Leute zurück, die Maier, Mayer, Meier, Meyer oder Mayr heißen; natürlich werden auch Müller, Mecker, Monster etc. gefunden.

Für einfache Wertüberprüfungen, die keinen Mustervergleich verwenden, können Sie die Operatoren `=`, `<`, `>`, `<=`, `>=` und `<>` (ungleich) benutzen. Mehrere Überprüfungen können Sie mit `AND` oder `OR` verknüpfen.

Übrigens können Sie sowohl bei der `SELECT`-Anweisung selbst als auch bei der `WHERE`-Klausel beliebige Berechnungen ausführen. Bei `WHERE` müssen Sie allerdings darauf achten, dass das Endergebnis ein boolescher Wahrheitswert sein muss. Beispielsweise ist die Klausel `WHERE preis * 2` unvollständig und damit verboten; `WHERE preis * 2 < 10` ist dagegen zulässig und gibt alle Felder zurück, deren doppelter Preis kleiner als 10 ist.

Wenn Sie in der `SELECT`-Anweisung keine einzelnen Felder auswählen, sondern Berechnungen anstellen, können Sie der Ergebnisspalte über die `AS`-Klausel einen Namen zuweisen (wobei Sie das Schlüsselwort `AS` selbst auch weglassen können). Beispielsweise könnten Sie folgendermaßen die Nettopreise aller Waren in der Tabelle *artikel* ermitteln:

```
SELECT artname, preis / (100 + mwst) * 100
AS netto
FROM artikel;
```

Der entsprechende Mehrwertsteuersatz wird also zu 100 addiert; das Teilen des Preises durch diesen Gesamtwert und die Multiplikation mit 100 ergibt natürlich den Nettopreis. Die Spalte in der Ergebnistabelle der Abfrage wird als *NETTO* bezeichnet, was erheblich lesefreundlicher ist als »`PREIS / (100 + MWST) * 100`«.

Neben den einfachen arithmetischen Berechnungen bietet SQL auch eine Reihe von Funktionen an. Die wichtigsten von ihnen werden als *Aggregatfunktionen* bezeichnet, da sie die Anzahl der zurückgegebenen Datensätze verkleinern können, indem sie mehrere zusammenfassen. Hier einige Aggregatfunktionen im Überblick:

- `SUM` gibt die Summe der Werte in der Spalte zurück, auf die `SUM` angewendet wird. Wenn Sie weitere Spalten in das `SELECT` aufnehmen, erhalten Sie so viele Einzelergebnisse, wie es unterschiedliche Wertepaare in diesen Spalten gibt. Beispielsweise ergibt die folgende Abfrage die Summe aller Artikelpreise in der Tabelle `artikel`:

  ```
  SELECT SUM(preis) AS summe FROM artikel;
  ```

  Diese Abfrage liefert dagegen die Summen der beiden Artikelgruppen mit unterschiedlicher Mehrwertsteuer getrennt:

  ```
  SELECT SUM(preis) AS summe, mwst FROM ARTIKEL;
  ```

  Sinnvollere Beispiele erfordern die Kombination mehrerer Tabellen; Sie finden sie im Abschnitt »Joins«.

- `MIN` gibt den kleinsten Wert eines Feldes innerhalb einer Gruppe zurück.
- `MAX` liefert entsprechend das Feld mit dem höchsten Wert.
- `COUNT` schließlich gibt die Anzahl der Felder einer Spalte oder Gruppe zurück. Beispielsweise gibt die folgende Abfrage die Anzahl aller Kunden zurück:

  ```
  SELECT COUNT(*) AS kundenzahl FROM adressen;
  ```

  Wenn Sie die Anzahlen der Artikel mit den beiden unterschiedlichen Mehrwertsteuersätzen getrennt voneinander erhalten möchten, funktioniert dies folgendermaßen:

  ```
  SELECT COUNT(artnr) AS anzahl, mwst FROM artikel GROUP BY mwst
  ```

Wichtig ist noch, wie Sie die Ergebnisdatensätze in einer Auswahlabfrage sortieren können. Dies funktioniert mithilfe der `ORDER BY`-Klausel. Anzugeben ist dabei die Spalte, nach deren Werten sortiert werden soll, sowie `ASC` (*ascending*) für aufsteigende Reihenfolge und `DESC` (*descending*) für absteigende Reihenfolge. Das folgende Beispiel zeigt, wie Sie die Kunden nach ihren Namen alphabetisch sortieren:

```
SELECT * FROM adressen
ORDER BY name ASC;
```

Joins

Für die praktische Anwendung von Beziehungen ist es wichtig, das Konzept der Joins zu verstehen. Der Beziehungstyp, der bei einer 1:n-Beziehung zwischen dem Primärschlüssel der einen und einem Fremdschlüssel in einer anderen Tabelle besteht, wird als *Inner Join* bezeichnet, wenn nur diejenigen Ergebnisse gewünscht werden, die aus Datensätzen beider Tabellen stammen.

Um beispielsweise die Namen aller Kunden auszugeben, die überhaupt etwas gekauft haben, wird folgende Syntax verwendet:

```
SELECT name FROM adressen
INNER JOIN  kaeufe ON adressen.adressnr = kaeufe.adressnr;
```

Dies gibt eine Liste der Kundennamen in der Reihenfolge aus, in der die Kunden in der Tabelle *kaeufe* über das Feld *nr* referenziert werden. Eine alternative Schreibweise, die auch mit älteren und seltener verwendeten Datenbanksystemen kompatibel ist, verwendet eine `WHERE`-Klausel anstelle der `INNER JOIN`-Angabe. Die zuvor formulierte Abfrage lässt sich also auch folgendermaßen formulieren:

```
SELECT name FROM adressen
WHERE adressen.adressnr = kaeufe.adressnr;
```


Auf dieselbe Art und Weise erhalten Sie auch den Gesamtpreis jedes einzelnen Kaufs:

```
SELECT preis * stueck FROM artikel, kaeufe
WHERE artikel.artnr = kaeufe.artnr;
```

Auch Aggregatfunktionen lassen sich mit Joins kombinieren. Das folgende Beispiel gibt den Gesamtumsatz jedes einzelnen Tages aus:

```
SELECT SUM(preis * stueck) AS tagesumsatz, datum
FROM artikel.kaeufe GROUP BY datum
WHERE artikel.artnr = kaeufe.artnr;
```

Das folgende, etwas komplexere Beispiel stellt jeden Kauf jedes Kunden mit allen interessanten Zusatzdaten dar:

```
SELECT kaufnr, name, artname, stueck,
       stueck * preis AS gesamtpreis
  FROM adressen, artikel, kaeufe
 WHERE adresssen.adressnr = kaeufe.adressnr
   AND artikel.artnr = kaeufe.artnr;
```

Das Ergebnis dieser Abfrage können Sie sich in Tabelle 13.5 ansehen.

Neben dem Inner Join existieren auch Left (Outer) Join und Right (Outer) Join. Der Unterschied besteht darin, dass das Ergebnis auf jeden Fall alle infrage kommenden Datensätze der links beziehungsweise rechts von der Join-Klausel stehenden Tabelle enthält und für die Felder der jeweils anderen Tabelle gegebenenfalls `NULL`.

13.3.3 Einfüge-, Lösch- und Änderungsabfragen

Mithilfe von `INSERT` werden Daten in eine Datenbanktabelle eingefügt. Die Syntax lautet folgendermaßen:

```
INSERT INTO tabelle (spalte1, spalte2, ...)
VALUES (WERT1, WERT2, ...);
```

Beachten Sie, dass Sie mindestens alle Spalten nennen müssen, die die Bedingung `NOT NULL` aufweisen. Die folgende Anweisung fügt beispielsweise einen neuen Artikel hinzu:

```
INSERT INTO artikel (artname, preis, mwst)
     VALUES ("Gurke", 39, "7");
```

Falls alle Spalten der Tabelle einen Wert erhalten sollen, funktioniert auch die folgende Kurzfassung:

```
INSERT INTO tabelle
VALUES (wert1, wert2, ...);
```

Sie können sogar das aktuelle Ergebnis einer SELECT-Abfrage permanent in einer neuen Tabelle ablegen. Das folgende Beispiel speichert jede einzelne Postleitzahl aufsteigend sortiert in der neuen einspaltigen Tabelle *plzs*:

```
CREATE TABLE plzs (
   plz INT
);
INSERT INTO plzs
SELECT DISTINCT plz FROM adressen
ORDER BY plz ASC;
```

Um Datensätze aus einer Tabelle zu löschen, wird die Anweisung DELETE verwendet. Welche Datensätze Sie entfernen möchten, können Sie wie bei einer Auswahlabfrage über eine WHERE-Klausel angeben:

```
DELETE FROM tabelle
 WHERE kriterium;
```

Wenn Sie beispielsweise alle Kunden aus der Tabelle *adressen* löschen möchten, die nicht im PLZ-Gebiet 5 wohnen, funktioniert dies folgendermaßen:

```
DELETE FROM adressen
 WHERE plz NOT LIKE "5%"
```

Wenn Sie die Werte von Feldern ändern möchten, geschieht dies durch die Anweisung UPDATE. Wichtig ist auch hier die WHERE-Klausel, damit Sie dem gewünschten Feld nicht einfach in allen Datensätzen einen neuen Wert zuweisen. Angenommen, der Kunde Schmidt ist aus dem Kleinen Weg 1 in die Große Allee 25 gezogen. Eine entsprechende Änderungsabfrage sieht folgendermaßen aus:

```
UPDATE adressen
   SET strasse="Große Allee", hausnr="25"
 WHERE adressnr=1;
```

Selbstverständlich müssen Sie auf den Kunden über den Primärschlüssel (hier die Kundennummer) zugreifen, da der Name doppelt vorkommen könnte.

Die folgende Abfrage zeigt dagegen ein Beispiel, in dem bewusst mehrere Datensätze geändert werden: Alle Artikel mit 19 % Mehrwertsteuer werden für eine Sonderaktion um 20 % billiger:

```
UPDATE artikel
SET preis = 0.8 * preis
WHERE mwst = "19";
```


13.3.4 Transaktionen

Moderne relationale Datenbanksysteme bieten eine interessante Erweiterung normaler SQL-Abfragen: Transaktionen ermöglichen es, beliebig viele Einzelschritte zusammenzufassen und am Ende zu bestätigen (Commit) oder rückgängig zu machen (Rollback). Eine vollwertige Implementierung von Datenbanktransaktionen genügt einem Standard namens *ACID*, bestehend aus den folgenden vier Komponenten:

- *Atomicity* – die Transaktion verbindet alle enthaltenen MySQL-Anweisungen zu einer atomaren Einheit, die nach außen hin einer einzelnen Anweisung entspricht.
- *Consistency* – wenn die Transaktion durch Commit oder Rollback abgeschlossen wird, muss die Datenbank in einem konsistenten Zustand verbleiben.
- *Isolation* – jede Transaktion muss gegenüber allen anderen Datenbankoperationen und Transaktionen isoliert ausgeführt werden; während sie abläuft, bemerken andere Operationen nichts von ihren Änderungen und umgekehrt.
- *Durability* – nach einem Commit müssen die Änderungen durch die Transaktion dauerhaft in der Datenbank gespeichert bleiben.

In MySQL werden Transaktionen bisher nur durch den speziellen Tabellentyp InnoDB unterstützt. InnoDB gehört schon seit Jahren zum Oracle-Konzern, der 2009 auch den MySQL-Besitzer Sun Microsystems aufgekauft hat. Damit gehören MySQL und InnoDB seit einigen Jahren derselben Firma.

Um eine InnoDB-Tabelle zu erzeugen, müssen Sie eine CREATE TABLE-Abfrage wie folgt ergänzen:

```
CREATE TABLE tabellenname (
  ...
) ENGINE=InnoDB
```

In MySQL für Windows ist InnoDB der Standardtabellentyp. Unter Unix wird dagegen automatisch die MySQL-eigene Engine MyISAM gewählt. Da diese performanter ist als InnoDB, lohnt es sich, Engine=MyISAM explizit anzugeben, wann immer Sie keine Transaktionen oder andere InnoDB-spezifischen Features benötigen.

Die Durchführung von Transaktionen ist sehr einfach. Geben Sie zunächst

```
START TRANSACTION;
```

ein, um eine neue Transaktion zu beginnen. Wenn Sie alle zur Transaktion gehörenden Anweisungen ausgeführt haben, können Sie entweder

```
COMMIT;
```

eingeben, um die Änderungen endgültig zu bestätigen, oder aber

```
ROLLBACK;
```

falls Sie alle Modifikationen auf den Ursprungszustand zurücksetzen wollen. Mit der Anweisung

```
SAVEPOINT name;
```

können Sie übrigens einen benannten Zwischenspeicherstand der Transaktion erstellen, zu dem Sie jederzeit mithilfe von

```
ROLLBACK TO name;
```

zurückgehen können.

Wenn Sie Transaktionen in Ruhe ausprobieren möchten, öffnen Sie einfach zwei `mysql`-Client-Fenster, und starten Sie in einem der Fenster eine Transaktion. Wie Sie feststellen werden, können Sie die Veränderungen aus der Transaktion in dem anderen Fenster bis zum `COMMIT` nicht sehen.

13.4 MySQL-Administration

Ebenso wichtig wie die Pflege der im RDBMS gespeicherten Daten ist die Administration des Datenbankservers selbst. Dazu gehören unter anderem Benutzerverwaltung, Datensicherung und Konfiguration. In diesem Abschnitt erhalten Sie eine Einführung in die Aufgaben der Datenbankadministration, wieder am Beispiel von MySQL.

Für die MySQL-Administration stehen im Wesentlichen drei Arten von Werkzeugen zur Verfügung:

- mit dem Server gelieferte Konsolen-Tools wie *mysql*, *mysqladmin* und *mysqldump*
- MySQL Administrator, ein grafisches Administrationstool der MySQL-Entwickler. Sie können es von der MySQL-Website herunterladen. Es wird hier nicht beschrieben, da es weitgehend intuitiv und selbsterklärend ist – erst recht, wenn Sie den folgenden Informationen zur manuellen Administration folgen.
- grafische Clients, die nicht auf die Administration spezialisiert sind (zum Beispiel phpMyAdmin)

13.4.1 mysqladmin

Einige Verwaltungsaufgaben lassen sich mithilfe des Kommandozeilen-Tools *mysqladmin* erledigen. Genau wie den Konsolen-Client *mysql* müssen Sie es mit `-u Benutzername` (in der Regel `root`) und `-p` für die Passwortanforderung aufrufen, sodass seine Syntax so aussieht:

```
mysqladmin Befehl -u Benutzername -p
```


Die wichtigsten Befehle, die Sie eingeben können, sind:

- `create Datenbank` – Erzeugt die angegebene Datenbank wie die SQL-Anweisung `CREATE DATABASE`.
- `drop Datenbank` – Löscht die gewünschte Datenbank.
- `extended-status` – erweiterte Statusinformationen. Da die Liste sehr lang ist, sollten Sie sie durch `|less` (Unix) oder `|more` (Windows) filtern.
- `ping` – Überprüft, ob der MySQL-Server läuft.
- `reload` – Lädt die Benutzerinformationen neu (siehe nächsten Abschnitt).
- `shutdown` – Beendet den MySQL-Server.
- `version` – Gibt die Version des MySQL-Servers aus.

Hier ein Beispiel, das den Server beendet:

```
$ mysqladmin shutdown -u root -p
```


13.4.2 Benutzerverwaltung

Die Benutzeridentifikation in MySQL erfolgt anhand der drei Komponenten Host, Benutzername und Passwort. Die entsprechenden Daten werden in der Verwaltungsdatenbank *mysql* gespeichert. Hier dienen insbesondere folgende Tabellen der Verwaltung von Benutzerrechten:

- *user* – Enthält Benutzernamen und verschlüsselte Passwörter zur Überprüfung der Anmeldung sowie globale Benutzerrechte.
- *host* – hostbasierte Berechtigungen
- *db* – Berechtigungen an einzelnen Datenbanken
- *tables_priv* – Rechte an einzelnen Tabellen
- *columns_priv* – Benutzerrechte an einzelnen Tabellenspalten

Sie können sich die Inhalte und Strukturen dieser Tabellen mithilfe der zuvor vorgestellten Abfragen in Ruhe anschauen. Wie Sie sehen, besitzt in *user*, *host* und *db* jedes Recht eine eigene `ENUM`-Spalte mit den möglichen Werten `'Y'` (Recht gewährt) oder `'N'` (Recht verweigert – dies ist aus naheliegenden Gründen der Standardwert). In *tables_priv* und *columns_priv* gibt es dagegen für alle Rechte je eine einzelne `SET`-Spalte; die darin für den einzelnen User aufgelisteten Rechte werden gewährt, alle anderen verweigert.

Die Überprüfung jedes Datenbankzugriffs erfolgt in zwei Stufen:

1. Prüfung, ob der fragliche Benutzer sich vom entsprechenden Host aus anmelden darf (Tabellen *user* und *host*). Scheitert dies aus einem der möglichen Gründe (unbekannter Benutzer, unberechtigter Host, falsches Passwort), bricht der Anmeldeversuch mit einer Fehlermeldung ab. Beispiel:

```
$ mysql -u wronguser -p
ERROR 1045 (28000): Access denied for 'wronguser'@'localhost' (using
password: YES)
```

2. War die grundlegende Legitimationsprüfung erfolgreich, wird als Nächstes getestet, ob der nunmehr angemeldete Benutzer die gewünschte Operation vornehmen darf. Dies geht schrittweise von oben nach unten: Hat er das entsprechende Recht global (in der Tabelle *user*), dann ist bereits alles klar. Wenn nicht, wird in der Tabelle *db* überprüft, ob der Benutzer die entsprechende Berechtigung für die gesamte aktuelle Datenbank besitzt. Ist auch dies nicht der Fall, dann geht es mit der Tabelle und schließlich mit den einzelnen Spalten weiter. Erst wenn alle diese Prüfungen versagen sollten, erhalten Sie eine Fehlermeldung wie diese:

```
mysql> SHOW TABLES FROM test;
ERROR 1044 (42000): Access denied for user 'darfnix'@'localhost' to
database 'test'
```

Es ist sehr wichtig, dieses Verfahren richtig zu verstehen: Wenn ein Recht auf einer übergeordneten Ebene besteht, ist die Berechtigungsüberprüfung unwiderruflich abgeschlossen. Es gibt also keine Möglichkeit, einem Benutzer zuerst eine allgemeine Erlaubnis zu erteilen und diese dann im Einzelnen wieder einzuschränken. Es ist also umso unerlässlicher, hier »geizig« zu sein und Benutzern stets nur die unbedingt notwendigen Berechtigungen zu erteilen.

Um einen neuen Benutzer zu erzeugen, starten Sie zunächst den Kommandozeilen-Client als User *root*:

```
$ mysql -u root -p
```

Wie es nun weitergeht, hängt von der MySQL-Version ab. Seit Version 5.0 verwendet MySQL die Anweisung `CREATE USER`, um ein Benutzerkonto anzulegen. Die allgemeine Syntax lautet:

```
CREATE USER username@hostname IDENTIFIED BY 'Passwort'
```

Hier ein konkretes Beispiel:

```
mysql> CREATE USER someuser@localhost
    -> IDENTIFIED BY 'AnyPa55';
```

Diese Anweisung erzeugt einen neuen Datensatz in der Verwaltungstabelle *mysql.user*, in der User- und Hostname, verschlüsseltes Passwort und keinerlei Rechte (alles auf `'N'`) eingetragen sind:

```
mysql> SELECT * from mysql.user WHERE user='someuser'\G
*************************** 1. row ***************************
      Host: localhost
      User: someuser
  Password: *426A9DFF6005EE6609101D651C4E70F51F52E12B
Select_priv: N
Insert_priv: N
Update_priv: N
[...]
```


Um ein Passwort nachträglich zu ändern, wird die Anweisung `SET PASSWORD` verwendet. Dabei wird für das eigentliche Passwort die Verschlüsselungsfunktion `PASSWORD()` aufgerufen. Hier ein Beispiel:

```
SET PASSWORD FOR user@localhost = PASSWORD('geheim');
```

Um bestehenden Benutzern neue Rechte zuzuweisen, wird in jedem Fall die Anweisung `GRANT` verwendet, deren allgemeine Syntax so aussieht:

```
GRANT Recht [(Spalte)][, Recht [(Spalte)] ...]
ON Datenbank.Tabelle
TO username@hostname
```

Die wichtigsten Rechte, die Sie Benutzern erteilen können, sind in Tabelle 13.8 aufgelistet. Eine Liste aller möglichen Privilegien erhalten Sie übrigens mit folgender MySQL-Anweisung:

```
mysql> SHOW PRIVILEGES;
```

MySQL-Benutzerrecht	Bedeutung
`USAGE`	nur Anmeldung; keine Rechte
`SELECT`	Auswahlabfragen
`INSERT`	Einfügeabfragen
`UPDATE`	Datenänderungsabfragen
`DELETE`	Datenlöschabfragen
`CREATE`	Tabellenerstellungsabfragen
`DROP`	Tabellenlöschabfragen
`ALTER`	Strukturänderungsabfragen
`INDEX`	Indexverwaltung mit `CREATE`/`DROP INDEX`
`CREATE VIEW`	Erstellen von Views

Tabelle 13.8 Benutzerrechte für GRANT- und REVOKE-Anweisungen

MySQL-Benutzerrecht	Bedeutung
FILE	Import/Export mit Textdateien
SHOW DATABASES	Anzeigen der Datenbankliste
SHOW VIEW	Anzeigen einer View-Definition (SHOW CREATE VIEW)
CREATE ROUTINE	Erstellen von Stored Procedures/Functions
ALTER ROUTINE	Ändern von Stored Procedures/Functions
EXECUTE	Ausführen von Stored Procedures/Functions
CREATE USER	Benutzer erstellen
SHUTDOWN	MySQL-Server beenden
REPLICATION CLIENT	Replikationseinstellungen ermitteln
REPLICATION SLAVE	als Replikations-Slave fungieren
SUPER	Prozessverwaltung (CHANGE MASTER, KILL etc.)
ALL [PRIVILEGES]	alle Rechte außer GRANT
GRANT OPTION	Rechteverwaltung mit GRANT/REVOKE – wird über die Zusatzoption WITH GRANT OPTION vergeben

Tabelle 13.8 Benutzerrechte für GRANT- und REVOKE-Anweisungen (Forts.)

Das folgende Beispiel erteilt dem zuvor erstellten Benutzer *someuser@localhost* das Recht, Daten aus beliebigen Datenbanken und Tabellen auszuwählen:

```
mysql> GRANT SELECT ON *.* TO someuser@localhost;
```

Zusätzlich soll er die Berechtigung erhalten, Datensätze in beliebige Tabellen der Datenbank *supermarkt* einzufügen:

```
mysql> GRANT INSERT ON supermarkt.*
    -> TO someuser@localhost;
```

In der Tabelle *artikel* soll er auch Datensätze löschen dürfen:

```
mysql> GRANT DELETE ON supermarkt.artikel
    -> TO someuser@localhost;
```

Schließlich soll er auch noch das Recht haben, den Inhalt der Spalte *artname* in der Tabelle *artikel* (den Artikelnamen) zu ändern:


```
mysql> GRANT UPDATE (artname) ON supermarkt.artikel
    -> TO someuser@localhost;
```

Angenommen, dieser Benutzer meldet sich an und führt folgende Abfrage durch:

```
mysql> UPDATE artikel SET artname="PC"
    -> WHERE artname="Computer";
```

Dann überprüft der MySQL-Server nacheinander folgende Werte:

- Besitzt der User *someuser@localhost* das globale Recht zum Ändern (Update_priv) in der Tabelle *user*? – Nein.
- Besitzt er das allgemeine Änderungsrecht (Update_priv) für die Datenbank *supermarkt* in der Tabelle *db*? – Nein.
- Besitzt er das Recht, beliebige Spalten in der Tabelle *artikel* zu ändern (Wert Update in der Spalte *Table_priv* der Tabelle *Tables_priv*)? – Nein.
- Besitzt er die Berechtigung, die Spalte *artname* in der Tabelle *artikel* zu ändern (Wert Update in der Spalte *Column_priv* – sowohl in der Tabelle *Tables_priv* als auch in *Columns_priv*; in Letzterer mit dem Spaltennamen)? – Ja.

Um einem User dieselben Privilegien wie *root* zu erteilen (wovon Sie in der Praxis in der Regel absehen sollten), genügt die folgende Anweisung übrigens nicht:

```
mysql> GRANT ALL PRIVILEGES ON *.*
    -> TO verwalter@localhost;
```

Dieser Benutzer besitzt nämlich nicht das Recht, seinerseits mithilfe von GRANT Benutzerrechte zu erteilen. Soll dies der Fall sein, ist folgende Variante erforderlich:

```
mysql> GRANT ALL PRIVILEGES ON *.*
    -> TO verwalter@localhost WITH GRANT OPTION;
```

Um einem Benutzer ein zuvor erteiltes Recht wieder zu entziehen, wird eine REVOKE-Anweisung verwendet. Ihre allgemeine Syntax lautet:

```
REVOKE Recht [(Spalte)][, Recht [(Spalte)] ...]
ON datenbank.tabelle FROM username@hostname
```

Das folgende Beispiel entzieht dem Benutzer *someuser@localhost* das zuvor erteilte Recht, in alle Tabellen der Datenbank *supermarkt* Daten einzufügen:

```
mysql> REVOKE INSERT ON supermarkt.*
    -> FROM someuser@localhost;
```

Möchten Sie dagegen einen Benutzer mit allen seinen Berechtigungen entfernen, können Sie in MySQL 5 folgende Anweisung verwenden:

```
mysql> DROP USER ex_user@localhost;
```

Falls Sie den Client nach dem Ändern von Benutzerrechten nicht beenden (oder zur Sicherheit sogar auch dann), sollten Sie die folgende Anweisung ausführen:

```
mysql> FLUSH PRIVILEGES;
```

Dies lädt die Benutzerinformationen neu; alternativ funktioniert auch folgende Konsolenanweisung:

```
$ mysqladmin reload -u root -p
```

13.4.3 Import und Export von Daten, Backups

Für einfache, manuelle Backups von MySQL-Datenbanken und -Tabellen kann das Konsolenprogramm *mysqldump* verwendet werden. Es erzeugt SQL-Dumps der entsprechenden Strukturen und Daten, also Dateien mit SQL-Anweisungen, die die entsprechenden Tabellen erstellen und die Daten einfügen.

Die Syntax dieser Anweisung ist etwas unterschiedlich, je nachdem, ob Sie einzelne Tabellen, einzelne Datenbanken oder alle Datenbanken Ihres Servers exportieren möchten. Für einzelne Tabellen lautet die Syntax:

```
mysqldump [Optionen] Datenbank [Tabelle ...]
```

Wenn Sie keine Tabellenbezeichnungen angeben, wird die gesamte Datenbank exportiert.

Falls Sie mehrere Datenbanken exportieren möchten, müssen Sie folgende Schreibweise verwenden:

```
mysqldump [Optionen] --databases Datenbank
          [Datenbank ...]
```

Für einen Export des gesamten Datenbestands des Servers gilt schließlich diese Syntax:

```
mysqldump [Optionen] --all-databases
```

Wie alle MySQL-Konsolenhilfsprogramme benötigt auch *mysqldump* Benutzername und Passwort, sodass Sie unter den Optionen auf jeden Fall `-u Benutzername` (hier meist `root`) und `-p` (Passworteingabeaufforderung) angeben müssen.

Angenommen, Sie möchten die Datenbank *supermarkt* exportieren. Dazu ist folgende Eingabe erforderlich:

```
$ mysqldump -u root -p supermarkt
```


Wie Sie feststellen werden, erfolgt die Ausgabe auf `STDOUT`, sodass Sie sie in eine Datei umleiten müssen, um etwas Sinnvolles damit anfangen zu können:

```
$ mysqldump -u root -p supermarkt >supermarkt.sql
```

Die Dateiendung *.sql* ist keine Bedingung, bietet sich aber an, da die entstehende Datei eben SQL-Anweisungen enthält.

Wenn Sie »Live-Datenbanken« betreiben, die in frequentierten Netzwerk- oder Webanwendungen eingesetzt werden, genügt der einfache Aufruf von `mysqldump` nicht mehr. Das Problem ist, dass sich noch während des Backup-Vorgangs Daten ändern können, was zu Inkonsistenzen führt. Hier müssen Sie dafür sorgen, dass der Backup-Vorgang vor Änderungen geschützt abläuft. Wie das im Einzelnen funktioniert, hängt davon ab, welche Storage Engine die Tabelle der entsprechenden Datenbank verwendet.

InnoDB-Tabellen sind transaktionsfähig und können daher innerhalb einer Transaktion exportiert werden, die komplett vor eventuellen gleichzeitigen Änderungen geschützt abläuft. Dazu wird die Option `--single-transaction` verwendet. Insgesamt hat ein solcher Aufruf folgende Syntax:

```
mysqldump -u root -p --single-transaction \
Datenbank >Datei
```

Ein korrektes Backup von MyISAM-Tabellen bei einem frequentierten Server benötigt dagegen zusätzliche Anweisungen, um während des Dump-Vorgangs Sperren (Locks) auf die entsprechenden Tabellen zu setzen:

```
mysql -e "flush tables with read lock" -u root -p
mysqldump -u root -p DATENBANK >DATEI
mysql -e "unlock tables" -u root -p
```

Die `mysql`-Option `-e` startet den `mysql`-Client nicht interaktiv, sondern führt die angegebene Anweisung aus. Die MySQL-Anweisung `FLUSH TABLES WITH READ LOCK` schreibt alle zurzeit »schwebenden« Änderungen ordnungsgemäß auf die Festplatte und sperrt dann alle Tabellen sämtlicher Datenbanken vollständig. `UNLOCK TABLES` setzt die Sperren wieder zurück.

Leider müssen Sie bei dieser Anweisungsfolge dreimal das *root*-Passwort des MySQL-Servers eingeben. Daher lässt es sich in dieser Form auch nicht per Cronjob automatisieren (siehe Kapitel 7, »Linux«). Zwar können Sie das Passwort notfalls im Klartext ohne Abstand hinter die Option `-p` schreiben – aber das sollten Sie auf keinen Fall mit dem *root*-Passwort machen! Dafür empfiehlt es sich eher, einen speziellen Backup-Benutzer mit eingeschränkten Rechten zu erstellen. Er benötigt die globalen Rechte `RELOAD` (für `FLUSH TABLES`), `LOCK TABLES` (für das Sperren und Entsperren) sowie `SELECT` für das Auslesen der Daten zum eigentlichen Backup:

```
mysql> CREATE USER backupuser@localhost
    -> IDENTIFIED BY '84ckUp2';
```

```
mysql> GRANT RELOAD, LOCK TABLES, SELECT ON *.*
    -> TO backupuser@localhost;
```

Anschließend können Sie die drei Anweisungen für MyISAM-Backups wie folgt als Shell-Skript speichern:

```
mysql -e "flush tables with read lock" \
-u backupuser -p84ckUp2
mysqldump -u backupuser -p84ckUp2 Datenbank >Datei
mysql -e "unlock tables" -u backupuser -p84ckUp2
```

Alternativ können Sie `mysqldump` auch gleich mit der zusätzlichen Option `-x` (Langform: `-lock-all-tables`) aufrufen, um alle Tabellen während des gesamten Backup-Vorgangs zu sperren. Dies sollte allerdings eher in Stunden mit relativ geringfügigem Zugriff geschehen.

