Einführung in die maschinennahe, imperative, funktionale, relationale und objektorientierte Programmierung

-

EMIFROP 0.26

Markus Alpers
B.Sc. und Ausbilder f. Industriekaufleute

29. Februar 2016

Inhaltsverzeichnis

Hinweis bezüglich diskriminierender Formulierungen

In diesem Text wurde darauf geachtet Formulierungen zu vermeiden, die diskriminierend verstanden werden können. Im Sinne der Lesbarkeit wurden dabei Formulierungen wie „Informatiker und Informatikerinnen“ durch „InformatikerInnen“ (mit großem i) ersetzt. An anderen Stellen habe ich Formen wie eine/einer durch eineR zusammengefasst. Hier berufe ich mich auf den Artikel „Sprache und Ungleichheit“ der Bundeszentrale für politische Bildung, kurz BpB, vom 16. April 2014, insbesondere auf den Absatz „Zum Umgang mit diskriminierender Sprache“, online abrufbar unter:

`http://www.bpb.de/apuz/130411/sprache-und-ungleichheit?p=all`

Sollten Sie dennoch Formulierungen entdecken, die diesem Anspruch nicht entsprechen, möchte ich Sie bitten, mir eine entsprechende Nachricht zu senden, denn es ist mir wichtig, Ihnen mit diesem Buch eine wertvolle Unterstützung beim Start in die faszinierende Welt der Informatik zu bieten. Das sollte nicht durch verletzte Gefühle in Folge missverständlicher Formulierungen torpediert werden.

Sie erreichen mich unter `markus.alpers@haw-hamburg.de`.

Hinweis zur Lizenz

zenz"´. Wenn Sie einen Teil finden, in dem diese „Hinweise zur Lizenz"´ nicht zu finden ist, dann dürfen Sie für den persönlichen Gebrauch dennoch Kopien davon anfertigen und Sie dürfen diese Kopien außerhalb von kommerziellen Projekten frei verwenden.

Hinweis zur Verwendbarkeit in wissenschaftlichen Arbeiten

Bitte beachten Sie dabei aber, dass die Verwendung dieses Textes im Rahmen wissenschaftlicher Publikationen zurzeit aus anderen Gründen problematisch ist: Wie viele andere Quellen, die frei im Internet verfügbar sind, wurde auch dieser Text bislang nicht durch einen nachweislich entsprechend qualifizierten Lektor verifiziert. Damit genügen Zitate aus diesem Band streng genommen noch nicht den Ansprüchen wissenschaftlicher Arbeiten.

Bitte beachten Sie außerdem, dass dieses Buch eine Konvention nutzt, die in wissenschaftlichen Arbeiten verpönt ist: Wenn in einer wissenschaftlichen Arbeit ein Begriff hervorgehoben wird, dann wird dazu kursive Schrift verwendet. In diesem Buch verwende ich dagegen Fettdruck, da es vielen Menschen schwer fällt, einen kursiv gedruckten Begriff schnell zu finden und ich mir wünsche, dass Sie es möglichst effizient auch als Nachschlagewerk nutzen können.

Teil I

Einführung in die Programmierung für alle Studierenden im Bereich MINT

(Mathematik, Ingenieurswissenschaften, Naturwissenschaften, Technik)

Kapitel 1

Typische Irrtümer darüber, was Programmieren ist.

Kapitel 2

Von der Nachrichtentechnik zur Programmierung

Kapitel 3

Vorbereitung fürs Programmieren

Kapitel 4

Ausgewählte Programmiersprachen

In diesem Kapitel stelle ich Ihnen einige Programmiersprachen vor und erkläre, wo die Unterschiede liegen.

Der Begriff der höheren Programmiersprachen wird heute eigentlich nur noch am Rande verwendet, weil die Abgrenzung zur maschinennahen Programmierung (und genau dafür steht der Begriff) zum Normalfall geworden ist. Sollten Sie also über diesen Begriff stolpern und sich fragen, was denn eine höhere Programmiersprache ist, dann merken Sie sich einfach: Es ist eine Abgrenzung, die für uns weitestgehend irrelevant geworden ist.

4.1 Nach B kam C

Die mangelnde Verständlichkeit von maschinennahen Programmen führte dazu, dass Programmiersprachen entwickelt wurden, die Befehle und Zeilenstrukturen beinhalteten, die leichter lesbar waren.

Die Zeile

```
if (a < b) then print "a ist kleiner als b"
```

dürfte auch von Menschen lesbar sein, die lediglich über grundlegende Englischkenntnisse, aber kaum über Computerkenntnisse verfügen.

Im Gegensatz dazu dürfte die Zeile

```
CMP R6 MSP
```

selbst bei denjenigen unter Ihnen für Stirnrunzeln sorgen, die bereits Projekte in Java oder C++ entwickelt haben. (Hier handelt es sich um eine maschinennahe Programmzeile, die bei einem ARM-Prozessor einen Vergleich zwischen zwei Zahlen durchführt.)

Eine dieser höheren Sprachen wurde von ihrem Entwickler **Dennis Ritchie** schlicht **C** genannt. Es gab vorher unter anderem eine Sprache namens B, die als Vorlage für C diente. Die Informatiker der Anfangszeit waren weniger an Marketing interessiert, weshalb Bezeichnungen wie Java, Ruby, Python usw. erst ab den 90er Jahren üblich wurden. Vorher wurden häufig einzelne Buchstaben oder Abkürzungen wie im Falle der Sprache **PROLOG** genutzt, was schlicht für programmable logic steht.

C ist bis heute eine sehr wichtige Sprache, weil sie dafür entwickelt wurde, um **Betriebssystem**e zu entwickeln und dabei möglichst wenig maschinennah programmieren zu müssen. Zusätzlich können Sie mit ihr grundsätzlich jede Form imperativer Programme entwickeln. Für Sprachen, die wie C für alle möglichen Zwecke eingesetzt werden können, wird die Bezeichnung **general purpose programming** (kurz GPP) verwendet.

Das Buch „**The C programming language**"von **Kernighan** und **Ritchie** ist eines der ersten Bücher zur Einführung in die Programmierung mit C. Es ist eher schwer zu nutzen, aber wenn Sie sich durch diesen Band durchgearbeitet haben, dann beherrschen Sie die Grundlagen der imperativen Softwareentwicklung, die InformatikerInnen beherrschen müssen.

Wenn Sie in einer Statistik nachsehen, wie viele Programmierer C nutzen, dann werden Sie feststellen, dass diese einen immer geringeren Anteil aller Programmierer ausmachen. Das hat damit zu tun, das C für die Entwicklung verteilter Anwendungen relativ wenig Unterstützung anbietet. Aber denken Sie deshalb nicht, C sei belanglos geworden; es gibt schlicht wesentlich mehr Bereiche, in denen heute programmiert wird, als in den 70er Jahren, in denen C entwickelt wurde.

