BZT Bildungszentrum für Technik Frauenfeld

HERDT Campus

Webcode - Übungsdateien

B8AC-36C0-CD34

BZT Bildungszentrum für Technik Frauenfeld

Programmierung Grundlagen

– mit Beispielen in Java

(Stand 2017)

PG

Plus

Ralph Steyer

4. Ausgabe, Dezember 2017

ISBN: 978-3-86249-770-6

HERDT

Schulversion

1 Informationen zu diesem Buch

In diesem Kapitel erfahren Sie

- an wen sich dieses Buch richtet
- welche Vorkenntnisse Sie mitbringen sollten
- welche Hard- und Software Sie für die Arbeit mit diesem Buch benötigen
- wie dieses Buch aufgebaut ist
- welche Konventionen verwendet werden

1.1 Voraussetzungen und Ziele

Zielgruppe

- Programmiereinsteiger
- Auszubildende in IT-Berufen

Empfohlene Vorkenntnisse

Folgende Kenntnisse werden für eine erfolgreiche Benutzung dieses Buchs vorausgesetzt:

- Grundkenntnisse im Umgang mit Windows
- Grundkenntnisse in der Bedienung von Anwendungsprogrammen

Lernziele

Zu Beginn erhalten Sie einen Überblick über die bekanntesten Programmiersprachen und lernen Methoden und Werkzeuge für die Entwicklung von Programmen kennen.

Im weiteren Verlauf werden grundlegende Elemente von Programmiersprachen und deren Verwendung vorgestellt. Anschließend arbeiten Sie mit einigen elementaren Algorithmen.

Im letzten Kapitel erhalten Sie einen Überblick über die Grundlagen zur Softwareentwicklung, wie Sie für umfangreiche Software-Projekte benötigt werden, und lernen die verschiedenen Methoden des Softwareentwurfs kennen.

Hinweise zur Software

Bei den Beispielen im Buch wird die Programmiersprache Java verwendet. Etliche Beispiele sind zusätzlich in JavaScript codiert.

- Im Buch wird von einer Erstinstallation der Software Java Platform, Standard Edition (Java SE), in der Version 9 ausgegangen, die Sie über das Internet downloaden können. Die Software wurde als Entwicklungsumgebung JDK (**J**ava **D**evelopment **K**it) heruntergeladen. Hauptbestandteile des JDK sind ein Java-Compiler zum Übersetzen der Programme, ein Java-Interpreter zum Ausführen der Programme, Bibliotheken mit fertigen Programmteilen und Programme zur Erstellung von Dokumentationen.

- Als Editor wird der Freeware-Editor PSPad verwendet. Hinweise zur Installation und Konfiguration finden Sie im Anhang.
- Als Beispiel für eine komfortable Entwicklungsumgebung wird Eclipse vorgestellt.
- Für das Testen von Java-Code-Passagen bietet sich der Hamster-Simulator an, der im Buch vorgestellt wird.
- Bei einigen Ausblicken werden bei Bedarf verschiedene weitere Programme und Tools verwendet.
- Das verwendete Betriebssystem ist Windows 10.
- Als Browser (für JavaScript) kommen Firefox 57 und der Internet Explorer 11 zum Einsatz. Die Beispiele funktionieren aber auch in älteren oder anderen Browsern, sofern sie JavaScript unterstützen.

Die Beispiele lassen sich auch anhand einer früheren Java-Version umsetzen.

1.2 Aufbau und Konventionen

Aufbau und inhaltliche Konventionen des Buchs

- Am Anfang jedes Kapitels finden Sie die Lernziele und am Ende einiger Kapitel eine Schnellübersicht mit den wichtigsten Funktionen im Überblick.
- Die meisten Kapitel enthalten Übungen, mit deren Hilfe Sie die erlernten Kapitelinhalte einüben können.
- Die Notizseiten im Buch geben Ihnen die Möglichkeit, eigene Anmerkungen und Erkenntnisse sowie praktische Arbeitstechniken einzutragen.

Hervorhebungen im Text

Im Text erkennen Sie bestimmte Programmelemente an der Formatierung. So werden z. B. Bezeichnungen für Programmelemente wie Register immer *kursiv* geschrieben und wichtige Begriffe **fett** hervorgehoben.

Kursivschrift	kennzeichnet alle von Programmen vorgegebenen Bezeichnungen für Schaltflächen, Dialogfenster, Symbolleisten, Menüs bzw. Menüpunkte (z. B. *Datei - Schließen*) sowie alle vom Anwender zugewiesenen Namen wie Dateinamen, Ordnernamen, eigene Symbolleisten, Hyperlinks und Pfadnamen.
`Courier New`	kennzeichnet Programmtext, Programmnamen, Funktionsnamen, Variablennamen, Datentypen, Operatoren etc.
`Courier New kursiv`	kennzeichnet Zeichenfolgen, die vom Anwendungsprogramm ausgegeben oder in das Programm eingegeben werden.
[ ]	Bei Darstellungen der Syntax einer Programmiersprache kennzeichnen eckige Klammern optionale Angaben.
|	Bei Darstellungen der Syntax einer Programmiersprache werden alternative Elemente durch einen senkrechten Strich voneinander getrennt.

Was bedeuten die Symbole im Buch?

 Hilfreiche Zusatzinformation

 Praxistipp

 Warnhinweis

1.3 HERDT BuchPlus – unser Konzept

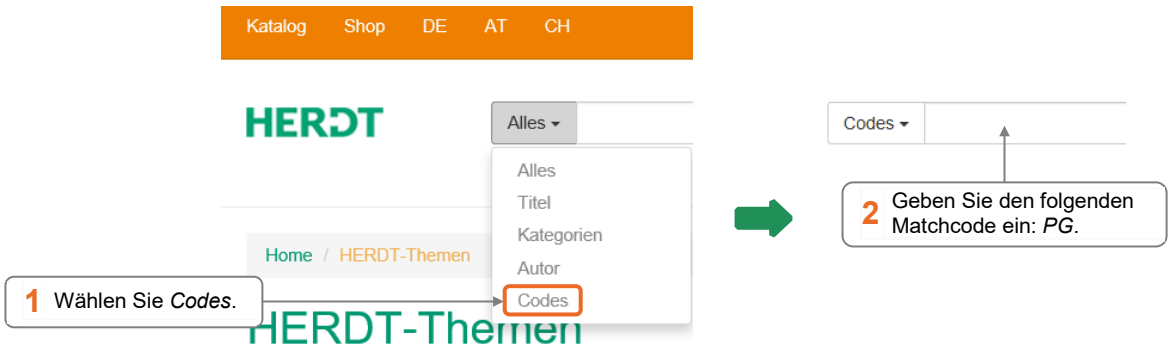
Problemlos einsteigen – Effizient lernen – Zielgerichtet nachschlagen

(weitere Infos unter *www.herdt.com/BuchPlus*)

Nutzen Sie dabei unsere maßgeschneiderten, im Internet frei verfügbaren Medien:

So können Sie schnell auf die BuchPlus-Medien zugreifen:

- Rufen Sie im Browser die Internetadresse *www.herdt.com* auf.



2 Grundlagen zu Programmen und Programmiersprachen

In diesem Kapitel erfahren Sie

- was Programme sind
- wie sich die Programmiersprachen geschichtlich entwickelt haben
- nach welchen Kriterien Programmiersprachen eingeteilt werden

2.1 Grundlagen zu Programmen

Computer unterstützen Sie in vielen Bereichen des täglichen Lebens, z. B. bei der Informationsbeschaffung, bei Bankgeschäften, beim Einkaufen (Onlineshopping) oder im Unterhaltungsbereich.

Der Lösungsweg zur Bearbeitung einer Aufgabe wird dem Computer in Form eines Programms oder Skripts mitgeteilt.

Im Folgenden wird **Programm** als Synonym für Programm und Skript verwendet, wenn nicht ausdrücklich von einem Skript die Rede ist. Ein Programm wird auch **Applikation (engl. application)** oder **Anwendung** genannt. Die Kurzform der englischen Version von Applikation – **App** – ist vor allem bei der Programmierung für mobile Geräte gebräuchlich.

Vom Algorithmus zum Programm

Ein **Programm** bzw. Skript ist eine Beschreibung der Lösung einer vorgegebenen Aufgabe in einer spezifischen Programmiersprache (vgl. Abschnitt 2.3).

Die Lösung kann (und wird in der Regel) aus einzelnen Bearbeitungsvorschriften bestehen. Eine solche Bearbeitungsvorschrift wird **Algorithmus** genannt. Ein Algorithmus muss bei jeder möglichen Eingabe von Daten die Verarbeitung nach endlich vielen Schritten beenden und einen eindeutigen Ablauf besitzen. Ein Programm besteht aus Algorithmen, deren Arbeitsschritte in einer Programmiersprache (z. B. in Java, C#, JavaScript oder PHP) formuliert sind.

Was ist Software?

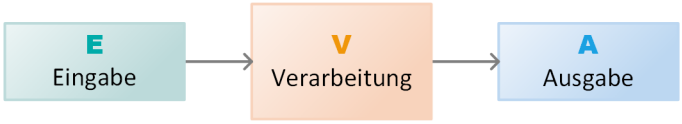
Der Begriff **Software** wird meist umfassender als der Begriff Programm verstanden. Als Software werden alle immateriellen Teile eines Computers verstanden. Das umfasst Programme, aber auch die zugehörigen Daten. Im täglichen Sprachgebrauch werden die Begriffe Software und Programm oft synonym verwendet.

Programme verarbeiten Daten

Ein Programm kann Ihnen beispielsweise Steuern berechnen. Sie teilen dem Programm die Daten mit, die für die Berechnung benötigt werden. Das Programm rechnet nach seinem Algorithmus mit Ihren Daten und liefert am Ende ein Ergebnis.

Programme verarbeiten **Daten**, z. B. Benutzereingaben, und liefern Daten zurück. Diesen Datenfluss durch ein Programm nennt man **EVA-Prinzip** (**E**ingabe – **V**erarbeitung – **A**usgabe).

- **Eingabe**: In das Programm werden Daten eingegeben, z. B. über eine Tastatur oder eine Datenbank.
- **Verarbeitung**: Das Programm verarbeitet diese Daten nach einem vorgegebenen Algorithmus.
- **Ausgabe**: Die Ergebnisse des Programms werden ausgegeben, z. B. auf einen Bildschirm, einen Drucker oder in eine Datenbank.



2.2 Warum programmieren?

Bestehende Programme decken viele Einsatzgebiete ab, wie z. B. die Verwaltung von Geschäftsprozessen, die Textverarbeitung, die Tabellenkalkulation, Spiele usw. Warum sollten Sie noch selbst programmieren?

Standard-Software anpassen

Standardisierte Software, die für ein bestimmtes Einsatzgebiet konzipiert wurde, stößt an ihre Grenzen, wenn spezielle Anforderungen an sie gestellt werden bzw. wenn die Anwendungsgebiete erweitert werden.

Beispiel

Sie haben sich mit einem Programm eine Datenbank mit Ihren Kundendaten aufgebaut. Später müssen Sie auch mit anderen Programmen auf diese Datenbank zugreifen. In diesen Programmen ist keine Möglichkeit vorgesehen, auf die Daten Ihrer Kundendatenbank zuzugreifen. Sie müssten also die Daten wieder manuell eingeben. Oft ist es dann einfacher, wenn Sie ein Programm schreiben, das den Zugriff auf die Daten ermöglicht.

Individual-Software

Hierbei handelt es sich um Software, die eigens für einen bestimmten Anwendungsbereich oder für spezielle Abteilungen innerhalb einer Firma erstellt wird.

Für verschiedene Branchen und Einsatzgebiete gibt es spezielle Anforderungen, sodass nur eine genau auf die Bedürfnisse abgestimmte Software infrage kommt. Individual-Software kann, auf Kundenwunsch hin, erweitert und verändert werden.

2.3 Klassifizierung von Programmiersprachen

Im Gegensatz zu natürlichen Sprachen, z. B. Deutsch, Englisch, gehören **Programmiersprachen** zu den **formalen Sprachen** (künstlichen Sprachen).

Programmiersprachen lassen sich nach verschiedenen Kriterien einordnen. Nach ihrer **historischen Entwicklung** werden verschiedene Generationen unterschieden:

- Erste Generation: Maschinensprachen (Maschinencode)
- Zweite Generation: Assembler-Sprachen

- Dritte Generation: Prozedurale Sprachen
- Vierte Generation: 4GL (**G**eneration **L**anguage)
- Fünfte Generation: Künstliche Intelligenz

Programmiersprachen unterscheiden sich erheblich in der zugrunde liegenden **Programmiertechnik**, auch Programmierparadigma genannt. Nach Programmiertechniken und Konzepten können die Programmiersprachen wie folgt klassifiziert werden, wobei diese Einteilung weder strikt zu trennen noch zwingend ist:

- Prozedurale Programmiersprachen
- Objektorientierte Programmiersprachen
- Hybride Programmiersprachen
- Skriptsprachen
- Funktionale Programmiersprachen
- Logische Programmiersprachen
- Erziehungsorientierte Programmiersprachen und Minisprachen

Sprachen lassen sich also nach verschiedenen Kriterien klassifizieren und zum Teil ist die Zuordnung zu einer bestimmten Gruppe nicht eindeutig.

Funktionale und logische Programmiersprachen werden auch als **deklarative Programmiersprachen** bezeichnet.

2.4 Die Klassifizierung nach Generationen

Erste Generation: Maschinensprachen (Maschinencode)

Damit ein Computer Probleme lösen kann, muss ihm der Lösungsweg in einer ihm verständlichen Art und Weise mitgeteilt werden. Eine Maschinensprache erfüllt genau diese Bedingung.

Beschreibung	Sowohl die Operationen als auch die Daten werden vom Programmierer ausschließlich als Bitfolge aus Nullen und Einsen (vgl. Abschnitt 6.2) eingegeben. Eine übliche Operation ist z. B. der Transport von Daten aus dem Speicher in ein Register.
Nachteile	Für jeden Computer müssen die Maschinenbefehle neu entwickelt werden, da diese Sprache von den Eigenschaften der Hardware (Prozessoren) abhängig ist. Programme in Maschinensprache sind schwer lesbar und mit einem hohen Programmieraufwand verbunden. Deshalb wird diese Sprache heute kaum mehr direkt eingegeben (allerdings werden auch heute noch alle Befehle für Computer letztendlich in Maschinenbefehle umgewandelt).
Beispiel	`00011010 0011 0100`

Zweite Generation: Assembler-Sprachen

Beschreibung	Assembler-Sprachen sind wie Maschinensprachen an bestimmte Prozessoren gebunden. Übersetzungsprogramme, die Assembler-Programme in Maschinencode umwandeln, werden ebenfalls als **Assembler** bezeichnet.
Einsatz	Assembler-Sprachen werden überwiegend zur Programmierung der Hardware oder für schnelle, zeitkritische Programme eingesetzt.

Vorteile	Im Vergleich zur Maschinensprache bieten Assembler-Sprachen dem Programmierer durch Operationskürzel, z. B. `ADD` für addieren, wesentliche Erleichterungen. Da Assembler-Sprachen auf die maschinenspezifischen Besonderheiten des jeweiligen Computers abgestimmt sind, verbrauchen die Programme im Allgemeinen weniger Speicherplatz und sind meist auch schneller als ein entsprechendes Programm in einer anderen Programmiersprache.
Besonderheit	Die einzelnen Befehle der Assembler-Sprachen verwenden direkt die internen Befehle des Prozessors. Wer eine Assembler-Sprache erlernt, erfährt dabei viel über die Arbeitsweise des jeweiligen Prozessortyps.
Beispiel	ADD ax, 10

Dritte Generation: Prozedurale Sprachen

Anstoß zur Weiterentwicklung der Programmiersprachen gaben die mangelhafte Eignung der maschinenorientierten Sprachen zum Erstellen komplexer Anwendungsprogramme und die schlechte Lesbarkeit für Menschen.