Um eines der zuvor angelegten Backups wieder zurückzuspielen, gibt es zwei Möglichkeiten. Die erste wird auf der Konsole angewendet:

```
$ mysql -u root -p <Datei
```

Innerhalb des `mysql`-Clients können Sie dagegen die bereits besprochene Syntax

```
mysql> source Datei
```

oder kurz

```
mysql> \. Datei
```

verwenden.

Eine etwas andere Möglichkeit bietet das SQL-Anweisungspaar `SELECT ... INTO OUTFILE` und `LOAD DATA INFILE`: Hier werden die Daten in sogenannte *CSV-Textdateien* (*Comma-separated Values*) exportiert beziehungsweise aus diesen importiert. CSV ist ein beliebtes Datenaustauschformat für Tabellenkalkulationsprogramme wie Excel oder Open-Office.org Calc.

Um Daten in eine CSV-Datei zu exportieren, hängen Sie an eine beliebige `SELECT`-Abfrage `INTO OUTFILE Dateiname` an. Beispiel:

```
mysql> SELECT * FROM artikel INTO OUTFILE
    -> "supermarkt.csv";
```

Der Import aus einer entsprechenden Datei erfolgt dagegen so:

```
mysql> LOAD DATA INFILE "supermarkt.csv"
    -> INTO TABLE artikel;
```

Über diverse Parameter, die hier allerdings zu weit führen würden, können Sie jeweils den genauen Aufbau der CSV-Dateien bestimmen.

13.4.4 Konfigurationsdateien

Wie bereits erwähnt, kommt MySQL im Standardbetrieb recht gut ohne Konfigurationsdateien aus. Sollten Sie dennoch irgendwelche Aspekte seines Verhaltens ändern wollen, können Sie eine der folgenden Dateien anlegen:

- */etc/my.cnf* – globale Konfigurationsdatei
- *~/.my.cnf* – benutzerspezifische Konfigurationsdatei

Der Aufbau dieser Dateien entspricht den bekannten Windows-INI-Dateien (ähnlich wie *smb.conf* oder *php.ini*, siehe Kapitel 7, »Linux«, beziehungsweise Kapitel 14, »Server für Webanwendungen«): Abschnitte in eckigen Klammern kennzeichnen die Themen (in diesem Fall die einzelnen MySQL-Programme); die einzelnen Konfigurationsanweisungen werden im Format `Parameter = Wert` angegeben.

Der wichtigste Abschnitt ist `[mysqld]` mit Parametern für den Server selbst. Aber auch alle anderen mit MySQL gelieferten Programme lesen die für sie geschriebenen Abschnitte in diesen Dateien aus, zum Beispiel `[mysqladmin]`, `[mysqldump]` oder der Client `[mysql]`.

Hier einige wenige Parameter für den Server im Überblick:

- `port` – der TCP-Port, an dem der Server lauscht (Standard 3306; kann geändert werden, um mehrere MySQL-Server auf einem Host zu betreiben)
- `socket` – das Unix-Domain-Socket (Dateipfad) für die lokale Kommunikation (Standard: `/tmp/mysql.sock`)
- `character-set-server` – der Zeichensatz des Servers selbst
- `collation-server` – die Sortierfolge des Servers
- `language` – die Sprache für Fehlermeldungen und Warnungen (Sie können beispielsweise auf `german` umschalten)
- `sql-mode` – der SQL-Kompatibilitätsmodus; zum Beispiel `mysql` (Standard) oder `ansi` (ändert etwa die zuvor diskutierte Bedeutung der verschiedenen Anführungszeichen)

In den Konfigurationsdateien können Sie jeden Wert ändern, dessen aktuelle Einstellung die folgende MySQL-Anweisung zeigt:

```
mysql> SHOW VARIABLES;
```

MySQL wird mit einigen Vorlagen für mögliche Konfigurationsdateien geliefert, von *my-small.cnf* (optimiert für sehr kleine Datenbanken) bis hin zu *my-huge.cnf* für gigantische Datenbanken. Sie können bei Bedarf eine dieser Dateien nach */etc/my.cnf* kopieren und sie noch etwas an Ihre Bedürfnisse anpassen.

13.4.5 Log-Dateien

MySQL ist in der Lage, verschiedene Log-Dateien anzulegen, die eine wichtige Basis zur Wiederherstellung verlorener Daten, für die Replikation (siehe nächsten Abschnitt) oder zur Fehlersuche bilden.

Standardmäßig wird nur eine Error-Log-Datei angelegt. Sie befindet sich – wie alle Log-Dateien – im MySQL-Datenverzeichnis (weil das *mysql*-Systembenutzerkonto nur dort Schreibrechte besitzt) und trägt den aktuellen Hostnamen und die Endung *.log* (Beispiel: *tux.log*). Mit folgendem Eintrag unter dem Abschnitt `[mysqld]` in der Datei *my.cnf* können Sie ihren Ort ändern:

```
log-error = Pfad
```

Eine weitere sehr wichtige Log-Datei ist das binäre Update-Log. Es protokolliert alle Änderungsabfragen und dient somit als Basis für die Replikation oder auch für die Wiederherstellung seit dem letzten richtigen Backup.

Der entsprechende Eintrag in */etc/my.cnf* lautet:

```
[mysqld]
...
log-bin = mylog
```

Dies legt im MySQL-Datenverzeichnis die Dateien *mylog.000001* und *mylog.index* an (anstelle von *mylog* können Sie auch einen beliebigen anderen Namen eingeben). Bei jedem Neustart des Servers, bei explizitem Flush oder wenn die Datei eine bestimmte Größe überschreitet, wird die nächste Datei (*mylog.000002* etc.) angelegt.

Um geplant die nächste Log-Datei zu beginnen, können Sie im `mysql`-Client Folgendes eingeben:

```
mysql> FLUSH LOGS;
```

Eine Konsolenalternative ist:

```
$ mysqladmin flush-logs -u root -p
```

Wenn Sie eine automatische Log-Rotation per Cronjob (siehe Kapitel 7, »Linux«) durchführen möchten, legen Sie zunächst einen speziellen MySQL-User an, der nur das globale Recht `RELOAD` besitzt. Danach können Sie ein Skript mit folgender Zeile erstellen:

```
mysqladmin -u reloaduser flush-logs -p Reloadpasswort
```

Da die Update-Log-Datei binär ist, können Sie sie nicht in einem Texteditor lesen. Wenn Sie sie im Klartext einsehen möchten, müssen Sie das mit MySQL gelieferte Tool *mysqlbinlog* verwenden. Beispiel:

```
$ mysqlbinlog -u root -p mylog.000001
```


Es gibt noch einige andere mögliche Logs, die Sie durch folgende Einträge in */etc/my.cnf* erzeugen können (die Bedeutung wird jeweils durch #-Kommentare beschrieben):

```
[mysqld]
...
# Zu langsame Abfragen unter HOSTNAME-slow.log
# protokollieren:
log-slow-queries
# Dazu muss definiert werden, wie lange ZU LANGE ist
# (variiert stark je nach Datenbankgröße):
long_query_time = SEKUNDEN
# Ganz allgemein Abfragen protokollieren, die
# keinen Index verwenden (und daher oft optimierbar sind)
log-queries-not-using-indexes
```

13.4.6 Replikation

Die MySQL-Replikation ermöglicht die automatische Übernahme aller Datenbankänderungen von einem MySQL-Server, dem Master, auf beliebig viele andere MySQL-Server, die Replikations-Slaves. Dies dient zum einen der Ausfallsicherheit – Sie haben stets ein oder gar mehrere sekundenaktuelle Backups zur Hand –, zum anderen auch dem Load-Balancing (Lastverteilung) frequentierter Datenbankanwendungen: Schreibvorgänge müssen zwar weiterhin auf den Master erfolgen, aber Auswahlabfragen können auf die Slaves verteilt werden.

Zur Einrichtung der Replikation sind einige Schritte erforderlich, aber danach läuft diese vollautomatisch. Selbst wenn ein Slave vorübergehend ausfällt, bringt er sich nach einem Neustart des MySQL-Servers selbstständig wieder auf den neuesten Stand.

Auf dem Rechner, der Master werden soll (dieser kann seinerseits auch durchaus Slave eines anderen Masters sein), müssen Sie folgende Schritte ausführen:

- Erstellen Sie einen User für den Replikations-Slave:

  ```
  mysql> CREATE USER repl_user@Slave-Host
      -> IDENTIFIED BY "Passwort";
  ```

  Ob Sie den Slave-Host als einfachen Hostnamen (etwa *heartofgold*) angeben können oder ob Sie seinen FQDN (*heartofgold.test.local*) benötigen, hängt von Ihrer Netzwerkkonfiguration ab; im Zweifelsfall funktioniert Letzteres immer.

- Erteilen Sie dem neuen User das Recht `REPLICATION SLAVE`:

  ```
  mysql> GRANT REPLICATION SLAVE ON *.*
      -> TO repl_user@Slave-Host;
  ```

- Richten Sie eine Update-Log-Datei ein, falls Sie noch keine haben. Nehmen Sie in beiden Fällen folgende Schritte vor, um sich die aktuelle Datei und deren Position zu notieren:

```
mysql> FLUSH TABLES WITH READ LOCK;
mysql> SHOW MASTER STATUS;
```

  Merken Sie sich die Angaben `File` (zum Beispiel *mylog.000003*) und Position (etwa `345`).
- Stoppen Sie den MySQL-Server, zum Beispiel so:

```
# mysqladmin shutdown -u root -p
```

- Erstellen Sie einen Snapshot des gesamten Datenverzeichnisses; hier ein Unix-Beispiel:

```
# cd <MySQL-Datenverzeichnis>
# tar czvf snapshot.tgz */* ibdata*
```

  Die ibdata*-Dateien existieren übrigens nur, wenn Sie InnoDB-Tabellen verwenden.
- Nehmen Sie die folgende Änderung in */etc/my.cnf* vor:

```
[mysqld]
...
server-id = 1
```

  Anstelle der `1` ist auch eine beliebige andere Nummer möglich, solange sie sich von den IDs aller Slaves unterscheidet; beim ersten Master ist `1` aber Standard.
- Zum Schluss müssen Sie den MySQL-Server wieder starten.

Auf dem Slave sind dagegen folgende Vorbereitungen erforderlich:

- Stoppen Sie den MySQL-Server.
- Sichern Sie Ihre bisherigen Daten; unter Unix beispielsweise wie folgt:

```
# mv <MySQL-Datenverzeichnis> <NeuerVerzeichnisName>
```

- Spielen Sie den Snapshot des Masters ein (den Sie zuvor per SCP, NIS, FTP, E-Mail oder wie auch immer auf den Slave kopiert haben):

```
# cd <Verzeichnis-über-MySQL-Datenverzeichnis>
# tar xzvf snapshot.tgz
```

- Passen Sie die Rechte des neuen Datenverzeichnisses an (nur Unix):

```
# chown -R mysql:mysql <MySQLDaten>
```

- Nehmen Sie die folgende Änderung in */etc/my.cnf* vor:

```
[mysqld]
...
server-id = 2
```

  Auch hier kann die ID im Grunde beliebig sein; sie muss sich nur vom Master und von allen anderen Slaves unterscheiden.

- Nun können Sie den MySQL-Server wieder starten.
- Geben Sie im MySQL-Kommandozeilen-Client schließlich noch folgende Anweisungen ein:

```
mysql> CHANGE MASTER TO
    -> MASTER_HOST =     'Master-Host',
    -> MASTER_USER =     'repl_user',
    -> MASTER_PASSWORD = 'Passwort',
    -> MASTER_LOG_FILE = 'ermittelte Log-Datei'
    -> MASTER_LOG_POS =  ermittelte Position;
mysql> START SLAVE;
```

Natürlich müssen Sie die Platzhalter wie `'Master-Host'` oder `'ermittelte Log-Datei'` durch konkrete Werte ersetzen.

Ob die Replikation nun funktioniert, können Sie leicht überprüfen, indem Sie auf dem Master Daten ändern und dann versuchen, diese auf dem Slave zu lesen. Wenn Sie alles richtig gemacht haben, müsste es funktionieren. Falls nicht, überprüfen Sie noch einmal alle Schritte; gegebenenfalls müssen Sie auch nachschauen, ob die Firewall auf einem der beteiligten Hosts den MySQL-Port (standardmäßig 3306) blockiert.

13.5 Grundlagen der Datenbankprogrammierung

Beinahe jede bedeutende Programmiersprache besitzt eine Schnittstelle zu einer oder sogar mehreren Datenbanken. Es ist erheblich praktischer, Daten für verteilte Anwendungen zentral in einer Datenbank abzulegen und aus Programmen darauf zuzugreifen, als diese Daten in einer selbst entwickelten Datenstruktur zu speichern.

Die meisten Datenbankanbindungen für Programmiersprachen sind in der Lage, auf mehrere unterschiedliche SQL-Datenbanken zuzugreifen. In diesem Abschnitt finden Sie eine kurze Einführung in die Datenbankschnittstelle JDBC, über die Sie aus Java-Programmen heraus Zugriff auf beinahe beliebige Datenbanken haben.

JDBC ist keine Schnittstelle zu einer bestimmten Datenbank, sondern eine unabhängige, allgemeine API, die mit vielen unterschiedlichen Treibern zusammenarbeitet. Die meisten dieser Treiber stammen von den jeweiligen Datenbankanbietern selbst, andere werden von Drittanbietern geliefert. An dieser Stelle sollen zwei dieser Treiber explizit erwähnt werden:

- Die JDBC-ODBC-Bridge ist bereits von Haus aus in das Java SDK eingebaut und bietet Zugriff auf beliebige Datenbanken mit ODBC-Anbindung. ODBC (Open Database Connectivity)[3] ist eine von Microsoft entwickelte Technologie, die seit Langem in alle Windows-

3 Die Entwickler von Java werden nicht müde zu betonen, dass JDBC nicht für »Java Database Connectivity« steht, sondern für nichts Konkretes.

Versionen eingebaut ist und den Zugriff auf unzählige Datenbanken bereitstellt. Die JDBC-ODBC-Bridge war vor allem bei der Einführung von JDBC wichtig, weil dadurch indirekt eine Vielzahl von Datenbanktreibern zur Verfügung stand.

Wenn Sie ODBC verwenden möchten, müssen Sie zunächst über die Windows-Systemsteuerung eine Datenbank als ODBC-Datenquelle einrichten.

- MySQL Connector/J stammt von den MySQL-Entwicklern, ist ein Open-Source-Projekt wie MySQL selbst und kann von der MySQL-Website heruntergeladen werden. Nachdem Sie das Archiv entpackt haben, finden Sie im äußersten Verzeichnis eine Datei namens *mysql-connector-java-5.1.36.tar.gz* (oder eine höhere Version als 5.1.36). Nach dem Entpacken müssen Sie den Pfad zum entsprechenden Verzeichnis zu Ihrem `CLASSPATH` hinzufügen (siehe dazu Kapitel 9, »Grundlagen der Programmierung«).

Um JDBC in Ihrem Java-Programm zu verwenden, müssen Sie zuerst die JDBC-Klassen importieren:

```
import java.sql.*;
```

Einen JDBC-Treiber können Sie über `Class.forName()` laden. Das sieht für die JDBC-ODBC-Bridge folgendermaßen aus:

```
try {
   Class.forName ("sun.jdbc.odbc.JdbcOdbcDriver");
}
catch (ClassNotFoundException e) {
   System.out.println
        ("JDBC-ODBC-Treiber nicht gefunden.");
}
```

Die entscheidende Zeile für das Laden von MySQL Connector/J sieht entsprechend so aus:

```
Class.forName("com.mysql.jdbc.Driver");
```

Ein einfaches `Class.forName ("Treiber")` funktioniert unter Umständen nicht richtig. Sicherer ist folgende Variante:

```
Class.forName("Treiber").newInstance();
```

Der nächste Schritt besteht darin, über den Treiber eine Verbindung zur gewünschten Datenbank herzustellen. Dafür besitzt jeder JDBC-Treiber sein eigenes JDBC-URL-Schema. Eine Verbindung über die JDBC-ODBC-Bridge, in diesem Beispiel zu einer Datenbank namens *supermarkt*, funktioniert folgendermaßen:

```
Connection conn = DriverManager.getConnection
     ("jdbc:odbc:supermarkt", "user", "pass");
```


Anstelle von `user` und `pass` müssen Sie selbstverständlich Ihren Usernamen und Ihr Passwort eintragen, die Sie zum Zugriff auf die Datenbank berechtigen. Einige Datenbanksysteme besitzen keinen Passwortschutz; hier müssen Sie zwei leere Strings (`""`) übergeben.

MySQL-Connector/J-URLs sehen ein wenig anders aus. Hier müssen Sie den Netzwerknamen des Rechners angeben, auf dem die Datenbank läuft. Wenn der MySQL-Server auf Ihrem eigenen Rechner läuft, erfolgt ein Zugriff auf die Datenbank *supermarkt* (in diesem Fall ohne Benutzernamen und Passwort) wie folgt:

```
Connection conn = DriverManager.getConnection
     ("jdbc:mysql://localhost/supermarkt", "", "");
```

`localhost` ist die Netzwerkbezeichnung für den lokalen Rechner. Falls sie auf Ihrem Rechner nicht unterstützt wird, können Sie stattdessen die numerische Adresse `127.0.0.1` verwenden.

Nachdem die Verbindung hergestellt wurde, benötigen Sie ein `Statement`-Objekt, das dazu verwendet wird, SQL-Abfragen an die Datenbank weiterzugeben. Das Objekt wird folgendermaßen eingerichtet:

```
Statement st = conn.createStatement();
```

Über das `Statement`-Objekt (hier `st`) können Sie SQL-Abfragen ausführen. Es besitzt die Methoden `executeQuery()` für Auswahlabfragen und `execute()` für alle Abfragen, die Änderungen an der Datenbank vornehmen. Beispielsweise können Sie folgendermaßen einen Datensatz in die Tabelle *artikel* einfügen:

```
try {
   st.execute ("INSERT INTO ARTIKEL (ARTNAME, PREIS,
        MWST) VALUES (\"Tomaten\", 199, \"7\")");
}

catch (SQLException e) {
   // SQL-Abfrage-Ausnahme behandeln
}
```

Die Methode `executeQuery()` liefert ein `ResultSet`-Objekt zurück, aus dem Sie die Ergebnisdatensätze auslesen können. Die folgende Auswahlabfrage liest alle Datensätze aus der Tabelle *artikel*:

```
ResultSet rs = st.executeQuery ("SELECT * FROM ARTIKEL");
```

Mit der `ResultSet`-Methode `next()` können Sie das Ergebnis nun Zeile für Zeile abarbeiten. Um die einzelnen Felder eines Datensatzes auszulesen, bietet `ResultSet` eine Reihe verschiedener `get`-Methoden an, die die verschiedenen SQL-Datentypen in passende Java-Daten-

typen umsetzen. Beispielsweise können Sie die Inhalte von *artikel* aus dem soeben ermittelten `ResultSet` folgendermaßen auf der Konsole ausgeben:

```
while (rs.next()) {
   System.out.print ("Artikel: "
        + rs.getString ("ARTNAME") + "   ");
   System.out.print ("Preis: "
        + rs.getInt ("PREIS") + "   ");
   System.out.println ("MWSt-Satz: "
        + rs.getString ("MWST"));
}
```

Tabelle 13.9 zeigt die wichtigsten SQL-Datentypen, die korrespondierenden Java-Datentypen und die zuständigen `get`-Methoden, um sie aus einem `ResultSet` zu lesen.

SQL-Datentyp	Java-Datentyp	get-Methode
CHAR	String	getString()
VARCHAR	String	getString()
INT	int	getInt()
FLOAT	double	getDouble()
DOUBLE	double	getDouble()
DATE	java.sql.Date	getDate()
TIME	java.sql.Time	getTime()
TIMESTAMP	java.sql.Timestamp	getTimestamp()
BLOB	java.sql.Blob	getBlob()

Tabelle 13.9 Wichtige SQL-Datentypen, ihre Java-Entsprechung und die passenden ResultSet-get-Methoden

Falls Sie die Datenbankressourcen in Ihrem Java-Programm nicht mehr benötigen, sollten Sie sie nacheinander mithilfe ihrer `close()`-Methoden schließen, und zwar zuerst das `ResultSet`, dann das `Statement` und zum Schluss die Datenbankverbindung selbst:

```
rs.close();
st.close();
conn.close();
```


Ein zusammenhängendes Beispiel für die JDBC-Programmierung finden Sie übrigens in Kapitel 16, »XML«, wo die aus einer XML-Datei gelesenen Daten in einer Datenbank gespeichert werden. Weitere Beispiele für Datenbankschnittstellen befinden sich in Kapitel 19, »Webserveranwendungen«, in dem die MySQL-Datenbankanbindungen der Webprogrammiersprache PHP erläutert werden.

CouchDB im Schnellüberblick

Einen ganz anderen Weg als relationale Datenbanken gehen die dokumentbasierten Datenbanken. Es gibt sie im Grunde seit vielen Jahren; ein bekanntes kommerzielles Beispiel ist etwa Lotus Notes. Erst in letzter Zeit beginnt sich dieses Paradigma jedoch auch im Bereich der Webanwendungen zu verbreiten. Hier kommt vor allem das Open-Source-Projekt *Apache CouchDB* zum Einsatz. Die Datenbank wurde 2005 veröffentlicht und kurz darauf von der Apache Software Foundation als Projekt aufgenommen. Die Software ist in der funktionalen Programmiersprache Erlang geschrieben.

In einer dokumentenbasierten Datenbank wie CouchDB gibt es keine Datensätze, sondern nur einzelne Dokumente. Es handelt sich um eine Sammlung frei definierbarer Felder, die jeweils einen Namen und einen Wert mit einem bestimmten Datentyp enthalten. Selbstverständlich besteht die Möglichkeit, beliebig viele Dokumente mit identischer Struktur zu erstellen. Auf diese Weise lassen sich im Grunde auch relationale Abfragen abbilden, allerdings ohne die Flexibilität des dokumentenbasierten Ansatzes einzubüßen.

CouchDB kommuniziert als eigenständiger Webserver mit der Außenwelt; sein Standard-TCP-Port ist 5984. Anfragen werden im REST-Format gestellt, das heißt, es kommen je nach Anfragetyp verschiedene HTTP-Methoden zum Einsatz:

- `PUT` zum Erzeugen von Datenbanken und Dokumenten
- `POST` zum Ändern von Datenbanken und Dokumenten
- `DELETE` zum Löschen von Datenbanken und Dokumenten
- `GET` zum Lesen von Informationen, zum Beispiel für Abfragen

Alle Antworten von CouchDB erfolgen im JSON-Format (JavaScript Object Notation). Dieses Format wird in Abschnitt 20.3, »Ajax«, genau erläutert. Aufgrund der HTTP-REST-JSON-Architektur wird CouchDB auch als *NoSQL-Datenbank* bezeichnet.

Obwohl praktisch alle wichtigen Programmiersprachen mit Bordmitteln HTTP-Anfragen stellen und JSON verarbeiten können, stellen die CouchDB-Entwickler Bibliotheken für eine Reihe wichtiger Sprachen bereit, zum Beispiel für JavaScript, Java, PHP, Python und Ruby.

Die meisten CouchDB-Installationen sind mit einer Administrationsoberfläche namens *Futon* ausgestattet, die sich im Browser über den besagten Port 5984 ansprechen lässt. Ansonsten kann Futon leicht nachinstalliert werden.

13.6 Übungsaufgaben

13.6.1 Praktische Übungen

1. Erstellen Sie eine MySQL-Datenbank, in der Informationen über Musikalben gespeichert werden. Für jedes Album sollen der Titel, das Erscheinungsjahr, die vollständige Trackliste mit Titeln und Spieldauern in Sekunden sowie der Interpret erfasst werden. Auch einzelne Tracks können eine Interpreten-Information haben, wenn es sich um ein Album von mehreren Interpreten handelt. Überlegen Sie zunächst im Sinne der Normalisierung, auf wie viele Tabellen Sie die Informationen sinnvollerweise aufteilen sollten, und erzeugen Sie erst danach die konkreten Tabellen. Erfassen Sie darin einige Beispieldatensätze.
2. Schreiben Sie `JOIN`-Abfragen, um die kombinierten Informationen über die Alben aus der Datenbank herauslesen zu können.
3. Implementieren Sie eine Java-Anwendung, die die Informationen aus der Musikdatenbank auf der Konsole anzeigt, sortiert nach Interpretennamen und anschließend nach Albentiteln. Über Kommandozeilenargumente soll ein optionaler Suchbegriff übergeben werden können, nach dem in den Alben- und Track-Titeln sowie Interpretennamen gefiltert wird. Dabei soll der Suchbegriff automatisch mit `%`-Zeichen umgeben werden, falls noch keine vorhanden sind, und die jeweilige Suche soll mit `LIKE` erfolgen, um auch Teiltreffer zu finden.

13.6.2 Kontrollfragen

Im Folgenden ist jeweils genau eine Antwort richtig.

1. Welche Art von Daten gehört nicht zu denjenigen, die im Zusammenhang mit einer Datenbank unterschieden werden?
   - Rechendaten
   - Bewegungsdaten
   - Stammdaten
   - Messdaten
2. Welche Kombination von Datenarten ist der Einzelpreis eines Artikels?
   - Bewegungsdatum und Messdatum
   - Stammdatum und Messdatum
   - Stammdatum und Rechendatum
   - Bewegungsdatum und Rechendatum
3. Welche der folgenden Informationen ist Bewegungsdatum und Rechendatum?
   - Kalendertag
   - Stückzahl auf Lager

   - Skonto
   - Börsenkurs
4. Welche Information in einer Datenbanktabelle ist der Datensatz?
   - ein einzelnes Feld
   - eine Zeile
   - eine Spalte
   - zusammengehörende Felder
5. Welche Bedeutung hat der Primärschlüssel in einer relationalen Datenbank?
   - eindeutige Kennzeichnung einer Tabelle
   - Schutz der Datenbank durch Verschlüsselung
   - eindeutige Kennzeichnung eines Datensatzes
   - Vorlage für neue Datensätze
6. Welche Beziehung kann in einer relationalen Datenbank nur indirekt bestehen?
   - die m:n-Beziehung
   - die 1:n-Beziehung
   - die 1:1-Beziehung
   - Alle Beziehungen können direkt bestehen.
7. Welche Bedeutung hat die zweite Normalform (2NF) einer relationalen Datenbank?
   - Die Information innerhalb eines Feldes muss atomar sein.
   - Die Felder eines Datensatzes müssen funktional unabhängig sein.
   - Ein Datensatz darf nur direkte Informationen über denselben Sachverhalt enthalten.
   - Bei mehrteiligen Primärschlüsseln muss jeder Datensatz von allen Elementen dieses Schlüssels abhängen.
8. Welche Normalform enthält die Forderung, dass eine Tabelle nur triviale Join-Abhängigkeiten enthalten darf?
   - die dritte Normalform (3NF)
   - die Boyce-Codd-Normalform (BCNF)
   - die fünfte Normalform (5NF)
   - die vierte Normalform (4NF)
9. Was ist ein Vorteil der objektorientierten gegenüber der relationalen Datenbank?
   - Nur in einer OO-Datenbank können die Felder verschiedene Datentypen besitzen.
   - Die Beziehungen sind nicht auf Schlüssel beschränkt und können freier gestaltet werden.
   - Nur die OO-Datenbank ist programmierbar.
   - In OO-Datenbanken funktioniert die Suche schneller.

10. Was benötigen Sie, um eine MySQL-Datenbank über phpmyadmin zu verwalten?
    - einen Windows-Rechner mit PHP
    - einen Webserver und PHP
    - einen Browser und PHP
    - einen zweiten Rechner, auf dem die Serversoftware läuft
11. Welche Indexart besitzt die Einschränkung, dass jedes Feld einer Spalte einen individuellen Wert benötigt?
    - `INDEX`
    - `FULLTEXT`
    - `AUTO_INCREMENT`
    - `UNIQUE`
12. Welche der folgenden "Abfragen" gibt es in SQL nicht?
    - Auswahlabfrage
    - Einfügeabfrage
    - Erweiterungsabfrage
    - Löschabfrage
13. Welchen Fehler enthält die folgende SQL-Anweisung zur Erstellung der Tabelle `TESTCREATE TABLE TEST (NR PRIMARYKEY, NAME CHAR(50) NOT NULL)`?
    - Die Felder müssen durch ein Semikolon anstelle eines Kommas getrennt werden.
    - `PRIMARYKEY` muss in zwei Wörtern geschrieben werden: `PRIMARY KEY`.
    - Die Angabe der Zeichenanzahl bei `CHAR` muss in eckigen Klammern stehen.
    - Bei `NOT NULL` fehlt der Unterstrich; es heißt `NOT_NULL`.
14. Welche Wortbreite besitzt der MySQL-Datentyp MEDIUMINT?
    - 16 Bit
    - 24 Bit
    - 32 Bit
    - 48 Bit
15. Welcher der folgenden SQL-Datentypen ist kein Fließkommatyp?
    - `FLOAT`
    - `REAL`
    - `DOUBLE`
    - `DECIMAL`

16. Was ist der Unterschied zwischen den SQL-Datentypen `CHAR` und `VARCHAR`?
    - `VARCHAR` kann beliebig viele Zeichen enthalten, `CHAR` nur eine bestimmte Anzahl.
    - `VARCHAR` belegt nur Speicherplatz für die tatsächlich verwendeten Zeichen, `CHAR` für alle Zeichen der vereinbarten Gesamtgröße.
    - `VARCHAR` kann mehr Zeichen enthalten als `CHAR`.
    - Die beiden Typen sind synonym.
17. Wie lassen sich alle Felder aller Datensätze der Tabelle `INFO` auswählen?
    - `SELECT () FROM INFO`
    - `SELECT ALL FROM INFO`
    - `SELECT * FROM INFO`
    - `SELECT FROM INFO WHERE *`
18. Was ist zu beachten, wenn Sie alle Elemente einer Tabelle mittels `COUNT` zählen möchten?
    - Sie müssen alle Felder auswählen.
    - Sie dürfen jedes Feld außer dem Primärschlüssel auswählen.
    - Sie dürfen nur den Primärschlüssel auswählen.
    - Sie können jedes beliebige Feld auswählen, aber nur eines.
19. Wie lässt sich die folgende SQL-Join-Abfrage mittels `WHERE` darstellen?
    `SELECT * FROM TEST, INFO INNER JOIN TEST.NR ON INFO.NR`
    - `SELECT * FROM TEST, INFO WHERE NR=NR`
    - `SELECT * FROM TEST, INFO WHERE NR`
    - `SELECT * FROM TEST, INFO WHERE NR.NR`
    - `SELECT * FROM TEST, INFO WHERE TEST.NR=INFO.NR`
20. Welche der folgenden Angaben ist eine formal gültige MySQL-Connector/J-URL?
    - *jdbc.mysql://host/database*
    - *jdbc:mysql://host:database*
    - *jdbc.mysql://host.database*
    - *jdbc:mysql://host/database*
21. Was ist der Unterschied zwischen den Methoden `st.execute()` und `st.executeQuery()` eines JDBC-Statement-Objekts?
    - `execute()` funktioniert nur mit der JDBC-ODBC-Bridge.
    - `executeQuery()` wird für Einfügeabfragen verwendet, `execute()` für die anderen.
    - `executeQuery()` gibt ein `ResultSet` zurück und wird daher für Auswahlabfragen eingesetzt.
    - `execute()` ist veraltet.