Eine **verteilte Anwendung** ist nichts anderes als ein Programm, das auf mehreren miteinander vernetzten Rechnern aktiv ist. Die Probleme, die dabei durch die Kommunikation zwischen den Rechnern entstehen sind eines der anspruchsvollsten Themen, mit denen Sie sich auseinander setzen können.

Kontrolle
Höhere Programmiersprachen sind Programmiersprachen, die für Menschen leichter lesbar sind, als das bei Assembler der Fall ist. C ist hier einer der wichtigsten Vertreter, auch wenn es insbesondere bei der Entwicklung

von verteilten Systemen eher nicht eingesetzt wird.

4.2 C++ : C mit Objektorientierung

Auch wenn höhere Programmiersprachen übersichtlicher und verständlicher als maschinennahe Programmiersprachen sind, ändert das nichts daran, dass irgendwann der Punkt erreicht ist, an dem auch sie nicht genug Übersichtlichkeit bieten. Vielen Informatikern war das bereits in der Frühzeit der Programmierung klar. Seit Mitte der 50er Jahre wurden deshalb immer neue Konzepte erarbeitet, die dann die Basis für verschiedene Sprachen bildeten, die mehr Strukturierungsmöglichkeiten beinhalten, als das bei C der Fall ist.

Für die Zwecke dieser Einführung soll es genügen, wenn Sie wissen, dass C++ der Sprache C entspricht, aber zusätzlich Möglichkeiten zur objektorientierten Programmierung bietet. Wenn nun von **objektorientierter Programmierung** die Rede ist, dann gibt es das Problem, dass es hierfür zwei Interpretationen gibt, die zu gänzlich unterschiedlichen Programmierstilen führen:

4.2.1 Objektorientierung nach Alan Kay

Alan Kay war ein Forscher am **MIT**, der den Begriff der Objektorientierung mitprägte aber diese Bezeichnung später als einen großen Fehler bezeichnete. Denn was er meinte war eine Programmierung, bei der der Fokus auf dem **Nachrichtenaustausch zwischen virtuellen Objekten** liegt. Wohlgemerkt, der Fokus liegt auf dem Nachrichtenaustausch, nicht auf den Objekten selbst.

Wenn Sie sich jetzt daran erinnern, was das wichtigste bei einem Computer ist (die Datenübertragung zwischen den Komponenten des Rechners), dann verstehen Sie auch, warum dieses Konzept der logische Schluss ist. Wenn Sie dann noch an den Aufbau des Internet denken, dann können Sie sich vorstellen, wie grundsätzlich und vorausschauend dieses Konzept ist. Und nochmal: In diesem Bereich kommen wir nur dann zu sinnvollen und effizienten Programmen, wenn **Informatik**erInnen und **Nachrichtentechnik**erInnen zusammen arbeiten.

Aufgabe:
Können Sie jetzt nachvollziehen, warum Kay die Bezeichnung Objektorientierung als großen Fehler bezeichnet hat?

Dieser Begriff suggeriert, dass die virtuellen Objekte das wichtige sind und lassen naive ProgrammiererInnen die Bedeutung des Nachrichtenaustauschs

vergessen. Da wäre der Begriff des **Message Sending** wesentlich passender gewesen. Aber so ist das eben, wenn ein neues Konzept entwickelt wird; da wird eine einprägsame Bezeichnung genutzt und dann gerät alleine dadurch das eigentliche Konzept in Vergessenheit.

Aus diesem Grund sind auch praktisch alle Programmiersprachen, die im Internet zum Einsatz kommen für dieses Einsatzgebiet praktisch nicht geeignet: Für die Probleme beim Nachrichtenaustausch, namentlich zeitliche Verzögerungen und Verluste bieten sie im Regelfall nur beschränkte Lösungsmöglichkeiten, was dementsprechend eher zu mittelmäßigen Programmen führt. Und hier gilt wieder, dass InformatikerInnen und NachrichtentechnikerInnen leider kaum zusammen arbeiten. Täten Sie das, dann würden just die Probleme wesentlich besser gehandhabt werden, die beim programmierten Nachrichtenaustausch auftreten: Die NachrichtentechnikerInnen würden die Probleme bei der Datenübertragung sinnvoll lösen und die InformatikerInnen würden die Probleme bei der Softwareentwicklung sinnvoll lösen.

Eine Sprache, die Message Sending bzw. Objektorientierung nach Kay umsetzt, heißt **Erlang**. Es handelt sich hier um eine Sprache, die mehrere Paradigmen unterstützt. Richtig gelesen: Es gibt Sprachen, die mehrere **Paradigmen** umsetzen. Und tatsächlich ist das bei den meisten Sprachen der Fall. **Java** war beispielsweise bis zur Version 7 eine rein imperative und klassenbasiert objektorientierte Sprache. Seit Version 8 beinhaltet Sie mit der funktionalen Programmierung aber auch ein Konzept der deklarativen Programmierung. Die Version 9 wird kein neues Paradigma einführen, sondern es wird eine massive Restrukturierung geben, die den Speicherbedarf von Java-Programmen deutlich reduzieren wird, die aber auch bei der Programmierung in Java Folgen haben wird.

4.2.2 Objektorientierung nach Lieschen Müller

Damit kommen wir jetzt zu dem, was heute üblicherweise unter Objektorientierung verstanden wird:

Anstelle eines Programms entwickeln wir im Kern lauter kleine Programme, die jeweils einen gewissen Funktionsumfang anbieten und grundsätzlich als **Klasse**n bezeichnet werden. Soll eine bestimmte Funktionalität genutzt werden, erhält die Klasse, die sie enthält einen entsprechenden Befehl, was dann als Methodenaufruf bezeichnet wird. Bitte beachten Sie: Ein Methodenaufruf ist mehr als nur ein einfacher Befehl, aber zu den Unterschieden kommen wir bei der Einführung in die imperative Programmierung, wenn wir uns die sogenannten Funktionen ansehen.

Und auch wenn Klassen, Methoden und Methodenaufrufe zentrale Themen der objektorientierten Programmierung sind, hat dieses Verständnis ungefähr so viel mit Objektorientierung zu tun, wie das Verleimen zweier Holzleisten mit dem Tischlerhandwerk: Es gibt noch wesentlich mehr, was Sie verstanden haben müssen, um wirklich zu verstehen, was Objektorientierung ist.