Beschreibung	Prozedurale Sprachen sind (weitgehend) unabhängig von einem Computersystem. Da Programmiersprachen ab der dritten Generation vom spezifischen Computersystem abstrahieren, werden sie auch als **höhere Programmiersprachen** bezeichnet. Damit ein Computer ein Programm in einer höheren Programmiersprache versteht, muss das Übersetzungsprogramm (Compiler oder Interpreter) an das jeweilige System angepasst sein und den entsprechenden Maschinencode erzeugen.
Einsatz	Die Sprachen der dritten Generation sind in ihrer Struktur und ihrem Befehlsvorrat auf bestimmte Anwendungsbereiche zugeschnitten.
Vorteile	Höhere Programmiersprachen sind im Allgemeinen leichter zu erlernen als maschinenorientierte Sprachen. Der Programmcode kann auch bei anderen Rechnersystemen wieder verwendet werden – bei einigen (älteren) Sprachen müssen allerdings bei einer Portierung einige Anpassungen vorgenommen werden.
Nachteile	Das Programm der höheren Programmiersprache verbrauchte früher mehr Speicherplatz und war meist auch langsamer als das vergleichbare Maschinenprogramm. Durch optimierte Übersetzung verschwinden diese Nachteile aber bei vielen modernen höheren Programmiersprachen.
Programmiersprachen	Cobol, RPG, Fortran, Pascal, PL/1, Basic, Ada, C/C++, Java und viele mehr
Beispiel (Java)	`flaeche = laenge * breite;`

Vierte Generation: 4GL (Generation Language)

Während die ersten drei Generationen noch relativ klar voneinander getrennt werden können, fehlen bei der folgenden Generation eindeutige Kriterien.

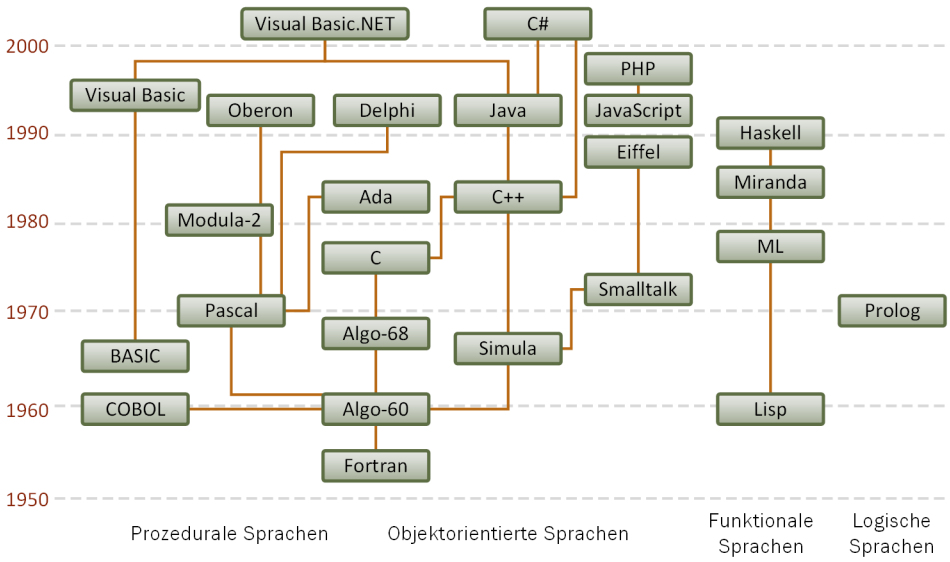
Beschreibung	Bei 4GL-Programmiersprachen beschreibt der Programmierer lediglich, was das Programm leisten soll, ohne den genauen algorithmischen Weg anzugeben. Eine einzelne Anweisung löst eine ganze Folge von internen Einzelschritten aus.
Einsatz	4GL-Programmiersprachen sind auf spezielle Anwendungsgebiete ausgerichtet, z. B. auf das Bearbeiten und Auswerten von Dateien, Datenbanken, Tabellenkalkulationen oder auf das Erstellen von Bildschirmformularen.

Vorteile	Zum Programmieren stehen fortschrittliche, einfach zu bedienende und leistungsfähige Entwicklungssysteme zur Verfügung. Die Erstellung eines Programms wird dadurch wesentlich erleichtert.
Nachteile	Die größtmögliche Effizienz der Programme ist nicht gegeben. Da bestimmte Folgen von Arbeitsschritten innerhalb einer Anweisung automatisch ausgeführt werden, hat der Programmierer kaum einen Einfluss auf die internen Abläufe. Die Ausführungsgeschwindigkeit dieser Programme ist aufgrund der mächtigeren Sprache langsamer als bei prozeduralen Sprachen.
Programmiersprachen	SQL, Natural, Symphony, Open Access
Beispiel (SQL)	`CREATE Adresskartei` `SELECT Kunde FROM Tabelle WHERE KdNr = 10`

Fünfte Generation: Künstliche Intelligenz

Beschreibung	Der Grundgedanke des Forschungsgebietes der künstlichen Intelligenz ist es, zu untersuchen, unter welchen Bedingungen Computer menschliche Verhaltensweisen, die auf Intelligenz beruhen, nachvollziehen können. Für diese Forschungszwecke sind einige spezielle Programmiersprachen entwickelt worden. Diese Sprachen gehören meist zu den logischen oder funktionalen Programmiersprachen, die Sie im weiteren Verlauf dieses Kapitels kennen lernen.
Einsatz	Inzwischen umfasst dieses Gebiet mehrere Fachbereiche, beispielsweise die Robotik, die zur Entwicklung von Robotern und deren komplizierten Bewegungsabläufen geführt hat, die Wissensverarbeitung und die Spracherkennung.
Programmiersprachen	Prolog, Lisp, Smalltalk
Beispiel (Prolog)	`grossvater(X,Y) :- vater(X,Z), vater(Z,Y).`

2.5 Die Klassifizierung nach Sprachtypen



Entwicklung und Verwandtschaft verschiedener Programmiersprachen

2.6 Prozedurale Programmiersprachen

BeginneProgramm
Anweisung1
Anweisung2
Anweisung3
...
BeendeProgramm

Als prozedurale Programmiersprachen werden höhere Programmiersprachen bezeichnet, die den Weg zur Problemlösung als eine Folge von Anweisungen angeben, die nacheinander abgearbeitet werden.

Die Programmiersprache Fortran

Bedeutung	Fortran steht für **For**mula **Tran**slator.
Entwicklung	Fortran wurde 1954 von IBM entwickelt und gilt als die erste Sprache der dritten Generation. In der aktuellen Version Fortran 2008 ist Fortran auch objektorientiert.
Einsatz	Fortran ist technisch-wissenschaftlich orientiert und wird hauptsächlich für mathematische oder technische Anwendungen eingesetzt.

Die Programmiersprache COBOL

Bedeutung	COBOL steht für **Co**mmon **B**usiness **O**riented **L**anguage.
Entwicklung	COBOL wurde speziell für betriebswirtschaftliche Anwendungen entwickelt und 1959 eingeführt. 1968 wurde COBOL durch das American National Standard Institute (ANSI) genormt. Seitdem wurde es kontinuierlich erweitert und modifiziert. Der letzte gültige Stand ist Cobol 2002. Diese Version enthält objektorientierte Erweiterungen.
Einsatz	COBOL ist eine noch immer weit verbreitete Programmiersprache. Im Bereich der kaufmännischen Großrechner sind weit über die Hälfte aller Anwendungen in COBOL geschrieben. Selbst auf dem PC werden betriebswirtschaftliche Anwendungen, die auf Großrechnerdaten zugreifen, in COBOL entwickelt.

Die Programmiersprache BASIC/Visual Basic

Bedeutung	BASIC steht für **B**eginners **A**ll-Purpose **S**ymbolic **I**nstruction **C**ode.
Entwicklung	BASIC wurde 1965 für Schulungszwecke entwickelt. Die Weiterentwicklung von BASIC führten viele Hersteller allerdings im Alleingang durch, weshalb für viele Rechnertypen eine eigene BASIC-Version entstand. 1991 wurde BASIC zu Visual Basic weiterentwickelt und ab 2002 auch in das .NET Framework, eine Entwicklungsumgebung für Windows-basierte Anwendungen von Microsoft, integriert.
Einsatz	Haupteinsatzbereich für BASIC waren Mikrocomputer. Die objektorientierte Programmiersprache Visual Basic 2008 wird zur Programmierung von Windows-basierten Anwendungen verwendet.

Die Programmiersprache C

Entwicklung	Die Programmiersprache C wurde Anfang der 70er-Jahre von B. W. Kernighan und D. M. Ritchie im Auftrag der Bell Laboratories während der Arbeit an dem Betriebssystem UNIX entwickelt.
Einsatz	C eignet sich für die Entwicklung von systemnahen Programmen. So ist beispielsweise das Betriebssystem UNIX in C geschrieben worden. Die letzte Variante der Sprache ist C99 von 1999.

Hinweise	C hat alle Vorteile einer höheren Programmiersprache. Gleichzeitig kann damit sehr hardwarenah programmiert werden, was sonst nur bei Assembler-Sprachen möglich ist. Bedingt durch die Normierung von C durch die ANSI-Kommission können Programme relativ leicht auf andere Rechnertypen übertragen (portiert) werden, sofern sich die Programmierer an die normierten Konventionen halten.

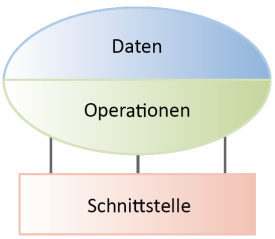
Die Programmiersprache Pascal

Bedeutung	Die Programmiersprache Pascal wurde nach dem französischen Philosophen Blaise Pascal (1623-1662) benannt.
Entwicklung	Entwickelt wurde Pascal 1971 von dem Schweizer Computerwissenschaftler Nikolaus Wirth, um die strukturierte Programmierung zu lehren. Bei der strukturierten Programmierung wird ein Programm in logische Einheiten zerlegt. Zur Steuerung des Ablaufs werden bestimme Kontrollstrukuren definiert, in Pascal Sequenz, Auswahl und Wiederholung (vgl. Kapitel 8). Eine besonders weite Verbreitung erfuhr Pascal durch die von Borland entwickelte Version TURBO Pascal und deren Nachfolger DELPHI, die eine leicht bedienbare Programmierumgebung bestehend aus Editor und Compiler besitzen und bezüglich ihres Befehlssatzes gegenüber der Standardversion von 1971 stark erweitert worden sind. Seit Pascal 7 hat Pascal auch objektorientierte Ansätze, die in DELPHI zu einer objektorientierten Sprache ausgebaut wurden.

2.7 Objektorientierte Programmiersprachen

Die Weiterentwicklung der Computer führte auch zur Entwicklung komplexerer Software. Entsprechend wurde die prozedurale Programmierung zur objektorientierten Programmierung weiterentwickelt. Das Problem der prozeduralen Programmierung ist, dass globale Daten in jedem Teil des Programms manipuliert und überschrieben werden können. Es fehlt eine Verbindung zwischen den Daten eines Programms und den sie manipulierenden Funktionen. Dies führt dazu, dass große Programme sehr leicht unübersichtlich werden und sich schwerer testen lassen.

1970 erkannte David Parnas das Problem und hatte die Idee, jedes einzelne Datenelement in einem Modul zu kapseln. Der direkte Zugriff auf diese Daten wurde nur über eine bestimmte Schnittstelle mit einem Satz von Operationen, wie z. B. über Prozeduren oder Funktionen, erlaubt. Sollen andere Module ebenfalls auf die Variable zugreifen, können sie dies nur indirekt über die Schnittstelle des Moduls tun.



Die Programmiersprache C++

Entwicklung	1982 begann Bjarne Stroustrup eine Erweiterung der prozeduralen Programmiersprache C zu entwickeln. 1989 wurde die Basissprache definiert, 1996 der internationale Standard (ISO/IEC 14882) verabschiedet. Die letzte Überarbeitung der Sprache wurde 2003 veröffentlicht.
Einsatz	C++ eignet sich für die Entwicklung von systemnahen Programmen und von komplexen Anwendungen.
Hinweise	Es gibt zur objektorientierten Programmierung mit C++ mächtige Programmierwerkzeuge. C++-Compiler stehen praktisch auf jeder Rechnerplattform zur Verfügung. C++ besitzt mächtige Sprachmittel, um komplexe Anwendungen zu erzeugen. Außerdem stehen umfangreiche Klassenbibliotheken zur Verfügung.

Die Programmiersprache Java

Entwicklung	Eine Gruppe von Ingenieuren bei Sun Microsystems entwickelte 1991 Software für interaktives Fernsehen und andere Geräte der Konsumelektronik. Diese Programmiersprache nannte sich „Oak“. Mit der zunehmenden Verbreitung des Internet wurde Oak dafür angepasst und in **Java** umbenannt. Java wurde an C/C++ angelehnt, wobei auf verschiedene (fehlerträchtige) Konstrukte verzichtet wurde.
Einsatz	Anwendungen, die auf unterschiedlichen Rechnersystemen laufen sollen, sowie für verteilte Anwendungen
Hinweise	Da nicht alle Sprachelemente von C++ realisiert sind, ist Java eine relativ schlanke und übersichtliche Sprache. Java-Compiler sind für Windows und Linux/Unix sowie MacOS kostenlos erhältlich und erzeugen maschinenunabhängigen Code, sogenannten Bytecode (vgl. Abschnitt 5.2). Dieser wird in einer virtuellen Maschine ausgeführt. Java besitzt eine umfangreiche Programmbibliothek für verschiedene Bereiche.

Bei den Programmcode-Beispielen in diesem Buch wird überwiegend die Programmiersprache Java verwendet.

.NET und die Programmiersprache C#

Entwicklung	Mit .NET bezeichnet Microsoft seine Software-Plattform zur Entwicklung und Ausführung von Anwendungsprogrammen. C# ist eine objektorientierte Programmiersprache für .NET. C# greift Konzepte der Programmiersprachen Java, C++, C und Delphi auf und wurde von Microsoft erstmals im Jahr 2000 standardisiert.
Einsatz	C# eignet sich besonders für eine bequeme Erstellung von sicheren, stabilen und gut an die Zielplattform angepassten Windows-Programmen. Aber auch für andere Betriebssysteme (etwa Linux) gibt es Laufzeitumgebungen.
Hinweise	.NET besteht aus einer Laufzeitumgebung (Common Language Runtime – CLR), in der die Programme ausgeführt werden, sowie einer Sammlung von Klassenbibliotheken, Programmierschnittstellen und Dienstprogrammen (analog Java und seiner virtuellen Maschine). .NET unterstützt neben C# die Verwendung weiterer Programmiersprachen. Unabhängig von der verwendeten Programmiersprache werden .NET-Programme in eine Zwischensprache (Common Intermediate Language – CIL) übersetzt, bevor sie von der Laufzeitumgebung ausgeführt werden (vgl. Abschnitt 4.2).

2.8 Hybride Programmiersprachen und Skriptsprachen

Die strenge Aufteilung in prozedurale und objektorientierte Sprachen ist bei vielen modernen Sprachen nicht möglich oder sinnvoll. Solche Sprachen können sowohl prozedural als auch objektorientiert eingesetzt werden. Man nennt diese Sprachen deshalb oft auch **hybrid**. Können hybride Sprachen mit Objekten umgehen, stellen jedoch nicht den vollen Umfang von objektorientierten Sprachen bereit, werden sie **objektbasiert** oder **objektbasierend** genannt. Gerade sogenannte Skriptsprachen sind oftmals sowohl objektbasierend als auch hybrid. Skriptsprachen sind in der Regel von einfacherer Natur und werden meist über einen Interpreter ausgeführt, der die Anweisungen erst zur Laufzeit in Maschinenbefehle übersetzt.