Kapitel 19
Webserveranwendungen

PHP is about as exciting as your toothbrush. You use it every day, it does the job, it is a simple tool, so what? Who would want to read about toothbrushes?[1]
– Rasmus Lerdorf

Das Grundprinzip von Webserveranwendungen ist immer dasselbe: Wenn ein Benutzer eine bestimmte URL anfordert, die auf einen Teil einer solchen Anwendung verweist, liefert der Webserver nicht einfach ein fertiges Dokument aus. Stattdessen startet er irgendeine Art von Programm, das aus einer Vorlage und variablen Daten »on the Fly« eine Webseite erstellt, und liefert diese dynamisch erzeugte Seite an den Browser des Besuchers aus.

Bei dem Programm, das der Webserver aufruft, handelt es sich je nach verwendeter Serverlösung um ein externes Programm, das separat gestartet wird, oder aber um ein Modul des Webservers selbst. Letzteres ist erheblich effizienter – der Webserver kann die Anfrage selbst bearbeiten und muss kein separates Programm starten. Bedenken Sie, dass bei einem externen Programm für jeden Aufruf ein neuer Prozess gestartet wird, was bei vielen zeitgleichen Benutzern zu erheblichen Engpässen führen kann.

In diesem Kapitel wird zuerst die beliebte Webserver-Programmiersprache PHP vorgestellt. Da größere Webanwendungen so gut wie immer auf einer Datenbank basieren, wird zusätzlich die Zusammenarbeit mit dem Datenbankserver MySQL beschrieben. Im zweiten Abschnitt wird – ebenfalls mit PHP – eine REST-API implementiert, also ein moderner Webservice.

19.1 PHP

Die Sprache *PHP* ist eines der beliebtesten Werkzeuge zur Erstellung dynamischer Webinhalte für kleine und mittlere Websites. Der Name dieser 1995 von Rasmus Lerdorf unter der ursprünglichen Bezeichnung *Personal Homepage Tools* entwickelten Server-Skriptsprache steht inzwischen für das rekursive Akronym *PHP*: *Hypertext Preprocessor*.

Die Sprache ist für viele verschiedene Plattformen wie Windows und etliche Unix-Varianten verfügbar. Besonders verbreitet ist die Kombination aus dem Betriebssystem Linux, dem

1 »PHP ist ungefähr so aufregend wie deine Zahnbürste. Du benutzt sie jeden Tag, sie tut ihren Dienst, sie ist ein einfaches Werkzeug, also was soll's? Wer würde etwas über Zahnbürsten lesen wollen?«

Webserver Apache, der freien Datenbank MySQL und der Programmiersprache PHP (manchmal auch Perl oder Python) – kurz ein *LAMP-System*. Unter dem Betriebssystem Windows wird dieselbe Softwarekombination *WAMP* genannt. Wie Sie PHP in Ihrem Apache-Webserver installieren, wird in Kapitel 14, »Server für Webanwendungen«, beschrieben.

19.1.1 Sprachgrundlagen

Der PHP-Interpreter akzeptiert gewöhnliche HTML-Dateien, in denen speziell markierte PHP-Abschnitte verarbeitet und durch ihre Ausgabe ersetzt werden. Sie müssen also kein HTML ausgeben, sondern können PHP-Anweisungen an die passende Stelle des HTML-Dokuments schreiben. In größeren Anwendungen ist es allerdings sehr zu empfehlen, Logik und Ausgabe voneinander zu trennen. Solche Konstrukte sollten daher höchstens in reinen Ausgabedateien verwendet werden – wenn nicht ohnehin ein eigenständiges Template-System zum Einsatz kommt, das für die Ausgabe selbst kein PHP verwendet.

Der PHP-Code wird in einen Bereich hineingeschrieben, der folgendermaßen gekennzeichnet wird:

```
<?php
  // PHP-Anweisungen
?>
```

Ein solcher Bereich kann an einer beliebigen Stelle im HTML-Dokument stehen, sogar innerhalb von HTML-Tags oder ihren Attributwerten. Außerdem können sich HTML- und PHP-Blöcke an einer beliebigen Stelle und selbst innerhalb derselben Zeile abwechseln. Konstrukte wie das folgende sind ohne Weiteres möglich und sind in den besagten Ausgabedateien mitunter praktisch:

```
<?php if ($punkte > 100) { ?>
  <h2>Herzlichen Gl&uuml;ckwunsch!</h2>
<?php } else { ?>
  <h2>Sie sollten noch &uuml;ben!</h2>
<?php } ?>
```

Dieses Beispiel gibt die Überschrift »Herzlichen Glückwunsch!« aus, falls die Variable `$punkte` einen höheren Wert als 100 hat, ansonsten den Text »Sie sollten noch üben!«. Die folgende Schreibweise ist synonym, aber für eine reine Ausgabedatei unhandlicher:

```
<?php

  if ($punkte > 100) {
    echo "<h2>Herzlichen Gl&uuml;ckwunsch!</h2>";
  } else {
    echo "<h2>Sie sollten noch &uuml;ben!</h2>";
  }

?>
```


Das schließende PHP-Tag `?>` können Sie übrigens beim letzten (oder einzigen) PHP-Block in einer Datei weglassen. Bei PHP-Dateien, die ausschließlich Programmlogik enthalten, ist dies sogar empfehlenswert. Whitespace hinter dem schließenden Tag könnte nämlich als Ausgabeinhalt interpretiert werden, sodass Mechanismen wie Weiterleitungen oder Cookies, die auf HTTP-Headern basieren, dann unter Umständen nicht mehr funktionieren.

Der Sprachkern von PHP wurde stark von Programmiersprachen wie C und Perl inspiriert. PHP war ursprünglich eine prozedurale Sprache, wurde aber in neueren Versionen um objektorientierte Merkmale erweitert.

Wie die meisten anderen Skriptsprachen ist PHP nicht typisiert, eine Variable kann also nacheinander Werte beliebiger Datentypen annehmen. Grundlegende Kontrollstrukturen wie Fallunterscheidungen und Schleifen funktionieren genau wie in C und in allen davon abgeleiteten Sprachen.

Variablenbezeichner beginnen grundsätzlich mit einem `$`-Zeichen. Anders als in Sprachen wie Perl gibt es keine besonderen Zeichen, die Arrays oder Hashes kennzeichnen (diese sind in PHP ohnehin dasselbe; ein Array kann sowohl Zahlen als auch andere Objekte als Indizes und Schlüssel verwenden). Hinter dem `$` können Buchstaben, Ziffern und Unterstriche folgen; das erste Zeichen darf allerdings keine Ziffer sein. Es wird zwischen Groß- und Kleinschreibung unterschieden.

Funktions- und Klassennamen kommen dagegen ohne Dollarzeichen aus, abgesehen davon, werden Groß- und Kleinschreibung weder bei selbst definierten noch bei eingebauten Funktions- und Klassennamen unterschieden. Allerdings wäre es sehr schlechter Programmierstil, dies auszunutzen. Üblicherweise sollten Sie Variablen- und Funktionsbezeichner mit kleinem und Klassennamen mit großem Anfangsbuchstaben schreiben. Bestehen die Bezeichner aus mehreren Wörtern, sollten Sie CamelCase (Binnenmajuskeln für jedes neue Wort) verwenden und keine Unterstriche (also beispielsweise `$myVariable` anstelle von `$my_variable`).

Variablendefinition und -verwendung

Eine Variable wird in PHP durch die erste Wertzuweisung (Initialisierung) erzeugt. Wertzuweisungen sehen genauso aus wie in den meisten anderen Programmiersprachen:

```
$test = 9;
$text = "hallo";
```

In dem Moment, in dem einer Variablen zum ersten Mal ein Wert zugewiesen wird, existiert sie. Eine Deklaration im eigentlichen Sinn gibt es nicht.

Variablen haben in PHP auch keinen festgelegten Datentyp. Sie können einer Variablen nacheinander verschiedene Arten von Werten zuweisen, zum Beispiel:

```
$a = 5;     // Ganzzahl
$a = 3.78;  // Fließkommazahl
$a = "hi";  // String
```

Mitunter werden die Werte von Variablen in einem neuen Zusammenhang automatisch anders interpretiert. Hier sehen Sie zwei Beispiele:

```
$b = "67";      // String, wegen Anführungszeichen
$c = $b + 9;    // Ergebnis: 76
$a = 22;        // Ganzzahl
$b = $a . "33"; // "2233"
```

Der Verkettungsoperator für Strings ist in PHP der Punkt (`.`) und nicht das in Java und JavaScript verwendete Pluszeichen, das häufig wegen der Verwechslung mit der numerischen Addition Ärger bereitet. Die meisten anderen Operatoren entsprechen ihrer Verwendung in C, Java und ähnlichen Sprachen. Genaueres über deren Operatoren erfahren Sie in Kapitel 9, »Grundlagen der Programmierung«.

Variablen gelten in PHP ab dem Zeitpunkt ihrer Wertzuweisung im gesamten Dokument. Es gibt keine untergeordneten Gültigkeitsbereiche innerhalb von Blöcken wie Fallunterscheidungen oder Schleifen. Die Ausnahme bilden Variablen, die innerhalb der im weiteren Verlauf des Kapitels behandelten Funktionen definiert werden. Sie sind lokal und gelten nur innerhalb der jeweiligen Funktion.

Ein wertvoller Helfer bei der Entwicklung von PHP-Skripten ist die Funktion `var_dump($ausdruck, $ausdruck ...)`. Sie gibt die Datentypen und Inhalte von Variablen und anderen Ausdrücken in einer für Menschen lesbaren Form aus, selbst wenn es sich um boolesche Werte, Arrays oder sogar Objekte handelt. Hier ein Beispiel:

```
$a = "Hallo";   // String
$b = 42;        // Integer
$c = TRUE;      // boolescher Wert
$d = [1, 2, 3]; // Array – siehe nächsten Abschnitt
var_dump($a, $b, $c, $d);
```

Die Ausgabe lautet wie folgt:

```
string(5) "Hallo"
int(42)
bool(true)
array(3) {
  [0]=>
  int(1)
```



```
  [1]=>
  int(2)
  [2]=>
  int(3)
}
```

Arrays

Wie in den meisten anderen Programmiersprachen gibt es auch in PHP die Möglichkeit, *Arrays* zu bilden. Ein Array ist eine Variable, die eine Liste von Werten enthält, auf die über einen *Index* zugegriffen werden kann. PHP unterscheidet nicht grundsätzlich zwischen einem gewöhnlichen Array mit numerischen Indizes und einem Hash, bei dem die Indizes Strings (oder andere Objekte) sind. In jedem Array können beide Indexarten gleichzeitig existieren.

Um ein klassisches Array mit numerischen Indizes zu erzeugen, genügt es, einem einzelnen Element dieses Arrays einen Wert zuzuweisen:

```
$monate[0] = "Januar";
```

Mithilfe der eingebauten Funktion `array()` können Sie auch gleich ein numerisches Array mit mehreren Elementen erzeugen:

```
$jahreszeiten = array(
  "Fruehling",
  "Sommer",
  "Herbst",
  "Winter"
);
```

Seit PHP 5.4 ist alternativ auch die folgende Schreibweise erlaubt:

```
$jahreszeiten = [
  "Fruehling",
  "Sommer",
  "Herbst",
  "Winter"
];
```

Besonders interessant ist im Übrigen die Tatsache, dass Sie leere eckige Klammern anstelle eines konkreten Indexes verwenden können, um ein Element am Ende des Arrays anzufügen:

```
$wochentage[] = "Sonntag";
$wochentage[] = "Montag";
//etc.
```

Über die Funktion `array_push($array, $wert1, $wert2, ...)` können Sie aber auch eine Liste mehrerer Elemente am Ende des Arrays einfügen. Umgekehrt liefert die Funktion `array_pop($array)` das letzte Element des Arrays zurück und entfernt es aus dem Array.

Um einen Hash oder ein assoziatives Array zu erzeugen, können Sie ebenfalls mit der Zuweisung des Wertes für ein einzelnes Element beginnen:

```
$monate['jan'] = "Januar";
```

Möchten Sie die Werte mehrerer Elemente gleichzeitig zuweisen, funktioniert dies mithilfe der folgenden Form der Funktion `array()`:

```
$wochentage = array(
  'So' => "Sonntag",
  'Mo' => "Montag",
  'Di' => "Dienstag",
  'Mi' => "Mittwoch",
  'Do' => "Donnerstag",
  'Fr' => "Freitag",
  'Sa' => "Samstag"
);
```

Die Verwendung einfacher Anführungszeichen für die Indizes und doppelter für die Werte ist nicht vorgeschrieben, aber eine gängige Konvention. Innerhalb doppelter Anführungszeichen werden Variablen und alle üblichen Escape-Sequenzen ausgewertet, innerhalb von einfachen aber nicht (lediglich `\'` für ein einzelnes Anführungszeichen als solches und `\\` für einen Backslash werden erkannt):

```
$geld = 100;
echo "Ich habe $geld \$.";
   // Ausgabe: Ich habe 100 $.
echo 'Ich habe auch $geld $.';
   // Ausgabe: Ich habe auch $geld $.
```

Die Funktion `count($array)` – ein gültiges Synonym ist `sizeof($array)` – liefert die Anzahl der Elemente im Array zurück. Auf diese Weise können Sie alle Elemente eines numerischen Arrays in einer Schleife ausgeben, zum Beispiel folgendermaßen:

```
$zimmer = array(
  "Wohnzimmer",
  "Schlafzimmer",
  "Kinderzimmer",
  "Arbeitszimmer"
);
$zahl = sizeof($zimmer);
```



```
for ($i = 0; $i < $zahl; $i++) {
   echo $zimmer [$i]."<br />";
}
```

Dieses kleine Beispiel gibt untereinander die Bezeichnungen der vier Zimmer aus. Bei Hashes sollten Sie dagegen eine andere Methode verwenden, die Elemente aufzuzählen: Die Funktion `each($array)` gibt bei jedem Aufruf das nächste Schlüssel-Wert-Paar zurück. Dieses Paar können Sie mithilfe von `list()` einer Liste aus zwei Variablen zuweisen. Das Ganze funktioniert so:

```
$rechner = array (
  'cpu' => "Intel Core i7",
  'ram' => "8192 MB DDR3 RAM",
  'hdd' => "Maxtor 500 GB",
  'dvd' => "16xDVD / 52xCD"
);
reset($rechner);  // Sicherheitshalber auf Anfang
while (list($key, $val) = each ($rechner)) {
   echo "$key: <b>$val</b><br />";
}
```

Noch klarer und einfacher sind `foreach`-Schleifen, die für die beiden Arrays wie folgt aussehen:

```
foreach ($zimmer as $z) {
  echo "$z<br />";
}
foreach ($rechner as $key => $val) {
   echo "$key: <b>$val</b><br />";
}
```

Interessant ist, dass die Elemente eines PHP-Arrays stets eine von den Schlüsseln oder Indizes unabhängige, festgelegte Reihenfolge haben – nämlich diejenige, in der die Elemente hinzugefügt wurden. Betrachten Sie etwa folgendes Beispiel:

```
$a = array();
$a[3] = "Wert auf 3";
$a[1] = "Wert auf 1";
var_dump($a);
```

In den meisten Programmiersprachen ergäbe dies ein Array mit vier Elementen, bei dem die Indizes 0 und 2 leere Elemente (`NULL` oder dergleichen) enthalten, in der Reihenfolge `[NULL, "Wert auf 1", NULL, "Wert auf 3"]`. In PHP sieht das Ganze dagegen anders aus, wie die Ausgabe von `var_dump()` zeigt:

```
array(2) {
  [3]=>
  string(10) "Wert auf 3"
  [1]=>
  string(10) "Wert auf 1"
}
```

Für `foreach()` wird intern ein Zeiger oder Cursor verwendet, der auf das aktuelle Element in der inneren Reihenfolge zeigt. Sie können die entsprechenden Funktionen auch manuell von außen aufrufen, wie das folgende Beispiel zeigt:

```
<?php

$person = array(
  'name' => "Schmitz",
  'vorname' => "Heinz",
  'stadt' => "Köln",
  'beruf' => "Sachbearbeiter"
);
reset($person);
do {
  printf("%s: %s\n", key($person), current($person));
} while (next($person));
```

Die Funktionen, die hier im Einzelnen zum Einsatz kommen, sind folgende:

- `reset($array)` setzt den Zeiger auf das erste Element zurück.
- `key($array)` liefert den Schlüssel an der aktuellen Position des Zeigers zurück.
- `current($array)` liefert den Wert an der aktuellen Position des Zeigers zurück.
- `next($array)` rückt den Zeiger um eine Position weiter und liefert den dortigen Schlüssel zurück oder `FALSE`, falls kein weiteres Element mehr vorhanden ist. Deshalb kann `next()` im Beispiel als Bedingung für die `do`/`while`-Schleife verwendet werden.

Die Ausgabe des Beispiels sieht so aus:

```
name: Schmitz
vorname: Heinz
stadt: Köln
beruf: Sachbearbeiter
```

Diese internen Details sind wichtig, wenn Sie eine eigene Klasse so aufbereiten möchten, dass ihre Objekte mit `foreach()` iterierbar sind. Wie dies funktioniert, erfahren Sie im Unterabschnitt »Interfaces der Standard PHP Library (SPL)« .

Interessant ist auch noch die Funktion `explode()`. Sie funktioniert nach folgendem Schema:

```
$array = explode($trennString, $string);
```

Die Funktion zerlegt den String `$string` an den Stellen, an denen `$trennString` vorkommt, in die einzelnen Elemente des Arrays `$array`. Zum Beispiel:

```
$string = "a,b,c,d";
$array = explode(",", $string);
/* $array[0] ist "a"
   $array[1] ist "b"
   $array[2] ist "c"
   $array[3] ist "d" */
```

Wenn Sie anstelle eines einfachen Strings einen regulären Ausdruck als Trennzeichen angeben möchten, müssen Sie die Anweisung `preg_split($regexp, $array)` anstelle von `explode()` verwenden. Näheres zur Verwendung regulärer Ausdrücke in PHP lesen Sie im nächsten Abschnitt.

Die umgekehrte Aufgabe erledigt die Funktion `implode()`, die die Elemente eines Arrays – getrennt durch die angegebene Zeichenfolge – zu einem String zusammenfasst:

```
$string = implode($trennString, $array);
```

Hier sehen Sie ein Beispiel:

```
$array = array("So", "Mo", "Di", "Mi", "Do", "Fr", "Sa");
$string = implode(", ", $array);
// $string ist "So, Mo, Di, Mi, Do, Fr, Sa";
```

Auch mehrdimensionale Arrays sind kein Problem. Das folgende Beispiel definiert die Variable `$jahr` als Aufzählung der Jahreszeiten mit ihren Monaten:

```
$jahr = array(
  'fruehling' => array("März", "April", "Mai"),
  'sommer' => array("Juni", "Juli", "August"),
  'herbst' => array("September", "Oktober", "November"),
  'winter' => array("Dezember", "Januar", "Februar")
);
```

Wenn Sie nun beispielsweise auf den zweiten Monat im Sommer zugreifen möchten, können Sie die Schreibweise `$jahr['sommer'][1]` verwenden – die Rückgabe lautet natürlich »Juli«.

Da Arrays eines der wichtigsten Elemente von PHP sind, gibt es zahllose Funktionen, um diese zu verarbeiten. Hier nur einige wichtige im Überblick:

- `array_push($array, $wert1, $wert2, ...)` hängt einen oder mehrere Werte am Ende des Arrays an. Wenn Sie nur einen Wert anhängen möchten, ist die Schreibweise mit den leeren eckigen Klammern allerdings bequemer.
- `array_pop($array)` entfernt das letzte Element des Arrays und liefert es als Wert zurück.
- `array_unshift($array, $wert1, $wert2, ...)` fügt einen oder mehrere Werte am Anfang des Arrays ein.
- `array_shift($array)` entfernt das erste Element aus dem Array und liefert es als Wert zurück.
- `sort($array)` sortiert das Array. Dabei wird das Original-Array sortiert und nicht etwa eine sortierte Fassung zurückgegeben. Es gibt zahlreiche Varianten der Sortierfunktion wie `rsort()` für absteigendes Sortieren, `ksort()` für Sortieren nach Schlüsseln anstelle von Werten oder sogar `usort()`, das mithilfe einer benutzerdefinierten Funktion sortiert. Betrachten Sie für Letzteres das folgende Beispiel:

```
<?php

$personen = array(
  array('name' => "Freeman", 'vorname' => "Morgan"),
  array('name' => "Freeman", 'vorname' => "Martin"),
  array('name' => "Cumberbatch", 'vorname' => "Benedict"),
  array('name' => "Dormer", 'vorname' => "Natalie"),
  array('name' => "Clarke", 'vorname' => "Emilia"),
  array('name' => "Jackman", 'vorname' => "Hugh")
);
usort($personen, function($a, $b) {
  if ($a['name'] < $b['name']) {
    return -1;
  }
  if ($a['name'] > $b['name']) {
    return 1;
  }
  if ($a['vorname'] < $b['vorname']) {
    return -1;
  }
  if ($a['vorname'] > $b['vorname']) {
    return 1;
  }
  return 0;
});
foreach ($personen as $person) {
  printf("%s %s\n", $person['vorname'], $person['name']);
}
```


Wie Sie sehen, enthält das zu sortierende Array wiederum Arrays, die die Namen und Vornamen diverser Schauspielerinnen und Schauspieler enthalten, jeweils mit entsprechend benannten Schlüsseln. Das Array soll nun aufsteigend nach Nachnamen sortiert werden, und wenn die Nachnamen identisch sind, hilfsweise nach Vornamen. Dazu wird eine anonyme Funktion definiert, die zwei Argumente (zu vergleichende Werte) entgegennimmt, hier – wie meist üblich – als `$a` und `$b` bezeichnet. Sie soll –1 zurückgeben, wenn `$a` vor `$b` einsortiert werden soll, 1 für die umgekehrte Reihenfolge und 0, wenn die Werte identisch sind.

Die Schreibweise mit der verschachtelten anonymen Funktion ist erst seit PHP 5.3 zulässig, davor konnte man entweder den Namen einer Funktion als String angeben oder ein Array in der Form `array($objekt, "methodenname")` für die Verwendung in Klassen; beides ist alternativ immer noch möglich. Näheres über Funktionen sowie Klassen und Methoden erfahren Sie später in diesem Kapitel, und die Verwendung solcher Callbacks wird in Abschnitt 19.1.3, »Include-Dateien, Autoloader und Namespaces« näher beschrieben.

Die Ausgabe des Beispiels sieht so aus:

```
Emilia Clarke
Benedict Cumberbatch
Natalie Dormer
Martin Freeman
Morgan Freeman
Hugh Jackman
```

- `shuffle($array)` erledigt das Gegenteil von `sort()` – das Array wird zufällig durchgemischt. Dazu hier ebenfalls ein kleines Beispiel, das ein 32er-Kartenspiel zunächst ungemischt und dann gemischt ausgibt:


```
<?php

$kartendeck = array();

$farben = ['♣', '♠', '♥', '♦'];
$werte = ['7', '8', '9', '10', 'B', 'D', 'K', 'A'];

foreach ($farben as $farbe) {
  foreach ($werte as $wert) {
    $kartendeck[] = "$farbe$wert";
  }
}

echo implode(" ", $kartendeck);
echo "\n\n";
```

```
shuffle($kartendeck);
echo implode(" ", $kartendeck);
echo "\n";
```

Die Sonderzeichen für die Kartenfarben können Sie unter OS X in den meisten Programmen über den Menüpunkt Bearbeiten • Emojis und Symbole einfügen; auf Windows-Systemen ist die Zeichentabelle (*charmap.exe*) zuständig.

Hier der Vollständigkeit halber noch eine mögliche Ausgabe des Beispiels (natürlich werden die Karten jedes Mal anders gemischt):

```
♣7 ♣8 ♣9 ♣10 ♣B ♣D ♣K ♣A ♠7 ♠8 ♠9 ♠10 ♠B ♠D ♠K ♠A
♥7 ♥8 ♥9 ♥10 ♥B ♥D ♥K ♥A ♦7 ♦8 ♦9 ♦10 ♦B ♦D ♦K ♦A

♠8 ♣B ♦9 ♠A ♣8 ♦8 ♦7 ♣10 ♥A ♣A ♥8 ♣9 ♥10 ♦A ♠10 ♣D
♥7 ♣7 ♠B ♦D ♦K ♣K ♥9 ♠K ♦10 ♥D ♠7 ♥K ♠9 ♥B ♦B ♠D
```

Perl-kompatible reguläre Ausdrücke

PHP besitzt traditionell mehrere Implementierungen regulärer Ausdrücke. Die leistungsfähigste von ihnen besteht aus Funktionen, deren Namen mit `preg_` beginnen – es handelt sich um Perl-kompatible reguläre Ausdrücke (PCRE), also eine weitgehende Übernahme der besonders umfangreichen Regex-Bibliothek der Programmiersprache Perl. Die anderen Implementierungen gelten inzwischen als veraltet und sollten nicht mehr verwendet werden.

Die Funktion `preg_match($regexp, $string)` überprüft, ob der reguläre Ausdruck `$regexp` auf den String `$string` passt. Das folgende Beispiel gibt eine Meldung aus, falls `$eingabe` nicht komplett aus Ziffern besteht:

```
if (preg_match('/\D/', $eingabe)) {
  echo "Dies ist keine Zahl!";
}
```

Innerhalb der String-Anführungszeichen steht der reguläre Ausdruck zwischen zwei // (Slashes) oder wahlweise zwischen anderen Trennzeichen wie ~~ oder Klammerpaaren. Hinter dem schließenden Trennzeichen können Modifikatoren stehen. Der wichtigste ist `i`, um Unterschiede zwischen Groß- und Kleinschreibung zu ignorieren. Das folgende Beispiel sucht nach »PHP« in beliebiger Schreibweise:

```
if (preg_match('(php)i', $eingabe)) {
   echo("PHP gefunden!");
} else {
   echo("PHP nicht vorhanden!");
}
```


`preg_replace($regexp, $ersatz, $string[, $limit])` ersetzt den regulären Ausdruck `$regexp` in `$string` durch den Ersatztext `$ersatz`. Normalerweise wird jedes Vorkommen von `$regexp` ersetzt; der optionale Parameter `$limit` gibt die maximale Anzahl von Ersetzungen an. Die folgende Anweisung ersetzt in `$eingabe` jedes Vorkommen von »Perl« in beliebiger Schreibweise durch »PHP«:

```
$eingabe = preg_replace('(perl)i', 'PHP', $eingabe);
```

Mithilfe von `$n` (mit `n`-Werten zwischen 0 und 99) können Sie sich im Ersatztext auf geklammerte Ausdrücke aus dem regulären Ausdruck beziehen. Das folgende Beispiel fügt an den korrekten Stellen einer alten, zehnstelligen ISBN Bindestriche an den gängigsten Stellen ein:

```
$isbn = preg_replace(
  '{(\d)(\d{5})(\d{3})(\d)}',
  "$1-$2-$3-$4",
  $isbn
);
```

Aus 3836214202 (einer unformatierten ISBN) würde so beispielsweise 3-83621-420-2.

Kommentare

PHP unterstützt drei Arten von Kommentaren, um es Programmierern leicht zu machen, die verschiedene Sprachen beherrschen. Als Erstes wird der einzeilige Kommentar im C++-Stil unterstützt:

```
// einzeiliger Kommentar
```

Dabei wird der Rest der Zeile ignoriert. Auch der mehrzeilige Kommentar im C-Stil ist erlaubt:

```
/* mehr-
   zeiliger
   Kommentar */
```

Hier werden alle betroffenen Zeilen ignoriert. Eine spezielle Variante dieses Kommentars ist der sogenannte *Docblock-Kommentar*, der von einem Dokumentationsgenerator wie PHP-Documentor oder der empfehlenswerteren Neuentwicklung phpdox ausgewertet wird:

```
/**
* Dokumentation ...
* Mehr Dokumentation ...
*/
```

Docblock-Kommentare werden vor Klassen, Methoden, Funktionen oder Attribute geschrieben, und für jedes dieser Elemente gibt es eine Reihe von Annotationen, die mit einem @-Zei-

chen beginnen und spezielle Aspekte des jeweiligen Elements beschreiben. Hier zum Beispiel der Docblock-Header einer Methode oder Funktion mit einem Parameter und einem Rückgabewert:

```
/**
* Return the square of the argument
*
* @param integer $value
* @return integer
*/
function square($value) {
  return $value * $value;
}
```

Schließlich ist noch der einzeilige Kommentar im Stil von Python und der Unix-Shell-Sprachen zulässig:

```
# einzeiliger Kommentar
```

Funktionen

Eine PHP-Funktion ist ein benannter Codeblock, der mit seinem Namen aufgerufen wird. Ständig wiederkehrende Anweisungsfolgen in Funktionen zu verpacken schafft Übersicht und schont Ressourcen, da eine Funktion innerhalb eines Skripts nur einmal kompiliert wird. Sie sollten Funktionen in einem PHP-Block zu Beginn Ihres Dokuments unterbringen oder sie in einer externen Include-Datei (wird in Abschnitt 19.1.3, »Include-Dateien, Autoloader und Namespaces«, beschrieben) speichern. Die allgemeine Syntax ist einfach:

```
function funktionsname() {
  // Anweisungen ...
}
```

Nach Ausführung der letzten Anweisung wird die Kontrolle an die aufrufende Stelle zurückgegeben. Wie bereits erwähnt, können Funktionen lokale Variablen enthalten: Jede Variable, die in einer Funktion verwendet wird, ist lokal – sogar dann, wenn im globalen Code eine gleichnamige Variable existiert. Betrachten Sie etwa den Wert der beiden Variablen namens `$a` im folgenden Beispiel:

```
function a_aendern() {
  $a = 9;   // lokales $a hat den Wert 9.
}

$a = 7;  // globales $a hat den Wert 7.
a_aendern();
// $a hat hier immer noch den Wert 7.
```


Möchten Sie innerhalb einer Funktion auf eine globale Variable zugreifen, müssen Sie am Anfang der Funktion ausdrücklich das Schlüsselwort `global` verwenden. Hier sehen Sie das zuvor gezeigte Beispiel noch einmal, allerdings wird innerhalb der Funktion auf die globale Variable `$a` zugegriffen:

```
function a_aendern() {
  global $a;
  $a = 9;   // globales $a hat nun den Wert 9.
}

$a = 7;  // globales $a hat den Wert 7.
a_aendern();
// $a hat jetzt den Wert 9.
```

Eine Funktion kann Parameter entgegennehmen. Zu diesem Zweck müssen Sie bei der Definition die Namen der gewünschten Parametervariablen in den Klammern des Funktionskopfes angeben:

```
function topCell($inhalt) {
  echo "<td valign=\"top\">$inhalt</td>";
}
```

Diese Funktion gibt den übergebenen Wert, der in der Parametervariablen `$inhalt` gespeichert wird, als Tabellenzelle mit der häufig verwendeten vertikalen Ausrichtung oben aus. Parametervariablen haben innerhalb der Funktion dieselbe Bedeutung wie lokale Variablen – der Standardfall beim Funktionsaufruf ist in PHP der *Call by Value*, das heißt die Übergabe eines Wertes ohne Rückbezug auf die aufrufende Stelle. Betrachten Sie dazu das folgende Beispiel:

```
function halbieren($wert) {
  $wert /= 2;
}

$zahl = 9;  // $zahl hat den Wert 9.
halbieren ($zahl);
// $zahl hat weiterhin den Wert 9.
```

Dies gilt jedoch nur für einfache Datentypen; beim Einsatz von objektorientiertem PHP sind die übergebenen Objekte stets Referenzen und keine Kopien. Das folgende kurze Beispiel definiert zunächst eine Klasse namens `Test` mit einem öffentlichen Attribut `$value`, das den Anfangswert 41 erhält. Anschließend wird außerhalb der Klasse eine globale Funktion namens `modifyValue()` definiert, die das Attribut `$value` eines übergebenen Objekts um 1 erhöht. Schließlich wird `Test` instanziiert, `modifyValue()` wird mit der neuen Instanz als Argument aufgerufen, und der Wert des Attributs wird ausgegeben:

```
<?php
class Test {
  public $value = 41;
}

function modifyValue($test) {
  $test->value++;
}

$test = new Test();
modifyValue($test);
echo $test->value."\n";
```

Sie brauchen das Beispiel nicht im Browser aufzurufen, sondern können es auch mit

```
> php Dateiname
```

auf der Konsole starten. Diese Verwendung von PHP wird als *CLI* (*Command Line Interface*) bezeichnet; beachten Sie, dass es dafür eigene Konfigurationsdateien gibt, die Sie gegebenenfalls zusätzlich zu denjenigen für PHP im Webserver ändern müssen.