Das ist auch der Grund, warum man mit der Einführung in die Objektorientierung bereits eine vollwertige Vorlesung für ein oder zwei Semester füllen kann.

Kontrolle
Wenn C Programme zu umfangreich werden, kann man auf C++ zurückgreifen, da es den gleichen Umfang an Befehlen und Strukturen bietet, aber mit der Objektorientierung weitere Strukturierungsmöglichkeiten anbietet. Java ist ebenfalls in diesem Sinne eine objektorientierte Sprache. Beachten Sie bitte, dass bei allen dreien Objektorientierung nicht im Sinne von Alan Kay umgesetzt und angewendet wird, auch wenn das durchaus möglich wäre.

Darüber, was das im Detail bedeutet und wozu es gut ist, haben Sie jetzt noch nichts erfahren. Zerbrechen Sie sich da also bitte nicht den Kopf. Es braucht im Regelfall mehrere Jahre, um diese Punkte verinnerlicht und weitgehend verstanden zu haben.

4.3 Java – C++ ohne maschinennähe

C und damit C++ bieten wie beschrieben die Möglichkeit recht nah an der Maschine zu programmieren, auf der ein Programm laufen soll. Das bringt einen großen Nachteil mit sich: Angreifer können durch geschickt entwickelte Programme in laufende Prozesse eingreifen. Außerdem muss ein C bzw. C++ Programm individuell auf jeden Prozessor zugeschnitten werden, auf dem es laufen soll. Bei der Vielzahl an Prozessoren, die heute in mobilen Endgeräten zum Einsatz kommt ist aber genau dieser letzte Punkt ein ernstes Problem: Nicht nur müssen die Entwickler die Software für jeden Prozessor anpassen, sie müssen insbesondere die Details jedes dieser Prozessoren kennen, sonst entwickeln Sie im besten Falle ineffiziente, im schlimmsten Fall leicht angreifbare Programme. Und wer möchte schon, dass die neueste App ein Einfallstor für Viren und Trojaner wird?!

Deshalb wurde u.a. **Java** entwickelt: Zu einem Zeitpunkt, zu dem nicht nur für Frau Merkel das Internet Neuland war (Anfang der 90er Jahre) entwickelte ein Team bei **Sun Microsystems** diese neue Sprache zusammen mit einem passenden mobilen Endgerät. Um Entwicklern die Umgewöhnung

zu erleichtern, wurden viele Konventionen und Regeln in Java so umgesetzt, wie das bereits in C und C++ der Fall war.

Wenn Sie also bislang dachten, **Apple** sei das Unternehmen, das (mit dem iPhone) das erste Smartphone entwickelt hat, dann liegen Sie schlicht falsch. Hier wie in mehreren anderen Fällen hat Apple (genau wie **Microsoft**) ein Konzept, das andere bereits zuvor ausgearbeitet hatten schlicht zum richtigen Zeitpunkt in einem Produkt umgesetzt und es zum Verkauf angeboten, als es ausreichend Menschen gab, die bereit waren, dafür Geld auszugeben. Das gleiche gilt für grafische Nutzeroberflächen und die Bedienung eines Computers mit der Mouse. Die wurden ebenfalls nicht von Apple entwickelt, sondern von einem Unternehmen namens Xerox Parc. Allerdings kam dort (im Gegensatz zu Steve Jobs, der das Gelände besuchte) niemand auf die Idee, dass mit so etwas Geld verdient werden könnte.

Übrigens ist auch die Möglichkeit, ein Javaprogramm unverändert auf unterschiedlichen Systemen zu nutzen ein Kriterium der Objektorientierung. Dabei spricht man von **Portabilität**.

Insbesondere bei Entwicklern, die vorrangig in C oder C++ aber auch in anderen imperativen Sprachen entwickeln, herrscht bis heute das Vorurteil vor, Java sei eine viel zu langsame Sprache und deshalb überflüssig, ja generell sei **Objektorientierung** unsinnig.

Hier sollten Sie sich merken, dass es im Regelfall keine unsinnigen Sprachen gibt; **Sprachen werden entwickelt, um einen bestimmten Zweck zu erfüllen.** Ist dieser Zweck tatsächlich nützlich und ist die Sprache sinnvoll und für den Zweck effizient konzipiert, dann wird sie im Regelfall einige Jahrzehnte verwendet. Wer dann einer solchen Sprache die Sinnhaftigkeit abspricht zeigt damit lediglich, dass er den Zweck nicht versteht. Und natürlich kann Java nicht die Geschwindigkeit einer Sprache wie C++ erreichen: Java übernimmt die Arbeit, jedes Programm auf einer möglichst großen Anzahl von Rechnern und Smartphones laufen zu lassen. Das bedeutet einen teilweise höheren Aufwand und damit laufen diese Programme in Java langsamer als in C++, wenn es fähige C++-ProgrammiererInnen in C++ umsetzen. Andererseits müssen diese eben auch sehr fähig sein und umfangreiche Kenntnisse über die Unterschiede zwischen rund dreißig Betriebssystemen und Prozessoren kennen und sich kontinuierlich in neue Systeme einarbeiten. Da das kaum jemand leisten kann, der als Entwickler bezahlbar ist, werden die meisten Spiele nur für ein System entwickelt oder sie sind nicht gut auf die einzelnen Systeme angepasst.

Einige von Ihnen werden jetzt einwenden, dass es doch von **Steam** eine Plattform gibt, auf der Spiele unabhängig vom System laufen. Hier gilt

das gleiche, was schon bei Java gilt: So lange diese Spiele nicht individuell für jede Plattform entwickelt werden, kann auch nicht die volle Palette an Möglichkeiten genutzt werden, die das System bietet. Also werden einige Spiele auf dieser Plattform langsamer laufen als wenn Sie speziell an das System angepasst wären.

Die meisten Spieleentwickler nutzen heute allerdings keine Programmiersprache mehr, sondern sogenannten **Game Engines**. Das sind Softwarepakete, die bereits eine Vielzahl an **Bibliotheken** beinhalten, sodass die Mitglieder von Entwicklerstudios sich nur noch auf den Ablauf des Spiels konzentrieren müssen und ein Team von Designern für die Grafik und den Sound benötigen. Um mit einer Game Engine zu arbeiten brauchen Sie deshalb kein Informatikstudium mehr abschließen. Im Gegenteil: Da diese Softwarepakete genau das übernehmen, was fähige InformatikerInnen tun, gibt es in der Spielebranche nur wenige Stellen für vollwertige InformatikerInnen. Im Gegenteil: Als Absolvent z.B. von Media Systems ist die Nutzung einer Game Engine eigentlich ein Rückschritt: Das System übernimmt nicht nur vieles, was Sie sonst umsetzen müssten, es verhindert auch vieles, das Sie kennen und schätzen gelernt haben.