Programme, die in Skriptsprachen geschrieben sind, werden Skripte (engl. scripts) genannt. Teilweise wird auch die Bezeichnung **Makro** verwendet.

Die Anwendungsgebiete und Eigenschaften konventioneller Programmiersprachen und Skriptsprachen überschneiden sich mittlerweile stark, weshalb eine strikte Trennung nur selten möglich ist.

JavaScript

Entwicklung	**JavaScript** ist eine Skriptsprache, die ursprünglich von der Firma Netscape für die Erweiterung von HTML in Webbrowsern entwickelt wurde, um Benutzerinteraktionen auszuwerten und dynamisch Inhalte der Webseite zu verändern. Im Jahr 1995 erschien mit dem Netscape Navigator 2.0 der erste Browser mit einer eingebetteten Skriptsprache, die zu diesem Zeitpunkt **LiveScript** genannt wurde. Im Rahmen einer Kooperation zwischen Netscape und Sun Microsystems wurde LiveScript in JavaScript umbenannt. Mittlerweile unterstützen alle modernen Webbrowser JavaScript, wobei Microsoft einen eigenen Dialekt mit Namen **JScript** verwendet und Anwender JavaScript im Browser auch deaktivieren können.
Einsatz	JavaScript wurde früher fast ausschließlich clientseitig im Webbrowser eingesetzt. Heutzutage finden Sie JavaScript-Implementierungen aber auch auf Servern, im Umfeld von Java oder .NET, in Datenbanken, Microcontrollern oder bei Apps.
Hinweise	**ECMAScript** (ECMA 262) bezeichnet den standardisierten Sprachkern von JavaScript. Die Syntax von JavaScript basiert auf C und stimmt in großen Bereichen mit der von Java überein. Aber trotz der Namens- und syntaktischen Ähnlichkeit hat JavaScript **konzeptionell** nur geringe Gemeinsamkeiten mit Java. JavaScript ist nicht streng objektorientiert, sondern objektbasierend und verwendet etwa bei der Vererbung Prototypen statt Klassen. Man kann mit JavaScript sowohl prozedural als auch rein objektorientiert arbeiten, wenn man dabei gewisse Konventionen einhält, die aber nicht zwingend sind.

Python

Entwicklung	**Python** ist eine universelle, höhere Programmiersprache, die üblicherweise interpretiert wird. Es gibt also in der Laufzeitumgebung von Python einen Interpreter samt notwendiger weiterer Ressourcen. Python unterstützt sowohl die objektorientierte, die aspektorientierte, die strukturierte als auch die funktionale Programmierung. Das bedeutet, Python zwingt den Programmierer nicht zu einem einzigen Programmierstil, sondern erlaubt das für die jeweilige Aufgabe am besten geeignete Paradigma zu wählen. Objektorientierte und strukturierte Programmierung werden vollständig unterstützt, funktionale und aspektorientierte Programmierung werden zumindest durch einzelne Elemente der Sprache unterstützt. Ein zentrales Feature ist in Python die dynamische Typisierung samt dynamischer Speicherbereinigung. Damit kann man Python auch als reine Skriptsprache nutzen.
Einsatz	Das Einsatzgebiet von Python ist breit gestreut. Man kann damit so gut wie jede Form von Applikation schreiben und es gibt für die wichtigsten Betriebssysteme Implementierungen.
Hinweise	Python hat seine Grundlagen in C und ist den meisten gängigen Programmiersprachen verwandt, wurde aber mit dem Ziel größter Einfachheit und Übersichtlichkeit entworfen. Zentrales Ziel bei der Entwicklung der Sprache ist die Förderung eines gut lesbaren, knappen Programmierstils. So wird beispielsweise der Code nicht durch geschweifte Klammern (wie in fast allen anderen C-basierenden Sprachen), sondern durch zwingende Einrückungen strukturiert. Zudem ist die gesamte Syntax reduziert und auf Übersichtlichkeit optimiert. Wegen dieser klaren und überschaubaren Syntax ist Python einfach zu erlernen, zumal die Sprache mit relativ wenigen Schlüsselwörtern auskommt. Es wird immer wieder zu hören sein, dass sich Python-basierte Skripte deutlich knapper formulieren lassen, als es in anderen Sprachen der Fall ist. In vielen Ländern hat Python bei Anfängerkursen in informatikbezogenen Studiengängen an Universitäten Java, was über viele Jahre die Szene beherrscht hatte und im professionellen Umfeld immer noch das Maß aller Dinge darstellt, abgelöst.

Bei den Programmcode-Beispielen in diesem Buch wird teilweise auch JavaScript verwendet. Diese Skripte können in jedem modernen Browser ausgeführt werden, sofern nicht JavaScript deaktiviert ist.

PHP

Entwicklung	**PHP** (rekursives Akronym für „PHP: Hypertext Preprocessor“ – ursprünglich Personal Home Page Tools) ist eine Skriptsprache, die 1995 von Rasmus Lerdorf entwickelt wurde. PHP war ursprünglich als Ersatz für eine Sammlung von Perl-Skripten gedacht, hat sich aber mittlerweile zu einer sehr verbreiteten Sprache bei der serverseitigen Webprogrammierung entwickelt.
Einsatz	PHP wird hauptsächlich zur serverseitigen Erstellung dynamischer Webseiten oder Webanwendungen verwendet. PHP ist OpenSource und zeichnet sich durch breite Datenbankunterstützung sowie die Verfügbarkeit zahlreicher Funktionsbibliotheken aus.
Hinweise	Die Syntax von PHP basiert auf C. Anfangs war PHP nicht objektorientiert, sondern rein prozedural. Mittlerweile wurde PHP erweitert, so dass Sie bei Bedarf mit PHP rein objektorientiert arbeiten können. PHP ist somit eine hybride Sprache, wobei neuere Sprachversionen immer mehr in Richtung Objektorientierung tendieren.

2.9 Funktionale und logische Programmiersprachen

Funktionale Programmiersprachen

Bei funktionalen Programmiersprachen ist das Programm eine Funktion, die sich typischerweise auf einfachere Funktionen stützt, daher auch der Name „funktionale Programmiersprache“. Die Beziehungen zwischen den Funktionen sind einfach: Eine Funktion kann eine andere aufrufen, oder das Ergebnis einer Funktion kann als Parameter für eine andere Funktion genutzt werden.

Programme werden wie mathematische Funktionen geschrieben. Eine Funktion hat einen Definitions- und einen Wertebereich. Die Funktion erhält einen Eingabewert und berechnet, mathematisch gesehen, den Wert der Funktion.

Die Programmiersprache LISP

Entwicklung	Die Programmiersprache LISP wurde von John McCarthy und einer Arbeitsgruppe am Massachusetts Institute of Technology Anfang der 60er-Jahre entwickelt. Es war die erste implementierte funktionale Sprache. Heute gibt es viele Dialekte von LISP, wie z. B. die Sprache Scheme.
Einsatz	LISP wird in der Informatik im Forschungsbereich eingesetzt und dort vorwiegend im Bereich der künstlichen Intelligenz und zur Lösung mathematischer Probleme.
Vorteil	Die Sprache ermöglicht es, komplexe Programme, die Zeichenketten bearbeiten, leichter als in anderen Programmiersprachen zu schreiben.
Beispiel zur Fakultätsberechnung	`>(defun fakultaet (x)` → Antwort: Funktionsdefinition `  (if ( x > 0) (* x (fakultaet (- x 1))) 1 ) )` `FAKULTAET` → Antwort: definierter Funktionsname `>(fakultaet 5)` → Funktionsaufruf mit dem Wert 5 `120` → Antwort: errechneter Wert

Logische Programmiersprachen

Bei logischen Programmiersprachen wird kein Algorithmus zum Lösen eines Problems angegeben, sondern es werden lediglich die Bedingungen für eine korrekte Lösung bestimmt. Stellen Sie dann eine Frage, erhalten Sie aufgrund dieser Wissensbasis eine entsprechende Antwort.

Die Programmiersprache PROLOG

Bedeutung	Der Name PROLOG ist eine Abkürzung für „**PRO**gramming in **LOG**ic“.
Entwicklung	Die Programmiersprache PROLOG wurde 1970 von Alain Colmerauer und Philippe Roussel entwickelt. PROLOG hat sich in den 80er-Jahren verbreitet, jedoch wurde durch die fehlende Entwicklung von Anwendungen die weitere Verbreitung behindert.
Einsatz	Dient als experimentelles Werkzeug für künstliche Intelligenz, wird jedoch hauptsächlich an Universitäten benutzt

Beispiel	`Links_von( Apfel, Birne ).  | Factum`
	`Links_von( Pflaume, Apfel ).  | Factum`
	`Links_von( X, Y ) :- Rechts_von( Y, X ).  | Regel`
	`...`
	`Frage: ?-Rechts_von(X, Apfel).  | Frage`
	`Antwort: X = Pflaume  | Antwort`

2.10 Erziehungsorientierte Programmiersprachen und Minisprachen

Heutzutage werden bereits kleine Kinder mit Computern konfrontiert. Erziehungsorientierte Programmiersprachen und Minisprachen sind als Lerninstrumente für Programmieranfänger konzipiert, die spielerisch mit der Funktionsweise und den Prinzipien der Programmierung vertraut machen sollen.

Solche Sprachen sind nicht zwingend auf Kinder beschränkt, sondern können sowohl für Jugendliche als auch für die Erwachsenenbildung sinnvoll sein. Ziel vieler Sprachen ist die Erstellung visuell interessanter Anwendungen und Oberflächen ohne „echte“ Programmierung. Andere erziehungsorientierte Konzepte agieren als virtuelle Roboter oder Umgebungen. Dabei werden Objekte wie ein Roboter, ein Hamster oder eine Schildkröte in einer Umgebung gesteuert. Das kann mit einer „erwachsenen“ Programmiersprache im Umfeld einer virtuellen Umgebung erfolgen, aber auch mit vereinfachten Techniken.

Erziehungsorientierte Programmiersprachen sind meist mit integrierten Entwicklungsumgebungen (IDE, vgl. Abschnitt 4.3) verknüpft, um Programme möglichst einfach und menügeleitet zu erstellen. Manchmal wird sogar ganz auf die Eingabe von Text durch den Anwender verzichtet und mit Symbolen gearbeitet, um Kinder mit den Grundzügen der Programmierung vertraut zu machen, ohne die Syntax einer Programmiersprache lernen zu müssen.

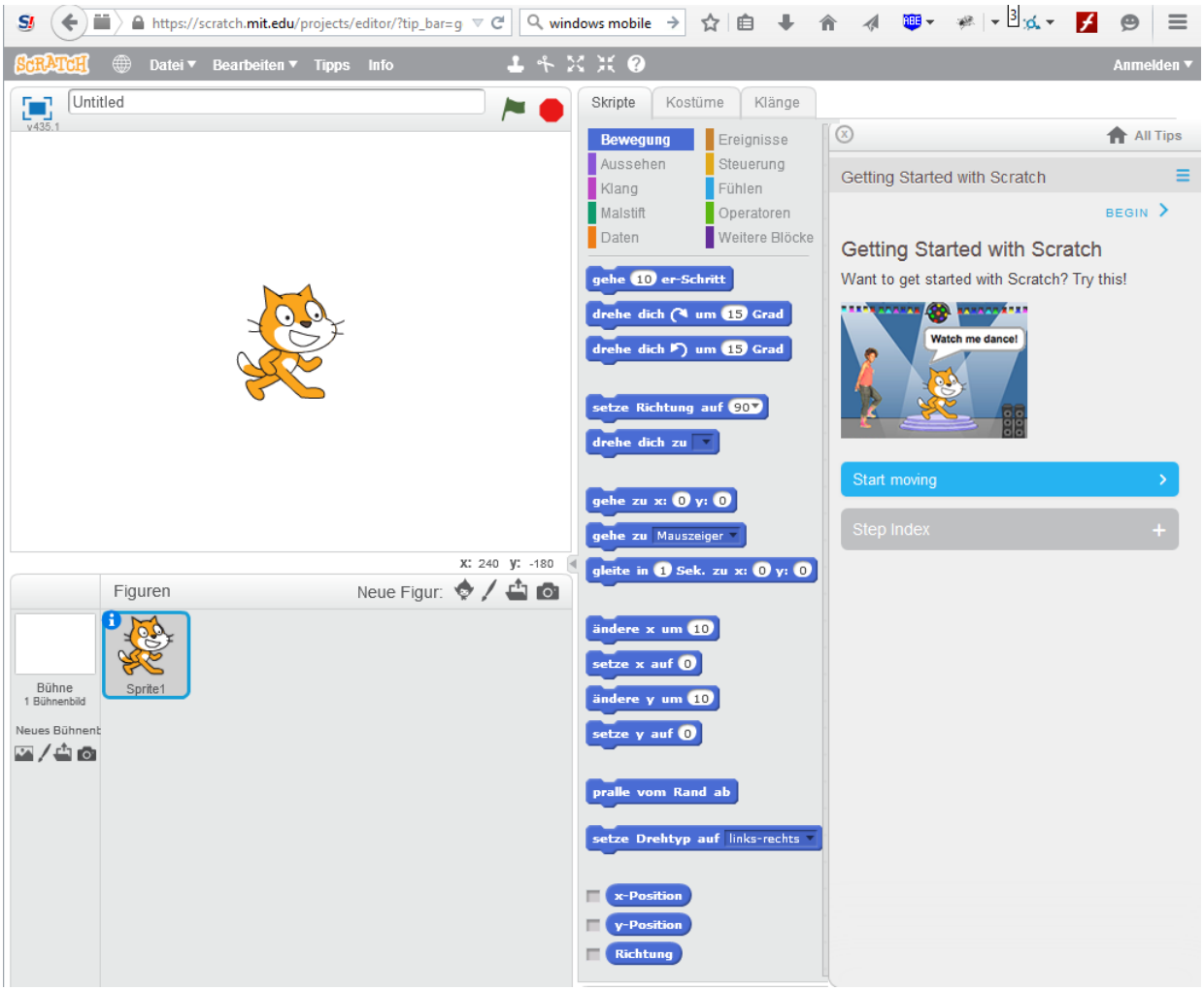
Die Programmiersprache Logo

Entwicklung	Als die älteste kindgerechte Programmiersprache gilt die von Seymour Papert 1967 auf der Grundlage von Lisp entwickelte Programmiersprache **Logo**.
Einsatz	Mit Logo wird die **Turtle-Grafik** erstellt, bei der eine virtuelle Schildkröte über den Bildschirm kriecht und dabei eine farbige Linie hinter sich herzieht. In einigen Ländern wird Logo noch heute in Schulen eingesetzt.

Die Programmiersprache Scratch

Entwicklung	Im Jahr 2007 wurde Scratch veröffentlicht. Mittlerweile gibt es die Version 2.0. Entwickelt wurde Scratch unter der Leitung von Mitchel Resnick am MIT Media Lab. Die ersten Implementierungen der Scratch-Entwicklungsumgebung basierten auf Squeak. Den folgenden Scratch-Web-Player gibt es auf Basis von Java oder Flash. Scratch 2.0 erschien im Mai 2013 und basiert komplett auf Flash.
Einsatz	Scratch soll Anfänger – besonders Kinder und Jugendliche – motivieren, die Grundkonzepte der Programmierung spielerisch und über interessante Experimente zu erlernen. Die Programmiersprache verwendet eine visuelle Entwicklungsumgebung, die Wert auf eine intuitive Bedienung und leichte Übersetzbarkeit legt und vorgefertigte Bausteine zur Programmierung bietet.