In jedem Fall lautet die Ausgabe 42, da die Instanz `$test` als Referenz übergeben wird. Näheres zur objektorientierten Programmierung in PHP erfahren Sie in Abschnitt 19.1.2, »Klassen und Objekte«.

Falls Sie für einfache Datentypen einen *Call by Reference* benötigen, also den Wert der Variablen verändern möchten, mit der die Funktion aufgerufen wird, müssen Sie der entsprechenden Parametervariablen bei der Funktionsdefinition ein `&`-Zeichen voranstellen. Wenn Sie die Funktion `halbieren()` folgendermaßen umschreiben, wird `$zahl` also tatsächlich halbiert:

```
function halbieren(&$wert) {
   $wert /= 2;
}

$zahl = 10;  // $zahl hat den Wert 10.
halbieren($zahl);
// $zahl hat nun selbst den Wert 5.
```

Beachten Sie bei einem Call by Reference, dass Sie der Funktion eine Variable übergeben müssen. Ein literaler Wert ist nicht gestattet und erzeugt eine Fehlermeldung.

Sie können Funktionsparametern Standardwerte zuweisen, um sie optional zu machen. Beim Aufruf können diese dann von rechts an weggelassen werden. Hier eine Variante der

Tabellenzellen-Ausgabe, bei der Sie sich die vertikale Ausrichtung aussuchen können; `'top'` ist Standard:

```
function tableCell($inhalt, $valign = 'top') {
  echo ("<td valign=\"$valign\">$inhalt</td>");
}
```

Eine oben ausgerichtete Zelle können Sie dabei ohne Angabe des zweiten Arguments erzeugen:

```
tableCell("Oben ausgerichtet");
```

Es schadet aber auch nichts, den Wert `"top"` dennoch anzugeben:

```
tableCell("Explizit oben ausgerichtet", "top");
```

Eine Funktion kann auch einen Wert zurückgeben, sodass Sie das Ergebnis an der aufrufenden Stelle in einem Ausdruck einsetzen können. Dafür ist die Anweisung `return` zuständig. Sie verlässt die Funktion sofort und gibt den entsprechenden Wert zurück. Hier ein Beispiel:

```
function verdoppeln($wert) {
   return $wert * 2;
}
```

Wenn Sie diese Funktion aufrufen, wird der Wert des übergebenen Arguments verdoppelt. Der fertig berechnete Ausdruck wird anschließend zurückgegeben. Sie können einen Aufruf dieser Funktion in einem beliebigen Ausdruck verwenden. Bevor dieser Ausdruck ausgewertet wird, erfolgt die Ausführung der Funktion, und es wird mit dem zurückgegebenen Wert weitergerechnet. Beispiel:

```
echo verdoppeln($zahl);
```

Diese Anweisung gibt den doppelten Wert der Variablen `$zahl` aus.

Die folgende Funktion zieht die n-te Wurzel aus einer Zahl. Wird der zweite Parameter weggelassen, wählt sie die Quadratwurzel (zweite Wurzel) als Standard:

```
function wurzel($zahl, $n = 2) {
  return pow($zahl, 1 / $n);
}
```

Probieren Sie die Funktion aus, indem Sie zum Beispiel die zweite und die dritte Wurzel aus 64 ziehen:

```
$w = wurzel(64);      // ergibt 8
$w = wurzel(64, 3);   // ergibt 4
```

Beachten Sie, dass eine Funktion mehrere `return`-Anweisungen enthalten kann, die von Fallentscheidungen abhängen. Sie benötigen noch nicht einmal `else`-Blöcke für solche `if`-Anweisungen, weil `return` die Ausführung der Funktion unmittelbar beendet. Die folgende Funktion macht sich das zunutze, um einen übergebenen Ausdruck daraufhin zu überprüfen, ob er ein Integer zwischen 1 und 4 ist:[2]

```
function antworttest($antwort) {
  if (!is_int($antwort)) {
    return FALSE;
  }
  if ($antwort < 1 || $antwort > 4) {
    return FALSE;
  }
  // An dieser Stelle kann $antwort nur OK sein.
  return TRUE;
}
```

In dem Beispiel wird als Erstes überprüft, ob `$antwort` überhaupt ein Integer ist – falls nicht, wird sofort `FALSE` zurückgegeben. Anschließend wird getestet, ob `$antwort` außerhalb des zulässigen Bereichs liegt – in diesem Fall erfolgt ebenfalls die Rückgabe von `FALSE`. Wenn keine der beiden `if`-Abfragen zutrifft, wird automatisch `TRUE` zurückgegeben.

Die verwendete eingebaute Funktion `is_int()` liefert übrigens `true` zurück, wenn es sich bei dem übergebenen Wert um einen Integer handelt, andernfalls `false`. In PHP steht eine Reihe solcher Funktionen zur Verfügung, um die Datentypen von Ausdrücken zu testen, beispielsweise `is_string()`, `is_float()`, `is_numeric()`, `is_array()`, `is_object()` oder `is_null()`, die das übergebene Argument daraufhin überprüfen, ob es ein String, eine Fließkommazahl, eine allgemeine Zahl, ein Array, ein Objekt (Instanz einer Klasse) oder `NULL` (leeres Element) ist.

Die verwandte Funktion `empty()` liefert `TRUE` zurück, wenn das untersuchte Element ein leerer String, die Zahl 0, ein leeres Array, `FALSE` oder `NULL` ist.

Eine ähnliche Aufgabe erfüllen die Funktionen `isset()` und `isunset()`, die überprüfen, ob eine Variable überhaupt jemals definiert wurde. Mithilfe der Anweisung `unset()` können Sie PHP sogar anweisen, eine Variable oder auch ein Element eines Arrays (anhand seines Indexes) vollständig zu vergessen.

2 Wichtig: Benutzereingaben aus Webformularen sind grundsätzlich Strings, und eine Prüfung mit `is_int()` liefert `FALSE` zurück. Allerdings können Sie mit `is_numeric()` überprüfen, ob die Eingabe Zahlen enthält.

19.1.2 Klassen und Objekte

Auch wenn PHP ursprünglich als prozedurale Skriptsprache entwickelt wurde, unterstützt sie seit der Version 4 eine einfache Art der Objektorientierung, die in PHP 5 – und nochmals in Version 5.3 – erheblich erweitert wurde. Sie können Klassen mit Eigenschaften, Methoden und Konstruktoren definieren und voneinander ableiten. Wenn Sie mit diesen Grundbegriffen der Objektorientierung nichts anfangen können, lesen Sie bitte Abschnitt 9.2, »Java«.

Das folgende Beispiel definiert eine Klasse namens `Hyperlink`, die einen HTML-Hyperlink kapselt, erzeugt zwei Objekte dieser Klasse und gibt deren Inhalt aus:

```
<?php

class Hyperlink {
  private $href = '';

  private $caption = '';

  public function __construct($href, $caption = '') {
    $this->href = $href;
    $this->caption = $caption;
  }

  public function __toString() {
    $caption = ($this->caption != '') ? $this->caption : $this->href;
    return sprintf(
      '<a href="%1$s" title="%2$s">%2$s</a>',
      $this->href,
      htmlspecialchars($caption)
    );
  }
}

$linkWithCaption = new Hyperlink(
  'http://www.rheinwerk-verlag.de',
  'Rheinwerk Verlag'
);

$linkWithoutCaption = new Hyperlink('http://www.heise.de');

printf("%s\n%s\n", $linkWithCaption, $linkWithoutCaption);
```

Wie Sie in diesem Beispiel sehen, wird eine Klasse mit dem Schlüsselwort `class` deklariert. Die Methoden sind einfache Funktionen, die in den Block der Klasse hineinverschachtelt

werden; die Veröffentlichungsstufe `public` gibt dabei an, dass die Methoden von außen sichtbar sind. Ein Objekt einer Klasse wird über den Operator `new` erzeugt und ruft, falls vorhanden, den Konstruktor der Klasse auf (mehr darüber erfahren Sie im Folgenden). Der Zugriff auf Methoden und Attribute eines Objekts erfolgt mithilfe des Operators `->`.

Das Beispiel erzeugt die folgende HTML-Ausgabe:

```
<a href="http://www.rheinwerk-verlag.de" title="Rheinwerk Verlag">Rheinwerk Verlag
</a>
<a href="http://www.heise.de" title="http://www.heise.de">http://www.heise.de</a>
```

Eine Besonderheit in der gezeigten Klasse bildet die Methode `__toString()`. Es handelt sich um eine sogenannte *magische Methode*; sie wird automatisch aufgerufen, wenn die Instanz in einem String-Kontext verwendet wird – hier beispielsweise in der Ausgabe mithilfe von `printf()`. Der Konstruktor ist übrigens ebenfalls eine magische Methode; im weiteren Verlauf des Kapitels werden Sie noch weitere kennenlernen.

Beachten Sie in der Methode `__toString()` noch die beiden folgenden Besonderheiten:

- Es wird eine erweiterte Form von `sprintf()` verwendet, um die Ausgabe zu formatieren. Da die Link-Beschriftung `$caption` (explizite Beschriftung oder, falls sie leer ist, der Link selbst) zweimal benötigt wird – einmal für das Attribut `title` und einmal für die Textausgabe –, wird bei den jeweiligen Platzhaltern die Position angegeben: `%1$s` ist das erste Ersetzungselement (durch das `s` als String formatiert) und `%2$s` das zweite. Denken Sie daran, dass Sie das Dollarzeichen in doppelten Anführungszeichen escapen müssten – also beispielsweise `%1\$s` anstelle von `%1$s`.
- Die Link-Beschriftung wird mithilfe von `htmlspecialchars()` escapet – das heißt, eventuell enthaltene HTML-Sonderzeichen werden durch ungefährliche Entity-Referenzen ersetzt; konkret `<` durch `<`, `>` durch `>`, `&` durch `&` und `"` durch `&quot;`. Natürlich bedeutet dies, dass die Link-Beschriftung hier kein verschachteltes HTML enthalten könnte. Wann immer Benutzereingaben oder andere dynamische Inhalte in einem HTML-Kontext ausgegeben werden, sollten Sie diese jedoch in jedem Fall escapen. Alternativ können Sie auch mithilfe von `strip_tags($string)` die enthaltenen HTML-Tags entfernen. Die Funktion nimmt einen optionalen zweiten Parameter an, der die erlaubten HTML-Tags enthält – als String in einem Format wie beispielsweise `"<a><b><i>"`.

Als Attribute einer Klasse gelten automatisch alle Variablen, die in der Klasse, aber nicht innerhalb einer Methode definiert werden. Die Deklaration erfolgt entweder mit einer der Veröffentlichungsstufen `public`, `protected` oder `private` oder mit dem Schlüsselwort `var` (veraltet). Aus einer Methode heraus wird das Schlüsselwort `$this` verwendet, um auf Attribute zuzugreifen. Im zuvor gezeigten Beispiel sind `$href` und `$caption` die beiden privaten Attribute. Das folgende Beispiel zeigt den Umgang mit einem Attribut genauer:


```
class Test {
  private $value = 0;

  public function setValue($value) {
    $this->value = $value;
  }

  public function getValue() {
    return $this->value;
  }
}

$object = new Test();
$obj->setValue(2);
echo $obj->getValue();
```

Zunächst wird das private Attribut `$value` deklariert und mit dem Wert 0 initialisiert. Die öffentliche Methode `setValue()` ändert sie auf einen von außen angegebenen Wert. `getValue()` gibt den Inhalt von `$value` dagegen zurück. Achten Sie darauf, dass Sie den Variablennamen hinter `$this` und dem Zugriffsoperator `->` nicht mit einem weiteren Dollarzeichen versehen.

Falls Sie einen expliziten Konstruktor definieren möchten: Es handelt sich um eine Funktion mit dem speziellen Namen `__construct()` – mit zwei Unterstrichen. Innerhalb des Konstruktors werden typischerweise Initialisierungsarbeiten vorgenommen, die zu Beginn des Lebenszyklus eines Objekts erforderlich sind. Insbesondere können die übergebenen Parameter als Anfangswerte für die Attribute des Objekts gesetzt werden.

Das folgende Beispiel definiert drei Klassen namens `Table`, `Row` und `Cell`; sie definieren HTML-Tabellen sowie deren Zeilen und Zellen. Zwei der drei Klassen besitzen explizite Konstruktoren:

```
<?php

class Table {
  private static $allowedAttributes = array(
    'title', 'style', 'border', 'cellpadding', 'cellspacing'
  );

  private $attributes = array();

  private $rows = array();

  public function __construct($attributes = array()) {
    if (!empty($attributes)) {
```

```
      foreach ($attributes as $name => $value) {
        $this->addAttribute($name, $value);
      }
    }
  }

  public function addAttribute($name, $value) {
    if (in_array($name, self::$allowedAttributes)) {
      $this->attributes[$name] = $value;
    }
  }

  public function addRow(Row $row) {
    $this->rows[] = $row;
  }

  public function __toString() {
    $result = '<table';
    foreach ($this->attributes as $name => $value) {
      $result .= sprintf(
        ' %s="%s"',
        $name,
        htmlspecialchars($value)
      );
    }
    $result .= ">\n";
    foreach ($this->rows as $row) {
      $result .= $row->__toString()."\n";
    }
    $result .= "</table>\n";
    return $result;
  }
}

class Row {
  private $cells = array();

  public function addCell(Cell $cell) {
    $this->cells[] = $cell;
  }

  public function __toString() {
    $result = "<tr>\n";
```



```
    foreach ($this->cells as $cell) {
      $result .= $cell->__toString()."\n";
    }
    $result .= "</tr>";
    return $result;
  }
}

class Cell {
  private static $allowedAttributes = array(
    'title', 'style', 'align', 'valign'
  );

  private $type = 'td';

  private $attributes = array();

  private $contents = '';

  public function __construct($contents, $type = 'td',
      $attributes = array()) {
    $this->contents = $contents;
    if ($type == 'th') {
      $this->type = 'th';
    }

    if (!empty($attributes)) {
      foreach ($attributes as $name => $value) {
        $this->addAttribute($name, $value);
      }
    }
  }

  public function addAttribute($name, $value) {
    if (in_array($name, self::$allowedAttributes)) {
      $this->attributes[$name] = $value;
    }
  }

  public function __toString() {
    $result = '<'.$this->type;
    foreach ($this->attributes as $name => $value) {
      $result .= sprintf(
```

```
            ' %s="%s"',
            $name,
            htmlspecialchars($value)
          );
        }
        $result .= '>';
        $result .= $this->contents;
        $result .= sprintf("</%s>", $this->type);
        return $result;
      }
}

$table = new Table(array('border' => 2, 'cellpadding' => 4));
$row = new Row();
$row->addCell(new Cell('Zeile 1, Zelle 1'));
$row->addCell(new Cell('Zeile 1, Zelle 1'));
$table->addRow($row);
$row = new Row();
$row->addCell(new Cell('Zeile 2, Zelle 1'));
$row->addCell(new Cell('Zeile 2, Zelle 1'));
$table->addRow($row);
echo $table;
```

Zum Testen können Sie alle Klassen und den globalen Code in eine einzelne Datei schreiben. In realen Projekten empfiehlt es sich dagegen, jede Klasse in einer eigenen Datei zu speichern, die den Klassennamen trägt. In diesem Fall müssen die jeweils verwendeten Klassen mithilfe von Include-Anweisungen oder per Autoloader geladen werden (Näheres dazu erfahren Sie später).

Wie Sie sehen, beginnt das Beispiel mit der Klasse `Table`. Diese besitzt die Instanzattribute `$attributes` für die Attribute des `<table>`-Tags und `$rows` für die in der Tabelle enthaltenen Zeilen, also `<tr>`-Elemente.

Zusätzlich ist das statische Attribut `$allowedAttributes` enthalten, das die Liste der zulässigen Attribute enthält. Statische Attribute oder Klassenvariablen werden durch das Schlüsselwort `static` gekennzeichnet, und sie gehören nicht zu einzelnen Instanzen, sondern besitzen für die gesamte Klasse denselben Wert. Der Zugriff erfolgt innerhalb der Klasse mithilfe von `self::$attribut` und von außen – falls das Attribut `public` ist – mit `Klasse::$attribut`. Statische Methoden können übrigens auf dieselbe Weise definiert und angesprochen werden.

Der Konstruktor von `Table` nimmt optional ein assoziatives Array mit Attributen entgegen. Diese werden – falls vorhanden – mithilfe der öffentlichen Methode `addAttribute()`

hinzugefügt. Diese überprüft zunächst anhand von `self::$allowedAttributes`, ob ein Attribut mit dem angegebenen Namen zulässig ist, und fügt es in diesem Fall zur Liste der Attribute hinzu.

Die Methode `addRow()` wird verwendet, um eine Tabellenzeile hinzuzufügen. Interessant ist hier die Angabe des Klassennamens `Row` vor dem eigentlichen Parameter. Es handelt sich um einen sogenannten *Type Hint*; wenn ein Argument übergeben wird, das keine Instanz von `Row` ist, führt dies zu einem Fatal Error. Type Hints sind seit PHP 5.1 verfügbar, und zwar für Klassen und Arrays (Schlüsselwort `array`). Wird einem solchen Parameter der Standardwert `NULL` zugewiesen, ist dieser Wert zusätzlich erlaubt. In Version 5.4 wurden auch Type Hints für einfache Datentypen wie Integer, String etc. eingeführt.

Auch hier wird die Methode `__toString()` verwendet, um die fertige Tabelle als String zurückzugeben. Bei der Ausgabe der Attribute wird nur der Wert escapet, der Name dagegen nicht, weil dieser bereits durch `addAttribute()` gefiltert wurde. Zum Hinzufügen der Zeilen wird wiederum deren `__toString()`-Methode aufgerufen.

Die Klasse `Row` ist wesentlich weniger umfangreich als `Table`, weil hier beispielsweise keine Attribute vorgehalten werden. Instanzen der Klasse enthalten ein Array von Zellen, die mithilfe von `addCell()` hinzugefügt werden können, und `__toString()` erledigt die Ausgabe unter Zuhilfenahme der gleichnamigen Methode in `Cell`.

`Cell` schließlich besitzt wieder Attribute, einen String mit dem Zellinhalt und schließlich das Attribut `$type`, das den Zelltyp (`<td>` oder `<tr>`) kapselt. Der Konstruktor nimmt den Inhalt, den Typ und ein Array mit Attributen entgegen. Die Parameter `$type` und `$attributes` sind optional und können von rechts an weggelassen werden. Beachten Sie, dass Zellen in dieser Implementierung keinen verschachtelten HTML-Code enthalten können, da der Inhalt mithilfe von `htmlspecialchars()` escapet wird.

Zum Schluss wird eine Instanz der Klasse `Table` erzeugt, und es werden Zeilen und Zellen hinzugefügt. Die Ausgabe erfolgt durch ein einfaches `echo $table`, da `__toString()` in diesem Fall wieder automatisch aufgerufen wird. Die HTML-Ausgabe des Beispiels sieht folgendermaßen aus:

```
<table border="2" cellpadding="4">
<tr>
<td>Zeile 1, Zelle 1</td>
<td>Zeile 1, Zelle 1</td>
</tr>
<tr>
<td>Zeile 2, Zelle 1</td>
<td>Zeile 2, Zelle 1</td>
</tr>
</table>
```

Vererbung

Die Vererbung funktioniert in PHP, ähnlich wie in Java, mithilfe des Schlüsselworts `extends`. Das folgende Beispiel leitet die Klassen `Table`, `Row` und `Cell` von der neuen Klasse `HtmlTag` ab, die die gemeinsame Funktionalität wie beispielsweise die Verwaltung der Attribute bereitstellt:

```
<?php

class HtmlTag {
  protected static $allowedAttributes = array('title');

  protected $tagName = '';

  protected $attributes = array();

  protected $content = '';

  public function __construct($tagName, $attributes = array(),
      $content = '') {
    $this->tagName = $tagName;
    if (!empty($attributes)) {
      foreach ($attributes as $name => $value) {
        $this->addAttribute($name, $value);
      }
    }
    $this->content = $content;
  }

  public function addAttribute($name, $value) {
    if (in_array($name, self::$allowedAttributes)) {
      $this->attributes[$name] = $value;
    }
  }

  public function __toString() {
    $result = sprintf('<%s', htmlspecialchars($this->tagName));
    foreach ($this->attributes as $name => $value) {
      $result .= sprintf(
        ' %s="%s"',
        htmlspecialchars($name),
        htmlspecialchars($value)
      );
    }
```



```
    if (!empty($this->content)) {
      $result .= ">\n";
      $result .= $this->content;
      $result .= sprintf("</%s>\n", htmlspecialchars($this->tagName));
    } else {
      $result .= "/>\n";
    }
    return $result;
  }
}

class Table extends HtmlTag {
  private $rows = array();

  public function __construct($attributes = array()) {
    parent::__construct('table', $attributes);
    self::$allowedAttributes = array(
      'title', 'style', 'border', 'cellpadding', 'cellspacing'
    );
  }

  public function addRow(Row $row) {
    $this->rows[] = $row;
  }

  public function __toString() {
    foreach ($this->rows as $row) {
      $this->content .= $row->__toString();
    }

    return parent::__toString();
  }
}

class Row extends HtmlTag {
  private $cells = array();

  public function __construct($attributes = array()) {
    parent::__construct('tr', $attributes);
  }

  public function addCell(Cell $cell) {
    $this->cells[] = $cell;
```

```
  }

  public function __toString() {
    foreach ($this->cells as $cell) {
      $this->content .= $cell->__toString();
    }
    return parent::__toString();
  }
}

class Cell extends HtmlTag {
  public function __construct($content, $type = 'td',
      $attributes = array()) {
    if ($type != 'td' && $type != 'th') {
      $type = 'td';
    }
    parent::__construct($type, $attributes, $content);
    self::$allowedAttributes = array(
      'title', 'style', 'align', 'valign'
    );
  }
}

$table = new Table(array('border' => 1, 'cellpadding' => 4));
$table->addAttribute('style', 'background-color: #ff0;');
$row1 = new Row();
$row1->addCell(new Cell('Hello World!'));
$row1->addCell(new Cell('New Table'));
$table->addRow($row1);
$row2 = new Row();
$row2->addCell(new Cell('Top content', 'td', array('valign' => 'top')));

$row2->addCell(new Cell('Normal content'));
$table->addRow($row2);
echo $table;
```

Die Attribute von `HtmlTag` sind als `protected` deklariert, damit der Zugriff von den abgeleiteten Klassen aus möglich bleibt. Das Überschreiben der Inhalte in den statischen Attributen ist übrigens nicht durch erneute Deklaration möglich, sondern muss – wie in den Klassen `Table` und `Cell` gezeigt –im Konstruktor erfolgen.

Der Konstruktor der Elternklasse wird nicht automatisch aufgerufen, sondern muss implizit in der Form `parent::__construct()` aufgerufen werden, falls er benötigt wird. Entsprechend werden Methoden der Elternklasse mit `parent::methode()` angesprochen.

Die HTML-Ausgabe des Beispiels sieht so aus:

```
<table style="background-color: #ff0;">
<tr>
<td>
Hello World!</td>
<td>
New Table</td>
</tr>
<tr>
<td valign="top">
Top content</td>
<td>
Normal content</td>
</tr>
</table>
```

Magische Methoden

Zuvor wurde bereits die magische Methode `__toString()` gezeigt, die automatisch aufgerufen wird, wenn Sie eine Instanz im String-Kontext verwenden. PHP kennt noch andere magische Methoden. Hier die wichtigsten im Überblick:

- `__get($name)` wird aufgerufen, wenn ein Lesezugriff auf ein nicht vorhandenes öffentliches Attribut einer Instanz erfolgt, also `$instanz->attribut`. Innerhalb der Methode können Sie anhand des in `$name` stehenden Attributnamens entscheiden, was Sie zurückgeben möchten – sogar die nachträgliche Initialisierung von Attributen ist auf diese Weise möglich.
- `__set($name, $value)` kommt entsprechend beim Schreibzugriff auf ein nicht vorhandenes Attribut zum Tragen, das heißt bei einer Wertzuweisung der Form `$instanz >attribut = $value`. Dabei steht der Attributname in `$name` und der Wert in `$value`.
- `__isset($name)` wird aufgerufen, wenn Sie `isset()` auf ein nicht vorhandenes Attribut aufrufen, also `isset($instanz->attribut)`.
- `__unset($name)` schließlich kommt bei einem `unset()`-Aufruf auf ein nicht definiertes Attribut zum Einsatz, das heißt bei `unset($instanz->attribut)`.
- `__call($name, $arguments)` wird beim Zugriff auf eine nicht definierte Methode einer Instanz aufgerufen. Der Methodenname steht dabei in `$name`, während sämtliche Argumente im Array `$arguments` gesammelt werden.

Hier eine kurze Beispielklasse, die alle diese magischen Methoden implementiert, und etwas Hauptprogrammcode, der die Methoden testet:

```
<?php

class Magics {
  private $data = array();

  public function __get($name) {
    printf("Attribut '%s' angefordert.\n", $name);
    return $this->data[$name];
  }

  public function __set($name, $value) {
    printf("Attribut '%s' wird auf Wert '%s' gesetzt.\n", $name, $value);
    $this->data[$name] = $value;
  }

  public function __isset($name) {
    printf("Ist Attribut '%s' gesetzt?\n", $name);
    return isset($this->data[$name]);
  }

  public function __unset($name) {
    printf("Attribut '%s' wird vergessen.\n", $name);
    unset($this->data[$name]);
  }

  public function __call($name, $arguments) {
    printf("Aufruf der Methode %s() mit folgenden Argumenten:\n", $name);
    foreach ($arguments as $argument) {
      printf("- %s\n", $argument);
    }
  }
}

$magics = new Magics();
// __get
$magics->testAttribute = 42;
// __set
printf("%d\n", $magics->testAttribute);
// __isset
var_dump(isset($magics->testAttribute));
// __unset
unset($magics->testAttribute);
// Noch einmal __isset
var_dump(isset($magics->testAttribute));
// __call
$magics->testMethod("Argument 1", "Argument 2", "Argument 3");
```


Das Programm ist für die Ausführung auf der Kommandozeile konzipiert – ansonsten sollten Sie die Zeilenumbrüche durch `<br />` anstelle von `\n` darstellen – und gibt dort Folgendes aus:

```
Attribut 'testAttribute' wird auf Wert '42' gesetzt.
Attribut 'testAttribute' angefordert.
42
Ist Attribut 'testAttribute' gesetzt?
bool(true)
Attribut 'testAttribute' wird vergessen.
Ist Attribut 'testAttribute' gesetzt?
bool(false)
Aufruf der Methode testMethod() mit folgenden Argumenten:
- Argument 1
- Argument 2
- Argument 3
```

Wie Sie sehen, verwendet die Klasse `Magics` das Array-Attribut `$data`, um die durch `__set()` gesetzten Werte zu speichern, wobei die verwendeten Attributnamen als Schlüssel verwendet werden. Die anderen Attribut-Methoden greifen ebenfalls auf dieses Array zu, um den Wert zu einem Schlüssel zurückzuliefern (`__get`), einen bestimmten Schlüssel zu entfernen (`__unset`) beziehungsweise zu überprüfen, ob der Schlüssel existiert (`__isset`). Alle magischen Methoden nehmen eine Debug-Ausgabe vor, damit Sie sehen, dass sie aufgerufen werden. `__call()` enthält sogar nur diese Debug-Ausgabe und erfüllt in dieser Implementierung keinen praktischen Nutzen.

Interfaces der Standard PHP Library (SPL)

Die *Standard PHP Library* (*SPL*) ist eine mit PHP 5 eingeführte und im Laufe der Unterversionen erweiterte Bibliothek zur Lösung alltäglicher Programmieraufgaben. Die SPL gehört zum automatisch vorhandenen Sprachkern von PHP und braucht nicht importiert zu werden. Sie definiert diverse Konstanten, Klassen, Interfaces und Funktionen, die Sie benutzen können. In diesem Unterabschnitt werden drei einfache Interfaces vorgestellt, die sogar schon vor der formalen Definition der SPL in PHP vorhanden waren, aber die Grundlage für viele SPL-Funktionen bilden:

- `ArrayAccess` ermöglicht es Ihnen, Klassen so zu schreiben, dass man auf ihre Instanzen wie auf Arrays zugreifen kann, nämlich mit dem Indexoperator **[]**.
- `Iterator` liefert die nötigen Hilfsmittel, um die »Array-Elemente« einer Instanz der Klasse mit `foreach()` iterieren zu können.
- `Countable` schließlich dient dem Zählen der Array-Elemente mit `count()`.

Ein Interface funktioniert in PHP genauso wie in Java: Es deklariert eine Reihe von Methodensignaturen, und Klassen, die ein Interface implementieren, müssen die entsprechenden Methoden enthalten. Dafür wird – ebenfalls wie in Java – das Schlüsselwort `implements` verwendet.