Dagegen müssen Sie anspruchsvolle Aufgaben als (Medien-)InformatikerIn erfüllen können, um eine Game Engine zu entwickeln oder Ihren Funktionsumfang zu erweitern. In Ihrem Studium können Sie das später ausprobieren: Sie werden eine Game Engine namens **Blender** kennen lernen. Diese wird von den meisten Studierenden abgelehnt, da die Nutzeroberfläche nicht wie die von vielen Game Engines oder Programmpaketen für Computergrafik aussieht. Tatsächlich ist Blender die wahrscheinlich beste Game Engine für (Medien-)InformatikerInnen: Da Sie hier alles und ohne Beschränkung erweitern oder verändern können und da Blender vollständig kostenlos und frei verfügbar ist, können Sie genau das tun, was Sie später als professionelle EntwicklerIn tun müssten. (Zum Vergleich: Wenn Sie keine akademische Lizenz erhalten, dann zahlen Sie für Engines wie Maya 3D mehrere tausend Euro. Doch selbst wenn Sie die Software erhalten, dürfen Sie daran nahezu nichts ändern.)

Kontrolle
Java ist eine imperative und klassenbasierte objektorientierte Programmiersprache, deren Programme leicht auf andere Systeme portiert werden können. Es unterstützt zusätzlich seit Version 8 die funktionale Programmierung und damit ein deklaratives Paradigma.

4.4 Verteilte Anwendungen

Die drei Sprachen, mit denen wir uns bislang beschäftigt haben setzen nicht voraus, dass unser Rechner sich in einem Netzwerk befindet, und dass es möglich ist, Daten mit den anderen Rechnern dieses Netzwerks auszutauschen. Nun wissen Sie aber, dass heute annähernd jedes computerbasierte System (also auch Smartphones) zumindest zeitweilig vernetzt ist. Wie Sie durch die einleitenden Kapitel wissen, wurden Rechner im Regelfall schon immer vernetzt und nur im Heimbereich hatten Nutzer einen Computer ohne Netzzugang. (Hieraus resultiert auch die veraltete Unterteilung in Heimcomputer und PCs.)

Wenn wir nun ein Programm entwickeln wollen, das vernetzte Rechner nutzen soll oder sogar nur bei vernetzten Rechnern einsetzbar sein soll, dann müssen wir die Strukturen, die sich daraus ergeben auch in unseren Programmen integrieren. Ein solches Programm wird übrigens als **verteilte Anwendung** bezeichnet, vor allem wenn es genau genommen aus mehreren individuell agierenden Programmen besteht. Wir müssen dann (siehe Alan Kay und die Objektorientierung) beachten, dass Daten zwischen Rechnern transportiert werden müssen. Und das bedeutet, dass wir eine Absicherung für die Fälle schaffen müssen, in denen Daten nicht das Ziel (also einen anderen Rechner) erreichen oder in denen das Ziel aus irgendwelchen Gründen nicht versteht, was es mit diesen Daten tun soll. Wir müssen insbesondere bei Verbindungen über das Internet auch beachten, dass die Datenübertragung einen Zeitversatz hat, und dass wir keine genaue Zeitabstimmung zwischen den Rechnern realisieren können. Die genauen Ursachen und möglichen Auswirkungen verstehen Sie, wenn Sie Veranstaltungen zum Thema **Netzwerk**e und **Nachrichtentechnik** belegen. Es folgen in Kürze einige Beispiele, um Ihnen einen ersten Eindruck zu vermitteln.

Aus der dafür nötigen Denkweise resultieren auch zwei Begriffe: Server und Client. Die naive Vorstellung lautet hier, dass ein Server ein Rechner im Netz ist, der eine bestimmte Funktionalität anbietet, und dass ein Client ein anderer Rechner im Netz ist, der vom Server eine solche Leistung anfordert. Das ist allerdings nicht richtig; ein **Server** ist lediglich ein Programm, das eine bestimmte Funktion anbietet, und ein **Client** ist ein Programm, das eine Funktion abruft. Sie können also auf einem Rechner verschiedene Server und Clients betreiben, wobei bei Betriebssystemen in aller Regel eine Vielzahl an Servern und Clients aktiv ist. (Hier gibt es noch andere Programmarten über die wir aber erst im Rahmen von Veranstaltungen wie „Betriebssysteme"reden werden.) Einsteiger, die aus der Apple- oder Microsoftwelt kommen sind häufig bei der Installation von Linux überrascht, dass Sie Mailserver und andere Server installieren können, aber Sie wissen

jetzt, warum das so ist.

Dennoch werden häufig einzelne Rechner als Server oder Client bezeichnet. Das ist insbesondere dann kein Problem, wenn ein solcher Rechner ausschließlich eine entsprechende Funktion im Netz übernimmt. Aber Sie wissen jetzt, dass Sie kein Netz benötigen, wenn Sie ein netzbasiertes Programm entwickeln wollen, weil Sie ja auf einem Rechner sowohl den Server als auch den Client betreiben können. Und ja: Sie können dann einen Datenaustausch zwischen Client und Server auf Ihrem Rechner praktisch genauso durchführen, als wenn beide auf unterschiedlichen Rechnern installiert und über ein Netz verbunden wären. Deshalb können Sie z.B. eine Webanwendung, die später im Internet nutzbar sein soll auf Ihrem Rechner entwickeln und sie dort auch testen, selbst wenn keine Internetverbindung vorhanden ist.

Machen Sie sich in solchen Fällen aber bewusst, dass Sie dann die zentrale Fehlerquelle bei verteilten Anwendungen ausblenden: Da Sie keine Daten über das Netz austauschen, wissen Sie nicht, ob die Anwendung am Ende auch tatsächlich so funktioniert, wie Sie sich das vorstellen: Die Datenübertragung kostet Zeit und diese Zeit ist deutlich höher, wenn die Daten über ein Netzwerk übertragen werden, als wenn Sie innerhalb eines Rechners übertragen werden.