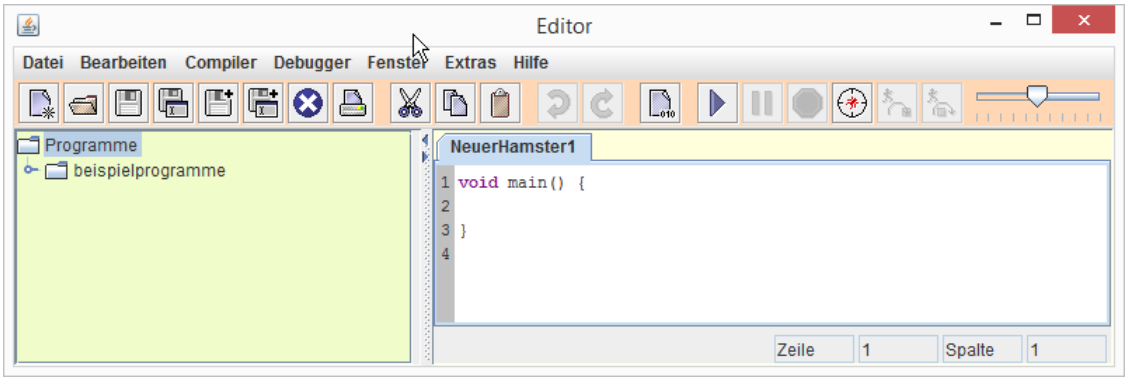
Zur Verdeutlichung verschiedener Konzepte in diesem Buch kann Scratch eine sinnvolle Ergänzung sein. Unter *https://scratch.mit.edu/* finden Sie die wichtigsten Informationen zu Scratch inklusive der Entwicklungsumgebung, die Sie direkt aus einem Browser heraus ausführen können. Informationen zu Scratch finden Sie im HERDT-Buch *Scratch 2.0 - Spielend programmieren lernen*.



Der Editor von Scratch kann direkt im Browser gestartet werden

Das Hamster-Modell

Entwicklung	Das Hamster-Modell gehört zu den Minisprachen mit pädagogischem Hintergrund. Ziel des Modells ist es, Programmieranfängern das Erlernen grundlegender Programmierkonzepte durch den Einsatz spielerischer Mittel zu erleichtern. Seit 2010 gibt es die Version 2.x, die immer noch aktuell ist. Seit Oktober 2016 gibt es die Version 2.9.5 des Hamster-Simulators, die allerdings nicht mit der Java SE 9 Version lauffähig ist. Deshalb benötigt man dafür noch Java SE 8!
Einsatz	Grundsätzlich geht es beim Hamster-Modell darum, einen Hamster mit einem minimalen Satz an Befehlen durch eine virtuelle Umgebung zu steuern und Körner sammeln zu lassen. Programme können hierbei mittlerweile in verschiedenen Programmiersprachen erstellt werden, wobei die ursprüngliche Ausrichtung auf Java zielte. Auch Scratch-Anwendungen werden von dem Modell unterstützt. Eine spezielle Entwicklungsumgebung, der sogenannte **Hamster-Simulator**, hilft bei der Entwicklung und visualisiert den Ablauf der Hamster-Programme. Der kostenlose Simulator ist bewusst an bekannte Entwicklungsumgebungen für Java angelehnt, z. B. als Plugin für Eclipse, aber auch als eigenständiges Tool.



Der Hamster-Simulator

Im Buch wird oft mit Java gearbeitet. Wenn Sie Hamster verwenden wollen, können Sie den Simulator als komprimiertes Archiv laden (*http://www.java-hamster-modell.de/*) und extrahieren. Das Tool ist sofort einsatzfähig. Einzelne Java-Code-Passagen können Sie im Hamster-Modell ausführen, wenn Sie keine vollständige Entwicklungsumgebung oder das JDK verwenden wollen. Sie erhalten eine Art Gerüst, um sich auf die Codepassagen konzentrieren zu können, die gerade behandelt werden. Das Hamster-Modell ist für kleinere Anwendungen beim Programmieren lernen entwickelt. Für größere Anwendungen in der Praxis ist dieses Tool nicht geeignet.

2.11 Entwicklung der Webprogrammierung

In den letzten Jahren hat die Entwicklung von Programmen eine massive Veränderung erfahren. Bis vor wenigen Jahren konnten Programme überwiegend in Desktop- und Großrechnerprogramme eingeteilt werden. Mittlerweile sind zwei weitere Programmarten von Bedeutung: interaktive Web-Applikationen (RIAs) und mobile Apps.

Interaktive Web-Applikationen (RIAs)

Seit Anfang der 1990er-Jahre hält das Internet Einzug in das tägliche Leben und die Bedeutung wächst stetig. Dabei nutzen Anwender neben **E-Mail** hauptsächlich das World Wide Web (**WWW**) über einen Browser.
Im Browser werden Webseiten angezeigt, die von einem Webserver geladen werden. Die Webseiten werden fast ausschließlich mit der Dokumentenbeschreibungssprache **HTML** (**H**yper**t**ext **M**arkup **L**anguage) erstellt.
Das Layout bei modernen Webseiten wird mittels Formatvorlagen (Style Sheets) festgelegt. Im WWW kommen hauptsächlich **CSS** (**C**ascading **S**tyle **S**heets) als Formatvorlagen zum Einsatz. Standardisiert werden die Techniken für das WWW vom W3C (*http://www.w3.org*).

HTML ist **explizit keine Programmiersprache**. Im Laufe der Jahre hat sich das WWW jedoch zu einem System entwickelt, in dem Anbieter von Inhalten sowohl auf dem Webserver als auch dem Client (Browser) programmieren können. Im Client werden dazu HTML-Seiten durch **JavaScript** erweitert, während auf dem Webserver sehr oft **PHP**, aber auch Java oder Sprachen aus dem .NET-Umfeld zum Einsatz kommen. Das einfache Konzept des WWW mit der Anforderung von Webseiten führte durch die Entwicklung von Webseiten mit interaktiven Inhalten zu Problemen. Bei der Anfrage eines Browsers an einen Webserver nach neuen Daten muss dieser immer eine **vollständige Webseite** als Antwort senden. Oder genauer – der Browser versteht die Antwort so, dass er die bisher im Browser angezeigte Seite durch diesen neuen Inhalt vollständig ersetzt. Diese Vorgehensweise ist aufwändig, vor allem wenn nur kleine Änderungen notwendig sind.

Es wurde ein Verfahren benötigt, um eine Reaktion bei einer interaktiven Webseite in (nahezu) Echtzeit zu gewährleisten, obwohl neue Daten vom Webserver angefordert werden. **Ajax** (**A**synchronous **J**avaScript **a**nd **X**ML) wurde entwickelt. Ändern sich Daten einer im Browser vorhandenen Webseite, führt eine Ajax-Datenanfrage dazu, dass nur die neuen Daten vom Webserver geschickt werden und diese dann mit JavaScript in die bereits beim Client geladene Webseite „eingebaut“ werden. Dabei wird in der Regel die Interaktion des Benutzers mit der Webseite durch das Laden neuer Daten nicht unterbrochen. Dieses Verfahren erlaubt nun auch im Web die Erstellung von Angeboten, die sehr stark auf Interaktion mit dem Anwender setzen. **Web 2.0** wird als Oberbegriff für die meisten interaktiven Webangebote verwendet. Interaktive Webangebote werden als **RIAs** (**R**ich **I**nternet **A**pplications) bezeichnet und sind – im Gegensatz zu statischen Webseiten ohne Programmierung – Programme bzw. Applikationen (Web-Applikationen).

Mobile Apps

Mobile Endgeräte und die darauf laufenden Anwendungen (Programme) – sogenannte **Apps** als Abkürzung von Applications – gehören mittlerweile zum Alltag. Mobilität zeigt sich vielfältig, in Form von Handys, Smartphones oder Tablets über E-Book-Reader bis hin zu Netbooks, Ultrabooks und Notebooks. Die Grenzen der Gerätetypen wie auch der mobilen Apps zu klassischen Desktopanwendungen verschwimmen. Bei mobilen Anwendungen ist eine permanente Verbindung mit dem Internet mehr oder weniger ein Muss. Die Anwendungen (Apps) müssen den Besonderheiten mobiler Endgeräte Rechnung tragen.

Sind Netbooks, Ultrabooks oder Notebooks dabei noch „echte“ Computer, die mit Tastatur und Maus bedient werden können, müssen E-Book-Reader, Smartphones, Tablets oder gar Smart-Watches meist ohne solche externen Eingabegeräte auskommen. Die Eingabe bei kleinen Geräten wird auf wenige Hardwaretasten, Touchscreens, Sensoren und Spracheingabe übertragen. Dazu werden spezifische Designs der Benutzeroberflächen benötigt. Mobile Geräte besitzen oft spezifische Hardware, die bei stationären Geräten oder klassischen Computern nicht vorhanden ist. Besondere Hardwaresensoren, z. B. zum Erkennen der Orientierung des Bildschirms oder des Standortes eines Benutzers, unterstützen die mobile Verwendung.

Im mobilen Bereich werden viele verschiedene Techniken eingesetzt. Dies betrifft die Hardware ebenso wie die Bedienkonzepte, die verwendeten Programmiersprachen und die Betriebssysteme.

In der Praxis nutzen die meisten mobilen Endgeräte entweder Android, iOS oder Windows Phone als Betriebssystem.

Native Programmierung von mobilen Apps bedeutet, dass Sie sich auf eine Hardware samt speziellem Betriebssystem oder nur eine spezielle Version davon konzentrieren. Dazu verwenden Sie klassische Programmiersprachen wie **Object C** oder **Swift** (Regelfall unter iOS), **Java** (meist unter Android eingesetzt) sowie **C/C++** oder **C#** (Windows Phone).

Möchten Sie die App-Programmierung für Android lernen, kann die Webseite *http://www.app-entwickler-verzeichnis.de/apps-programmierung/24-android/297-android-programmierung-tutorial-der-grosse-android-newbie-guide* hilfreich sein. Im vorliegenden Buch werden die Grundlagen vermittelt, die Sie zu der Erstellung von Java-Apps benötigen. Als Entwicklungsumgebung für die App-Entwicklung für Android bietet sich vor allen Dingen das Android SDK bzw. das Android Studio (*https://developer.android.com/sdk/index.html*) an. Aber auch mit den neuen Versionen von Microsoft Visual Studio (*https://www.visualstudio.com/de/downloads*) kann man Apps für Android entwickeln.

Wollen Sie mehrere Systeme unterstützen, müssen Sie ggf. für jedes Zielsystem eine eigene App erstellen. Alternativ können Sie mit klassischen Web-Technologien (HTML, JavaScript, CSS) als Basis einer App arbeiten und damit weitgehend neutral von verschiedenen Zielplattformen. Insbesondere HTML5 und CSS3 werden in Verbindung mit JavaScript mittlerweile von allen relevanten Herstellern mobiler Endgeräte unterstützt. Dabei sind Sie erst einmal auf den jeweilig verfügbaren eingebetteten Browser beschränkt und können nicht auf die gesamte mobile Hardware zugreifen. Es gibt aber auch Umgebungen wie Cordova (*https://cordova.apache.org/*), um darüber auf die mobile Hardware zugreifen zu können und auch mit Web-Technologien „echte“ Apps zu erstellen.

2.12 Übungen

Übung 1: Fragen zu Grundlagen der Datenverarbeitung

Übungsdatei: -- **Ergebnisdatei: *uebung02.pdf***

1. Was ist ein Programm? Was ist Software?
2. Beschreiben Sie das EVA-Prinzip.

Übung 2: Fragen zu Programmiersprachen

Übungsdatei: -- **Ergebnisdatei: *uebung02.pdf***

1. Nach welchen Kriterien können Programmiersprachen klassifiziert werden?
2. Gehören Assembler-Sprachen zu den höheren Programmiersprachen? Begründen Sie Ihre Antwort.
3. Ordnen Sie Programmiertechniken und Programmiersprachen zu:

Programmiersprachen	prozedural	objektorientiert	funktional	logisch
Lisp				
C#				
Pascal				
C++				
Java				
Prolog				

3 Darstellungsmittel für Programmabläufe

In diesem Kapitel erfahren Sie

- wie Sie einen Programmablauf visualisieren können
- was ein Programmablaufplan, ein Datenflussdiagramm und ein Struktogramm sind
- wie Sie Pseudocode schreiben
- wie Sie Entscheidungstabellen verwenden können

3.1 Programmabläufe visualisieren

Viele Problemstellungen sind umfangreich und kompliziert. Deshalb erfordern sie eine systematische Vorarbeit, um den Lösungsweg zu beschreiben. Die strukturierte Programmierung verlangt eine gut durchdachte Vorbereitung, bevor Sie mit dem eigentlichen Programmieren (Codieren) beginnen.

Beim Entwurf eines Programms werden deshalb oftmals grafische Darstellungsmittel für die Logik des Programmablaufs verwendet, mit deren Hilfe sich Kontrollstrukturen und Verzweigungen anschaulich darstellen lassen (vgl. Kapitel 7). Es werden im Folgenden einige Entwurfstechniken vorgestellt, die Sie für Ihren Programmentwurf wählen können.

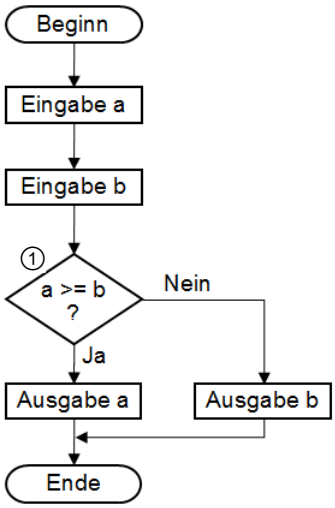
3.2 Programmablaufplan

Ein Programmablaufplan wird auch Ablaufdiagramm, Flussdiagramm oder Blockdiagramm genannt. Der Begriff **Programmablaufplan** (kurz: PAP) ist in der DIN 66001 genormt. Die aktuellste Veröffentlichung ist die Ausgabe 1983-12, siehe auch *http://www.din.de*.

Programmablaufpläne sind grafische Darstellungen mithilfe genormter Symbole. Sie sind weit verbreitete Hilfsmittel bei der Programmentwicklung. Die Programmstruktur lässt sich bildhaft als Ablauf darstellen. Die Pfeile zeigen dabei die Richtung der Verarbeitung an.

Programmablaufpläne werden meist in mehreren Stufen erstellt. Je nach Komplexität des Problems wird zunächst ein grober PAP entworfen, der immer weiter verfeinert wird, bis alle Befehle einzeln aufgeführt sind.

Die einzelnen Symbole haben unterschiedliche Bedeutungen. Die wichtigsten davon werden nachfolgend kurz erklärt.



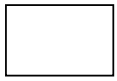
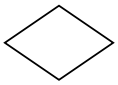
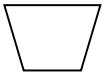
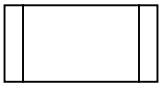
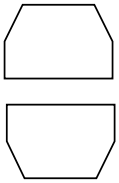
Programmablaufplan (PAP)

Erläuterung des PAP der obigen Abbildung

Nach Beginn des Programms werden die Werte für die Variablen `a` und `b` eingegeben. Dann folgt der Vergleich der Werte der Variablen `a` und `b` ①. Ist der Wert der Variablen `a` größer als der Wert der Variablen `b` oder gleich groß, ist die Bedingung erfüllt (der Weg *Ja* wird durchlaufen), und es wird der Wert der Variablen `a` ausgegeben. Anderenfalls, wenn der Wert der Variablen `b` größer ist, ist die Bedingung nicht erfüllt (Weg *Nein*) und der Wert der Variablen `b` wird ausgegeben. Danach laufen beide Wege wieder zusammen, und das Programm ist zu Ende.

PAPs wie auch die nachfolgend im Kapitel vorgestellten Diagramme können Sie mit Papier und Stift erstellen. Bequemer ist die Verwendung eines Grafikprogramms. Sie können auch spezialisierte Tools zur Erstellung von Diagrammen verwenden. Neben kommerziellen Produkten gibt es empfehlenswerte Freeware, z. B. den **PapDesigner** (*http://friedrich-folkmann.de/papdesigner/*), den **Dia Diagram Editor** (*http://dia-installer.de/*) oder den **Diagramm Designer** (*http://meesoft.logicnet.dk/*). Auch die Zeichnenfunktionen von Word, PowerPoint oder OpenOffice lassen sich sehr gut zum Erstellen verwenden.