Tabelle 19.1 enthält eine Übersicht aller Methoden, die Sie für die einzelnen Interfaces implementieren müssen.

Interface	Methode	Aufgabe
`ArrayAccess`	`offsetSet($key, $value)`	den Wert `$value` für den Schlüssel `$key` setzen; wird bei Schreibzugriffen mit Indexoperator aufgerufen
	`offsetGet($key)`	den Wert für den Schlüssel `$key` zurückliefern; wird bei Lesezugriffen mit Indexoperator aufgerufen
	`offsetExists($key)`	`TRUE` zurückliefern, wenn der Schlüssel `$key` existiert, ansonsten `FALSE`; wird bei `isset()` auf Elemente mit Indexoperator aufgerufen
	`offsetUnset($key)`	den Wert für den Schlüssel `$key` löschen; wird bei `unset()` auf Elemente mit Indexoperator aufgerufen
`Countable`	`count()`	die Anzahl der Array-Elemente; wird bei `count()` auf das Objekt aufgerufen
`Iterator`	`rewind()`	den internen Array-Zeiger auf die Anfangsposition setzen; wird bei `reset()` oder am Anfang von `foreach()` aufgerufen
	`current()`	den Wert für die aktuelle Position des Zeigers zurückliefern
	`key()`	den Schlüssel an der Position des Zeigers zurückliefern

Tabelle 19.1 Die Methoden der Interfaces ArrayAccess, Countable und Iterator

Interface	Methode	Aufgabe
	`next()`	den Zeiger um eine Position vorrücken und den Wert für diese Position zurückliefern
	`valid()`	`TRUE` zurückliefern, wenn sich an der aktuellen Zeigerposition ein Wert befindet, ansonsten `FALSE`

Tabelle 19.1 Die Methoden der Interfaces ArrayAccess, Countable und Iterator (Forts.)

Hier ein Beispiel, das die drei Interfaces implementiert und ein privates Array-Attribut für die Datenhaltung verwendet:

```
<?php

class LikeAnArray implements ArrayAccess, Countable, Iterator {
  private $data = array();

  // ArrayAccess-Methoden

  public function offsetSet($offset, $value) {
    $this->data[$offset] = $value;
  }

  public function offsetGet($offset) {
    return $this->data[$offset];
  }

  public function offsetExists($offset) {
    return isset($this->data[$offset]);
  }

  public function offsetUnset($offset) {
    unset($this->data[$offset]);
  }

  // Countable-Methode

  public function count() {
    return count($this->data);
  }
```

```
  // Iterator-Methoden

  public function rewind() {
    reset($this->data);
  }

  public function current() {
    return current($this->data);
  }

  public function key() {
    return key($this->data);
  }

  public function next() {
    return next($this->data);
  }

  public function valid() {
    return $this->current() !== FALSE;
  }
}

$arr = new LikeAnArray();
$arr['beispielschluessel1'] = 'Wert 1';
$arr['beispielschluessel2'] = 'Wert 2';
printf("Es gibt %d Elemente:\n", count($arr));
foreach ($arr as $key => $value) {
  printf("- %s: %s\n", $key, $value);
}
```

Nach den Erläuterungen der einzelnen Methoden in der Tabelle müsste der Code eigentlich selbsterklärend sein. Hinter der eigentlichen Klassendefinition steht wieder ein wenig Beispielcode, der die Verwendung der Klasse demonstriert. Seine Ausgabe lautet:

```
Es gibt 2 Elemente:
- beispielschluessel1: Wert 1
- beispielschluessel2: Wert 2
```

19.1.3 Include-Dateien, Autoloader und Namespaces

Damit die Objektorientierung ihren Hauptnutzen entfalten kann, nämlich den der einfachen Wiederverwendbarkeit von Code, sollten Sie die Klassendefinitionen in externe Dateien schreiben, die Sie zur Laufzeit importieren können. Eine externe Datei wird mithilfe der Anweisungen `require()` oder `include()` importiert. Der Unterschied besteht darin, dass `require()` für unkonditionale Includes am Dateibeginn eingesetzt wird, `include()` dagegen für konditionale innerhalb von Funktionen oder Methoden.

Bei größeren PHP-Projekten empfiehlt es sich, `include_once()` beziehungsweise `require_once()` anstelle des einfachen `include()` oder `require()` zu verwenden – diese Variante sorgt dafür, dass jede benötigte Datei nur einmal inkludiert wird, was Speicher und Rechenzeit spart.

Der Pfad wird durch die zuerst vom Interpreter aufgerufene Datei festgelegt; zusätzlich werden die Verzeichnisse des in der Konfigurationsdatei *php.ini* festgelegten `include_path` durchsucht. Es ist daher keineswegs sichergestellt, dass der Import ohne Pfadangabe funktioniert, nur weil sich zwei Dateien im selben Verzeichnis befinden. Deshalb sollten Sie dem Pfad in einem solchen Fall die Pseudokonstante `__DIR__` voranstellen, die für das Verzeichnis der aktuellen Datei steht.[3]

Das folgende Beispiel importiert die Datei *Table.php*, die sich im selben Verzeichnis befindet wie die aktuelle Datei:

```
require_once(__DIR__.'/Table.php');
```

Obwohl es in PHP keine Pflicht ist wie in Java, ist es auch hier empfehlenswert, Klassen- und Dateinamen übereinstimmen zu lassen. Üblicherweise wird ein mehrgliedriger CamelCase-Klassenname dabei in entsprechenden Unterverzeichnissen gespeichert. Angenommen, die Klasse `Table` hieße `OutputHtmlTable`. Dann wäre es eine gute Idee, sie innerhalb des Projektverzeichnisses als *Output/Html/Table.php* zu speichern.

Wenn Sie solche Namenskonventionen einhalten, können Sie einen *Autoloader* schreiben, der die benötigten Klassen automatisch lädt, sobald sie gebraucht werden. Es handelt sich dabei um eine magische Funktion[4] namens `__autoload()`; die Datei, in der sie steht, muss ihrerseits natürlich durch ein manuelles Include geladen werden. Für die hier behandelte Namenskonvention könnte die Funktion so aussehen, wenn sie im obersten Verzeichnis der Webanwendung steht, das heißt noch über dem Verzeichnis *Output*:

```
function __autoload($className) {
  $classPath = preg_replace('(([A-Z]))', '/$1', $className);
  require_once(__DIR__.$classPath.'.php');
}
```

3 `__DIR__` wurde erst in der PHP-Version 5.3 neu eingeführt; in älteren Versionen können Sie stattdessen `dirname(__FILE__)` schreiben – die Funktion `dirname()` gibt den Verzeichnisteil eines Pfad zurück, und `__FILE__` ist der absolute Pfad der aktuellen Datei. Beachten Sie, dass `__DIR__` und `__FILE__` vorn und hinten mit je zwei Unterstrichen geschrieben werden.

4 Im Unterschied zu `__get()` u. Ä. handelt es sich nicht um eine magische Methode, weil `__autoload()` nicht innerhalb einer Klasse deklariert wird.

Hier werden alle Großbuchstaben mithilfe von `preg_replace()` durch Slashes gefolgt von dem jeweiligen Großbuchstaben ersetzt.

Seit PHP 5.3 sind *Namespaces* verfügbar. Ein Namespace oder Namensraum dient der logischen Unterteilung von Klassenbezeichnern; das Konzept ähnelt beispielsweise den Packages in Java. Mit Namespaces könnten Sie die Klasse `Output\Html\Table` nennen, das heißt, sie läge im Unter-Namespace `Html` des Namespaces `Output`. Dazu muss der Namespace als erste Anweisung im Skript deklariert werden:

```
<?php
namespace Output\Html;

class Table {
  // ...
}
```

Der Autoloader müsste dann wie folgt angepasst werden:

```
function __autoload($className) {
  $classPath = str_replace('\\', '/', $className);
  require_once(__DIR__.$classPath.'.php');
}
```

Hier genügt das einfache `str_replace()` anstelle von `preg_replace()`, weil lediglich die Backslashes durch Slashes ersetzt werden.

Die SPL enthält eine Funktion namens `spl_autoload_register()`, mit der Sie beliebig viele Autoloader-Funktionen registrieren können. Diese werden als Callbacks angegeben, also in einer der folgenden Formen:

- `spl_autoload_register("funktionsname")` für eine alleinstehende Funktion außerhalb einer Klasse,
- `spl_autoload_register($objekt, "funktionsname")` für eine Methode eines gegebenen Objekts (einschließlich `$this` für das aktuelle Objekt),
- `spl_autoload_register("klassenname", "funktionsname")` für eine statische Methode der angegebenen Klasse oder
- `spl_autoload_register(function($klassanname) { ... })` als eingebettete anonyme Funktion, verfügbar seit PHP 5.3.

Der Aufbau der Autoloader-Funktion entspricht dem von `__autoload()`, das heißt, es wird der angeforderte Klassenname als Argument übergeben, und die Funktion soll die passende Datei inkludieren.

Hinter dem Callback können zwei optionale Boolean-Argumente folgen, die standardmäßig `FALSE` sind:

- Wenn Sie das Argument `$throw` auf `TRUE` setzen, löst `spl_autoload_register()` eine Exception aus, die Sie mit `try/catch` abfangen können, falls die angegebene Funktion nicht registriert werden kann (dies funktioniert in PHP im Wesentlichen wie in Java; Details dazu finden Sie in Kapitel 9, »Grundlagen der Programmierung«).
- Das zweite Argument, `$prepend`, sorgt dafür, dass die neu registrierte Funktion Vorrang vor den bisher registrierten haben soll – standardmäßig ist dies umgekehrt, sodass PHP die Funktionen in der Reihenfolge ihrer Registrierung durchprobiert, bis die gewünschte Klasse gefunden wird.

19.1.4 Webspezifische Funktionen

Die Hauptaufgabe von PHP ist das Erstellen dynamischer Webanwendungen. Zu diesem Zweck wird eine Reihe spezieller Funktionen und Fähigkeiten angeboten. Dazu gehören das Auslesen von Formulardaten, das Erzeugen und Lesen von Cookies oder auch das Session-Tracking. Letzteres steuert eine Funktionalität bei, die HTTP von Haus aus nicht besitzt: Es ermöglicht das Verfolgen der Aktivitäten eines Benutzers über mehrere besuchte Seiten hinweg. Dies ist wichtig für Warenkorbsysteme oder andere Anwendungen, bei denen eine Reihe aufeinanderfolgender Aktivitäten registriert werden muss.

Formulardaten auslesen

HTML-Formulare wurden im vorangegangenen Kapitel vorgestellt. In PHP stehen empfangene Formulardaten je nach Methode in einem der beiden Arrays `$_GET` oder `$_POST` zur Verfügung. Ein `name=wert`-Paar aus dem Formular wird dabei als `$_GET['name']` beziehungsweise `$_POST['name']` angesprochen.

Betrachten Sie als Beispiel das folgende kurze Formular:

```
<form action="auswert.php" method="get">
  Name: <input type="text" name="user" /><br />
  E-Mail: <input type="text" name="mail" /><br />
  <input type="submit" value="Abschicken" />
</form>
```

Klickt ein Benutzer den Button Abschicken an, wird das PHP-Skript *auswert.php* aufgerufen. Da die HTTP-Methode `GET` ausgewählt wurde, befinden sich die Formulardaten in dem Array `$_GET`. Der folgende Codeblock liest sie und gibt sie anschließend aus:

```
$user = $_GET['user'];
$mail = $_GET['mail'];
echo ("Hallo $user.<br />");
echo ("Best&auml;tigung geht an $mail.");
```

Sicherer ist es, wenn Sie zuerst überprüfen, ob die gewünschten Formularfelder überhaupt Daten enthalten. Prüfen Sie dazu zunächst mit `isset()`, ob ein erwartetes Feld überhaupt gesetzt ist, und anschließend, ob es einen Wert besitzt. Dazu können Sie beispielsweise folgende praktische Funktion verwenden:

```
function readParam($field, $default = '') {
  // Variable zunächst auf Default-Wert setzen
  $var = $default;
  if (isset($_POST[$field] && $_POST[$field] != '') {
    $var = $_POST[$field];
  } elseif (isset($_GET[$field] && $_GET[$field] != '') {
    $var = $_GET[$field];
  }
  // Ermittelten Wert zurückgeben
  return $var;
}
```

Die Funktion erwartet als Parameter den Namen des auszulesenden Feldes sowie optional einen Default-Wert, der geliefert wird, falls das Feld nicht verfügbar oder leer ist. Die beiden Felder aus dem zuvor gezeigten Formularbeispiel lassen sich mithilfe dieser Funktion folgendermaßen lesen:

```
// User auslesen, Standardwert "Anonymous"
$user = readParam("user", "Anonymous");
// E-Mail auslesen, Standardwert "" (automatisch)
$mail = readParam ("mail");
```

Schreiben Sie diese Funktion einfach in einen PHP-Block am Beginn jeder Datei, die Formulardaten auslesen muss, oder speichern Sie sie in einer Include-Datei. Bei einer modernen, objektorientierten Anwendung können Sie sie als Methode in die Basisklasse Ihres Frameworks einbauen.

Bei der HTTP-Methode `POST` und einigen anderen befinden sich die Formulardaten im Body der HTTP-Anfrage (siehe Kapitel 14, »Server für Webanwendungen«). Sie werden nur im Array `$_POST` bereitgestellt, wenn der Body einen der MIME-Types `application/x-www-form-urlencoded` (Standardformular) oder `multipart/form-data` (Formular mit eventuellen Datei-Uploads) hat. Andernfalls handelt es sich um Daten wie XML, reinen Text oder dergleichen, und solche Daten werden am einfachsten folgendermaßen eingelesen:

```
$data = file_get_contents("php://input");
```

Die Funktion `file_get_contents()` öffnet eine Ressource zum Lesen, liest ihren gesamten Inhalt in einem Durchgang ein und schließt die Ressource dann wieder. Als Ressource können

Sie sowohl einen lokalen Dateipfad auf dem Server als auch eine lesbare URL verwenden; `php://input` ist dabei eine spezielle URL.

Datei-Uploads

Bereits im vorangegangenen Kapitel haben Sie das spezielle HTML-Formularfeld kennengelernt, das Benutzern den Versand einer lokalen Datei mit dem Formular ermöglicht. Es handelt sich um ein `<input>`-Element mit dem Attribut `type="file"`. Damit der Upload funktionieren kann, muss das Formular mit der HTTP-Methode `POST` und der speziellen MIME-Codierung `multipart/form-data` versandt werden. PHP verarbeitet optional ein zusätzliches Formularfeld mit der Bezeichnung `MAX_FILE_SIZE`, das die maximal zulässige Dateigröße in Bytes enthält und meist als Hidden-Formularfeld gesetzt wird. Sollte der angegebene Wert größer sein als die *php.ini*-Einstellung `upload_max_filesize` (siehe Kapitel 14, »Server für Webanwendungen«), wird er ignoriert.

Das folgende Beispiel definiert ein Formular für den Versand einer GIF-Bilddatei (Dies wird erst auf dem Server geklärt!) von maximal 400 Kilobyte Größe:

```
<form action="bildupload.php" method="post" enctype="multipart/form-data">
  <input type="hidden" name="MAX_FILE_SIZE" value="409600" />
  Bitte eine GIF-Bilddatei w&auml;hlen:<br />
  <input type="file" name="bild" />
  <br />
  <input type="submit" value="Abschicken" />
</form>
```

Die Datei lässt sich in PHP über das globale Array `$_FILES` auslesen; der Index ist der Feldname. Das entsprechende Array-Element ist wiederum ein Array mit folgenden Elementen:

- `tmp_name`: der temporäre Dateiname, unter dem der Server die hochgeladene Datei gespeichert hat
- `type`: der MIME-Type der Datei (aus der Dateiendung oder aus dem `Content-Type`-Header des Formularabschnitts ermittelt)
- `size`: Größe der Datei oder 0 bei einem Fehler
- `name`: ursprünglicher Pfad und Dateiname der Datei auf dem Clientrechner
- `error`: Fehlercode bei einem Upload-Fehler. Die symbolischen Konstanten entsprechen dabei den in Klammern angegebenen numerischen Werten:
  - `UPLOAD_ERR_OK` (0): Es ist kein Fehler aufgetreten.
  - `UPLOAD_ERR_INI_SIZE` (1): Datei ist größer als die in *php.ini* angegebene Höchstgrenze `upload_max_filesize`.
  - `UPLOAD_ERR_FORM_SIZE` (2): Datei ist größer als die mit dem Formular übertragene Höchstgrenze `MAX_FILE_SIZE`.

- `UPLOAD_ERR_PARTIAL` (3): Die Datei wurde nur zum Teil hochgeladen.
- `UPLOAD_ERR_NO_FILE` (4): Es wurde gar keine Datei hochgeladen.

Das folgende PHP-Fragment kopiert die hochgeladene Datei unter dem Namen *bild.gif* in das Verzeichnis des Skripts selbst, falls es sich um eine Bilddatei vom Typ GIF handelt:

```
if (is_uploaded_file($_FILES['bild']['tmp_name'])) {
  if ($_FILES['bild']['type'] == 'image/gif') {
    move_uploaded_file (
      $_FILES['bild']['tmp_name'],
      __DIR__.'/bild.gif'
    );
  } else {
      echo "Falscher Dateityp!<br />";
  }
} else {
  echo "Upload-Fehler (Datei zu gro&szlig;)?<br />";
}
```

Beachten Sie, dass das Verzeichnis, in das Sie hochgeladene Dateien kopieren, für den User des Webservers beschreibbar sein muss.

Sessions

Eines der interessantesten Web-Features von PHP ist das *Session-Tracking*. HTTP ist ein zustandsloses Protokoll, das heißt, jede Clientanfrage steht für sich allein. Für größere Webanwendungen, die sich über viele Seiten erstrecken, wird deshalb ein Verfahren zur Datenweitergabe an nachfolgende Anfragen benötigt. PHP stellt sehr praktische Funktionen zur Verwaltung von Session-Daten bereit.

Beachten Sie, dass Sie Session-Befehle in einen PHP-Block zu Beginn Ihres Skripts setzen müssen: Session-Daten werden als Cookies übertragen, wenn der Client dies zulässt, ansonsten über eine Session-ID im Query-String. Beides erfordert, dass das eigentliche geparste Dokument noch nicht begonnen hat, sodass mit dem ersten Zeichen Ihres Skripts die Sequenz `<?php` einsetzen muss.

Jedes Skript, das auf Session-Daten zugreifen soll, muss zunächst folgende Anweisung enthalten:

```
session_start();
```

Anschließend können Sie das spezielle Array `$_SESSION` mit Name-Wert-Paaren bestücken oder Werte von einer zuvor besuchten Seite daraus lesen. Aus dem soeben erwähnten Grund muss das Lesen vor dem Schreibzugriff erfolgen:


```
// Stückzahl auslesen
$stueckzahl = $_SESSION['stueck'];
// GET-Formularfeld zahlungsart auslesen
$zahlungsart = $_GET['zahlungsart'];
// Zur zukünftigen Verwendung in Session speichern
$_SESSION['zahlungsart'] = $zahlungsart;
```

Cookies

Für die meisten Anwendungen sollten Sie die Verwendung von Session-Daten derjenigen von Cookies vorziehen. Der einzige Nachteil von Sessions besteht darin, dass die Daten beim nächsten Besuch eines Benutzers nicht mehr zur Verfügung stehen. Dies lässt sich oft durch eine persönliche Anmeldung und die Speicherung der Daten in einer Datenbank beheben.

Dennoch gibt es Fälle, in denen Cookies praktischer sind. Es ist für einen Besucher unter Umständen nützlich, Einstellungen, die er auf einer Seite vorgenommen hat, beim nächsten Besuch wieder vorzufinden. Sie sollten allerdings niemals eine Webanwendung programmieren, die von Cookies abhängt: Aufgrund des Missbrauchs, der häufig zu Werbe- und User-Tracking-Zwecken mit Cookies getrieben wird, schalten viele Benutzer Cookies generell ab.

In PHP wird ein Cookie mithilfe der folgenden Funktion gesetzt:

```
setcookie($name, $wert[, $verfallsdatum[, $pfad [, $domain[, $secure[, $httponly]]]]])
```

Hier eine Übersicht über die Parameter:

- `$name`: der Name des Cookies, über den es später wieder abgefragt werden kann
- `$wert`: der Wert, den Sie für den entsprechenden Namen festlegen möchten
- `$verfallsdatum`: der Verfallszeitpunkt für das Cookie in Sekunden seit EPOCH. Sie können einfach `time() + $sekundenzahl` verwenden, da die Funktion `time()` den aktuellen Zeitpunkt in diesem Format liefert. Lassen Sie diesen Wert weg, erzeugen Sie automatisch ein Session-Cookie, das nur während der aktuellen Clientsitzung gilt.
- `$pfad`: der URL-Pfad, unter dem das Cookie zur Verfügung steht. Wenn Sie den Wert nicht angeben, ist das Cookie nur in dem Verzeichnis verfügbar, in dem Sie es gesetzt haben (also das Verzeichnis der ersten PHP-Datei, die durch die HTTP-Anfrage aufgerufen wurde).
- `$domain`: die Domain, unter der das Cookie gültig ist. Standardwert ist der vollständige Hostname, von dem das Cookie gesetzt wurde. Haben Sie aber beispielsweise die Subdomains *www.example.com* und *cgi.example.com*, sollten Sie den Wert *.example.com* setzen, damit das Cookie für beide Domains gilt.
- `$secure`: Wenn Sie diesen Wert auf `TRUE` setzen, wird das Cookie nur über eine gesicherte HTTPS-Verbindung übertragen.
- `$httponly` (seit PHP 5.2.0): Der Wert `TRUE` sorgt dafür, dass das Cookie nur über eine HTTP(S)-Verbindung ausgelesen und verändert werden kann und beispielsweise nicht per JavaScript. Dies erschwert Cross-Site-Scripting-Attacken (XSS).

Wie bereits erwähnt, werden Cookies als HTTP-Header gesetzt. Deshalb muss dieser Befehl – genau wie die Session-Anweisungen – vor jeglichem HTML-Code in der PHP-Datei stehen. Die folgende Anweisung setzt ein sieben Tage gültiges Cookie namens `lastvisit`, dessen Wert die aktuelle Uhrzeit ist:

```
setcookie("lastvisit", time(), time() + 7 * 24 * 60 * 60);
```

Die Cookies, die der Client bei der Anfrage mitgeschickt hat, können Sie aus dem globalen Array `$_COOKIE` lesen. Dieses Beispiel liest das Cookie `lastvisit` und speichert seinen Wert in einer gleichnamigen Variablen:

```
$lastvisit = $_COOKIE['lastvisit'];
```

Diese Variable lässt sich im weiteren Verlauf des Skripts etwa so verwenden:

```
$lastvisitformat = date("d.m.Y, H:i", $lastvisit);
echo "Ihr letzter Besuch: $lastvisitformat <br />";
```

Die Funktion `date($format, $timestamp)` erzeugt formatierte Datumsangaben. Die Zeitangabe ist ein Unix-Timestamp, also ein Integer, der die Sekunden seit EPOCH (01.01.1970, 00:00 Uhr UTC) zählt. Im Format-String können unter anderem folgende Komponenten vorkommen:

- `j` ist der Tag im Monat.
- `d` steht ebenfalls für den Tag, gibt ihn allerdings zweistellig an (aus 9 wird etwa `09`).
- `w` gibt den Wochentag als Zahlenwert an, wobei 0 für Sonntag steht, 1 für Montag, bis 6 für Samstag.
- `n` ist der numerisch angegebene Monat.
- `m` ist noch einmal die Nummer des Monats, aber zweistellig.
- `y` gibt das zweistellige Jahr an.
- `Y` bedeutet ebenfalls das Jahr, allerdings vierstellig.
- `G` ist die Stunde im 24-Stunden-Format.
- `H` ist ebenfalls die Stunde im 24-Stunden-Format, jedoch mit erzwungener Zweistelligkeit.
- `i` gibt die Minuten zweistellig an.
- `s` steht für die zweistelligen Sekunden.

19.1.5 Zugriff auf MySQL-Datenbanken

Es gibt drei verschiedene PHP-Schnittstellen für den Zugriff auf MySQL-Datenbanken: die klassische `mysql`-Schnittstelle, die neuere Entwicklung `mysqli` und die Abstraktionsschicht *PHP Data Objects* (PDO). `mysqli` und PDO existieren nur in PHP 5 und arbeiten nur mit MySQL ab Version 4.1 zusammen. Diese Einführung beschränkt sich auf die Gegenüberstellung von `mysql` und `mysqli`.

Eine Testdatenbank

Die folgenden einfachen MySQL-Beispiele beziehen sich alle auf eine kleine Datenbank namens *programming_languages*, in der eine einzige Tabelle mit dem Namen *languages* enthalten ist. Die Felder dieser Datenbanktabelle sehen Sie in Tabelle 19.2. Damit die Beispiele nicht allzu umfangreich werden, ist die Tabelle sehr einfach gehalten.

Feldbezeichnung	Datentyp
language_id	`int`, `auto_increment` (Primärschlüssel)
language_name	`varchar(30)`
language_architecture	`enum('imperative', 'oop', 'other')`
language_implementation	`enum('compiler', 'interpreter', 'VM', 'mixed')`
language_system	`set('Unix', 'Windows', 'other')`
language_description	`varchar(255)`
language_year	`year`

Tabelle 19.2 Struktur der Beispieldatenbanktabelle »programming_languages«

Sie können diese Tabelle mit phpMyAdmin oder einem anderen grafischen Datenbankverwaltungstool anlegen. Alternativ öffnen Sie die `mysql`-Konsole (siehe Kapitel 13, »Datenbanken«) und tippen nacheinander folgende SQL-Befehle ein (die Ausgabe des Clients wurde hier weggelassen):

```
mysql> create database programming_languages;
mysql> use programming_languages
mysql> create table languages (
    ->   language_id int auto_increment,
    ->   language_name varchar(30),
    ->   language_architecture enum('imperative', 'oop',
    ->   'other'),
    ->   language_implementation enum('compiler', 'interpreter',
    ->   'VM', 'other'),
    ->   language_system set('Unix', 'Windows', 'sonstige'),
    ->   language_description varchar(255),
    ->   language_year year,
    ->   primary key(language_id),
    ->   index(language_name)
    -> );
```

`create database` und `create table` sind SQL-Standardbefehle zum Erzeugen einer neuen Datenbank beziehungsweise Tabelle; `use <Datenbank>` ist dagegen ein interner Befehl des MySQL-Clients, der zur Auswahl der gewünschten Datenbank verwendet wird.

Als Nächstes benötigt die Datenbank einige Beispielinhalte. Sie können entweder die Werte aus Tabelle 19.3 übernehmen oder Ihre eigenen benutzen.

Name	Arch.	Impl.	System	Description	Year
C	imperative	compiler	Unix, Windows	älteste weitverbreitete Sprache; Syntax in vielen Sprachen verbreitet	1970
C++	oop	compiler	Unix, Windows	Weiterentwicklung von C mit OOP-Fähigkeiten	1983
Java	oop	VM	Unix, Windows	OOP-Sprache mit Multi-Plattform-VM	1995
C#	oop	mixed	Windows	Sprache für die CLR des .NET Frameworks	2000
Python	oop	interpreter	Unix, Windows	Multiparadigmen-Skriptsprache mit vielen bequemen Modulen	1991
Perl	imperativ	interpreter	Unix, Windows	Skriptsprache für Admin- und Textbearbeitungsaufgaben; neuere Versionen bieten (umständliche) OOP.	1987
Ruby	oop	interpreter	Unix, Windows	Skriptsprache mit fast allen Perl-Features sowie sauberer, moderner OOP-Implementierung	1993

Tabelle 19.3 Beispieldatensätze für die Datenbanktabelle »languages«

Verwenden Sie am besten einen grafischen Client, um die Beispielwerte einzugeben. Manuelle `INSERT`-Abfragen gemäß der Anleitung in Kapitel 13, »Datenbanken«, sind ebenfalls möglich, aber recht aufwendig und fehlerträchtig. Alternativ können Sie auch die Datei *mklanguages.sql* aus der Listing-Sammlung zu diesem Buch (*http://buecher.lingoworld.de/fachinfo/listings*) ausführen. Dies funktioniert im MySQL-Kommandozeilen-Client beispielsweise folgendermaßen:

```
mysql> source mklanguages.sql
```


Die mysql-Schnittstelle

Hier wird zunächst einmal die Vorgehensweise für die klassische Schnittstelle vorgestellt; im weiteren Verlauf des Kapitels lernen Sie dieselben Anweisungen für `mysqli` kennen.

Beachten Sie, dass diese Schnittstelle seit PHP 5.5 als veraltet gilt und in zukünftigen Versionen vermutlich abgeschafft wird. Die `mysqli`-Schnittstelle ist aber ohnehin angenehmer zu benutzen.

Als Erstes müssen Sie eine Verbindung zum MySQL-Server herstellen. Dies funktioniert mithilfe der klassischen Schnittstelle schematisch so:

```
$connID = mysql_connect ($host[, $user[, $passwort]]);
```

Angenommen, der Datenbankserver läuft auf demselben Host wie der Webserver, der Benutzername ist *dbuser* und das Passwort »geheim«. Dann wird der Befehl folgendermaßen aufgerufen:

```
$connID = mysql_connect("localhost", "dbuser", "geheim");
```

Sie sollten es bei einem Produktivsystem dringend vermeiden, anonyme Zugriffe auf den Datenbankserver zuzulassen. Es sollte also keine Webanwendung geben, bei der ein Zugriff auf MySQL ohne Benutzername und Passwort möglich ist.

Die Verbindungskennung `$connID` ist übrigens im Erfolgsfall ein positiver Integer, andernfalls erhält die Variable den Wert 0. Deshalb könnte die konkrete Verwendung folgendermaßen aussehen:

```
$connID = mysql_connect("localhost", "dbuser", "geheim");
if ($connID) {
  // Datenbankverarbeitung ...
} else {
  echo("Fehler: Kein Datenbankzugrif möglich!<br />");
}
```

Als Nächstes müssen Sie die Datenbank auswählen, auf die zugegriffen wird. Dies geschieht mit dem Befehl `mysql_select_db()`. Der erste Parameter ist der Name der gewünschten Datenbank. Optional geben Sie die Verbindungskennung an – dies ist aber nur erforderlich, falls mehrere Datenbankverbindungen bestehen. Beispiel:

```
mysql_select_db("programming_languages");
// Falls mehrere Datenbankverbindungen geöffnet sind:
mysql_select_db("programming_languages", $connID);
```

Datenbankfehler lassen sich jeweils mit den Funktionen `mysql_errno()` und `mysql_ error()` überprüfen, die eine Fehlernummer beziehungsweise eine Fehlermeldung zurückliefern. Falls kein Fehler aufgetreten ist, liefert `mysql_errno()` den Wert 0 zurück und `mysql_error()`

einen leeren String. Auch bei diesen Funktionen können Sie optional die Verbindungs-ID angeben. Idealerweise überprüfen Sie sie nach jeder Datenbankoperation. Beispiel:

```
mysql_select_db('nicht_vorhandene_datenbank');
if (mysql_errno() > 0) {
  $errno = mysql_errno();
  $error = mysql_error();
  echo "Datenbankfehler $errno: $error<br />";
} else {
  // Datenbankoperation durchführen
}
```

Die tatsächliche Fehlermeldung sollten Sie nur während der Entwicklung ausgeben, weil sie den Enduser nur verwirren würde oder sogar mögliche Schwachstellen für Angriffe offenbaren könnte.