4.4.1 Entwicklung von Webanwendungen – MySQL und PHP versus Ruby on Rails

Bevor wir an dieser Stelle weiter machen, hier ein wichtiger Hinweis: Bis vor wenigen Jahren entwickelten die meisten Softwareentwickler Anwendungen, die auf einem System liefen und die eine Netzwerk nur nutzten, um Nachrichten darüber auszutauschen. **Webanwendung**en haben wie alle **verteilten Anwendungen**, mindestens einen Server- und einen Clientteil, die tatsächlich auf getrennten Rechnern aktiv sind. Die einfachste Form von Webanwendungen kennen Sie wahrscheinlich unter dem Namen Internetseiten. Doch das sind nicht einfach nur Dokumente mit Bildern und Videos, die Sie sich auf Ihren Rechner bzw. Ihr Smartphone herunterladen können, sondern es sind immer öfter komplette Anwendungen. Allerdings werden diese Anwendungen zum Teil auf dem Server und zum Teil auf dem Client ausgeführt. Wenn Sie sich intensiver mit diesem Bereich beschäftigen, werden Sie Sprachen wie **PHP** kennen lernen, die ausschließlich als Server bzw. auf einem Webserver genutzt werden können. Allerdings ist dieser Ansatz veraltet: Aktuelle Sprachen wie **JavaScript** können sowohl als Server als auch als Client eingesetzt werden. Das ist allerdings für Einsteiger bzw. Erstsemester meist nur schwer umsetzbar, da Sie hier bewusst und gut begründet entscheiden müssen, auf welche Programm-

teile Nutzer Zugriff haben dürfen.

Wenn Sie also denken, die Programmierung in **HTML** (der am häufigsten eingesetzten Sprache für Webanwendungen) sei langweilig und man könnte damit nur einfache Internetseiten programmieren, dann liegen Sie falsch; das war in der Version 4.01 so, die Ende 1999 veröffentlicht wurde. Mit der Version 5, die im Herbst 2014 veröffentlicht wurde, ist dieses Thema endgültig passé: Basierend auf HTML 5 ergänzt um Sprachen wie PHP oder JavaScript können Sie Anwendungen entwickeln, die genau das gleiche leisten wie eine beliebige Anwendung, die Sie auf einem einzelnen Rechner nutzen können. Der Begriff Webanwendung ist im Grunde ein Synonym für den Begriff der verteilten Anwendung.

Zu Beginn dieses Kapitels haben Sie erfahren, dass im Grunde keine Sprachen existieren, die die zentralen Probleme angehen, die bei der Datenübertragung im Netz aufkommen. Der Grund besteht darin, dass für die meisten InformatikerInnen eben das System im Mittelpunkt steht, auf dem eine Software ausgeführt wird. Die Kommunikationswege dazwischen und der Zeitfaktor bei der Übertragung werden im Grunde immer nur als lästiges Übel angesehen oder gleich gänzlich ignoriert. Das gleiche gilt für die Nutzung von Webanwendungen durch Menschen.

Ein anschauliches Beispiel konnten Sie bei ebay in der Anfangszeit erleben: Damals hatten die Entwickler ignoriert, dass kurz vor Abschluss einer Auktion besonders viele Aufrufe und Gebote für ein Angebot erfolgten. Also konnten die Server gar nicht alle Angebote „sofort"verarbeiten. Dementsprechend wurde es zu einer Art Glücksspiel, ein Gebot kurz vor Versteigerungsschluss abzugeben: Unter Umständen wurde Ihr Gebot scheinbar ignoriert, weil es erst nach Auktionsende von den ebay-Servern verarbeitet werden konnte. Dieser Fehler im System wurde inzwischen soweit als möglich bereinigt. Dies ist außerdem ein Beispiel für die Bedeutung des Begriffs **Skalierbarkeit**.

Damit wieder zurück zu den eingangs genannten Sprachen: **Ruby on Rails** ist nicht die Sprache selbst, sondern ein Framework namens Rails, das Ruby so erweitert, dass Sie damit **Webanwendungen** entwickeln können.

Eine zweite Möglichkeit (und deutlich älter), um Webanwendungen zu entwickeln besteht in der Kombination aus drei Sprachen: **MySQL** ist eine Sprache, mit der Sie Datenbanken nutzen können und **PHP** ist eine imperative Sprache, mit der Sie die Funktionalität von Elementen einer Webanwendung programmieren können. Dazu kommt noch **HTML**, was eine **Markup Language** ist. Markup Languages sind Programmiersprachen mittels derer sich die Struktur von Anwendungen unabhängig von der Dar-

stellung und der Funktion programmieren lassen. HTML ist eine Markup Language mit der sich die Struktur einer Webanwendung und seit Version 5 die Bedeutung der Inhalte programmieren lässt.

Viele Softwareentwickler reden in Bezug auf Markup Languages vom sogenannten **Script**en. Für diesen Begriff gibt es keine präzise Definition. Wenn ProgrammiererInnen ihn benutzen, dann geht es in aller Regel um etwas, das zwar programmiert werden muss, damit eine bestimmte Aufgabe erfüllt wird, das aber vom jeweiligen Entwickler keinerlei logisches Denkvermögen erfordert. Sie müssen im Falle vom Scripten also nur verschiedene Befehle aneinander reihen oder ineinander verschachteln, ohne sich weiter Gedanken darüber zu machen, wie die miteinander interagieren: Sie tun es schlicht nicht. Teilweise wird auch bei Konfigurationsdateien vom scripten gesprochen, obwohl hier (im Gegensatz zu HTML4.01 oder LaTeX) sehr viel Grundlagenwissen nötig ist. Wenn Sie beispielsweise nicht genau wissen, was der Unterschied zwischen SSH und SSL ist, dann sollten Sie von der Konfiguration eines Servers die Finger lassen.

Haben Sie im ersten Semester eine Veranstaltung zum Webpage Development besucht, dann können Sie eine Webpage entwickeln, denn das ist gar nicht so schwer. Aber viele Konzepte und Abläufe werden Ihnen kaum klar werden. Wenn Sie allerdings die nötige Zeit investieren, dann werden Sie sich basierend auf dieser Veranstaltung die Grundlagen erarbeiten können, um eine vollwertige Webanwendung zu entwickeln.

Ein Tipp für den Fall, dass Ihnen jemand zu **Ruby on Rails** rät: Ja, es ist richtig, dass Rails ein großartiges Framework ist, mit dem Sie selbst komplexe Anwendungen für multinationale Konzerne entwickeln können. Aber Sie merken es schon an der Formulierung: Es ist für Einsteiger schlicht zu komplex und die Vielzahl an Optionen, mit denen Sie von Beginn an konfrontiert werden, lenkt Sie von den Punkten ab, die Sie als Einsteiger verinnerlicht haben müssen. Hier würde ich eher empfehlen, dass Sie sich in **HTML5** und **JavaScript** einarbeiten. Leider sind aber die meisten Anleitungen im Netz (selbst wenn dort die Rede von HTML 5 ist) immer noch Einführungen in HTML 4, bei denen praktisch alles ignoriert wird, was an Version 5 so großartig ist. Teilweise werden hier sogar Techniken vermittelt, die bereits in der Version 4 als schlampig galten. Der Grund ist recht simpel: Wie so oft erklären dort Menschen die Programmierung, die zwar die alte Version beherrschen, aber die schlicht zu faul oder dumm sind, um zu erkennen, dass Version 5 keine kleine Erweiterung um ein paar nette Effekte ist, sondern eine vollständige Überarbeitung, bei der Aspekte berücksichtigt wurden, die nirgends in Version 4 auftauchen und auch nichts mit dem zu tun haben, was in Version 4 vorhanden ist.