Auswahl von Symbolen des PAP

	**Verarbeitung** Das Rechteck ist für Zuweisungen oder Ein- und Ausgabeoperationen vorgesehen.
	**Verbindung** Zur Verdeutlichung der Ablaufrichtung werden Linien mit einer Pfeilspitze benutzt.
	**Verzweigung** Bedingungen werden als Raute dargestellt. Eine Verbindungslinie führt hinein, zwei Verbindungslinien führen heraus. Je nach Wahrheitswert der Bedingung wird der Ablauf in die eine oder andere Richtung fortgeführt.
	**Manuelle Verarbeitung** Eingaben des Programmnutzers werden durch ein Trapez dargestellt.
	**Dokumentation an anderer Stelle** Durch dieses Symbol wird auf einen anderen PAP hingewiesen, der z. B. ein Unterprogramm darstellt.
	**Verbinder** Teilt sich ein Programmablauf und wird dann wieder zusammengeführt, wird der Verbinder (Konnektor) eingesetzt. Somit wird vermieden, dass sich Ablauflinien kreuzen. Damit wird die Übersichtlichkeit erhöht. Für größere PAPs, für die mehrere Seiten benötigt werden, wird der Verbinder am Seitenende der vorherigen und Seitenanfang der folgenden Seite verwendet. Zur Identifizierung wird er mit einer eindeutigen Zahl für diese Verbindungsstelle versehen.
	**Grenzstelle** Eine Grenzstelle kennzeichnet den Anfang und das Ende eines Programmablaufplans.
	**Schleifenbegrenzer** Zur Darstellung von Programmwiederholungen werden diese zwei Symbole benutzt, die den Anfang und das Ende einer Schleife kennzeichnen. In der Praxis verwendet man statt dieser Schleifensymbole oft auch Verbindungen, die wieder zu einem vorherigen Verarbeitungsschritt verweisen.

Geschachtelte Strukturen sind im PAP nicht gut zu erkennen. Ein weiterer Nachteil besteht darin, dass für objektorientierte Programmkonzepte keine Symbole definiert sind. Als vorteilhaft ist jedoch zu werten, dass …

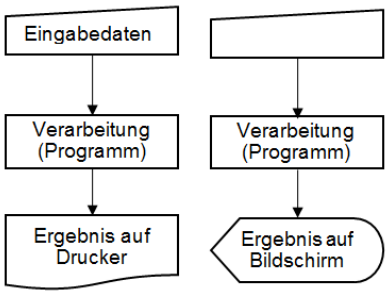
- Anweisungsteile der Algorithmen übersichtlich lesbar sind,
- die Terminierung überprüfbar ist,
- die Korrektheit überprüfbar ist,
- die Methodik der schrittweisen Verfeinerung unterstützt wird, da jede Kontrollstruktur als Blackbox betrachtet werden kann.

Entwerfen Sie bei der Erstellung des PAP zuerst die Kontrollstrukturen, bevor Sie die elementaren Anweisungen einbauen.

3.3 Datenflussdiagramm

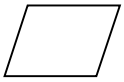
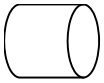
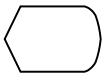
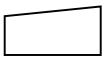
Ein **Datenflussplan** ist eine grafische Übersicht, welche die Programme und Daten, die zu einer Gesamtaufgabe gehören, miteinander verbindet. Er zeigt, welche Teile eines Programms von den Daten durchlaufen werden und welche Art der Bearbeitung innerhalb der Programme vorgenommen wird.

Ein Datenflussplan besitzt ähnliche Symbole wie ein Programmablaufplan. Zusätzliche Sinnbilder werden vor allem für die Daten eingeführt.



Datenflussplan

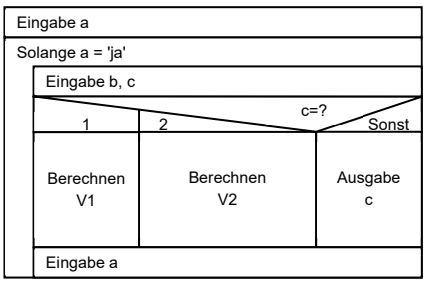
Symbole von Datenflussplänen

	Daten allgemein
	Daten auf einem Schriftstück (z. B. Druckliste)
	Daten auf einer Karte (z. B. Loch- oder Magnetkarte)
	Daten auf einem Speicher mit ausschließlich nacheinander zu verarbeitendem Zugriff (sequenziell), z. B. ein Magnetband
	Daten auf einem Speicher mit direktem Zugriff (wahlweise), z. B. eine Festplatte
	Maschinell erzeugte optische oder akustische Daten, z. B. die Bildschirmausgabe
	Manuelle Eingabe

3.4 Struktogramme

Struktogramme wurden 1973 von I. Nassi und B. Shneiderman als Darstellungsmittel für einen Algorithmus für den strukturierten Programmentwurf entwickelt. Sie sind in der DIN 66261 genormt.

Ein Struktogramm ist die grafische Darstellung eines Programmablaufs in Form eines geschlossenen Blocks, der entsprechend den einzelnen logischen Grundstrukturen in verschiedene untergeordnete Blöcke aufgeteilt werden kann. Struktogramme setzen sich aus verschiedenen Symbolen für die verschiedenen Operationsarten zusammen, die von oben nach unten betrachtet werden (top-down).



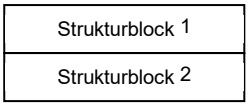
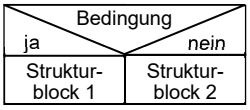
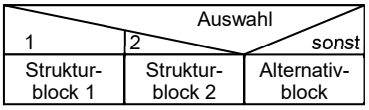
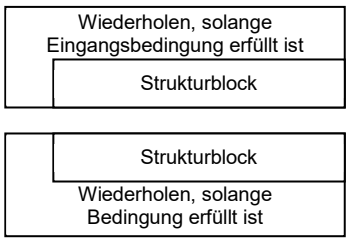
Nassi-Shneiderman-Struktogramm

Erläuterung des obigen Struktogramms

Zuerst wird der Wert `a` eingegeben. Gibt der Benutzer "`ja`" ein, wird er nach den Werten für die Variablen `b` und `c` gefragt. Die nachfolgende Verzweigung des Programms richtet sich nach dem Wert der Variablen `c`, also nach der Eingabe des Anwenders. Ist `c=1`, wird `V1` berechnet, ist `c=2`, wird `V2` berechnet, ansonsten wird der Wert `c` ausgegeben. Dies wird so lange durchgeführt, bis der Anwender bei der 1. Abfrage nicht "`ja`" eingibt.

Symbole von Struktogrammen

Ein Programmbaustein aus mehreren logisch zusammengehörenden Befehlen wird als Strukturblock bezeichnet. Ein einzelner Befehl heißt Elementarblock. Zur Darstellung von Programmabläufen verwendet man folgende Sinnbilder:

	**Verarbeitung (Prozess)** Mit dem Verarbeitungssymbol werden die Struktur- und Elementarblöcke dargestellt, die Ein- und Ausgabebefehle, Berechnungen und Unterprogrammaufrufe enthalten.
	**Folge (Sequenz)** Folgen mit zwei oder mehreren Arbeitsschritten werden durch aneinandergereihte Strukturblöcke dargestellt.
	**Alternative (Verzweigung)** Wird im Programmablauf eine Bedingung gestellt, wird dies mit dem Symbol „Alternative“ dargestellt. Ist die Bedingung erfüllt, wird der Strukturblock 1 ausgeführt, sonst Strukturblock 2.
	**Fallauswahl (Mehrfachverzweigung)** Eine mehrfache Bedingung im Programmablauf wird mit diesem Symbol dargestellt. Es wird kontrolliert, welche Auswahl vorgenommen wurde, und in den entsprechenden Strukturblock verzweigt. Trifft keine der Bedingungen zu, wird der Alternativblock ausgeführt.
	**Wiederholung (Schleife)** Diese Wiederholungssymbole dienen der Darstellung von Anweisungen in Schleifen. Der Anweisungsblock (Strukturblock) wird so lange von Neuem ausgeführt, bis die angegebene Bedingung erfüllt ist. Oben: kopfgesteuerte (abweisende) Schleife (while-Schleife oder Iteration) Unten: fußgesteuerte (annehmende) Schleife (do-while-Schleife)

Nach DIN 66261 sind insgesamt elf Symbole in Struktogrammen einsetzbar.

Die grafische Darstellung von linearen Kontrollstrukturen ist im Struktogramm optimal, da keine Sprünge darstellbar sind. Der zeichnerische Aufwand wird durch den Einsatz von Struktogramm-Generatoren begrenzt. Als besonders vorteilhaft ist zu werten, dass Alternativen nebeneinander angeordnet sind und so klare Linien des Programmablaufs erkennbar sind.

Bei Struktogrammen lässt sich die Methode der schrittweisen Verfeinerung anwenden, statt der konkreten Anweisungen wird nur eine grobe Beschreibung eingetragen (z. B. Datensatz verarbeiten). Für solch ein Rechteck wird dann wiederum ein Struktogramm entworfen, welches die Einzelheiten darstellt.

3.5 Pseudocode

Der Pseudocode ist eine halbformale, textuelle Beschreibung eines Programmablaufs, die sich an höhere Programmiersprachen anlehnt. Der Pseudocode ist nicht genormt wie Struktogramme oder Programmablaufpläne. Für Kontrollstrukturen werden meist ähnliche Worte verwendet, wie sie in der Syntax (vgl. Abschnitt 7.1) einer Programmiersprache vorkommen:

`if...then...else...end if, while...end while`

Für die Anweisungen werden entweder verbale Beschreibungen verwendet (z. B.: `erhöhe i um 1`) oder Anweisungen, die einer Programmiersprache ähneln (z. B.: `i := i + 1`). Im Pseudocode können auch Variablen- und Konstantendeklarationen enthalten sein.

```
begin
BetragPruefen
   Eingabe(a);
   Eingabe(b);
   if a >= b then
     Ausgabe(a);
   else
   Ausgabe(b);
   end if
 end BetragPruefen
```

Pseudocode

3.6 Entscheidungstabellen

Ein Hilfsmittel zur Darstellung von logischen Verknüpfungen sind Entscheidungstabellen (ET). Darin werden mehrere Bedingungen aufgeführt und die daraus resultierenden Aktionen festgehalten. Die Entscheidungstabellen sind leicht verständlich und können für Absprachen zwischen Programmierern und Anwendern bezüglich der Programmlogik verwendet werden.

Programmablaufpläne oder Struktogramme eignen sich für eine derartige Darstellung weniger, da mit der Anzahl der Bedingungen auch die Anzahl der Verzweigungen steigt, sodass die Pläne sehr unübersichtlich werden.

Aufbau von Entscheidungstabellen

Eine Entscheidungstabelle gibt eine Wenn-Dann-Beziehung wieder.

WENN eine bestimmte Bedingung erfüllt ist,
DANN ist eine bestimmte Aktion auszuführen.

Entscheidungstabellen werden in vier Bereiche eingeteilt. Die Bedingungen werden links oben formuliert. Die Aktionen werden darunter (links unten) aufgeführt. Im Bedingungsanzeigerteil wird die Kombination möglicher Wahrheitswerte eingetragen, und die entsprechend auszuführenden Aktionen werden im Aktionsanzeiger markiert.

Name der ET	Regelnummern
Bedingungen (WENN...)	Bedingungsanzeiger
Aktionen (DANN...)	Aktionsanzeiger

Die nachfolgende Abbildung zeigt Ihnen beispielhaft den schematischen Aufbau einer Entscheidungstabelle. Die Entscheidungsregeln bestehen aus Kombinationen von vier Bedingungen und drei Aktionen. Die im Anzeigerteil erfolgten Eintragungen haben folgende Bedeutung:

j = ja	Bedingung ist erfüllt
n = nein	Bedingung ist nicht erfüllt
x	Aktion wird ausgeführt
-	Aktion wird nicht ausgeführt, Bedingung ist für die Entscheidung nicht relevant

	ET: ET-Name	R1	R2	R3	R4	...
Bedingungsteil WENN	Bedingung 1	-	j	-	j	
	Bedingung 2	j	-	-	-	
	Bedingung 3	j	n	j	-	
	Bedingung 4	-	n	n	-	
Aktionsteil DANN	Aktion 1	-	x	-	-	
	Aktion 2	x	-	-	x	
	Aktion 3	x	x	x	-	

Schematische Darstellung einer Entscheidungstabelle

Die Entscheidungsregeln in der Tabelle sind spaltenweise zu lesen, z. B. für die Spalte R1:

WENN Bedingung 2 erfüllt ist und
Bedingung 3 erfüllt ist
(unabhängig davon, ob Bedingung 1 oder 4 erfüllt ist),

DANN führe Aktion 2 aus und
führe Aktion 3 aus.

Beispiel: Aufpreise bei Produktvarianten

	ET: Aufpreise	R1	R2	R3
Bedingungsteil *WENN*	Standardverpackung	j	n	n
	Geschenkverpackung	-	j	j
	Menge < 30	-	j	n
	Menge >= 30	-	n	j
Aktionsteil *DANN*	Aufpreis	-	x	-
	kein Aufpreis	x	-	x

Ein Unternehmen liefert ein Produkt in verschiedenen Verpackungen.

R1: Wird ein Produkt mit der standardmäßigen Verpackung gewünscht, zahlt der Kunde keinen Aufpreis.

R2: Bestellt er weniger als 30 Stück mit einer speziellen Geschenkverpackung, zahlt er einen Aufpreis.

R3: Bei mehr als 30 Stück entfällt dieser Aufpreis wieder.

3.7 Übung

Struktogramm und Entscheidungstabelle erstellen

Übungsdatei: -- **Ergebnisdatei: *uebung03.pdf***

1. Entwerfen Sie ein Struktogramm, um nach der Eingabe einer Schulnote die Bewertung in Textform auszugeben. Wird eine Note größer als 6 angegeben, soll eine Meldung ausgegeben werden. Die Bewertungen lauten:

   1: Sehr gut 3: Befriedigend 5: Mangelhaft
   2: Gut 4: Ausreichend 6: Ungenügend

2. Entwickeln Sie eine Entscheidungstabelle für die Versandbedingungen eines Versandhauses. Dieses berechnet bis zu einem Einkaufspreis von 150,00 EUR Versandkosten in Höhe von 8,00 EUR; ab einem Einkaufspreis von 300,00 EUR wird zusätzlich ein Gratisgeschenk verschickt.

4 Werkzeuge der Softwareentwicklung

In diesem Kapitel erfahren Sie

- welche Werkzeuge Sie beim Erstellen eines Programms benötigen
- was ein Compiler, Interpreter, Binder, Lader ist
- welchen Grundaufbau ein Programm hat
- wie Sie ein Programm kompilieren und ausführen
- wie Sie ein Skript interpretieren lassen

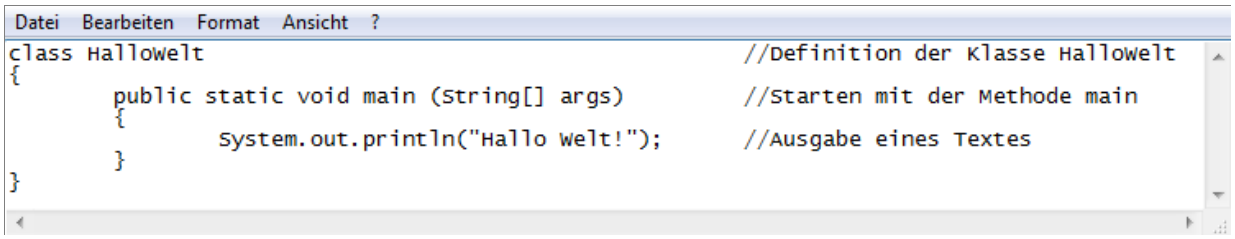
Voraussetzungen

- Grundlegende Computerkenntnisse

4.1 Programme erstellen

Quelltext eingeben

Bei der Programmierung wird ein Quelltext auch Code oder Quellcode genannt, in einer bestimmten Programmiersprache erstellt. Zur Eingabe dient ein Editor. Ein Editor ist eine Software zur Bearbeitung von Texten. Im Unterschied zu einer Textverarbeitung oder einem Layout-Programm fügt ein Editor keine Formatierungsanweisungen in den Text ein, um ihn z. B. in Absätze zu gliedern.