Nun können Sie der Datenbank SQL-Abfragen senden. Dies geschieht mithilfe der Funktion `mysql_query($abfrage[, $connID])`. Die Verbindungskennung braucht auch hier wieder nur dann angegeben zu werden, falls Sie mehrere Datenbankverbindungen gleichzeitig verwenden. Der Rückgabewert ist eine Ergebniskennung, deren Inhalt Sie mit weiteren Funktionen auslesen können, oder `FALSE`, wenn die Abfrage fehlschlägt. Im letzteren Fall liefert die Funktion `mysql_error()` wieder eine Fehlerbeschreibung.

Hier ein Beispiel für eine Auswahlabfrage; sie wählt alle Programmiersprachen aus, deren Name mit »C« beginnt:

```
$result = mysql_query("SELECT language_name, language_architecture,
                              language_implementation, language_system,
                              language_description, language_year
                         FROM languages
                        WHERE language_name LIKE 'C%'");
```

Die einzelnen Zeilen des Ergebnisses lassen sich mithilfe von `mysql_fetch_row($result)`, `mysql_fetch_assoc($result)` oder `mysql_fetch_array($result)` auslesen. Trotz der unterschiedlichen Bezeichnungen liefern alle drei ein Array. Der Unterschied ist, dass die Indizes bei `mysql_fetch_row()` numerisch und bei `mysql_fetch_assoc()` die eigentlichen Feldnamen sind; bei `mysql_fetch_array()` wird standardmäßig sogar jeder Datensatz mit beiden Indexarten zurückgeliefert. Sie können diese Funktionen in einer `while`-Schleife aufrufen, weil sie `FALSE` zurückgeben, falls kein weiterer Datensatz mehr vorhanden ist. Hier ein Beispiel für `mysql_fetch_row()`:

```
echo("<table border=\"2\">");
echo("<tr><th>Sprache</th>");
echo("<th>Architektur</th>");
echo("<th>Implementierung</th>");
```



```
echo("<th>Systeme</th>");
echo("<th>Kurzinfo</th>");
echo("<th>Erscheinungsjahr</th></tr>");
while ($record = mysql_fetch_row($result)) {
  $name = $record[0];
  $architecture = $record[1];
  $implementation = $record[2];
  $systems = $record[3];
  $description = $record[4];
  $year = $record[5];
  echo("<tr><td>$name</td>");
  echo("<td>$architecture</td>");
  echo("<td>$implementation</td>");
  echo("<td>$systems</td>");
  echo("<td>$description</td>");
  echo("<td>$year</td></tr>");
}
echo("</table>");
```

Eine kürzere Fassung lässt sich mithilfe der Funktion `list()` erreichen, die die Elemente eines Arrays einer Liste von Einzelvariablen zuweisen kann:

```
while (list($name, $architecture, $implementation, $system, $description, $year)
       = mysql_fetch_row($result)) {
  echo("<tr><td>$name</td>");
  echo("<td>$architecture</td>");
  echo("<td>$implementation</td>");
  echo("<td>$system</td>");
  echo("<td>$description</td>");
  echo("<td>$year</td></tr>");
}
```

Mit `mysql_fetch_assoc()` sieht der relevante Ausschnitt der `while()`-Schleife dagegen so aus:

```
while ($record = mysql_fetch_assoc($result)) {
  $name = $record['language_name'];
  $architecture = $record['language_architecture'];
  $implementation = $record['language_implementation'];
  $system = $record['language_system'];
  $description = $record['language_description'];
  $year = $record['language_year'];
  // ... wie gehabt ...
}
```

Bei den unterschiedlichen Arten von Änderungsabfragen (`UPDATE`, `INSERT` und `DELETE`) gibt es natürlich kein Ergebnis in Form zurückgegebener Datensätze. Stattdessen können Sie über die Funktion `mysql_affected_rows()` erfahren, wie viele Datensätze geändert (beziehungsweise hinzugefügt oder gelöscht) wurden. Beispiel:

```
// Neue Sprache hinzufügen
$name = "Scala";
$architecture = "other";
$implementation = "VM";
$system = "Unix,Windows";
$description = "Multiparadigmen-Skriptsprache mit vielen funktionalen Aspekten";
$jahr = 2003;
$result = mysql_query
         ("INSERT INTO sprachen (language_name, language_architecture,
                                 language_implementation,
                                 language_system, language_description,
                                 language_year)
                 VALUES ('$name', '$architecture',
                         '$implementation', '$system', '$description',
                         '$year')");
// Hat es funktioniert?
if (mysql_affected_rows() == 1) {
  echo("Sprache erfolgreich hinzugef&uuml;gt.<br />");
} else {
  echo("Sprache konnte nicht eingef&uuml;gt werden.<br />");
}
```

Wenn die Tabelle, in die Sie einen Datensatz einfügen, einen einspaltigen Primärschlüssel mit Auto-Increment besitzt, können Sie übrigens auch die Funktion `mysql_insert_id()` verwenden, um die zuletzt eingefügte ID zu erhalten, anstatt die Anzahl der eingefügten Datensätze zu überprüfen. Der Vorteil ist, dass Sie damit gleich die ID des neuen Datensatzes zur Hand haben. Wenn das Einfügen fehlschlägt, liefert diese Funktion `FALSE` zurück.

Wenn Sie eine Abfrage durchführen, die besonders viele Datensätze liefert, sollten Sie anschließend den Speicher leeren, den das Ergebnis belegt hat:

```
mysql_free_result($result);
```

Zum Schluss sollten Sie die Datenbankverbindung ordnungsgemäß schließen:

```
mysql_close($connID);
     // $connID entfällt, wenn eindeutig
```


Die mysqli-Schnittstelle

Die neuere Datenbankschnittstelle `mysqli` kann sowohl prozedural als auch objektorientiert verwendet werden. Wenn Sie die prozedurale Schreibweise verwenden, unterscheiden sich ihre Funktionen nicht sehr stark von der gerade vorgestellten klassischen Schnittstelle `mysql`. Sie sollten den objektorientierten Zugriff bevorzugen. Hier wird für jede Verbindung ein eigenes `mysqli`-Objekt erzeugt, dessen Methoden und Eigenschaften Sie für die Datenbankoperationen einsetzen können.

Hier sehen Sie das gesamte Beispiel: Auslesen aller mit »C« beginnenden Sprachen und Hinzufügen einer neuen – in objektorientierter `mysqli`-Syntax. Die ausführlichen Kommentare erläutern die Unterschiede:

```
// Datenbankverbindung herstellen,
// Standarddatenbank wählen
// (nun in einem Schritt)
$conn = new mysqli ("localhost", "dbuser",
                    "geheim", "programming_languages");
// Abfragen sind Instanzmethoden
// des mysqli-Objekts
$result = $conn->query("SELECT language_name, language_architecture,
                               language_implementation,
                               language_system, language_description,
                               language_year
                          FROM languages
                         WHERE language_name LIKE 'C%'");
echo("<table border=\"2\">");
echo("<tr><th>Sprache</th>");
echo("<th>Architektur</th>");
echo("<th>Implementierung</th>");
echo("<th>Systeme</th>");
echo("<th>Kurzinfo</th>");
echo("<th>Erscheinungsjahr</th></tr>");
// Das Auslesen der Datensätze ist
// eine Instanzmethode des Result-Objekts
while (list($name, $architecture, $implementation, $system, $description, $year)
       = $result->fetch_row()) {
  echo("<tr><td>$name</td>");
  echo("<td>$architecture</td>");
  echo("<td>$implementation</td>");
  echo("<td>$system</td>");
  echo("<td>$description</td>");
  echo("<td>$year</td></tr>");
}
```

```
echo("</table>");
// Neuen Datensatz einfügen
$name = "PHP";
$archiecture = "oop";
$implementation = "Interpreter";
$system = "Unix,Windows";
$description = "Objektorientierte Web-Skriptsprache";
$year = 1995;

$conn->query
   ("INSERT INTO languages (language_name, language_architecture,
                            language_implementation,
                            language_system, language_description,
                            language_year)
                    VALUES ('$name', '$architecture',
                            '$implementation', '$system', '$description',
                            '$year')");
// Hat es funktioniert?
// (affected_rows ist nun eine Eigenschaft
//  des mysqli-Objekts; insert_id ebenfalls)
if ($conn->affected_rows == 1) {
  echo ("Sprache erfolgreich hinzugef&uuml;gt.<br />");
} else {
  echo ("Sprache konnte nicht eingef&uuml;gt werden.<br />");
}
// Verbindung schließen
$conn->close();
```

Eine Klasse für vereinfachte Datenbankzugriffe

Bisher wurde gezeigt, wie Sie jede Abfrage manuell durchführen. Für Projekte ab einer gewissen Größe ist dies nicht mehr empfehlenswert. Stattdessen sollten Sie entweder eines der bestehenden PHP-Frameworks wie Zend oder Symfony nutzen oder aber eine eigene Klasse oder Klassenbibliothek für den Datenbankzugriff und andere wiederkehrende Aufgaben schreiben. Die folgende Klasse kapselt eine `mysqli`-Datenbankverbindung und stellt einige bequeme Methoden zur Verfügung:[5]

```
<?php
/*
* Database access class
*
```

5 Die Dokumentationskommentare sind hier in Englisch, da dies in größeren Programmierteams üblicher ist als Deutsch.


```
* @package Database
*/

require_once(__DIR__.'/config.inc.php');

/**
* Database access class
*
* @package Database
*/
class Database {
  /**
  * mysqli database connection
  * @var mysqli
  */
  private $con = NULL;

  /**
  * Set the mysqli database connection object to be used
  *
  * (Applying the Dependency Injection design pattern)
  *
  * @param mysqli $con
  */
  public function setCon($con) {
    $this->con = $con;
  }

  /**
  * Get (and, if necessary, initialize) the mysqli connection object
  *
  * (Applying the Lazy Initialization design pattern)
  *
  * @throws RuntimeException
  * @return mysqli
  */
  public function getCon() {
    if (!is_object($this->con)) {
      $this->con = new mysqli(HOST, USER, PASSWORD, DATABASE);
      if (!is_object($this->con)) {
        throw new RuntimeException("No database connection!!!");
      }
      if (defined("INIT_QUERY")) {
```

```
      $this->query(INIT_QUERY);
    }
  }
  return $this->con;
}

/**
* Create a condition to be applied to a WHERE clause
*
* @param array $filter
* @return string
*/
public function getCondition($filter) {
  $result = '';
  $con = $this->getCon();
  $fieldFirstRun = TRUE;
  foreach ($filter as $field => $value) {
    if (!$fieldFirstRun) {
      $result .= " AND";
    } else {
      $fieldFirstRun = FALSE;
    }
    $result .= sprintf(" %s", $con->real_escape_string($field));
    if (is_array($value)) {
      $result .= " IN (";
      $valFirstRun = TRUE;
      foreach ($value as $val) {
        if (!$valFirstRun) {
          $result .= ", ";
        } else {
          $valFirstRun = FALSE;
        }
        $result .= sprintf("'%s'", $con->real_escape_string($val));
      }
      $result .= ")";
    } elseif (strstr($value, '%') || strstr($value, '_')) {
      $result .= sprintf(
        " LIKE '%s'",
        $con->real_escape_string(
          str_replace('%', '%%', $value)
        )
      );
    } else {
```



```
      $result .= sprintf(" = '%s'", $con->real_escape_string($value));
    }
  }
  return $result;
}

/**
* Perform a database query
*
* @param string $sql sprintf()-style format string
* @param array $params parameters for the format string optional,
* default empty array
* @return mysqli_result
*/
public function query($sql, $params = array()) {
  $con = $this->getCon();
  if (!is_array($params)) {
    $params = array($params);
  }
  foreach ($params as $key => $param) {
    $params[$key] = $con->real_escape_string($param);
  }
  return $con->query(vsprintf($sql, $params));
}

/**
* Perform an insert query
*
* @param string $table
* @param array $values associative array ($field => $value)
* @return mixed last insert ID or affected rows
*/
public function insertQuery($table, $values) {
  $con = $this->getCon();
  $sql = "INSERT INTO %s (";
  $firstRun = TRUE;
  foreach (array_keys($values) as $field) {
    if (!$firstRun) {
      $sql .= ", ";
    } else {
      $firstRun = FALSE;
    }
    $sql .= $con->real_escape_string($field);
```

```
    }
    $sql .= ") VALUES (";
    $firstRun = TRUE;
    foreach ($values as $value) {
      if (!$firstRun) {
        $sql .= ", ";
      } else {
        $firstRun = FALSE;
      }
      $sql .= sprintf("'%s'", $con->real_escape_string($value));
    }
    $sql .= ")";
    $this->query($sql, array($table));
    return $con->insert_id ? $con->insert_id : $con->affected_rows;
  }

  /**
  * Perform an update query
  *
  * @param string $table
  * @param array $values associative array (field => value)
  * @param string $condition sprintf()-style pattern
  * (to be appended to WHERE... if not empty)
  * @param array $params arguments for the sprintf()-style pattern
  * @throws InvalidArgumentException
  * @return integer number of affected rows
  */
  public function updateQuery($table, $values, $condition = '',
       $params = array()) {
    if (empty($values) || !is_array($values)) {
      throw new InvalidArgumentException(
        "At least one value for update needed!"
      );
    }
    $con = $this->getCon();
    $sql = sprintf("UPDATE %s SET ", $con->real_escape_string($table));
    $firstRun = TRUE;
    foreach ($values as $field => $value) {
      if ($firstRun) {
        $firstRun = FALSE;
      } else {
        $sql .= ", ";
      }
```



```
      $sql .= sprintf(
       "%s = '%s'",
       $con->real_escape_string($field),
       $con->real_escape_string($value)
      );
    }
    if (trim($condition) != '') {
      if (!is_array($params)) {
        $params = array($params);
      }
      foreach ($params as $key => $param) {
        $params[$key] = $con->real_escape_string($param);
      }
      $sql .= sprintf(" WHERE %s", vsprintf($condition, $params));
    }
    $con->query($sql);
    return $con->affected_rows;
  }

  /**
  * Perform real_escape_string() on connection
  *
  * @param string $string
  * @return string
  */
  public function escape($string) {
    $con = $this->getCon();
    return $con->real_escape_string($string);
  }

  /**
  * Get the number of affected rows for previous query
  *
  * @return integer
  */
  public function getAffectedRows() {
    $con = $this->getCon();
    return $con->affected_rows;
  }

  /**
  * Get the last inserted auto_increment id
  *
```

```
  * @return integer
  */
  public function getLastInsertId() {
    $con = $this->getCon();
    return $con->insert_id;
  }
}
```

Wie Sie sehen, wird zunächst eine Konfigurationsdatei inkludiert, die die Verbindungsparameter in Form von Konstanten definiert. Diese Datei sieht beispielsweise so aus:

```
<?php
define('HOST', 'localhost');
define('DATABASE', 'mydatabase');
define('USER', 'dbuser');
define('PASSWORD', 'dbpassword');
define('INIT_QUERY', 'SET NAMES utf8'); // Sämtliche DB-Kommunikation in UTF-8
```

Lesen Sie sich den Quellcode der Klasse in Ruhe durch; in den Kommentaren wird vieles erläutert. Wenn Sie die Klasse in Ihrem Projekt verwenden möchten, stehen Ihnen folgende Methoden zur Verfügung:

- `getCondition(array $filter)` erzeugt eine durch `AND`-Verknüpfungen verbundene Abfolge von Prüfungen, die als `WHERE`-Bedingung verwendet werden können. In dem Array sind die Schlüssel Tabellenfelder, die auf die angegebenen Werte geprüft werden sollen. Dabei entscheidet die Methode selbst, ob `=`, `LIKE` oder `IN` verwendet wird – je nachdem, ob der Vergleichswert ein einfacher String ist, `LIKE`-Platzhalter enthält oder ein Array ist. Im Fall von `LIKE` werden einzelne Prozentzeichen mit `str_replace()` durch doppelte ersetzt, damit `vsprintf()` sie nicht als Formatplatzhalter missversteht.
- `query(string $sql, array $params)` führt eine Abfrage durch. Das erste Argument ist ein `sprintf()`-Platzhalter, der zweite ein optionales Array mit denjenigen Werten, die aus Sicherheitsgründen escapet werden sollten. Hier sind besonders Benutzereingaben betroffen.
- `insertQuery(string $sql, array $values)` versucht, einen neuen Datensatz einzufügen, wobei `$values` ein assoziatives Array ist, in dem die Schlüssel die Feldnamen und die Werte die einzufügenden Feldwerte sind.
- `updateQuery(string $table, array $values, string $condition, array $params)` führt eine `UPDATE`-Abfrage durch. `$table` ist die betroffene Tabelle, `$values` enthält die Namen und Werte der zu ändernden Spalten, und `$condition` und `$params` beschreiben die optionale Bedingung – diese Parameter funktionieren wie bei der Methode `query()`.

- `escape($string)` ruft die `mysqli`-Methode `real_escape_string()` auf den angegebenen String auf, die potenziell gefährliche Zeichen im String escapet, bevor dieser in einer Abfrage verwendet wird.
- `getAffectedRows()` liefert die Anzahl der bei der letzten Änderungsabfrage geänderten Datensätze zurück.
- `getLastInsertId()` gibt die zuletzt eingefügte Auto-Increment-ID zurück.

Beachten Sie die Art und Weise, wie das `mysqli`-Objekt von außen gesetzt werden kann und nur dann instanziiert wird, wenn es nicht zuvor gesetzt oder eben instanziiert wurde. Ein solches Getter-Setter-Paar implementiert die beiden Entwurfsmuster *Dependency Injection* und *Lazy Initialization*. Sie werden verwendet, um die Implementierung einer Abhängigkeit jederzeit austauschen zu können, vor allem aber für Unit-Tests, um echte abhängige Objekte durch Simulationen zu ersetzen.

19.1.6 Unit-Tests mit PHPUnit

In Kapitel 12, »Software-Engineering«, wurde bereits auf die Bedeutung und die allgemeinen Grundlagen des Unit-Testings hingewiesen. Für PHP steht dazu insbesondere das Test-Framework PHPUnit von Sebastian Bergmann zur Verfügung.

Installieren Sie PHPUnit zunächst anhand der Anleitung auf der Projektwebsite *www.phpunit.de*. Danach können Sie Tests schreiben, um das Funktionieren Ihrer Klassen und Methoden sicherzustellen. Unit-Tests werden von der Klasse `PHPUnit_Framework_TestCase` abgeleitet, die per `require_once()` eingebunden werden muss. Alle öffentlichen Methoden, deren Name mit `test` beginnt, werden beim Ausführen des Tests aufgerufen.

Innerhalb von Testmethoden werden *Assertions* verwendet. Dies sind spezielle Methoden, deren Name mit `assert` anfängt. Im Allgemeinen steht innerhalb der Assertion als erstes Argument der erwartete Wert, während weitere Argumente einen von der getesteten Methode zurückgegebenen Wert, ein Attribut oder Ähnliches enthalten. Schlägt eine Assertion fehl, gibt PHPUnit für die entsprechende Methode ein »F« (für »failure«) und eine Fehlermeldung aus. Andernfalls wird ein Punkt (`.`) ausgegeben.

Betrachten Sie als Beispiel noch einmal die Klasse `HtmlTag`, die im Abschnitt »Vererbung« behandelt wurde. Zur Verwendung von Unit-Tests sollte ein PHP-Projekt mit folgender Verzeichnisstruktur erstellt werden:

```
+ [Wurzelverzeichnis der Anwendung]
|
+---+ [source]
|   |
|   +---+ [lib]
|       |
|       +---+ [Html]
```

```
|           |
|           +---+ Tag.php
|
+---+ [tests]
|   |
|   +---+ [lib]
|       |
|       +---+ [Html]
|           |
|           +---+ TagTest.php
|
+---+ index.php (und weitere Ausgabeseiten)
```

Wie Sie sehen, befindet sich unter *tests* dieselbe Verzeichnisstruktur wie unter *source*. Auf diese Weise ist stets völlig klar, welcher Unit-Test sich auf welche Klasse bezieht. Hier also der Unit-Test für die Klasse `HtmlTag`:

```
<?php

require_once('/usr/lib/php/PHPUnit/Framework/TestCase.php');
require_once(__DIR__.'/../../../source/lib/Html/Tag.php');

class HtmlTagTest extends PHPUnit_Framework_TestCase {
  /**
  * @covers HtmlTag::__construct
  */
  public function testConstruct() {
    $tag = new HtmlTag('p', array('title' => 'Test'), 'Test text');
    $this->assertAttributeEquals('p', 'tagName', $tag);
    $this->assertAttributeEquals(
      array('title' => 'Test'), 'attributes', $tag);
    $this->assertAttributeEquals('Test text', 'content', $tag);
  }

  /**
  * @covers HtmlTag::addAttribute
  */
  public function testAddAttribute() {
    $tag = new HtmlTag('p');
    $tag->addAttribute('title', 'Test Title');
    $this->assertAttributeEquals(
      array('title' => 'Test Title'), 'attributes', $tag);
  }
```



```
  /**
  * @covers HtmlTag::__toString
  */
  public function testToString() {
    $tag = new HtmlTag('p', array('title' => 'Test'), 'Test text');
    $expected = '<p title="Test">
Test text</p>
';
    $this->assertEquals($expected, $tag->__toString());
  }
}
```

Den Pfad von `PHPUnit_Framework_TestCase` müssen Sie auf Ihrem System eventuell anpassen. Danach können Sie den Unit-Test ausführen, indem Sie Folgendes auf der Konsole eingeben:

```
> phpunit HtmlTest.php
```

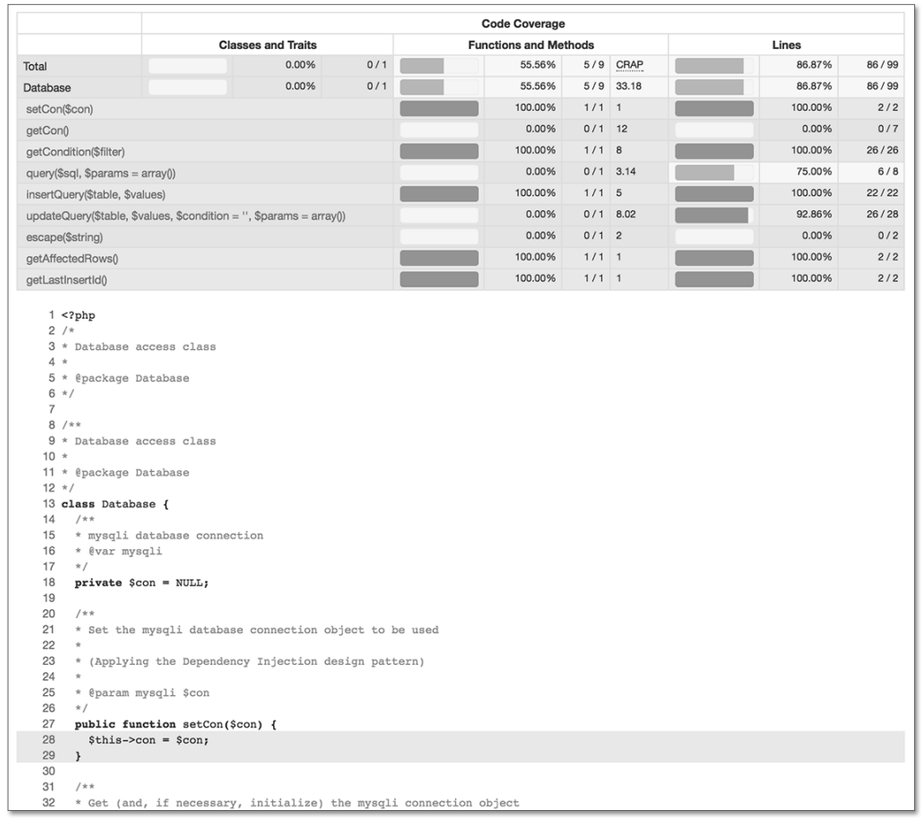
Wenn Sie alles korrekt abgespeichert haben, müssten drei Punkte herauskommen, die besagen, dass alle drei Methoden erfolgreich getestet wurden.

In diesem Beispiel kommen zwei Assertions zur Anwendung: `assertEquals($expected, $actual)` gilt als erfüllt, wenn der erwartete Wert `$expected` und der tatsächliche `$actual` übereinstimmen. `assertAttributeEquals($expected, $attribute, $instance)` funktioniert im Prinzip genauso, vergleicht jedoch den erwarteten Wert mit dem angegebenen Attribut der Instanz. Durch Reflexion ist dabei sogar der Zugriff auf `private`- oder `protected`-Attribute möglich.

Die `@covers`-PHPDoc-Annotations über den Testmethoden werden übrigens für den sogenannten *Coverage-Report* verwendet: Wenn auf dem Entwicklungssystem die Debug-Erweiterung XDebug installiert ist, können Sie sich anzeigen lassen, welche Teile der Klassen von Unit-Tests abgedeckt sind und wie der prozentuale Anteil ist. Das Ziel sollten 100 % sein, aber die sind nicht immer zu erreichen. Um einen Coverage-Report zu generieren, rufen Sie PHPUnit wie folgt auf:

```
$ phpunit --coverage-html reportDirectory FileOrDirectory
```

Dies führt die Unit-Tests in der Datei oder dem Verzeichnis `FileOrDirectory` aus und schreibt den Coverage-Report in das Verzeichnis `reportDirectory`, das bei Bedarf neu angelegt wird. Anschließend können Sie die in dem Verzeichnis enthaltene Datei *index.html* in einem Browser öffnen. Abbildung 19.1 zeigt einen Coverage-Report für die im nächsten Abschnitt getestete Klasse *Database.php*. Getestete Codezeilen werden darin grün angezeigt, ungetestete rot, und es wird eine Statistik für alle beteiligten Klassen, Methoden und Codezeilen angezeigt.



Code Coverage

	Classes and Traits			Functions and Methods			Lines		
Total		0.00%	0 / 1		55.56%	5 / 9	CRAP	86.87%	86 / 99
Database		0.00%	0 / 1		55.56%	5 / 9	33.18	86.87%	86 / 99
setCon($con)					100.00%	1 / 1	1	100.00%	2 / 2
getCon()					0.00%	0 / 1	12	0.00%	0 / 7
getCondition($filter)					100.00%	1 / 1	8	100.00%	26 / 26
query($sql, $params = array())					0.00%	0 / 1	3.14	75.00%	6 / 8
insertQuery($table, $values)					100.00%	1 / 1	5	100.00%	22 / 22
updateQuery($table, $values, $condition = '', $params = array())					0.00%	0 / 1	8.02	92.86%	26 / 28
escape($string)					0.00%	0 / 1	2	0.00%	0 / 2
getAffectedRows()					100.00%	1 / 1	1	100.00%	2 / 2
getLastInsertId()					100.00%	1 / 1	1	100.00%	2 / 2

```
 1 <?php
 2 /*
 3 * Database access class
 4 *
 5 * @package Database
 6 */
 7
 8 /**
 9 * Database access class
10 *
11 * @package Database
12 */
13 class Database {
14   /**
15   * mysqli database connection
16   * @var mysqli
17   */
18   private $con = NULL;
19
20   /**
21   * Set the mysqli database connection object to be used
22   *
23   * (Applying the Dependency Injection design pattern)
24   *
25   * @param mysqli $con
26   */
27   public function setCon($con) {
28     $this->con = $con;
29   }
30
31   /**
32   * Get (and, if necessary, initialize) the mysqli connection object
```

Abbildung 19.1 Ein PHPUnit-Coverage-Report für die Unit-Tests der Klasse »Database«

Mock-Objekte verwenden

Wenn eine Klasse andere Objekte als Abhängigkeiten enthält, sollten diese Objekte bei einem reinen Unit-Test nicht mitgetestet werden, denn Ziel des Unit-Tests ist das Testen möglichst kleiner, eigenständiger Codeeinheiten. Um diese Abhängigkeiten nicht mittesten zu müssen, werden sie, wenn möglich, durch eigene Objekte ersetzt, die sich ganz nach Wunsch verhalten – die sogenannten *Mock-Objekte*. Als Beispiel sehen Sie hier einen Unit-Test für die Datenbank-Abstraktionsklasse `Database`, der ein Mock-Objekt für die eigentliche Datenbankverbindung verwendet:

```
<?php

require_once('/usr/lib/php/PHPUnit/Framework/TestCase.php');
require_once(__DIR__.'/../../source/lib/Database.php');
```



```
class DatabaseTest extends PHPUnit_Framework_TestCase {
   /**
   * @covers Database::setCon
   */
   public function testSetCon() {
    $db = new Database();
    $con = $this->getMock('GenericDb');
    $db->setCon($con);
    $this->assertAttributeSame($con, 'con', $db);
  }

  /**
  * @covers Database::getCon
  */
  public function testGetCon() {
     $this->markTestSkipped('Real database connection cannot be tested.');
  }

  /**
  * @covers Database::getCondition
  */
  public function testGetCondition() {
    $db = new Database();
    $con = $this->getMock('GenericDb', array('real_escape_string'));
    $con
      ->expects($this->atLeastOnce())
      ->method('real_escape_string')
      ->will(
          $this->onConsecutiveCalls(
            $this->returnValue('field1'),
            $this->returnValue(1),
            $this->returnValue(2),
            $this->returnValue('field2'),
            $this->returnValue('test%'),
            $this->returnValue('field3'),
            $this->returnValue('test')
          )
        );
    $db->setCon($con);
    $expected = " field1 IN ('1', '2') AND field2 LIKE 'test%%' AND field3 = 'test'";
    $this->assertEquals(
      $expected,
      $db->getCondition(
```

```
        array('field1' => array(1, 2), 'field2' => 'test%', 'field3' => 'test')
      )
    );
}

/**
* @covers Database::query
*/
public function testQuery() {
  $db = new Database();
  $con = $this->getMock('GenericDb', array('real_escape_string', 'query'));
  $result = $this->getMock('GenericDbResult');
  $con
    ->expects($this->atLeastOnce())
    ->method('real_escape_string')
    ->will(
        $this->onConsecutiveCalls(
          $this->returnValue('table'),
          $this->returnValue(1)
        )
      );
  $con
    ->expects($this->once())
    ->method('query')
    ->will($this->returnValue($result));
  $db->setCon($con);
  $this->assertEquals(
    $result,
    $db->query('SELECT test FROM %s WHERE field = %d', array('table', 1))
  );
}

/**
* @covers Database::insertQuery
*/
public function testInsertQuery() {
  $db = new Database();
  $con = $this->getMock('GenericDb', array('real_escape_string', 'query'));
  $con
    ->expects($this->atLeastOnce())
    ->method('real_escape_string')
    ->will(
        $this->onConsecutiveCalls(
```



```
          $this->returnValue('table'),
          $this->returnValue('name'),
          $this->returnValue('SampleName'),
          $this->returnValue('value'),
          $this->returnValue('SampleValue')
        )
      );
  $con->insert_id = 7;
  $db->setCon($con);
  $this->assertEquals(
    7,
    $db->insertQuery(
      'table',
      array('name' => 'SampleName', 'value' => 'SampleValue')
    )
  );
}

/**
* @covers Database::updateQuery
*/
public function testUpdateQuery() {
  $db = new Database();
  $con = $this->getMock('GenericDb', array('real_escape_string', 'query'));
  $con
    ->expects($this->atLeastOnce())
    ->method('real_escape_string')
    ->will(
        $this->onConsecutiveCalls(
          $this->returnValue('table'),
          $this->returnValue('name'),
          $this->returnValue('SampleName'),
          $this->returnValue('value'),
          $this->returnValue('SampleValue'),
          $this->returnValue(1)
        )
      );
  $con->affected_rows = 1;
  $db->setCon($con);
  $this->assertEquals(
    1,
    $db->updateQuery(
      'table',
```

```
      array('name' => 'SampleName', 'value' => 'SampleValue'),
      'id = %d',
      array(1)
    )
  );
}

/**
* @covers Database::updateQuery
*/
public function testUpdateQueryExpectingException() {
  $db = new Database();
  try {
     $db->updateQuery('test', array());
     $this->fail('Expected InvalidArgumentException not thrown.');
  } catch (InvalidArgumentException $e) { }
}

/**
* @covers Database::getAffectedRows
*/
public function testGetAffectedRows() {
  $db = new Database();
  $con = $this->getMock('GenericDb');
  $con->affected_rows = 2;
  $db->setCon($con);
  $this->assertEquals(2, $db->getAffectedRows());
  }

  /**
  * @covers Database::getLastInsertId
  */
  public function testGetLastInsertId() {
    $db = new Database();
    $con = $this->getMock('GenericDb');
    $con->insert_id = 3;
    $db->setCon($con);
    $this->assertEquals(3, $db->getLastInsertId());
  }
}
```

Mock-Objekte werden mit der PHPUnit-Methode `getMock(string $classname, array $methods)` erstellt. Zusätzlich kann mit `expects()` festgelegt werden, was bei welchem

Methodenaufruf auf das jeweilige Mock-Objekt herauskommen soll. Als Argument von `expects()` wird die erwartete Anzahl der Aufrufe der Methode angegeben:

- `$this->once()` erwartet genau einen Aufruf.
- `$this->atLeastOnce()` benötigt mindestens einen Aufruf.
- `$this->any()` erlaubt beliebig viele Aufrufe, einschließlich keinen.
- `$this->exactly($anzahl)` besteht auf genau `$anzahl` Aufrufen.