Sie fragen, was eine Datenbank ist? Wie so oft, wenn **Informatik**erInnen es mit gleichartigen Daten oder Abläufen zu tun haben, entwickeln Sie entsprechende Strukturen, die letztlich dazu dienen, Fehler zu reduzieren und Abläufe effizienter zu gestalten. **Datenbank**en sind eine weitere Lösung, die so entstanden ist: Wann immer es um große Mengen gleichartiger oder gleichartig strukturierter Daten geht, die nach verschiedenen Kriterien untersucht oder geändert werden müssen, wird eine Datenbank verwendet. **Relationale Datenbanken** bestehen dabei aus Tabellen, bei denen jede Spalte einem Kriterium entspricht und jede Zeile einem sogenannten **Datensatz**. Beispielsweise würde bei einer relationalen Kundendatenbank jede Zeile einem Kunden entsprechen und Einträge in den Spalten wären nach Aspekten wie Name, Vorname, Anschrift, usw. unterteilt.

Bei Webanwendungen dienen Datenbanken (wie bei allen Anwendungen) verschiedenen Zwecken. Zum einen wäre da die klassische Kundendatenbank. Dann gibt es Datenbanken, in denen Einträge auf den Webanwendungen verwaltet werden. Auch die Speicherung jedes Klicks und jeder Taste, die NutzerInnen gedrückt haben wird mit einer Datenbank realisiert. Aber es gibt noch wesentlich mehr Einsatzmöglichkeiten. Wie oben genannt geht es schlicht darum, große Mengen gleichartiger Daten in einer strukturierten Form aufzubewahren, um möglichst schnell darauf zuzugreifen.

Kontrolle
Beginnen Sie beim Webapplication Development mit einer Einführung in MySQL und PHP sowie HTML5. Wenn Sie die objektorientierte und funktionale Programmierung beherrschen, dann können Sie auch JavaScript anstelle von PHP verwenden. Allerdings ist hier ein häufiger Fehler, dass dann HTML nur noch dazu genutzt wird, die JavaScript-Anwendung zu starten, sodass sie in einem beliebigen Browser genutzt werden kann. Ruby on Rails ist ein sehr mächtiges Werkzeug aber für Einsteiger nur beschränkt empfehlenswert. Dazu kommt, dass Rails häufig in Kombination mit weiteren Frameworks verwendet wird, was den Einstieg zusätzlich erschwert. Außerdem ist es leider noch immer nicht so effizient wie JavaScript.

4.5 Konzepte bei der Programmierung

Wie schon mehrfach angeführt gab und gibt es zu jeder Zeit eine Vielzahl von Programmiersprachen, die jeweils für bestimmte Zwecke ausgelegt sind. Zum Teil sind die Unterschiede nur in wenigen Details begründet. Um Ihnen einen kleinen Überblick darüber zu verschaffen, was es noch für Sprachen gibt und wofür diese nützlich sind, folgt eine kleine und dementsprechend unvollständige Aufstellung. Zum Teil werden hier weitere Be-

griffe eingeführt, die für die Programmierung insgesamt wichtig sind.

4.5.1 Dynamisch versus statisch – Ruby und Python versus C und Java

Ruby und **Python** sind Programmiersprachen, die in Konkurrenz zu Java stehen. Der auffälligste Unterschied besteht darin, dass Ruby **schwach typisiert** ist. (Alternativ spricht man auch von **dynamischer Typisierung**.) Im Gegensatz dazu sind **C** und **Java stark bzw. statisch typisiert**. Beide Varianten haben Vor- und Nachteile. Und leider neigen die meisten Entwickler dazu, die Variante als schlecht zu bezeichnen, die sie als zweites kennen lernen. Wer das tut hat aber leider nichts mit professionellen **Informatiker**Innen zu tun, selbst wenn er/sie in einer Sprache bzw. einem **Paradigma** wirklich gut ist.

4.5.2 Typisierung von Daten

Bislang haben wir lediglich über Codierung gesprochen aber nicht über Typisierung. Wie Sie bereits wissen werden alle möglichen Daten, die sie vom Computer verarbeiten lassen in einer anderen Form gespeichert als die, in der sie angezeigt werden. Wenn wir nun von statischer oder starker Typisierung sprechen, dann bedeutet das, dass Sie bei einem Wert, den Sie programmieren so etwas Ähnliches wie eine Codierung festlegen. Der Typ eines Wertes, den Sie so vergeben wird entsprechend als **Datentyp** bezeichnet.

Einer der ersten Fälle, in denen Sie mit Typisierung zu tun bekommen ist das Rechnen mit ganzzahligen und ganzrationalen Zahlen. Diese werden nämlich vom Rechner unterschiedlich gespeichert: Fließkommazahlen werden nicht in der Form gespeichert, die Sie aus dem Mathematikunterricht kennen, aber eine Einführung in diese Materie überlasse ich den Kollegen der **Technischen Informatik**. Jetzt aber ein Beispiel für die möglichen Varianten, wie eine Programmiersprache mit ganzen Zahlen umgehen kann:

Wenn Sie die Zahl 5 programmieren, als Typ der Zahl ganzzahlig (**Integer**) festlegen und anschließend durch 2 teilen (oder jede andere Zahl ungleich +/- 5 oder +/- 1), dann ergibt sich bekanntlich eine ganzrationale Zahl. Je nach Programmiersprache gibt es nun unterschiedliche Möglichkeiten, was dabei passiert:

1. Bei statisch typisierten Sprachen erfolgt in aller Regel eine Fehlermeldung, denn Sie haben definiert, dass Ihre Zahl ganzzahlig ist. Also muss das Ergebnis ebenfalls ganzzahlig sein. Es gibt dennoch

Möglichkeiten, um eine solche Aufgabe in einer solchen Sprache lösen zu lassen. Dazu sind jedoch zusätzliche Programmzeilen nötig.

2. Wenn keine Fehlermeldung erfolgt, ist das das Ergebnis häufig nicht das, was Sie erwarten. In den Fällen, wo die Sprache vorsieht, dass das Ergebnis als ganzzahliger Wert gespeichert wird, wird nun entweder auf- oder abgerundet.