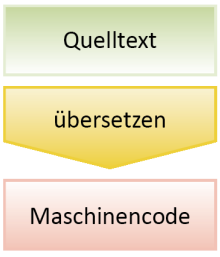


Editor (Windows) mit Quelltext (Java)

Editoren sind auf allen Computerplattformen verfügbar und besitzen unterschiedliche Funktionsausstattungen. Viele Editoren können z. B. programmiersprachenspezifische Sprachelemente hervorheben (vgl. Kapitel 7).

Entwicklungsumgebungen für eine bestimmte Programmiersprache besitzen einen eigenen Editor, der eine Hervorhebung der Syntax und Hilfestellungen für verschiedene Sprachelemente bietet (vgl. Abschnitt 5.3).

Programme übersetzen



Der Quelltext kann von einem Computer nicht direkt ausgeführt werden. Der Computer benötigt eine sogenannte Maschinensprache (Maschinencode). Deshalb muss jeder Quelltext in den rechnertypspezifischen Maschinencode übersetzt werden.

Abhängig von der verwendeten Programmiersprache lassen sich drei grundlegende Verfahren für den Übersetzungsvorgang unterscheiden.

- Vollständige Übersetzung **bei der Programmausführung** mit einem Interpreter, z. B. bei der Programmiersprache BASIC oder bei der Skriptsprache JavaScript
- Vollständige Übersetzung **bei der Programmerstellung** mit einem Compiler, z. B. bei der Programmiersprache C
- **Zweistufige Übersetzung**, z. B. bei der Programmiersprache Java:
  - Übersetzung in plattformunabhängigen Bytecode bei der Programmerstellung (Compiler)
  - Übersetzung des plattformunabhängigen Bytecode bei der Programmausführung (Interpreter)

4.2 Konzepte zur Übersetzung

Interpreter – Übersetzung während der Programmausführung

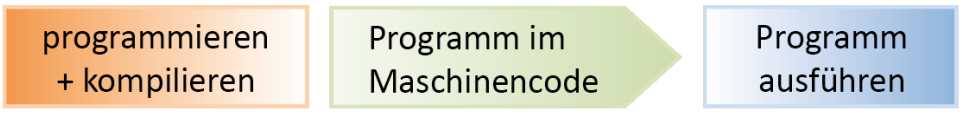
Während der Laufzeit eines Programms oder Skripts (Programmausführung) übersetzt ein **Interpreter** den Quelltext in Maschinencode. Interpreter übersetzen ein Programm zeilenweise, wobei erst dann jede Zeile einer Syntaxprüfung unterzogen wird.

- Sind die Befehle in der Zeile korrekt, werden sie vom Interpreter in Anweisungen für den Prozessor übersetzt und ausgeführt. Anschließend wird die nächste Zeile geprüft usw.
- Tritt während des Programmablaufs ein Syntaxfehler auf, wird der Übersetzungsvorgang abgebrochen und eine entsprechende Fehlermeldung ausgegeben. Allerdings ist das Programm schon bis zu der Fehlerstelle ausgeführt worden.

Bei jedem Übersetzungsvorgang wird die Fehlerüberprüfung automatisch durchgeführt. Interpretierte Programme werden aufgrund der Prüfung zur Laufzeit langsamer ausgeführt als vor der Laufzeit übersetzte Programme.

Mittlerweile gibt es Programmiersprachen, deren Interpreter bereits vor dem Start eines Programms die Syntax kontrollieren und verschiedene Optimierungen vornehmen. Diese Programme werden schneller abgearbeitet, da in diesem Fall keine Überprüfung während der Laufzeit erfolgen muss. Ein Vertreter dieser Art ist die Programmiersprache Perl, die zum Beispiel zum Erstellen von dynamischen Internetseiten benutzt wird.

Compiler – Quelltext vor der Laufzeit übersetzen

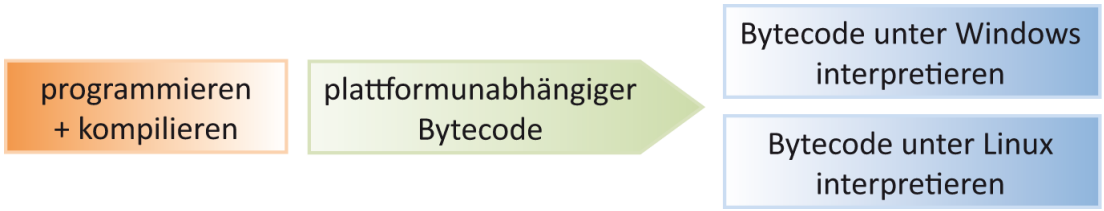


Um ein Programm schnell ausführen zu können, muss es **vor der Laufzeit** in Maschinencode umgewandelt werden, den der Prozessor ausführen kann. Dies übernimmt ein **Compiler**. Dieser Vorgang wird Kompilieren genannt.

Der Compiler übersetzt Programme einer Programmiersprache in einen Maschinencode, ohne jedoch die Befehle dabei schon auszuführen. Der zu übersetzende Code wird auch **Quellprogramm** (Quelltext) und das übersetzte Programm **Zielprogramm** oder einfach nur **Programm** genannt. Bei der Übersetzung eines Programms durch einen Compiler wird jede Anweisung auf die korrekte Syntax der Befehle geprüft. Im Fehlerfall wird der gesamte Vorgang des Übersetzens abgebrochen und der Fehler muss beseitigt werden, bevor das Zielprogramm erstellt werden kann.

Assembler	Ein Übersetzer von einer maschinenorientierten Programmiersprache in eine Maschinensprache wird als Assembler bezeichnet. Einige Compiler höherer Programmiersprachen haben die Möglichkeit, Assemblercode anstatt Maschinencode zu erzeugen.
Präprozessor	Ein Präprozessor modifiziert vor dem Kompilieren den Quellcode. Mit seiner Hilfe werden weitere Quelltextdateien eingebunden, Teile des Codes ignoriert oder ersetzt.
Compiler	Ein Compiler übersetzt ein Programm in der Regel direkt in die Maschinensprache des Zielsystems. Dabei werden verschiedene Optimierungen durchgeführt. Je nach Programmiersprache und Programmierumgebung können mehrere Compilerdurchläufe zur Erzeugung des eigentlichen Programms notwendig sein.
Cross-Compiler	Sie nehmen eine Sonderstellung ein, denn sie erzeugen einen Maschinencode für eine andere Computerplattform als die, auf der sie ausgeführt werden.

Zweistufige Übersetzung – Übersetzung mit Zwischencode

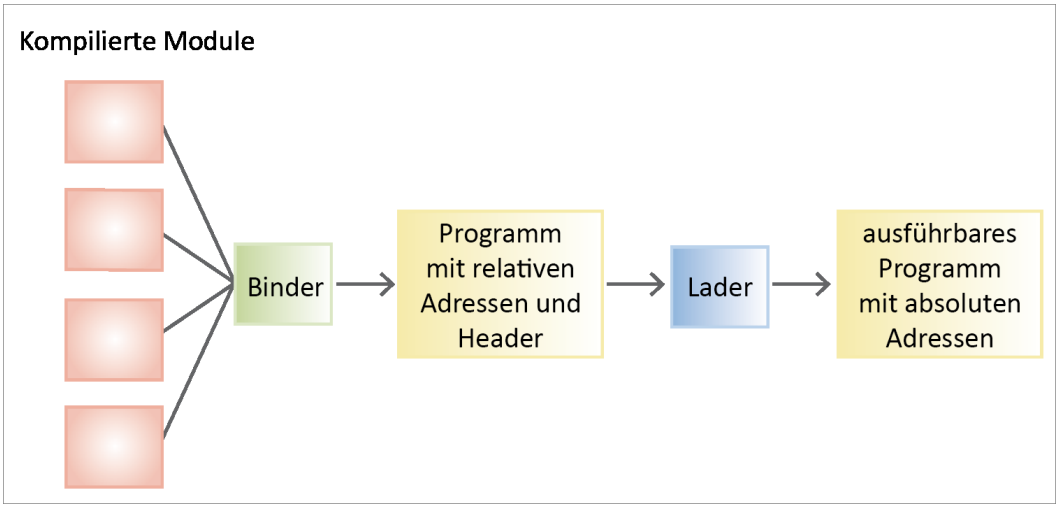


Ein Nachteil bei kompilierten Programmen ist, dass diese Programme maschinenabhängig sind und somit nicht auf jeder Computerplattform, z. B. Linux und Windows, ausgeführt werden können. Java oder die .NET-Plattform gehen in dieser Hinsicht einen anderen Weg. Der Programmcode in Java wird nicht zu einem ausführbaren Programm, sondern in einen **Zwischencode**, den sogenannten **Bytecode**, kompiliert (1. Schritt). Dieser Code ist für alle Plattformen gleich und kann mithilfe des entsprechenden **plattformspezifischen Interpreters** (2. Schritt) auf der jeweiligen Plattform ausgeführt werden. Java-Interpreter werden auch virtuelle Maschinen (JVM) genannt. Bei der .NET-Plattform geht man einen etwas anderen Weg, aber auch hier kommt ein Zwischenformat (Common Intermediate Language) zum Einsatz. Es werden verschiedene Programmiersprachen zunächst in die Common Intermediate Language übersetzt, bevor sie von der Laufzeitumgebung ausgeführt werden.

Binden und Laden

In der letzten Phase der Übersetzung werden die Teile des Codes aus der getrennten Übersetzung der einzelnen Module zum endgültigen ausführbaren Programm zusammengesetzt.

Binder (Linker)	Die einzeln kompilierten Module eines Programms sind noch nicht lauffähig, da die verwendeten Adressen von Funktionen, Prozeduren usw. angepasst werden müssen. Ein Binder verbindet diese einzelnen Programm-Module zu einem vollständigen Programm (relative Adressauflösung innerhalb der Module und Anfügen eines Programmheaders). Er zählt zu den Bibliotheksverwaltungsprogrammen, gehört aber inzwischen zu jedem Betriebssystem, das über Compiler verfügt, bzw. zur verwendeten Programmiersprache.
Lader	Ein Lader (oder auch Ladeprogramm) hat die Aufgabe, ein übersetztes und gebundenes Programm an eine Ladeadresse im Arbeitsspeicher zu bringen und alle Adressangaben dieses Programms auf diese auszurichten, d. h., relative Adressen in absolute Adressen umzusetzen. Das Programm kann von dort ausgeführt werden.
Bindelader	Über einen Bindelader werden die zwei Vorgänge zusammengefasst. Die einzelnen Module werden zu einem vollständigen Programm zusammengesetzt, anschließend wird das Programm in den Arbeitsspeicher geladen.



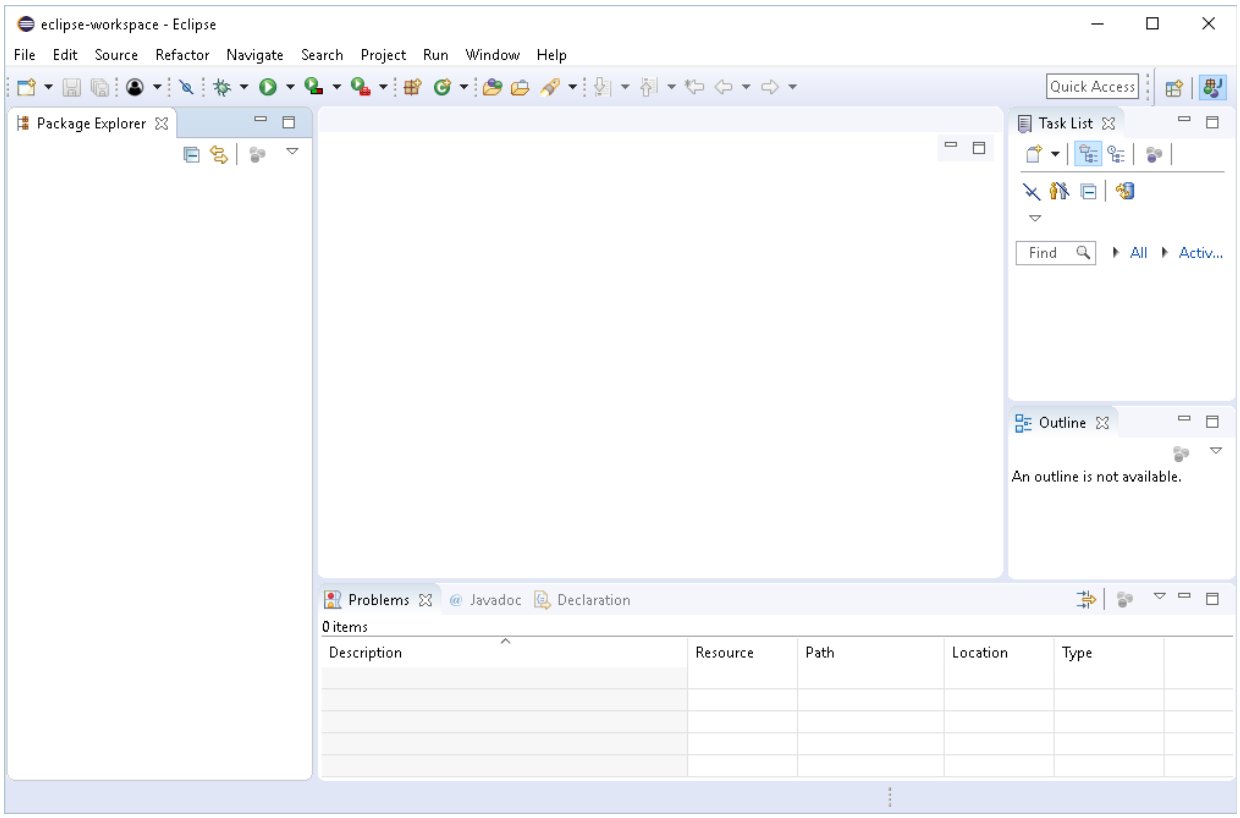
Ablauf des Bindens und Ladens der kompilierten Module

Ein Programm besitzt vor seiner Ausführung nur relative Adressangaben, d. h., alle Positionen innerhalb des Programms werden relativ zu einer Basisadresse angegeben. Wird ein Programm ausgeführt, werden diese relativen Adressangaben durch absolute ersetzt (wo sich das Programm im Hauptspeicher befindet).

Der Vorgang den Bindens und Ladens wird in modernen Entwicklungsumgebungen meist automatisch vorgenommen und bleibt dem Programmierer weitgehend verborgen.

4.3 Entwicklungsumgebungen

Für viele Programmiersprachen werden mächtige Werkzeuge für die Programmierung zur Verfügung gestellt, die **(integrierte) Entwicklungsumgebungen** (IDE, engl. Integrated Development Environment) oder auch Entwicklungstools genannt werden. Meist beinhalten sie unter einer grafischen Bedieneroberfläche den Quelltexteditor, eine Projektverwaltung, einen Compiler und einen Debugger zum einfachen Erstellen und Testen von Programmen. Professionelle aktuelle Entwicklungsumgebungen sind z. B. Microsoft Visual Studio für verschiedene Programmiersprachen wie Visual Basic, C#, C++ oder JScript und Eclipse für Java oder diverse andere Sprachen (über Plugins ist Eclipse erweiterbar).