Mit `will($this->returnValue($value))` geben Sie an, dass die Methode beim Aufruf den Wert `$value` zurückgeben soll. Die Variante `will($this->onConsecutiveCalls())` wird verwendet, wenn mehrere aufeinanderfolgende Aufrufe unterschiedliche Werte zurückgeben sollen.

Beachten Sie, dass hier eine Fantasieklasse namens `GenericDb` als Vorlage für die Mock-Objekte verwendet wird und nicht `mysqli`. Das liegt daran, dass `getMock()` nur mit sehr großem Aufwand das Umgehen des Konstruktors der Originalklasse erlaubt.

In neuere PHPUnit-Versionen ist eine Klasse namens `MockBuilder` eingebaut, die in dieser Hinsicht erheblich flexibler ist. Sie können sie (schematisch) wie folgt verwenden:

```
$mockObject = $this
  ->getMockBuilder($className)
  ->setMethods(array($method1, $method2...)
  ->disableOriginalConstructor()
  ->getMock();
```

Mit `setMethods()` legen Sie die Methoden fest, die das Mock-Objekt bereitstellen soll. Falls das Mock-Objekt nach einer lesbaren echten Klasse modelliert wird, können Sie diesen Aufruf weglassen. Die zusätzliche Methode `getOriginalConstructor()` ermöglicht es Ihnen, einen Aufruf des Konstruktors der Originalklasse zu umgehen, wenn Sie dessen Abhängigkeiten nicht erfüllen können oder wollen. Natürlich kann auch diese Zeile weggelassen werden, wenn der Original-Konstruktor keine Argumente verlangt oder Sie einen Mock einer nicht existierenden Klasse erzeugen.

Mit der `MockBuilder`-Variante ist es übrigens kein Problem mehr, die echte Klasse `mysqli` als Vorlage für das Mock-Objekt zu verwenden, denn der Schwierigkeiten bereitende Konstruktor lässt sich nun leicht ausblenden. Dafür brauchen Sie in diesem Fall nicht mehr die Methoden anzugeben, die das Mock-Objekt bereitstellen soll. Der Aufruf für alle Datenbank-Mock-Objekte sieht dadurch also wie folgt aus:

```
$con = $this
  ->getMockBuilder('mysqli')
  ->disableOriginalConstructor()
  ->getMock();
```

Wie gerade dieses etwas längere Beispiel zeigt, sind Unit-Tests unter anderem auch Teil der Dokumentation – denn Sie sehen hier ganz genau, wie die Klasse verwendet wird.

19.2 Eine REST-API implementieren

Immer mehr Webanwendungen werden in Form von Webservices bereitgestellt. Der große Vorteil ist, dass diese alle Arten von Clients bedienen können – von klassischen Websites mit serverseitiger Programmiersprache und HTML über moderne JavaScript-Frameworks bis hin zu Mobile Apps für iOS, Android und Windows Phone.

Der überwiegende Teil dieser Webservices wird heutzutage nach dem *REST*-Schema erstellt. REST ist eine Abkürzung für *REpresentational State Transfer*; es handelt sich um eine konsequente Nutzung der verschiedenen HTTP-Zugriffsmethoden. Dabei werden typischerweise die sogenannten *CRUD*-Funktionen für Ressourcen bereitgestellt (*Create*, *Read*, *Update*, *Delete*); auf der Serverseite werden diese Ressourcen oft als Datensätze in relationalen Datenbanken gespeichert. Die einzelnen HTTP-Methoden haben dabei folgende Aufgaben:

- `GET` wird zum Lesen verwendet, liefert also die auf dem Server gespeicherten Informationen über die gewünschten Ressourcen zurück.
- `POST` dient dem Neuanlegen von Ressourcen; der Body der Anfrage enthält die zu speichernden Daten.
- `PUT` wird zum Ändern bereits vorhandener Ressourcen verwendet; auch hier enthält der Body die entsprechenden Daten.
- `DELETE` schließlich löscht die angeforderte Ressource.

Anstelle von *Webservice* ist im Zusammenhang mit REST häufiger von einer *API* (*Application Programming Interface*) die Rede; diese Terminologie wird hier übernommen.

In diesem Abschnitt wird mit PHP eine REST-API entwickelt, die den CRUD-Zugriff auf die in Abschnitt Abschnitt 19.1.5, »Zugriff auf MySQL-Datenbanken«, entwickelte Tabelle mit Informationen über Programmiersprachen ermöglicht. Im nächsten Kapitel erfahren Sie, wie Sie mit JavaScript auf die API zugreifen können, und bereits in Kapitel 11, »Mobile Development«, wurde sie zum Servergegenstück für eine iOS-App.

19.2.1 Die API im Überblick

Jede vernünftige REST-API muss dokumentiert werden, damit Entwickler von Clients wissen, welche Zugriffe möglich sind. Hier also zunächst die Entwicklerdokumentation für die neu zu erstellende API.

Die einzige unterstützte Ressource ist *Language*, denn die API basiert nur auf einer Datenbanktabelle. Tabelle 19.4 enthält alle zulässigen Zugriffsmethoden und Pfade.

HTTP-Methode	Pfad	Erläuterung
`GET`	`/Language/`	liefert die Liste aller Programmiersprachen
`GET`	`/Language/?search=str`	liefert eine Lister der Sprachen, in deren Namen der String `str` vorkommt
`GET`	`/Language/ID`	liefert die Programmiersprache mit der angegebenen numerischen ID
`POST`	`/Language/`	erzeugt eine neue Programmiersprache mit den im Body angegebenen Daten
`PUT`	`/Language/ID`	ersetzt die Daten der Sprache mit der angegebenen ID durch diejenigen im Body
`DELETE`	`/Language/ID`	löscht die Sprache mit der angegebenen ID

Tabelle 19.4 Überblick über die Methoden der REST-API

Das Datenaustauschformat der API ist XML – die Ausgaben der `GET`-Anfragen erfolgen in dieser Sprache, und in den Bodys der `POST`- und `PUT`-Anfragen wird sie ebenfalls erwartet. XML ist eines der beiden am häufigsten verwendeten Datenaustauschformate für REST-APIs; das andere ist das im nächsten Kapitel vorgestellte JSON.

Eine Liste von Sprachen, wie sie von `GET /Language/` zurückgegeben wird, sieht so aus:

```
<?xml version="1.0" encoding="utf-8" standalone="yes"?>
<languages>
    <language id="1">
        <id>1</id>
        <name>C</name>
        <architecture>imperative</architecture>
        <implementation>compiler</implementation>
        <system>Unix,Windows,other</system>
        <description>älteste weitverbreitete Sprache; Syntax in vielen
        Sprachen verbreitet</description>
        <year>1970</year>
    </language>
    <!-- Weitere Sprachen -->
</languages>
```

Eine einzelne Sprache, wie sie etwa von `GET /Language/3` zurückgegeben wird, hat folgendes Format:

```
<?xml version="1.0" encoding="utf-8" standalone="yes"?>
<language>
    <id>3</id>
    <name>Python</name>
    <architecture>oop</architecture>
    <implementation>interpreter</implementation>
    <system>Unix,Windows,other</system>
    <description>Multiparadigmen-Skriptsprache</description>
    <year>1991</year>
</language>
```

Dasselbe Format wird auch bei `POST` und `PUT` erwartet, wobei Sie nur diejenigen Elemente angeben müssen, die Sie hinzufügen beziehungsweise ersetzten möchten. Falls Sie also beispielsweise nur die Beschreibung von Python ändern möchten, senden Sie eine `PUT`-Anfrage an den URL-Pfad `/Language/3`, und fügen Sie folgendes XML-Dokument in den Body ein:

```
<?xml version="1.0" encoding="utf-8" standalone="yes"?>
<language>
    <description>Moderne Multiparadigmen-Skriptsprache mit objektorientierten
    und funktionalen Aspekten</description>
</language>
```

Alle Schreibzugriffe, also `PUT`, `POST` und `DELETE`, benötigen eine Autorisierung in Form der URL-Parameter `user` und `key`, deren Inhalte in der Konfigurationsdatei der API festgelegt werden. Beim Key wird dabei der MD5-Hash (einfache Einwegverschlüsselung) gespeichert – keine Software mit Netzwerkanbindung sollte Passwörter oder Schlüssel jemals im Klartext speichern, um Crackern auch bei einem eventuell erfolgreichen Angriff keine brauchbaren Daten an die Hand zu geben.

Eine `DELETE`-Anfrage für eine Sprache mit der ID 25 sieht also vollständig so aus (die Platzhalter `USERNAME` und `KEY` müssen natürlich durch die korrekten Werte ersetzt werden):

```
DELETE /Language/25?user=USERNAME&key=KEY
```

In der Praxis erfolgt die Autorisierung oder Authentifizierung von REST-APIs über aufwendigere Verfahren wie *OAuth2*. Diese haben zwei Vorteile:

- Sie sind standardisiert, sodass die meisten Sprachen über Bibliotheken dafür verfügen.
- Der Inhalt der Anfrage selbst wird in die Berechnung des jeweiligen Schlüssels eingerechnet, sodass dieser Inhalt nicht durch Angreifer verfälscht werden kann.

Nach erfolgreichen Änderungsanfragen erhalten Sie den HTTP-Status `200 (OK)` und eine kurze XML-Erfolgsmeldung; bei der soeben gezeigten `PUT`-Anfrage beispielsweise diese:


```
<?xml version="1.0" encoding="utf-8" standalone="yes"?>
<success>Language successfully modified.</success>
```

Fehlermeldungen haben Status-Werte wie `404 (Not found)` oder `400 (Bad request)` und einen XML-Body wie beispielsweise diesen:

```
<?xml version="1.0" encoding="utf-8" standalone="yes"?>
<error>The requested resource could not be found.</error>
```

19.2.2 Die Grundarchitektur der API

Die REST-API besteht aus verschiedenen Dateien mit folgenden Aufgaben (Ausgabe des Unix-Befehls `tree -a` mit zusätzlich eingefügten Erläuterungen):

```
.
+-- .htaccess         Apache-Konfiguration: immer index.php laden
+-- config.inc.php    Konfiguration für Database.php und Autorisierung
+-- index.php         Hauptprogrammdatei, lädt Autoloader und Application
+-- Autoloader.php    enthält die Autoleader-Methode für die API
+-- Application.php   Hauptanwendungsklasse; Methode run() bearbeitet Anfragen
+-- Api.php           Basisklasse für API-Ressourcen
+-- Language.php      API-Ressource "Language"
+-- Database.php      Datenbank-Basisklasse (bereits beschrieben)
+-- Dba.php           Helferklasse, die DB-Feldnamen auf lesefreundliche Art abbildet
+-- Language
|   +-- Dba.php       DB-Abfragen und lesefreundliche Felder für Language
+-- Xml.php           Helferklasse zur XML-Verarbeitung und -Erzeugung
```

Die Klassen sind so gestaltet, dass sich API-Anfragen für weitere Ressourcen leicht hinzufügen lassen. Im Prinzip brauchen Sie dafür nur eine Datei namens *Ressourcenname.php* mit einer von `Api` abgeleiteten Klasse, die die konkreten Anfragen verarbeitet, und eine von `Dba` abgeleitete Klasse, sinnvollerweise wie hier mit dem Pfad *Ressourcenname/Dba.php*, die konkrete Datenbankabfragen für Ihre Ressource enthält und die internen Feldnamen der Datenbanktabelle auf lese- und schreibfreundliche Art für die API abbildet.

Nach dem ersten Überblick hier eine ausführlichere Beschreibung der Aufgaben aller Dateien:

- *.htaccess* enthält nur eine einzige Apache-Direktive, die dafür sorgt, dass alle Anfragen – ungeachtet ihres konkreten Pfades – von *index.php* behandelt werden. Sie sollten die API in einem eigenen Virtual Host mit separatem Hostnamen oder separatem TCP-Port betreiben und können diese Zeile auch in den Konfigurationskontext dieses Virtual Hosts kopieren. Informationen über die Einrichtung und Konfiguration von Apache erhalten Sie in Kapitel 14, »Server für Webanwendungen«.

Falls Sie die API mit einem anderen Webserver als Apache ausführen möchten, müssen Sie eine adäquate Konfiguration für diesen verwenden.

- *config.inc.php* enthält die Konfiguration für die Datenbank-Basisklasse `Database`, die in diesem Kapitel bereits gezeigt wurde, sowie die Autorisierungsdaten für Schreibzugriffe.
- *index.php* ist die erste Datei, die beim Eintreffen einer Anfrage aufgerufen wird. Sie importiert die Konfigurationsdatei, initialisiert den Autoloader und ruft anschließend die Methode `run()` der Klasse `Application` auf, die die eigentliche Arbeit erledigt.
- *Autoloader.php* enthält die statische Methode `autoload()`, die in *index.php* mithilfe von `spl_autoload_register()` registriert wird. Die Methode ergänzt Großbuchstaben im Klassennamen durch einen / vor diesen Großbuchstaben und verwendet das Ergebnis mit der Endung *.php* als den zu importierenden Dateinamen.
- *Application.php* enthält die Methode `run()`, mit der die eigentliche Verarbeitung der Anfrage durchgeführt wird. Dazu wird der Anfragepfad an /-Zeichen zerlegt; das erste Element wird als Klassenname instanziiert, und die restlichen werden an die zuständige Methode übergeben, die dem Namen der HTTP-Methode entspricht. Die Anfrage `GET /Language` würde also dazu führen, dass eine Instanz der Klasse `Language` erzeugt und die Methode `get()` dieser Instanz aufgerufen wird. In einer robusten Anwendung für die reale Welt müsste natürlich überprüft werden, ob die Klasse und die Methode überhaupt existieren, aber dies unterbleibt hier.
- *Api.php* ist die Basisklasse für API-Ressourcen-Klassen. Sie enthält Implementierungen der vier Anfragemethoden `get()`, `post()`, `put()` und `delete()`, die hier jedoch alle mit `503 Not implemented` antworten. Der Vorteil dieser Vorgehensweise ist, dass Sie automatisch die passende Antwort geben, wenn eine Ressourcenklasse nicht alle vier Methoden benötigt. Daneben enthält die Klasse einige Methoden für gängige HTTP-Statusmeldungen wie `notFound()`. Diese sind als `protected` gekennzeichnet, damit eine abgeleitete Klasse darauf zugreifen und sie bei Bedarf als Ergebnis ihrer öffentlichen Methoden zurückgeben kann.
- *Language.php* ist die konkrete API-Klasse für die Ressource `Language`; sie ist von `Api` abgeleitet, um die grundlegenden Funktionalitäten zu übernehmen. Hier sind alle vier Zugriffsmethoden implementiert, außerdem diverse Helfermethoden. Die Klasse verwendet eine Instanz von `LanguageDba` für die Zugriffe auf die Datenbanktabelle mit den Programmiersprachen.
- *Database.php* wurde bereits im vorangegangenen Abschnitt ausführlich beschrieben; es handelt sich um eine Klasse für bequemere MySQL-Datenbankabfragen.
- *Dba.php* ist die Basisklasse für konkrete Datenbankzugriffsklassen (DataBase Access) von Ressourcen. Sie enthält ein (hier leeres) Array namens `$fields`, das die internen Feldnamen der Tabellen, im Fall von Language etwa `language_description`, XML-freundlichere Namen wie `description` zuordnet. Die enthaltenen Methoden `recordToResult()` und `resultToRecord()` nehmen die entsprechenden Umwandlungen vor. Die Klasse ist von

`Database` abgeleitet, um den DBA-Klassen der Ressourcen die grundlegende Datenbankfunktionalität zur Verfügung zu stellen.

- *Language/Dba.php* enthält die wiederum von `Dba` abgeleitete Klasse `LanguageDba`, die die konkreten Datenbankoperationen für die Programmiersprachen-Tabelle bereitstellt. Dabei wandeln die lesenden Methoden die Feldnamen aus der Tabelle in die öffentlichen Kurznamen um, während `create()` und `update()` den umgekehrten Weg gehen.
- *Xml.php* stellt verschiedene Methoden zur Erzeugung und Verarbeitung von XML bereit. Zum Parsen von XML-Code aus Anfrage-Bodys verwendet die Klasse die PHP-XML-API `SimpleXML`, deren Ansatz dem in Kapitel 16, »XML«, vorgestellten Python-Zugriff auf XML-Dokumente ähnelt.

19.2.3 Der komplette Quellcode

In diesem Abschnitt finden Sie den Quellcode jeder Datei der API zusammen mit einigen Erläuterungen. Mit Ausnahme von *Language/Dba.php* liegen alle im selben Verzeichnis, das als DocumentRoot eines Virtual Hosts verwendet werden kann, um die API über den Webserver zugänglich zu machen.

Die Apache-Konfigurationsdatei .htaccess

Diese Datei enthält nur eine einzige Zeile:

```
FallbackResource /index.php
```

Die Direktive `FallbackResource` sorgt dafür, dass alle Anfragen für nicht existierende Dateien an die angegebene Ressource weitergeleitet werden. Der führende / bedeutet, dass sich die Datei in der DocumentRoot des aktuellen Virtual Hosts befindet.

Die Konfigurationsdatei config.inc.php

Der Datenbankteil dieser Datei wurde bereits im vorangegangenen Abschnitt gezeigt. Er enthält die Datenbank-Zugriffsparameter, die Sie natürlich an Ihre eigene Datenbank anpassen müssen. Die Konstante `INIT_QUERY` sollten Sie dagegen beibehalten, da sämtliche Kommunikation zwischen der API und ihren Clients den UTF-8-Zeichensatz verwendet, was auch die Datenbank tun sollte.

Die Konstanten `AUTH_USER` und `AUTH_KEY` sind die Werte, die bei Schreibanfragen in den Query-String-Parametern `user` beziehungsweise `key` mitgeliefert werden müssen. Der Key ist dabei, wie bereits erwähnt, ein MD5-Hash des eigentlichen Wertes; im folgenden Beispiel handelt es sich um den Hash des Strings `"my-key"`. Wenn Sie die Werte aus dem Beispiel übernehmen, müssen Sie an `POST`-, `PUT`- und `DELETE`-Anfragen also den Query-String `?user=apiuser&key=my-key` anhängen.[6] In PHP stehen diese Parameter übrigens auch dann im Array

$_GET zur Verfügung, wenn die Anfragemethode gar nicht GET ist, denn $_GET enthält grundsätzlich nur Query-String-Parameter.

Hier ein Beispiel für den Inhalt der Datei:

```
<?php
// Datenbank
define('HOST', 'localhost');
define('DATABASE', 'mydatabase');
define('USER', 'dbuser');
define('PASSWORD', 'geheim');
define('INIT_QUERY', 'SET NAMES UTF8');

// Autorisierung
define('AUTH_USER', 'apiuser');
define('AUTH_KEY', '515780610702189dabd912e9c9ef6f38');
```

Das »Hauptprogramm« index.php

Wie bereits erwähnt, sorgt die *.htaccess*-Datei dafür, dass bei jeder Anfrage diese Datei aufgerufen wird. Sie hat folgenden Inhalt:

```
<?php

require_once(__DIR__.'/config.inc.php');
require_once(__DIR__.'/Autoloader.php');

spl_autoload_register(array('Autoloader', 'autoload'));

$application = new Application();
$application->run();
```

Zunächst werden also die Konfigurationsdatei und die Autoloader-Klasse inkludiert. Dass die Konfigurationsdatei nochmals in *Database.php* eingebunden wird, spielt keine Rolle, denn die verwendete Anweisung require_once() sorgt dafür, dass sie in jedem Fall nur einmal hinzugefügt wird.

Die Autoloader-Klasse Autoloader.php

Die Klasse Autoloader enthält lediglich die statische Methode autoload(); der Quellcode sieht so aus:

6 Falls Sie die API – oder Ihre eigene davon abgeleitete Version – auf einem öffentlich zugänglichen Webserver bereitstellen sollten, ist es natürlich ratsam, Ihre eigenen Werte zu verwenden, denn dieses Buch hat eine gewisse Verbreitung.


```
<?php

class Autoloader {
  public static function autoload($classname) {
    $filename = preg_replace('(([A-Z]))', '/\\1', $classname);
    if (substr($filename, 0, 1) != '/') {
      $filename = '/'.$filename;
    }
    $filename .= '.php';
    include_once(__DIR__.$filename);
  }
}
```

Das Verfahren wurde im Prinzip bereits erläutert. Mithilfe von preg_replace() wird zunächst vor jedem Großbuchstaben ein / eingefügt, da Großbuchstaben in der Logik dieses Autoloaders Unterverzeichnisse einleiten. Wenn der Pfad danach nicht mit / beginnt (weil er nicht mit einem Großbuchstaben begann), wird dieses Zeichen davor eingefügt. Anschließend wird die Dateiendung *.php* angehängt, und die so gekennzeichnete Datei im aktuellen Verzeichnis __DIR__ wird geladen.

Die Haupt-Anwendungsklasse Application.php

Diese Klasse definiert die Hauptmethode der Anwendung mit dem Namen run(). Sie wird von *index.php* auf einer neuen Instanz von Application aufgerufen. Hier zunächst der Quellcode:

```
<?php

class Application {
  public function run() {
    $path = strtok($_SERVER['REQUEST_URI'], '?');
    $elements = preg_split('(/)', $path);
    while (empty($elements[0])) {
      array_shift($elements);
    }
    $classname = array_shift($elements);
    $instance = new $classname();
    $method = strtolower($_SERVER['REQUEST_METHOD']);
    $result = $instance->$method($elements);
    header(
      sprintf(
        "HTTP/1.1 %d %s",
        $instance->statusCode(),
        $instance->statusMessage()
```

```
      )
    );
    header(sprintf("Content-type: %s", $instance->contentType()));
    header(sprintf("Content-length: %d", strlen($result)));
    echo $result;
  }
}
```

Der echte Pfad der Anfrage wird aus der Servervariablen `REQUEST_URI` gelesen. Dieser wird mit `strtok()` vor einem eventuellen Fragezeichen, das einen Query-String einleitet, abgeschnitten, und dann mithilfe von `preg_split()` an /-Zeichen in ein Array zerlegt. Eventuelle leere Elemente, die durch einen führenden / entstehen, werden mit `array_shift()` vom Anfang des Arrays entfernt.

Das erste Element des Arrays wird als Klassenname interpretiert; es wird ebenfalls aus dem Array entfernt, und eine Instanz dieser Klasse wird erzeugt:

```
$classname = array_shift($elements);
$instance = new $classname();
```

Anschließend wird die mit `strtolower()` in Kleinbuchstaben umgewandelte HTTP-Methode als Methode der neuen Instanz aufgerufen; der Rest des Arrays wird als Argument übergeben. Das Ergebnis des Methodenaufrufs ist der Body der HTTP-Antwort:

```
$method = strtolower($_SERVER['REQUEST_METHOD']);
$result = $instance->$method($elements);
```

Die Anfrage `GET /Language/7` würde also beispielsweise in folgenden Code umgewandelt:

```
$instance = new Language();
$instance->get(array(7));
```

Wie bereits erwähnt fehlt an dieser Stelle eine vernünftige Fehlerbehandlung; es sollte überprüft werden, ob die angesprochene Klasse und die gewünschte Methode überhaupt existieren.

Zum Schluss erfolgt die HTTP-Ausgabe: der Status, der MIME-Type und die Länge des Bodys werden über die Funktion `header()` ausgegeben, und als Letztes folgt der Body mit `echo`.

Die API-Basisklasse Api.php

Diese Klasse stellt die Grundfunktionalität bereit, die jede API-Ressourcen-Klasse erben sollte. Der Code lautet wie folgt:

```
<?php

class Api {
```



```
  protected $xml = NULL;
  protected $contentType = "text/xml";
  protected $statusCode = 200;
  protected $statusMessage = "OK";

  public function get($elements) {
    return $this->notImplemented();
  }

  public function post($elements) {
    return $this->notImplemented();
  }

  public function put($elements) {
    return $this->notImplented();
  }

  public function delete($elements) {
    return $this->notImplemented();
  }

  public function contentType($type = NULL) {
    if ($type !== NULL) {
      $this->contentType = $type;
    }
    return $this->contentType;
  }

  public function statusCode($code = NULL) {
    if ($code !== NULL) {
      $this->statusCode = $code;
    }
    return $this->statusCode;
  }

  public function statusMessage($message = NULL) {
    if ($message !== NULL) {
      $this->statusMessage = $message;
    }
    return $this->statusMessage;
  }

  protected function notFound() {
```

```
    $this->statusCode(404);
    $this->statusMessage('Not found');
    return $this
      ->xml()
      ->getElement(
          'The requested resource could not be found.',
          'error'
        );
  }

  protected function badRequest($message = '') {
    $this->statusCode(400);
    $this->statusMessage('Bad request');
    return $this
      ->xml()
      ->getElement(
          empty($message) ? 'This request is formally incorrect.' : $message,
          'error'
        );
  }

  protected function notImplemented() {
    $this->statusCode(501);
    $this->statusMessage('Not implemented');
    return $this
      ->xml()
      ->getElement(
          'This request method is not implemented.',
          'error'
        );
  }

  protected function forbidden() {
    $this->statusCode(403);
    $this->statusMessage('Forbidden');
    return $this
      ->xml()
      ->getElement(
          'You are not allowed to access this resource.',
          'error'
        );
  }
```



```
  protected function checkAuthorization() {
    $result = FALSE;
    if (defined('AUTH_USER') && defined('AUTH_KEY')) {
      if (isset($_GET['user']) && $_GET['user'] == AUTH_USER &&
          isset($_GET['key']) && md5($_GET['key']) == AUTH_KEY) {
        $result = TRUE;
      }
    }
    return $result;
  }

  public function xml($xml = NULL) {
    if ($xml !== NULL) {
      $this->xml = $xml;
    } elseif ($this->xml === NULL) {
      $this->xml = new Xml();
    }
    return $this->xml;
  }
}
```

Die Klasse beginnt mit der Deklaration einiger Attribute. Sie haben die Sichtbarkeitsstufe `protected`, damit sie auch in abgeleiteten Klassen zur Verfügung stehen. `$xml` ist eine Instanz der Helferklasse `Xml`, die jeweils bei Bedarf initialisiert wird. `$contentType`, `$statusCode` und `$statusMessage` sind Felder für die verschiedenen HTTP-Header, die über verschiedene Methoden gesetzt und ausgelesen werden können.

Es folgen die öffentlichen Methoden `get()`, `post()`, `put()` und `delete()` für die gleichnamigen HTTP-Anfragemethoden. In dieser Basisklasse liefern sie alle den Status `501 Not implemented` zurück; in abgeleiteten Ressourcenklassen brauchen Sie also nur diejenigen zu überschreiben, die Sie tatsächlich benötigen.

Die Methoden `contentType()`, `statusCode()` und `statusMessage()` sind kombinierte Getter und Setter für die entsprechenden Attribute: Wenn das optionale Argument gesetzt ist, werden die Attribute auf den angegebenen Wert gesetzt, und er wird in jedem Fall zurückgegeben. Die Methode `xml()` am Ende der Klasse funktioniert ähnlich, initialisiert die Instanz jedoch zusätzlich mit `new Xml()`, falls sie nicht übergeben wird und noch nicht existiert. Die Möglichkeit, das `Xml`-Objekt von außen zu setzen, ist vor allem für Unit-Tests interessant, denn so können Sie der `Api`-Instanz anstelle des Originals ein Mock-Objekt übergeben.

Die `protected`-Methoden `notFound()`, `badRequest()`, `notImplemented()` und `forbidden()` können Sie in Ihren Ressourcen-Klassen aufrufen, um die entsprechenden Fehlermeldungen an Clients zurückzuliefern. Die HTTP-Methoden in der vorliegenden Klasse zeigen am Beispiel von `notImplemented()`, wie dies funktioniert. Alle vier Methoden erzeugen den Body mithilfe

der `Xml`-Methode `getElement()`, die einen einzelnen String in das angegebene XML-Tag verschachtelt (hier jeweils `<error>...</error>`). Im Fall von `badRequest()` können Sie die Original-Fehlermeldung optional überschreiben, um dem Client genauer mitzuteilen, was mit seiner Anfrage nicht stimmt. Neben der Rückgabe der jeweils passenden XML-Fehlermeldung setzen die Methoden auch Statuscode und -meldung entsprechend.