   ABER! Ob auf- oder abgerundet wird, das hat nichts mit den Rundungsregeln zu tun, die Sie aus der Schule kennen: Es gibt also vier Möglichkeiten, wie eine Programmiersprache damit umgeht, wenn Sie eine Division von zwei ganzzahligen Werten einprogrammieren und bei der keine Ganzzahl berechnet wird.

   (a) In der Sprache wird stets abgerundet:
   5 : 2 = **2**.
   -5 : 2 = **-3**,
   denn -2,2 abgerundet ergibt -3 und nicht -2!

   (b) In der Sprache wird stets aufgerundet:
   5 : 2 = **3**
   -5 : 2 = **-2**,
   denn -2,2 aufgerundet ergibt -2 und nicht -3!

   (c) In der Sprache sind die Rundungsregeln enthalten, die Sie aus dem Mathematikunterricht der Schule kennen. Das ist der seltenste Fall.

   (d) Die Sprache gibt in solchen Fällen eine Fehlermeldung oder eine **Exception** aus. Als EntwicklerIn müssen Sie dann das Programm entsprechend korrigieren.
   Wichtig:
   Eine Exception ist KEINE Fehlermeldung. Es ist vielmehr ein Komfortfaktor einzelner Programmiersprachen, der sie darauf hinweist, dass Ihr Programm in bestimmten Fehlern nicht so ablaufen wird, wie Sie das wahrscheinlich erwarten. Je nach Komfort der jeweiligen Sprache gibt es auch Exceptions, die auf Situationen hinweisen, die durchaus wie gewünscht verlaufen können, wo Ihnen die Programmiersprache also quasi den Tipp gibt, zu prüfen, ob Sie dieses Verhalten so haben wollen oder ob das nicht doch ein logischer Fehler ist. In Java haben Sie sogar die Möglichkeit, eigene Exceptions zu programmieren, um bestimmte Ausnahmen ganz bewusst anders verarbeiten zu lassen als das sonst der Falle wäre.

3. Bei **dynamisch typisierten Sprachen** wird der Datentyp je nach Bedarf automatisch von der Programmiersprache angepasst. Hier pro-

grammieren wir also in aller Regel nicht den Datentyp. Generell steht der Begriff des **Typecasting** für eine solche Anpassung, die in einigen statisch typisierten Sprachen möglich ist.

Typecasting gibt es noch für andere Datentypen, es ist also nicht nur auf die Umwandelung des Datentyps bei einer Zahl beschränkt, sondern bei allen denkbaren virtuellen Objekten.

Aber auch beim Typecasting ist das Ergebnis nicht automatisch das, was Sie denken. Das hat wiederum mit der Speicherung von Daten zu tun. Wie Sie wissen basiert die Speicherung von Daten in einem Computer auf Zahlen der Basis 2. Und damit basiert die Speicherung von Nachkommastellen auf Potenzen von $\frac{1}{2}$. In Informatikveranstaltungen werden Sie dazu einige Beispiele rechnen, hier seien nur zwei genannt: $\frac{5}{2} = 2 + 1/2$. Binär lässt sich das in der Form $[10, 1]_2$ darstellen: $(1 \cdot 2) + (0 \cdot 1) + (1 \cdot \frac{1}{2})$. Versuchen wir einmal, die Zahl 0,3 als Binärzahl darzustellen:

$$
\begin{aligned}
&(0 \cdot 1) + (0 \cdot \tfrac{1}{2}) + 0,3 \\
&= (0 \cdot 1) + (0 \cdot \tfrac{1}{2}) + (1 \cdot \tfrac{1}{4}) + 0,05 \\
&= (0 \cdot 1) + (0 \cdot \tfrac{1}{2}) + (1 \cdot \tfrac{1}{4}) + (0 \cdot \tfrac{1}{8}) + 0,05 \\
&= (0 \cdot 1) + (0 \cdot \tfrac{1}{2}) + (1 \cdot \tfrac{1}{4}) + (0 \cdot \tfrac{1}{8}) + (0 \cdot \tfrac{1}{16}) + 0,05 \\
&= (0 \cdot 1) + (0 \cdot \tfrac{1}{2}) + (1 \cdot \tfrac{1}{4}) + (0 \cdot \tfrac{1}{8}) + (0 \cdot \tfrac{1}{16}) + (1 \cdot \tfrac{1}{32}) + 0,01875 \\
&= ...
\end{aligned}
$$

Aber es gibt kein endgültiges Ergebnis. Dementsprechend kann weder der Computer noch die Programmiersprache eine Zahl wie 0,3 richtig darstellen. Er könnte sie als $3 * 10^{-1}$ darstellen, aber da das nur eine begrenzte Anzahl an Spezialfällen aller möglichen ganzrationalen Zahlen löst, ist das keine Lösung, die wir immer nutzen können.) Also wird eine Zahl wie 0,3 nur annäherungsweise gespeichert. Und wenn sie dann ausgegeben wird, kann so etwas wie 0,3000010002 oder 0,298991 herauskommen. Auch hierfür gibt es Lösungen, die aber auch wieder bedeuten, dass Sie zusätzliche Zeilen programmieren müssen.

Wenn wir mit der Programmierung in **C** beginnen, werden Sie häufig mit solchen Fällen zu tun haben. Sie werden dann (wie auch sonst grundsätzlich bei der Programmierung) Lösungen entwickeln müssen, damit der Rechner die Zahlen verwendet und speichert, die für Ihre Aufgabenstellung eine richtige Lösung darstellen.

Dies ist allerdings kein Beispiel dafür, was **Informatik**erInnen von ProgrammiererInnen unterscheidet: Gute ProgrammiererInnen wissen, dass es solche Probleme gibt, sie kennen die Ursachen und sie entwickeln Programme so, dass diese Probleme in allen Varianten gelöst werden. (Sonst sind es Dilletanten, was leider auf viele Quereinsteiger zutrifft.) Allerdings lernen InformatikerInnen in Ihrem Studium verschiedene systematische Methoden, um solche Probleme effizient anzugehen. Der entsprechende Bereich heißt **Praktische Informatik**, wobei die entsprechenden Veranstaltungen in aller Regel unter Titeln wie **Algorithmen und Datenstrukturen**, **Algorithmendesign** und **Algorithmik** angeboten werden.

Aber zurück zu Ruby: Wie gesagt handelt es sich hier um eine dynamisch typisierte Programmiersprache, während C, C++ und Java statisch typisiert sind. Es spielt keine Rolle, welche der beiden Varianten Ihnen lieber ist, das einzige, was eine Rolle spielt ist, dass Sie langfristig lernen, beide Formen der Typisierung zu beherrschen.