Entwicklungsumgebung Eclipse

Im Buch wird bei Programm-Beispielen meist mit Java oder JavaScript gearbeitet. Diese Arbeit können Sie z. B. mit **Eclipse** leichter erledigen. Die IDE liegt allerdings nur in Englisch vor. Unter *http://www.eclipse.org* können Sie die aktuellste Version von Eclipse kostenlos laden. Es gibt verschiedene Versionen von Eclipse. Am besten eignet sich für dieses Buch die Version für Java-Entwickler. Achten Sie auf die richtige Variante für Ihr Betriebssystem.

Wenn Sie das Programm aus dem Internet laden, können Sie ein gepacktes Archiv erhalten, das Sie extrahieren müssen – Eclipse muss in dem Fall nicht installiert werden und ist nach dem Extrahieren unmittelbar startfähig. Neue Versionen von Eclipse bringen jedoch ein Setup-Programm mit sich, das Ihnen einen einfachen Assistenten bereitstellt.

Früher musste zusätzlich ein Java Development Kit (JDK) vorhanden sein (siehe Abschnitt 4.6), in neuen Versionen bringt Eclipse bereits alles mit, was zum Entwickeln und Ausführen von Java-Programmen notwendig ist. Eine Java-Laufzeitumgebung (JRE – Java Runtime Environment) muss auf dem Rechner vorhanden sein. Das ist aber in der Regel der Fall, da viele Programme Java nutzen. Das JDK bringt aber auch eine JRE mit sich (siehe Abschnitt 4.6).

Integrierte Entwicklungsumgebungen bieten eine ganze Reihe an Unterstützung für die Programmierung.

Editoren

Integrierte Quelltexteditoren bieten umfangreiche Leistungsmerkmale, wie zum Beispiel

- die Hervorhebung der Syntaxstruktur durch Farben oder durch textliche Auszeichnungen,
- eine integrierte Programmierhilfe sowie
- eine automatische Code-Vervollständigung.

Mithilfe einer Projektverwaltung können größere Programme komfortabel verwaltet werden. Teilweise sind Versionsverwaltungssysteme und die Unterstützung für die Entwicklung in einem Team integriert.

Debugger

Logische Fehler sind Fehler, die bei der Programmausführung falsche Ergebnisse liefern. Da die Syntax des Programms fehlerfrei ist und auch bei der Programmausführung selbst kein Fehler auftritt, erkennt weder der Compiler den Fehler noch können Sie den Fehler durch einfaches Starten des Programms ermitteln. Der Fehler wird durch eine falsche Programmlogik oder Eingabefehler hervorgerufen. Um diese Fehler aufzufinden, können Sie einen Debugger verwenden.

Ein Debugger ist ein Werkzeug mit dessen Hilfe Sie Fehler suchen können. Mit einem Debugger können Sie beispielsweise die Programmausführung an einer definierten Stelle unterbrechen, indem Sie sogenannte **Haltepunkte** (Breakpoints) im Programm setzen. An den Haltepunkten können Sie z. B. Überprüfungen im Quellcode vornehmen, den Quellcode ändern, den Quellcode Zeile für Zeile weiterlaufen lassen und dabei bestimmte Teile des Quellcodes beobachten.

Beispiel: Logische Fehler

Sie haben ein Programm geschrieben, das einen Ausdruck berechnen soll. Die Progammausführung liefert jedoch nicht das gewünschte Ergebnis. Der Fehler wird hier durch fehlende Klammern hervorgerufen. Da das Programm syntaktisch richtig ist, wird beim Übersetzen keine Fehlermeldung erzeugt.

Statt

```
Ergebnis = (a+b)*(a/(b-a));
```

Gewünschte Klammerung

haben Sie

```
Ergebnis = (a+b)*(a/b-a);
```

Falsche Klammerung

eingegeben. Da bei der Reihenfolge der Berechnung die mathematischen Grundregeln (Punkt vor Strich) gelten, wird in der zweiten Klammer zuerst der Bruch `a/b` berechnet und vom Ergebnis wird `a` subtrahiert. In der gewünschten Berechnung wird durch Klammerung zuerst die Berechnung der Differenz erzwungen, bevor der Bruch berechnet wird.

Mit dem Debugger können Sie die Werte von `a` und `b` auslesen. Durch die Auswertung der Berechnungen innerhalb der Klammern werden Sie den logischen Fehler schnell finden.

Führen Sie deshalb stets nach erfolgreicher Kompilierung und Ausführung eines Programms Tests mit Beispieldaten durch, um derartige Fehler auszuschließen und die Funktionsweise Ihres Programms zu überprüfen.

4.4 Standardbibliotheken

Für die meisten Programmiersprachen existiert eine Reihe von Standardbibliotheken. Diese Bibliotheken sind Sammlungen von bereits implementierten Funktionalitäten, die Sie in Ihren eigenen Programmen verwenden können, z. B. Grafikroutinen, mathematische Funktionen oder Funktionen für den Dateizugriff. Dazu müssen diese in das eigene Programm eingebunden werden. Bei der Verwendung von Bibliotheken kann der Vorteil der Wiederverwendbarkeit ausgenutzt werden.

- Der Programmcode ist einfacher zu warten.
- Funktionen können mehrfach verwendet werden.
- Es muss weniger Quellcode geschrieben werden.
- Die Fehleranfälligkeit des Programms wird verringert, da der wiederverwendete Quellcode bereits ausführlich getestet wurde.

Es liegen für die meisten Programmiersprachen sehr umfangreiche Bibliotheken für die verschiedenen Anwendungsgebiete vor.

4.5 Grundaufbau eines Programms am Beispiel Java

Jedes Programm, das Sie in einer Programmiersprache schreiben, hat ein spezielles Grundgerüst, das immer verwendet werden muss. Zur Erklärung des Grundaufbaus der Programme am Beispiel der Programmiersprache Java wird das zu Demonstrationszwecken sehr oft verwendete "Hallo Welt"-Programm angewendet. Dieses gibt auf Ihrem Bildschirm "Hallo Welt!" aus.

Eine Java-Anwendung erstellen

In Java sind alle Informationen, die zur Ausführung eines Programms notwendig sind, in Klassen (vgl. Kapitel 10) enthalten. Diese Klassen umfassen Attribute (Daten) und Methoden (Funktionalität). Die Methoden beinhalten die Anweisungen, die ausgeführt werden sollen.

Jede einzelne Klasse wird in einer eigenen Datei mit der Dateinamenerweiterung *.java* gespeichert. Die Datei erhält den Namen der Klasse als eigenen Namen. Im Beispiel wird die einzige Klasse unter dem Namen *HalloWelt.java* als Datei gespeichert. Die Klasse `HalloWelt` enthält eine Methode mit dem „Hauptprogramm“, die sogenannte **main-Methode**. Die main-Methode besteht aus einer einzigen Anweisung, die den Text "Hallo Welt!" auf dem Bildschirm ausgibt.

Beispiel: *HalloWelt.java*

Geben Sie mit einem Editor das folgende Beispielprogramm ein, und probieren Sie es aus.

Der Quelltext wird hier im Beispiel nur kurz beschrieben, eine ausführlichere Erläuterung der einzelnen Bestandteile finden Sie in den nachfolgenden Kapiteln.

```
①  class HalloWelt
②  {
③  public static void main(String[] args)
     {
④      System.out.println("Hallo Welt!");
     }
⑤  }
```

① Mit dem Schlüsselwort `class` wird eine Klassendefinition eingeleitet. Danach folgt der Name der Klasse. Der Name der Klasse wird genau so geschrieben wie der Name der Datei, in der sie sich befindet, jedoch ohne Dateinamenerweiterung. Im Beispiel lautet der Name der Klasse `HalloWelt` und der Name der Datei *HalloWelt.java*.

②,⑤ Innerhalb von geschweiften Klammern { } steht die Definition einer Klasse.

③ In der Klasse, die beim Programmstart aufgerufen wird, muss immer die `main`-Methode implementiert sein. Die `main`-Methode hat dabei folgendes Aussehen: `public static void main (String[] args)`.

④ Bei der Zeile `System.out.println("Hallo Welt!");` handelt es sich um den Aufruf einer standardmäßig vordefinierten Methode, die zur Ausgabe von Zeichen auf dem Bildschirm eingesetzt wird. Jede Java-Anweisung wird durch ein Semikolon ; beendet, damit der Compiler erkennt, wo der auszuführende Befehl zu Ende ist.

Speichern Sie den Quelltext unter dem Dateinamen *HalloWelt.java*. Erstellen Sie dazu ein Verzeichnis sowie Unterverzeichnisse, z. B. kap04, in die Sie die Beispiele für das jeweilige Kapitel speichern können.

4.6 Ein Java-Programm kompilieren und ausführen

Um ein Java-Programm übersetzen und ausführen zu können, benötigen Sie entweder eine integrierte Entwicklungsumgebung wie Eclipse oder Sie müssen das JDK (Java Development Kit oder auch Java SE Development Kit) installieren. Das JDK erhalten Sie kostenlos bei Oracle (*http://www.oracle.com/technetwork/java/javase/downloads/index.html*). Über den Hyperlink *Documentation* finden Sie zusätzlich Installationshinweise und Tipps zur Fehlersuche. Die Firma Oracle hat vor geraumer Zeit den Java-Erfinder Sun übernommen und ist mittlerweile für Java „verantwortlich“.

- Das JDK gibt es für verschiedene Betriebssysteme (Linux, MacOS, Solaris und Windows).
- Java und das JDK werden ständig aktualisiert. Installieren Sie entsprechend den Angaben die jeweils aktuellste Version.
- Installieren Sie das JDK nicht über eine bereits vorhandene Version. Der Installationsassistent wird Ihnen standardmäßig unterschiedliche Verzeichnisse vorschlagen.

Die Installation

Wenn Sie das JDK aus dem Internet laden, erhalten Sie eine Installationsdatei, die mit einem komfortablen Assistenten das JDK installieren lässt.

- Starten Sie das Installationsprogramm, und bestätigen Sie die Lizenzbestimmungen.

Im folgenden Dialogfenster werden die Komponenten angezeigt, die Sie installieren können. Bei Bedarf können Sie Komponenten, die Sie nicht installieren möchten, deaktivieren. Es wird empfohlen, wie vorgegeben, **alle** Komponenten zu installieren:

- Klicken Sie auf *Change,* falls Sie das Verzeichnis ändern möchten.
- Klicken Sie auf *Next*, um die ausgewählten Komponenten zu installieren.

Standardmäßig werden folgende Komponenten installiert:

- die eigentliche Entwicklungsumgebung und die integrierte Laufzeitumgebung zur Ausführung von Java-Programmen;
- Beispiele zur Demonstration (Quellcode);
- ein Großteil des Quellcodes von Klassen der Entwicklungsumgebung;
- die zur Weitergabe bestimmte Laufzeitumgebung.

- Beenden Sie die Installation, indem Sie auf *Finish* klicken.

Sofern Sie, wie standardmäßig vorgegeben, die Installation der Komponente *Public JRE* (Java Runtime Environment) gewählt haben, wird automatisch eine weitere Anwendung zur Installation der Laufzeitumgebung (JRE) gestartet. Eine Laufzeitumgebung ist bereits in der Entwicklungsumgebung integriert. Daher benötigen Sie das *Public JRE* nicht zwingend für die Entwicklung (das Erstellen und Ausführen) von Java-Programmen. Wenn Sie die Laufzeitumgebung jedoch zusammen mit Ihren Programmen weitergeben möchten, benötigen Sie aus lizenzrechtlichen Gründen diese spezielle Laufzeitumgebung (*Public JRE*).

Auch bei dieser Installation können Sie Komponenten auswählen und das Installationsverzeichnis anpassen.

Installation der Dokumentation

Zur Programmiersprache Java stellt Sun als Referenz eine HTML-basierte Dokumentation zur Verfügung. Die Dokumentationsdateien werden nicht automatisch mit dem JDK installiert. Sie müssen diese separat downloaden und einrichten. Die Dokumentationsdateien liegen in gepackter Form (ZIP) vor.

- Extrahieren Sie den Ordner *Docs* in einen Ordner Ihrer Wahl – etwa in das Installationsverzeichnis des JDK.

Die Programme und Tools des JDK sind befehlszeilenorientiert. Das bedeutet, dass Sie sie nicht einfach über die grafische Oberfläche anklicken und damit starten können. Sie müssen in einer **Kommandozeile** (auch Shell, Konsole oder Eingabeaufforderung genannt) über Tastaturbefehle aufgerufen werden. Dabei müssen bei der Übersetzung und Ausführung von Java-Programmen zusätzlich Parameter an den Aufruf angehängt werden. Insgesamt ist der Umgang mit befehlszeilenorientierten Tools unbequem, fehleranfällig und nicht ganz einfach. In der Praxis wird deshalb bei der Programmierung fast ausschließlich mit integrierten Entwicklungsumgebungen wie Eclipse gearbeitet. Die nachfolgenden Schritte mit der Eingabeaufforderung werden aus didaktischen Gründen durchgeführt. Die nötige Folge an Anweisungen zeigt, was sich exakt hinter der zweistufigen Übersetzung von Java verbirgt. Verwenden Sie später eine IDE, werden Sie im Vergleich sehen, was eine IDE automatisch leistet.

Ein Java-Programm mit dem Java-Compiler javac kompilieren

Mithilfe des Java-Compilers **javac** , der Bestandteil des JDK ist, übersetzen Sie Java-Programme in Bytecode, wobei gilt:

- Die Dateinamen der zu übersetzenden Java-Programme müssen mit der Endung **.java* versehen sein. Der Compiler unterscheidet zwischen Groß- und Kleinschreibung in den Dateinamen.
- Die neu erzeugte **.class*-Datei wird vom Compiler automatisch in denselben Ordner wie die dazugehörige **.java*-Datei gespeichert.

Die Beispielanwendung *HalloWelt.java* übersetzen

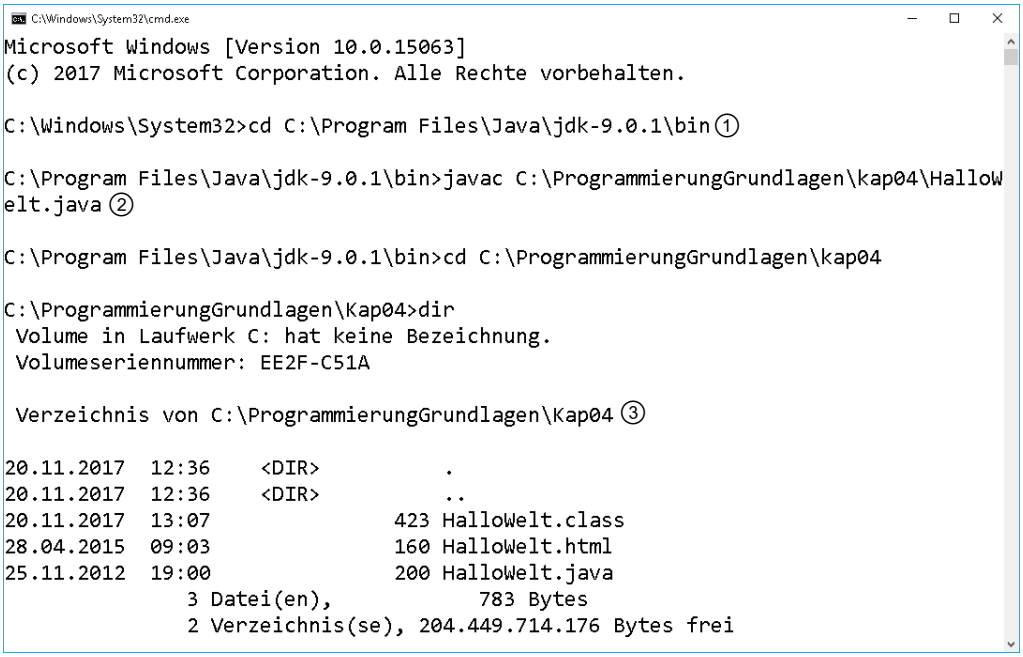
Wenn Sie ein Java-Programm über die Eingabeaufforderung übersetzen möchten, geben Sie den dazu notwendigen Befehl in die Kommandozeile ein.