Die Methode `checkAuthorization()` prüft schließlich, ob der API-User berechtigt ist, eine bestimmte Anfrage durchzuführen. Dazu müssen die Konstanten `AUTH_USER` und `AUTH_KEY` in der Konfigurationsdatei gesetzt sein. Sie werden mit den Werten der Query-String-Parameter `user` beziehungsweise `key` verglichen, wobei von Letzterem der MD5-Hash gebildet wird. Nur bei einer Übereinstimmung liefert die Methode das Ergebnis `TRUE` zurück, das heißt, der User ist autorisiert.

Wenn Sie `checkAuthorization()` in einer Methode einer von `Api` abgeleiteten Ressourcenklasse verwenden möchten, funktioniert dies beispielsweise so:

```
public function post($elements) {
  if (!$this->checkAuthorization()) {
    return $this->forbidden();
  }
  // Hier ist der Client autorisiert: POST-Anfrage weiterverarbeiten
}
```

Die API-Ressourcenklasse Language.php

Die Klasse `Language` wird von der soeben beschriebenen Klasse `Api` abgeleitet. Ihr PHP-Code sieht so aus:

```
<?php

class Language extends Api {
  private $dba = NULL;

  public function get($elements) {
    if (isset($elements[0]) && !empty($elements[0])) {
      return $this->getOne($elements[0]);
    }
    return $this->getAll();
  }

  public function post($elements) {
    if (!$this->checkAuthorization()) {
      return $this->forbidden();
    }
    $raw = file_get_contents('php://input');
```



```
    if (!empty($raw)) {
      $data = $this->dba()->parse($raw);
      $id = $this->dba()->create($data);
      if ($id) {
        return $this
          ->xml()
          ->getElement(
              sprintf('Created new language with ID %d.', $id),
              'success'
            );
      } else {
        return $this->badRequest("Could not create a new record; database error.");
      }
    }
    return $this->badRequest("You provided no data to insert.");
  }

  public function put($elements) {
    if (!$this->checkAuthorization()) {
      return $this->forbidden();
    }
    if (isset($elements[0])) {
      $id = $elements[0];
      $raw = file_get_contents('php://input');
      if (!empty($raw)) {
        $data = $this->dba()->parse($raw);
        $success = $this->dba()->update($id, $data);
        if ($success) {
          return $this
            ->xml()
            ->getElement(
                'Language successfully modified.',
                'success'
              );
        }
      }
    }
    return $this->badRequest();
  }

  public function delete($elements) {
    if (!$this->checkAuthorization()) {
      return $this->forbidden();
```

```
    }
    if (isset($elements[0])) {
      $id = $elements[0];
      $success = $this->dba()->delete($id);
      if ($success) {
        return $this
          ->xml()
          ->getElement(
              'Language successfully deleted.',
              'success'
            );
      }
    }
    return $this->badRequest();
  }

  protected function getOne($id) {
    $language = $this->dba()->getById($id);
    if (!empty($language)) {
      return $this->xml()->getRecord($language, 'language');
    }
    return $this->notFound();
  }

  protected function getAll() {
    if (isset($_GET['search'])) {
      $data = $this->dba()->getBySearch($_GET['search']);
    } else {
      $data = $this->dba()->getAll();
    }
    if (!empty($data)) {
      return $this->xml()->getRecords($data, 'languages', 'language');
    }
    return $this->notFound();
  }

  public function dba($dba = NULL) {
    if ($dba !== NULL) {
      $this->dba = $dba;
    } elseif ($this->dba === NULL) {
      $this->dba = new LanguageDba();
    }
```



```
    return $this->dba;
  }
}
```

Die Klasse deklariert zunächst ein privates Attribut namens `$dba`. Es handelt sich um eine Instanz der Klasse `LanguageDba` für die Datenbankzugriffe, die genau wie die `Xml`-Instanz in der Elternklasse über eine kombinierte Getter- und Setter-Methode namens `dba()` initialisiert wird.

Ansonsten werden im Wesentlichen die Methoden für die verschiedenen HTTP-Zugriffsarten definiert. `get()` überprüft dabei zunächst, ob der Pfad hinter `Language/` ein weiteres Element enthielt – dies wird als ID interpretiert und über die interne Helfermethode `getOne()` ausgelesen. Andernfalls wird der Rückgabewert von `getAll()` zurückgeliefert.

Die Methode `getOne()` ruft ihrerseits die Methode `getById()` der DBA-Klasse auf. Wenn diese ein Ergebnis zurückliefert, bedeutet dies, dass der Datensatz gefunden wurde; er wird mithilfe der `Xml`-Methode `getRecord()` zu XML mit dem Wurzelelement `<language>...</language>` verarbeitet und zurückgegeben. Andernfalls wird `notFound()` aufgerufen, um dem Client den Fehlerstatus `404 Not found` zu präsentieren.

`getAll()` funktioniert ähnlich, prüft aber vor dem eigentlichen Datenbankzugriff, ob der URL-Parameter `search` gesetzt ist. Ist dies der Fall, dann wird der Wert dieses Parameters an die DBA-Methode `getBySearch()` übergeben, die nur Datensätze zurückliefert, in denen `language_name` den angegebenen String enthält. Andernfalls werden mit der DBA-Methode `getAll()` alle Datensätze ausgelesen. Wenn ein Ergebnis vorhanden ist, wird es mit `getRecords()` aus der Klasse `Xml` in das Ausgabeformat umgewandelt und zurückgegeben, ansonsten gibt es wieder eine 404-Fehlermeldung.

Die Methoden `post()`, `put()` und `delete()` überprüfen zunächst nach dem bereits gezeigten Schema, ob der Client ordnungsgemäß autorisiert ist. Ist dies nicht der Fall, wird 403 Forbidden zurückgeliefert. Wenn die Autorisierung erfolgreich war, lesen `post()` und `put()` den Inhalt von *php://input* aus, um den Body der Anfrage zu erhalten. Ist dieser leer, wird `badRequest()` mit einer spezifischen Fehlermeldung zurückgegeben. Ansonsten parst die Methode `parse()` der Klasse `LanguageDba` das übergebene XML, und es wird mit den Methoden `create()` beziehungsweise `update()` derselben Klasse in die Datenbank geschrieben. Im Erfolgsfall wird `200 OK` mit einer spezifischen XML-Erfolgsmeldung zurückgegeben, ansonsten `badRequest()`.

`delete()` funktioniert so ähnlich wie die beiden anderen Schreibmethoden, überprüft aber keinen Anfrage-Body, sondern nur die ID des zu löschenden Datensatzes aus dem Pfad.

Die grundlegende Datenbank-Klasse Database.php

Der Quellcode dieser Datei wurde in diesem Kapitel bereits abgedruckt, und zwar im Abschnitt »Eine Klasse für vereinfachte Datenbankzugriffe«. Blättern Sie dorthin, falls Sie den Code noch nicht gelesen oder abgetippt haben.

Die allgemeine Datenbank-Zugriffsklasse Dba.php

Diese Klasse ist die Elternklasse für spezifische Datenbank-Zugriffsklassen der API-Ressourcen. Sie ist von `Database` abgeleitet und stellt die Umwandlung der internen Datenbank- in die öffentlichen API-Feldnamen und umgekehrt bereit. Hier der Inhalt:

```
<?php

class Dba extends Database {
  protected $fields = array();

  protected function recordToResult($record, $fields = NULL) {
    if ($fields === NULL) {
      $fields = $this->fields;
    }
    $result = array();
    if (is_array($record)) {
      foreach ($record as $field => $value) {
        if (isset($fields[$field])) {
          $result[$fields[$field]] = $value;
        } else {
          $result[$field] = $value;
        }
      }
    }
    return $result;
  }

  protected function resultToRecord($result) {
    return $this->recordToResult($result, array_flip($this->fields));
  }
}
```

Das in dieser Klasse zunächst leere Attribut `$fields` ordnet die Feldnamen der Datenbank den öffentlichen XML-Elementnamen der API zu; als Schlüssel werden die Datenbank-Feldnamen erwartet und als Werte die XML-Elemente. Die Methode `recordToResult()` wandelt Datenbank-Datensätze in Ergebnis-Arrays um; wenn ein Feld nicht vorhanden ist, wird der Original-Feldname beibehalten. Optional kann die Methode eine Liste von Feldern als Argument entgegennehmen – das macht sich die Umkehrfunktion `resultToRecord()` zunutze, indem sie Schlüssel und Werte mittels `array_flip()` umkehrt und wiederum `recordToResult()` aufruft.

Ausgewachsene Web-Frameworks verwenden oft mehr Automatisierung für Datenbankzugriffe, bis hin zu so genannten *ORM*-Schnittstellen (*Object-Relational Mapping*), die eine Datenbanktabelle mehr oder weniger automatisch einer Klasse zuordnen. Dies ist bequem, geht aber mitunter auf Kosten der Datenbank-Performance.

Die Datenbank-Zugriffsklasse Language/Dba.php

Diese Klasse stellt die spezifischen Datenbank-Zugriffsmethoden für die Tabelle `languages` bereit. Sie ist von `Dba` abgeleitet und enthält das folgende PHP:

```
<?php

class LanguageDba extends Dba {
  private $xml = NULL;

  protected $fields = array(
    'language_id' => 'id',
    'language_name' => 'name',
    'language_architecture' => 'architecture',
    'language_implementation' => 'implementation',
    'language_system' => 'system',
    'language_description' => 'description',
    'language_year' => 'year'
  );

  public function getAll() {
    $sql = "SELECT language_id, language_name, language_architecture,
                   language_implementation, language_system,
                   language_description, language_year
              FROM languages
             ORDER BY language_name";
    $result = array();
    $query = $this->query($sql);
    while ($row = $query->fetch_assoc()) {
      $result[$row['language_id']] = $this->recordToResult($row);
    }
    return $result;
  }

  public function getById($id) {
```

```
  $sql = "SELECT language_id, language_name, language_architecture,
                 language_implementation, language_system,
                 language_description, language_year
            FROM languages
           WHERE language_id = %d";
  $query = $this->query($sql, $id);
  return $this->recordToResult($query->fetch_assoc());
}

public function getBySearch($search) {
  $condition = $this->getCondition(array('language_name' => '%'.$search.'%'));
  $sql = "SELECT language_id, language_name, language_architecture,
                 language_implementation, language_system,
                 language_description, language_year
            FROM languages
           WHERE ".$condition;
  $result = array();
  $query = $this->query($sql);
  while ($row = $query->fetch_assoc()) {
    $result[$row['language_id']] = $this->recordToResult($row);
  }
  return $result;
}

public function create($data) {
  return $this->insertQuery('languages', $this->resultToRecord($data));
}

public function update($id, $data) {
  return $this->updateQuery(
    'languages',
    $this->resultToRecord($data),
    'language_id = %d',
    $id
  );
}

public function delete($id) {
  $sql = "DELETE FROM languages WHERE language_id = %d";
  $this->query($sql, $id);
  return $this->getAffectedRows();
}
```



```
  public function parse($rawXml) {
    return $this->xml()->parse($rawXml, $this->fields);
  }

  public function xml($xml = NULL) {
    if ($xml !== NULL) {
      $this->xml = $xml;
    } else {
      $this->xml = new Xml();
    }
    return $this->xml;
  }
}
```

Auch diese Klasse verwendet eine Instanz der Klasse `Xml`, um einen Aufruf der Methode `parse()` an diese durchzureichen. Da die gleichnamige Methode von `Xml` eine Liste der Felder benötigt, ist es sinnvoll, diesen Umweg zu gehen.

Die Felderliste `$fields` lässt im Fall dieser Klasse bei den XML-Elementnamen das Präfix `language_` der Tabellenfelder weg.

Die Methoden `getAll()`, `getById()` und `getBySearch()` erzeugen SQL-`SELECT`-Anfragen, um die gewünschten Datensätze auszulesen. `getAll()` benötigt keine Parameter, da stets alle Datensätze ausgelesen werden. Bei `getById()` wird die übergebene ID als einfacher Parameter einer `WHERE`-Bedingung verwendet. `getBySearch()` konstruiert die Suchbedingung zunächst mithilfe der Database-Methode `getCondition()`, da diese die entsprechende Sonderbehandlung für `LIKE`-Platzhalter bietet.

Die Methoden `getAll()` und `getBySearch()` können mehrere Datensätze zurückliefern, sodass das Ergebnis von `query()` jeweils in einer Schleife aufgerufen wird; `getById()` gibt dagegen höchstens einen Datensatz zurück. In jedem Fall wird für alle Datensätze `recordToResult()` aufgerufen, um die Feldnamen zu ersetzen.

Die Schreibmethoden sind recht kurz. Insbesondere `create()` und `update()` reichen die Daten nach einem Aufruf von `resultToRecord()` einfach an die Database-Methoden `insertQuery()` beziehungsweise `updateQuery()` durch. `delete()` sendet eine spezifisch erzeugte Abfrage an die Datenbank und gibt die Anzahl der betroffenen Datensätze zurück.

Die XML-Helferklasse Xml.php

Diese Klasse enthält Methoden zum Erzeugen und Parsen von XML. Dies ist ihr Quellcode:

```
class Xml {
  public function getRecords($records, $root = 'result', $line = 'record') {
    $result = '<?xml version="1.0" encoding="utf-8" standalone="yes"?>';
    $result .= sprintf('<%s>', $root);
```

```
    foreach ($records as $id => $record) {
      $result .= sprintf('<%s id="%d">', $line, $id);
      foreach ($record as $field => $value) {
        $result .= sprintf(
          '<%1$s>%2$s</%1$s>',
          htmlspecialchars($field),
          htmlspecialchars($value)
        );
      }
      $result .= sprintf('</%s>', $line);
    }
    $result .= sprintf('</%s>', $root);
    return $result;
  }

  public function getRecord($record, $line = 'record') {
    $result = '<?xml version="1.0" encoding="utf-8" standalone="yes"?>';
    $result .= sprintf('<%s>', $line);
    foreach ($record as $field => $value) {
      $result .= sprintf(
        '<%1$s>%2$s</%1$s>',
        htmlspecialchars($field),
        htmlspecialchars($value)
      );
    }
    $result .= sprintf('</%s>', $line);
    return $result;
  }

  public function getElement($text, $element = 'message') {
    $result = '<?xml version="1.0" encoding="utf-8" standalone="yes"?>';
    $result .= sprintf(
      '<%1$s>%2$s</%1$s>',
      htmlspecialchars($element),
      htmlspecialchars($text)
    );
    return $result;
  }

  public function parse($xml, $fields) {
    $result = array();
    $record = new SimpleXMLElement($xml);
    foreach ($fields as $field) {
```



```
      if ($record->{$field} != NULL && !empty((string)$record->{$field})) {
        $result[$field] = (string)$record->{$field};
      }
    }
    return $result;
  }
}
```

Die Methoden `getRecords()`, `getRecord()` und `getElement()` wandeln ein verschachteltes Array, ein einfaches Array beziehungsweise einen String in XML-Code um. Bei `getRecords()` können optional der Name des Wurzelelements und des Elements für jeden einzelnen Datensatz angegeben werden. `getRecord()` und `getElement()` nehmen nur einen Elementnamen entgegen, jeweils für das Wurzelelement.

`parse()` dürfte die interessanteste Methode der Klasse sein: Sie verwendet die PHP-Schnittstelle SimpleXML, um das vom Client übergebene XML zu parsen. Neben dem XML-String erwartet sie eine Liste von Feldern, die als Elemente unterhalb des Wurzelelements gesucht werden. Zunächst wird mit `new SimpleXMLElement($xml)` ein SimpleXML-Baum-Objekt erzeugt. Anschließend wird mit `foreach` über die Felderliste iteriert. Wenn ein XML-Element mit dem gesuchten Feldnamen vorhanden ist – es handelt sich bei SimpleXML um ein öffentliches Attribut dieses Namens –, dann wird der Textinhalt des Elements unter dem entsprechenden Schlüssel zum Ergebnis-Array hinzugefügt. Den Textinhalt erhalten Sie in SimpleXML, indem Sie das gewünschte Element mit `(string)$element` in einen String umwandeln.

19.2.4 Die API testen

Eine REST-API können Sie nicht testen wie eine gewöhnliche Webanwendung, die im Browser angezeigt und über Links und Formulare bedient wird. Am besten ist es, parallel zur API einen Client zu programmieren. Dieser ermöglicht zum einen das Testen der API und ist zum anderen eine Referenzimplementierung eines Clients, die Sie nach Fertigstellung der API an Entwickler verteilen können, die Ihre API benutzen sollen.

In den nächsten beiden Kapiteln werden verschiedene Clients für diese API entwickelt, aber zunächst wird hier eine einfachere Möglichkeit vorgestellt. Es handelt sich um die Verwendung des generischen REST-Clients *Postman*, der als kostenlose Erweiterung für den Browser Google Chrome verfügbar ist.

Um Postman zu installieren, wählen Sie im Menü von Chrome zunächst Fenster • Erweiterungen. Klicken Sie dann auf den Link Mehr Erweiterungen herunterladen. Geben Sie in das Suchfeld links oben `postman` ein, und klicken Sie im Suchtreffer Postman – angeboten von www.getpostman.com auf die Schaltfläche Hinzufügen. Nach der Installation steht Postman im Bereich Apps des Browsers zur Verfügung.

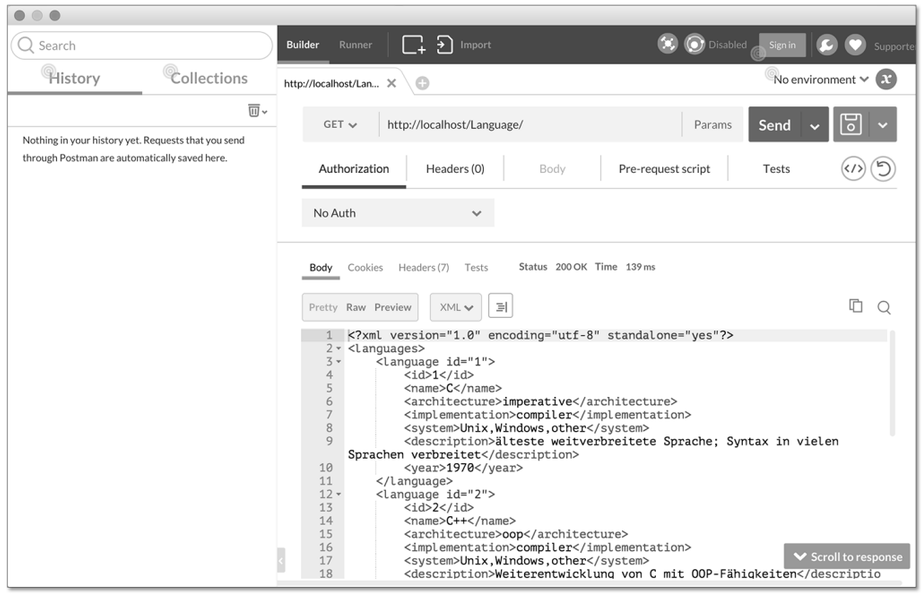


Abbildung 19.2 Der REST-Client »Postman« beim Ausführen der Anfrage GET /Language

Abbildung 19.2 zeigt den Postman bei der Anzeige der Anfrage GET /Language/, die die Liste aller Programmiersprachen liefert. Die Bedienung ist denkbar einfach:

- Im Hauptbereich (rechts) können Sie links oben zunächst die HTTP-Methode wählen; neben den vier klassischen REST-Methoden stehen auch alle anderen Methoden zur Verfügung.
- Rechts neben der Methode wird die URL eingegeben, an die die Anfrage geschickt werden soll – in diesem Fall muss sie vollständig sein, also etwa *http://localhost/Language/*.
- Wenn Sie einen Query-String benötigen, können Sie ihn einfach durch ein Fragezeichen getrennt hinter die Basis-URL schreiben oder aber auf PARAMS klicken, um die Felder und Werte in eine bequeme Tabelle einzugeben.
- Für den Body einer POST- oder PUT-Anfrage wechseln Sie auf die Registerkarte BODY. Wählen Sie für das Format des Bodys den Radiobutton RAW; im Pulldown-Menü Text rechts daneben können Sie auch den passenden Typ XML auswählen. Ob Sie den konkreten MIME-Type TEXT/XML oder APPLICATION/XML wählen, ist für die vorliegende API egal, aber andere REST-APIs bestehen möglicherweise auf einen der beiden.
- Wenn Ihre Anfrage fertig konfiguriert ist, klicken Sie auf SEND, um sie abzuschicken. Das Ergebnis wird im unteren Bereich angezeigt; eventuell müssen Sie vorher zurück auf die Registerkarte AUTHORIZATION wechseln.

- Die linke Spalte enthält die HISTORY aller bereits gesendeten Anfragen. Wenn Sie eine von ihnen anklicken, können Sie diese im rechten Bereich ganz nach Wunsch modifizieren und schließlich erneut abschicken.

Cloud Computing

In den letzten Jahren wird immer häufiger der Begriff *Cloud Computing* verwendet. Darunter versteht man die Möglichkeit, Server, Software und Datenspeicherplatz dynamisch zu mieten, oft auf stündlicher Basis. Im Einzelnen werden drei verschiedene Dienstleistungen unterschieden:

- *Infrastructure as a Service* (IaaS) stellt virtualisierte Hardware, also Serverrechner mit Betriebssystem, bereit. Kunden können meist zwischen diversen Linux- und Windows-Versionen wählen.
- *Platform as a Service* (PaaS) bietet gezielt bestimmte Serverdienste an, beispielsweise Datenbanken, Streaming-Server oder auch Storage.
- *Software as a Service* (SaaS) schließlich ermöglicht es dem Kunden, bestimmte, oft sehr teure Software flexibel zu mieten, anstatt zu kaufen. Dies ist unter Umständen billiger, als eigene Lizenzen zu erwerben, und zudem werden neue Versionen umgehend bereitgestellt, ohne dass der Kunde mehr bezahlen muss.

Wichtige Anbieter von Cloud-Computing-Diensten sind unter anderem:

- *Amazon Web Services* (AWS) mit Angeboten wie EC2 (IaaS und PaaS) und S3 (Storage)
- Microsoft mit der Cloud-Plattform *Azure*. Hier gibt es aus naheliegenden Gründen nur Windows- und keine Linux-Server.
- *Rackspace*, ursprünglich ein klassischer Hoster, hat seit einigen Jahren auch verschiedene Cloud-Dienste im Angebot. Eine Besonderheit ist die Möglichkeit, Cloud-Server gegen Aufpreis als Managed Service zu mieten, bei dem Mitarbeiter von Rackspace die Administration weitgehend übernehmen.
- Einen besonderen Weg geht NewServers mit der Bare Metal Cloud: Hier können anstelle der üblichen virtuellen Serverinstanzen dedizierte Server auf Stundenbasis gemietet werden.

Bei Cloud-Server-Angeboten können die Kunden zwischen verschiedenen Leistungsmerkmalen zu unterschiedlichen Preisen wählen – da es sich außer bei NewServers um virtuelle Instanzen handelt, bezieht sich die Leistung auf den zugesicherten Anteil an CPU, RAM und Festplattenspeicher.

Ein ernst zu nehmendes Cloud-Angebot für Enterprise-Anwendungen ist komplett über eine API steuerbar. In aller Regel kommen dabei REST-basierte *XML Web Services* zum Einsatz. Der Benutzer des Dienstes kann über die API Serverinstanzen in Betrieb nehmen oder freigeben, Daten mit ihnen austauschen, Programme starten und beenden, Daten zur Speicherung in einem Cloud-Storage-Service hochladen oder Statistikdaten erhalten.

Cloud-Anbieter garantieren in der Regel eine hohe Ausfallsicherheit; normalerweise werden 99 bis 100 % Uptime pro Jahr angeboten (geplante und angekündigte Unterbrechungen nicht mitgerechnet). Sollte der Dienstleister diese Quote nicht einhalten können, wird dem Kunden die entsprechende Zeit beziehungsweise Kapazität meist zur späteren Verwendung gutgeschrieben.

Dass Cloud Computing nicht unfehlbar ist (und stets durch lokale Backups ergänzt werden sollte), zeigte sich jedoch beispielsweise, als Amazons Storage-Service Anfang 2011 ausfiel und zum Teil unwiederbringliche Datenverluste verursachte. Auch auf die Sicherheit sollten die Nutzer solcher Dienste achten. Zwar erfolgt die Kommunikation mit den Cloud-Diensten in der Regel über eine gesicherte HTTPS-Verbindung, aber auf den Storage-Servern selbst werden die Daten unverschlüsselt gespeichert. Der Kunde sollte sie also selbst verschlüsseln und erst dann in der Cloud ablegen.

Gute Cloud-Computing-Dienste verfügen über mehrere Rechenzentren an verschiedenen wichtigen Orten der Welt, mindestens in Nordamerika, Europa und Ostasien. Die Kunden können dabei jeweils wählen, auf welche Rechenzentren sie die Last verteilen möchten.

Neben diesen Cloud-Computing-Diensten, die sich fast ausschließlich an Firmenkunden richten, gibt es auch Cloud-Dienste für Privatanwender. Meistens handelt es sich um Storage-Lösungen. Beispielsweise ist Apple gerade dabei, seinen iCloud-Dienst einzuführen, den Anwender von OS X seit Mac OS X 10.7 (Lion) und iOS seit Version 5 vollautomatisch und ohne Zusatzkosten nutzen können.

19.3 Übungsaufgaben

1. Schreiben Sie ein PHP-Skript, das einen beliebigen String an den Stellen, an denen die Satzzeichen Punkt (`.`), Ausrufezeichen (`!`) oder Fragezeichen (`?`), gefolgt von einem Leerzeichen oder dem Zeilenende, vorkommen, in ein Array zerlegt, also einzelne Sätze daraus macht. Anschließend sollen diese Sätze mittels `usort()` nach der Länge sortiert und ausgegeben werden (kürzester zuerst).
2. Finden Sie die Syntax- und Logikfehler in der folgenden PHP-Funktionsdefinition, möglichst ohne sie auszuführen:

```
function add($a = 5, $b) {
  $a += $b;
  return $a;
  if $b < 10 {
    return 0;
  }
}
```


3. Leiten Sie die Klassen `Html`, `Head`, `Title`, `Meta`, `Body`, `Hn` (`<h1>` bis `<h6>` mit zusätzlichem Parameter für die Hierarchiestufe) und `P` zur Ausgabe der gleichnamigen HTML-Tags von der Klasse `HtmlTag` in diesem Kapitel ab, und geben Sie mithilfe dieser Klassen das folgende HTML-Dokument aus:

```
<html>
  <head>
    <title>HTML-Ausgabe durch PHP-Klassen</title>
    <meta http-equiv="Content-type" content="text/html; charset=utf-8" />
    <meta name="description" content="Von PHP generiertes HTML" />
  </head>
  <body>
    <h1>HTML-Ausgabe</h1>
    <p>Dieses Dokument wird mithilfe von PHP-Klassen ausgegeben.</p>
    <h2>HtmlTag</h2>
    <p>Dies ist die Elternklasse aller HTML-Tags.</p>
    <h2>Html, Head, Body, P etc.</h2>
    <p>Diese Klassen sind von HtmlTag oder voneinander abgeleitet.</p>
    <p align="right">Aus: "IT-Handbuch für Fachinformatiker"</p>
  </body>
</html>
```

4. Schreiben Sie ein PHP-Skript, das das Pizzabestellformular aus dem vorangegangenen Kapitel ausgibt, dessen Daten nach dem Abschicken entgegennimmt und eine Meldung wie »Ihre Bestellung wurde entgegengenommen« zusammen mit dem Gesamtpreis der Bestellung ausgibt. Die Grundpreise für die Größen und der Preis pro Zutat (der Einfachheit immer derselbe) sollen im Skript als Variablen festgelegt werden, damit sie bei Bedarf geändert werden können.
5. Ergänzen Sie die Beispieldatenbank mit den Programmiersprachen um eine Tabelle `authors` mit den Feldern `author_id`, `author_firstname` und `author_lastname` sowie eine Tabelle `languages_authors`, mit der die m:n-Beziehung zwischen den Programmiersprachen und ihren Erfindern hergestellt wird (Felder `la_author_id` und `la_language_id`). Finden Sie (zum Beispiel aus der Wikipedia) heraus, wer die verschiedenen Sprachen erfunden hat, und fügen Sie diese Informationen in die Datenbank ein. Schreiben Sie anschließend ein PHP-Skript, das eine JOIN-Abfrage auf die Tabellen ausführt und die Sprachen sowie ihre Autoren in einer HTML-Tabelle ausgibt.
6. Fügen Sie PHPDoc-Kommentare zu allen Klassen der REST-API hinzu.
7. Schreiben Sie Unit-Tests für alle Klassen und Methoden der REST-API, soweit sie testbar sind; Mock-Objekte können Sie dabei über die kombinierten Getter/Setter-Methoden wie `dba()` oder `xml()` einfügen.
8. Erweitern Sie die REST-API um eine weitere Ressource namens Author (Klassendateien *Author.php* und *Author/Dba.php*). Diese soll Informationen über Programmiersprachen-

erfinder für die Tabelle aus Aufgabe 5 verwalten. Erweitern Sie die Language-Klassen anschließend so, dass sie Informationen über die Erfinder einer Sprache als verschachteltes XML in folgendem Format ausgeben:

```
<authors>
  <author>
    <firstname>Vorname</firstname>
    <lastname>Nachname</lastname>
  </author>
  <!-- evtl. weitere Autoren -->
</authors>
```

Zusätzlich wird eine Ressource namens `LanguageAuthor` benötigt, mit der die Informationen für die Tabelle `language_authors` in numerischer Form jeweils einzeln eingesetzt und gelöscht, aber nicht geändert oder ausgelesen werden können (Sie brauchen hier also nur `insert()` und `delete()` zu implementieren, während `get()` und `update()` von der Klasse `Api` geerbt werden und mit `501 Not implemented` antworten).

9. Wenn Sie das nächste Kapitel, »JavaScript und Ajax«, durchgearbeitet und JSON kennengelernt haben (oder bereits kennen), erweitern Sie die API so, dass sie zusätzlich dieses Datenaustauschformat beherrscht: Der `Accept`-Header der Clientanfrage soll `application/json` anfordern, wenn JSON als Ausgabeformat erwünscht ist; ansonsten wird wie bisher XML geliefert. Entsprechend soll der Body einer `POST`- oder `PUT`-Anfrage als JSON ausgewertet werden, wenn der Anfrage-Header `Content-type` den Wert `application/json` hat, und sonst XML.

Auf einen Blick

Inhalt

Anhang 1219

Index

A

B

C

D

E

F

G

H

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y

Z

Sascha Kersken arbeitet seit vielen Jahren als Softwareentwickler sowie als Trainer für EDV-Schulungen in den Themengebieten Netzwerke und Internet, interaktive Medien und Programmierung. Er hat zahlreiche Fachbücher und Artikel zu verschiedenen IT-Themen geschrieben.

Sascha Kersken

IT-Handbuch für Fachinformatiker – Der Ausbildungsbegleiter

1.304 Seiten, gebunden, 7. Auflage 2015
34,90 Euro, ISBN 978-3-8362-3473-3

www.rheinwerk-verlag.de/3756