Eine weitere Programmiersprache, die neben Ruby in den letzten Jahren immer bekannter wurde und dynamische Typisierung bietet, ist **Python**.

Kontrolle
Es gibt dynamisch und statisch typisierte Sprachen. Der Unterschied besteht darin, dass bei den dynamisch typisierten Sprachen die Programmiersprache Methoden hat, um den Datentyp eines Wertes automatisch anzupassen. Das ist komfortabel, aber es ist nicht intuitiv. Denn während Sie bei einer statisch typisierten Sprache selbst programmieren müssen, wie ein Typcasting ausgeführt wird, müssen Sie bei jeder dynamisch typisierten Sprache genau wissen, wie diese einzelne Sprache das „automatische"Typecasting durchführt. Wenn Sie das eine oder das andere nicht beherrschen, dann programmieren Sie den Computer nicht, um das zu tun, was er tun soll, sondern Sie programmieren Ergebnisse, die schlichtweg falsch sind. (Und glauben Sie mir, das wollen Sie nicht bei der Steueranlage eines Flugzeugs...)

4.5.3 Funktionen – Dynamisch versus statisch

Dieser Abschnitt dürfte auch für die meisten fortgeschrittenen unter Ihnen eine Überraschung beinhalten: Nicht nur Datentypen, sondern auch Funktionen können bei einzelnen Sprachen dynamisch während der Laufzeit eines Programms geändert werden.

Wichtig:
Bitte denken Sie jedoch nicht, dass das Programmieren einer Funktion eine Form der **funktionalen Programmierung** ist: Genau wie bei der **impera-**

tiven Programmierung nutzen Sie dort etwas, das als Funktion bezeichnet wird, um Abläufe aus dem Programm auslagern. Diese Art der Funktionsdefinition sorgt dafür, dass Sie den Inhalt der Funktion an beliebigen Stellen Ihres Programms verwenden können. Bei der funktionalen Programmierung ist eine Funktion dagegen eine Umsetzung des sogenannten **Lambda-Kalkül**s, das wir uns erst bei der Einführung in die **deklarative Programmierung** ansehen werden.

Doch für die Einsteiger zunächst die Erklärung, was eine Funktion ist: Eine **Funktion** fasst in der Programmierung mehrere Programmzeilen zusammen und lässt sich über einen Bezeichner von beliebigen Stellen eines Programms aus aufgerufen werden.

Die meisten Programmierer kennen Funktionen dagegen nur als ein Mittel der Programmierung, das sich nicht ändern kann, während das Programm läuft. Das ist aber ein Irrtum, der darauf basiert, dass das bei Programmiersprachen wie C, C++ oder Java so ist.

Tatsächlich werden Funktionen während des Programmablaufs wie alle anderen Daten eines Programms im Speicher des Rechners abgelegt und bei Bedarf von dort geladen. Und weil der Speicher eines Rechners zu beliebigen Zeiten geändert werden kann, ist es natürlich auch grundsätzlich möglich beliebige Teile eines Programms abzuändern, das dort abgelegt wurde.

Kontrolle
Auch wenn die meisten bekannten Sprachen das nicht können, ist es grundsätzlich möglich, dass auch Funktionen eines Programms dynamisch programmiert werden.

4.6 First-class Objects

Die meisten Programmierneulinge lernen das Programmieren mit Variablen kennen und entwickeln dabei die Vorstellung, dass eine Variable nur einen Wert oder eine Menge an Werten (z.B. die sogenannten Arrays) sein kann. Tatsächlich kann aber auch eine Funktion der Wert einer Variablen sein. Ist das bei einer Programmiersprache der Fall, dann können Sie nicht nur eine Funktion mit einem Wert aufrufen, sondern Sie können eine Funktion quasi wie einen beliebigen Wert an eine andere Funktion übergeben. Das bedeutet, dass Sie dann die Möglichkeit haben, den Ablauf einer Funktion während eines Programmablaufs dynamisch anpassen können.

Diejenigen von Ihnen, die bereits imperativ programmiert haben werden

jetzt behaupten, dass das doch klar sei, weil so „schon immer"der Wert einer Funktion an eine Variable übergeben wurde. Damit zeigen Sie, dass Sie den Absatz missverstanden haben: Dort steht, dass auch eine Funktion als ganzes und eben nicht nur der Wert, den sie berechnet in einer Variablen gespeichert werden kann. Warum das so ist werden wir uns ansehen, wenn wir klären, was genau eine Variable eigentlich ist.

Als Sammelbegriff für alles, was einer Funktion übergeben werden kann wird der Begriff des **first-class Object** verwendet. Wenn also die Rede davon ist, dass in einer Programmiersprache Funktionen first-class Objects sind, dann bedeutet das nichts anderes, als dass Sie in dieser Sprache eine Funktion genauso als Objekt an eine andere Funktion übergeben können, wie Sie das mit einer Variablen gewohnt sind.

Kontrolle
Im Gegensatz zur meist anzutreffenden Überzeugung von ProgrammiererInnen spricht eigentlich nichts dagegen, auch Funktionen als Argumente an Funktionen zu übergeben. Und wenn eine Programmiersprache das unterstützt, dann reden wir davon, dass in dieser Sprache Funktionen first-class objects sind.

4.7 Zusammenfassung

In diesem Kapitel haben Sie einen Überblick erhalten, wie die drei Sprachen C, C++ und Java zusammen hängen und wo die Unterschiede liegen. Sie haben eine Vielzahl an Begriffen kennen gelernt, die bei der Programmierung von Hochsprachen von Belang sind.

Sie haben verstanden, dass es keine beste Sprache oder sinnlose Sprachen gibt, sondern dass Sprachen jeweils für eine bestimmte Problemstellung entwickelt wurden. Deshalb gibt es dann auch ganz unterschiedliche Arten (Paradigmen) des Programmierens und hier haben Sie konkrete Fälle kennen gelernt, um den Begriff des Paradigmas mit Leben zu füllen.

Sie wissen jetzt, dass Sie es gelegentlich mit einem bestimmten Programmierparadigma zu tun haben und teilweise lediglich mit einem Spezialfall, der so nur in einer einzelnen oder bei einigen wenigen Sprachen umgesetzt wird. Diese Kenntnisse sind ein weiterer Unterschied zwischen einem dilletantischen Quereinsteiger und einem ernstzunehmenden Softwareentwickler.

Danach haben Sie etwas über Konzepte erfahren, die C-, C++- und JavaProgrammiererInnen nicht verstehen und die beispielsweise in Ruby, Py-

thon und JavaScript zum Einsatz kommen.

Stichwortverzeichnis