- Öffnen Sie z. B. in Windows 10 die Eingabeaufforderung über *Start - Alle Programme - Zubehör - Eingabeaufforderung*. Wenn Sie Windows 10 mit dem Kacheldesktop verwenden, geben Sie in dem Suchdialog *cmd* ein und öffnen dann die angezeigte Eingabeaufforderung.

  Wechseln Sie in den `bin`-Ordner des Verzeichnis, in dem Sie das JDK installiert haben ①. Haben Sie es etwa in dem Verzeichnis `C:\Program Files\Java\jdk-9.0.1\` installiert, nutzen Sie den Befehl `cd C:\Program Files\Java\jdk-9.0.1\bin` in das Verzeichnis. Dabei müssen Sie die Pfadangaben an die Verzeichnisstrukturen anpassen, die bei Ihnen vorliegen. Haben Sie das JDK in einem anderen Verzeichnis installiert, passen Sie die Angaben entsprechend an.
- Übersetzen Sie das Programm HalloWelt.java mit dem Befehl `javac C:\ProgrammierungGrundlagen\kap04\HalloWelt.java` ②.

  Sie können für Ihre Quellcodes auch ein anderes Verzeichnis oder einen anderen Laufwerksbuchstaben nutzen und müssen dann auch diese Pfadangaben anpassen.
- Wechseln Sie mit dem `cd`-Befehl in das Verzeichnis mit den Quellcodes. Etwa so:

  cd `C:\ProgrammierungGrundlagen\kap04`
- Überprüfen Sie mit dem Befehl `dir` (Windows), ob eine Datei *HalloWelt.class* ③ erzeugt wurde.



Übersetzung eines Java-Programms mithilfe der Eingabeaufforderung

Ein Java-Programm mit dem Interpreter java ausführen

Vom Compiler übersetzte **.class*-Dateien sind noch nicht lauffähig. Die Dateien müssen vom Java-Interpreter **java** ausgeführt werden. Der Interpreter übernimmt für den Bytecode (**.class*-Datei) die Aufgabe, einen virtuellen Computer (virtuelle Maschine – Virtual Machine – VM) zu simulieren, der immer die gleichen Eigenschaften hat. Der Interpreter ist dann dafür verantwortlich, die Instruktionen im Bytecode in Maschinenanweisungen oder Anweisungen an das Betriebssystem auf dem betreffenden Rechner umzusetzen.

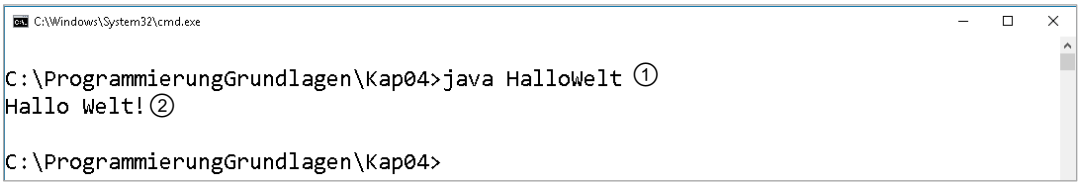
- Der Interpreter sucht nach einer Java-Klasse. Er unterscheidet dabei auch zwischen Groß- und Kleinschreibung in den Dateinamen. Beispielsweise sucht der Interpreter beim Starten der Beispieldatei *HalloWelt.class* nach einer Klasse `HalloWelt`, die sich in der Datei *HalloWelt.class* befinden muss.
- Solange die Anwendung ausgeführt wird, kehrt der Interpreter nicht zurück, d. h., Sie erhalten keinen Eingabeprompt.
- Die beim Aufruf des Interpreters angegebene **.class*-Datei muss eine Methode `public static void main(String[] args)` besitzen, damit sie der Interpreter starten kann.

Die Beispielanwendung *HalloWelt* aus der Eingabeaufforderung ausführen

Wenn Sie ein Java-Programm über die Eingabeaufforderung ausführen möchten, geben Sie den dazu notwendigen Befehl in die Kommandozeile ein.

- Um das Beispielprogramm *Hallo Welt* zu starten, wechseln Sie zum Beispiel in das Verzeichnis mit der **.class*-Datei. Etwa so: `cd C:\ProgrammierungGrundlagen\kap04`
- Geben Sie in die Kommandozeile der Eingabeaufforderung den Befehl `C:\"Program Files\Java\jdk-9.0.1\bin\java" HalloWelt` ein. Auch hier müssen Sie ggf. Ihre Pfadangaben anpassen. Aber oft genügt sogar einfach nur *`java HalloWelt`*, denn der Pfad zum Java-Interpreter ist in meist in den Suchpfad des Betriebssystems eingetragen ①.

  Die Anwendung gibt den Text "*Hallo Welt!*" ② in der folgenden Zeile aus.



Ausführung eines Java-Programms und Anzeige des Ergebnisses

4.7 Ein Java-Programm mit Eclipse erstellen, kompilieren und ausführen

Mächtigere Editoren bzw. IDEs wie Eclipse bieten Möglichkeiten, um die Befehle zur Ausführung des Compilers oder zum Start des Programms direkt aus der grafischen, integrierten Umgebung aufzurufen.

Wenn Sie Eclipse das erste Mal starten, müssen Sie meist die Frage beantworten, welchen **Workspace** Sie verwenden wollen? Das ist im Sinn von Eclipse ein übergeordnetes Verzeichnis, in dem Sie Ihre Programmieraufgaben in einzelnen Projekten verwalten. Bestätigen Sie die Grundeinstellung oder erstellen Sie ein individuelles Verzeichnis. Nach dem abgeschlossenen Start der IDE sehen Sie möglicherweise ein Fenster mit Willkommensmeldungen, das Sie schließen können.

In der Folge sollten Sie Eclipse in einer Ansicht sehen, die in der Regel für eine typische Java-Applikation besonders sinnvoll ist. Sie sehen eine Reihe integrierter kleiner Fenster, die zu Beginn weitgehend leer sind. Das wird die **Java-Perspektive** genannt.

Das Eclipse-Fenster mit allen Fenstersegmenten und Menüs und den dahinter liegenden Funktionalitäten heißt **Workbench**. Die einzelnen Fenstersegmente in Eclipse werden **Views** genannt. Sie können Sie individuell verschieben und auch schließen. Sollten sie geschlossen sein und Sie benötigen ein View erneut, können Sie das Fenstersegment unter dem Menüpunkt *Window* wieder öffnen.

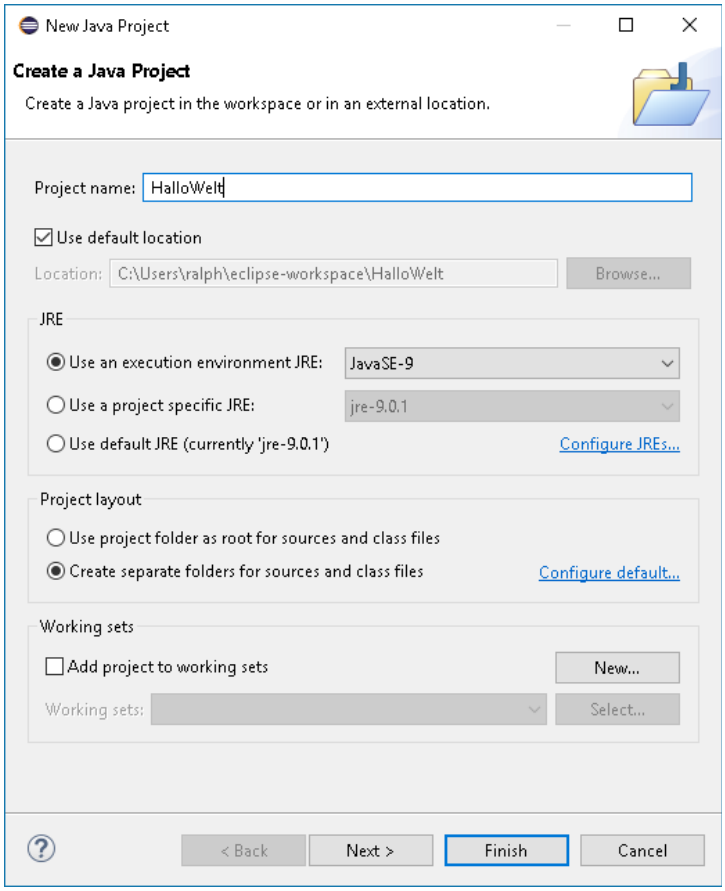
In Eclipse erstellen Sie in der Regel immer erst ein **Projekt** und nicht eine einzelne Quelltextdatei. Unter einem Projekt können Sie sich ein verwaltetes Verzeichnis vorstellen, in dem alle notwendigen Ressourcen zusammengefasst werden.

Grundsätzlich ist es zu empfehlen, eine eigene Verzeichnisstruktur für jedes Programm zu erstellen. Für die Arbeit mit diesem Buch erstellen Sie für jedes Kapitel ein eigenes Projekt.

- Klicken Sie auf *File - New - Java Project*, um ein Projekt in Eclipse zu erstellen.

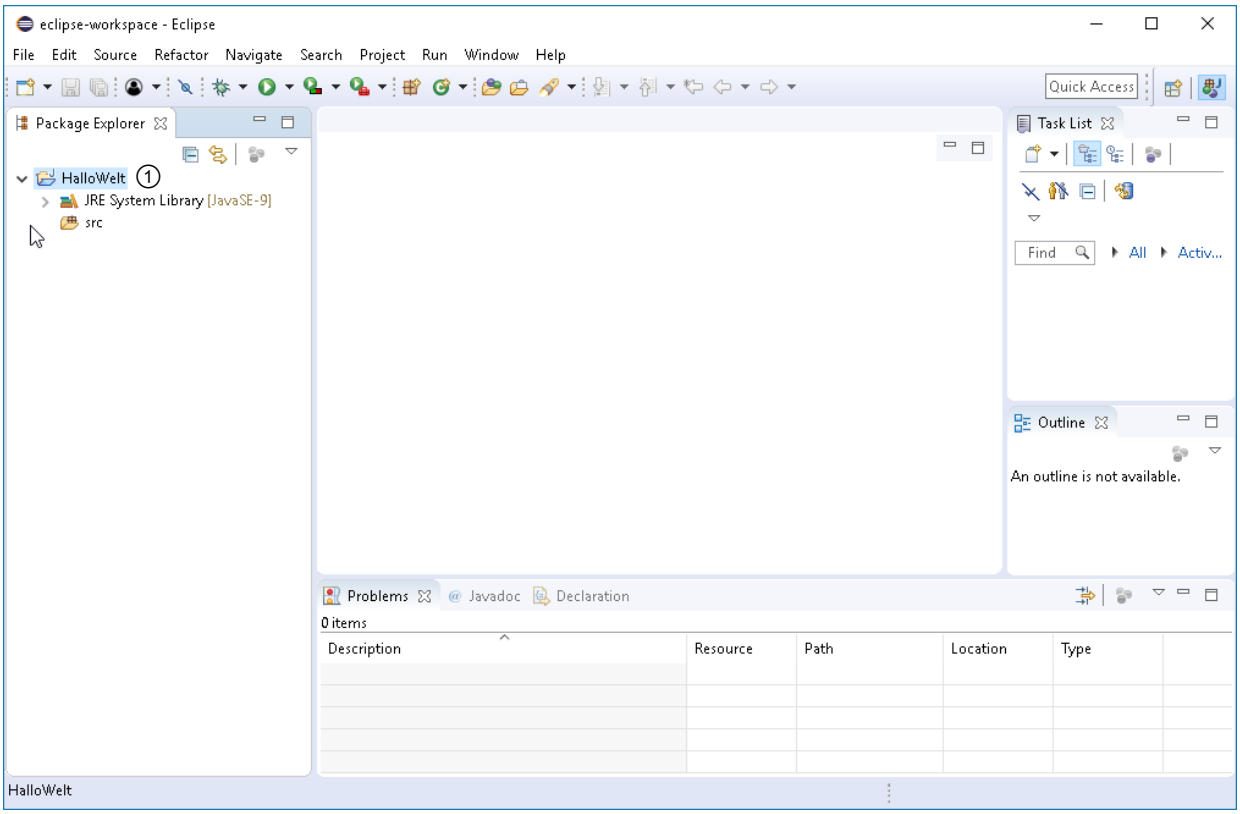
  Sie können im nachfolgenden Dialog ein Java-Projekt anlegen.
- Vergeben Sie für das Projekt in dem Dialog den Namen *HalloWelt*.

  Die weiteren Einstellungen lassen Sie in den Grundeinstellungen. Auch die weiteren Schritte des Assistenten, die Sie mit der Schaltfläche *Next >* aufrufen könnten, ignorieren Sie.
- Bestätigen Sie den Dialog mit *Finish*.



Ein Java-Projekt in Eclipse anlegen

Im sogenannten **Package Explorer** – standardmäßig links im Eclipse-Fenster – klicken Sie auf den Namen des Projektes ①, um die angezeigte Baumstruktur zu expandieren.



Projekt und seine Baumstruktur

In der Standardkonfiguration von Eclipse wird das Unterverzeichnis *src* angezeigt. Dort werden die Java-Quelltexte des Projektes gespeichert.

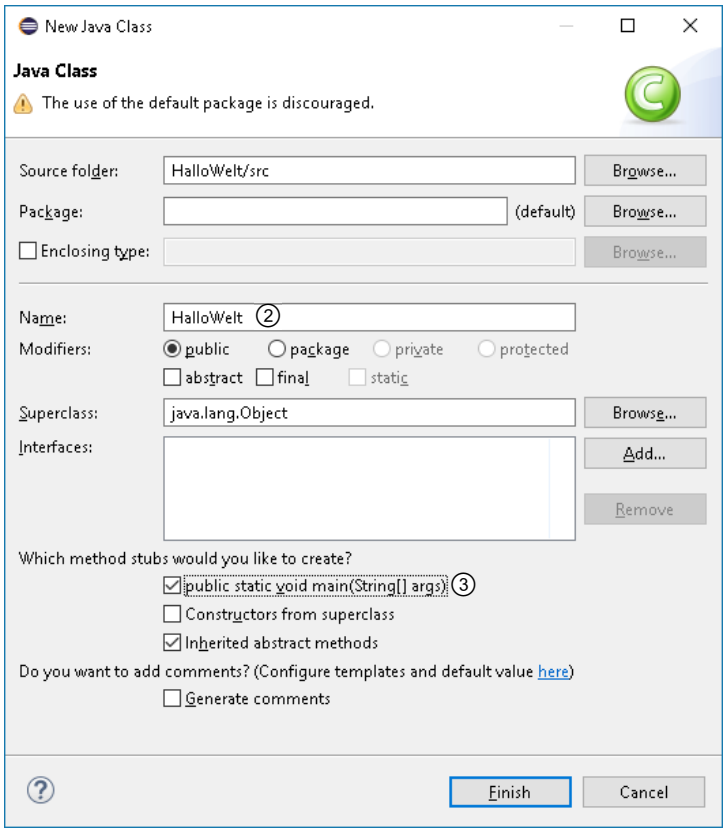
- Markieren Sie im Package Explorer das Projektverzeichnis und wählen Sie *File - New - Class*.

Folgende Angaben legen Sie zu Beginn fest:

- Den Namen der Klasse ② geben Sie an: *HalloWelt*. Damit legen Sie bei Eclipse immer auch den Namen der Java-Datei fest.
- Bei *public static void main(String[] args)* setzen Sie einen Haken ③.

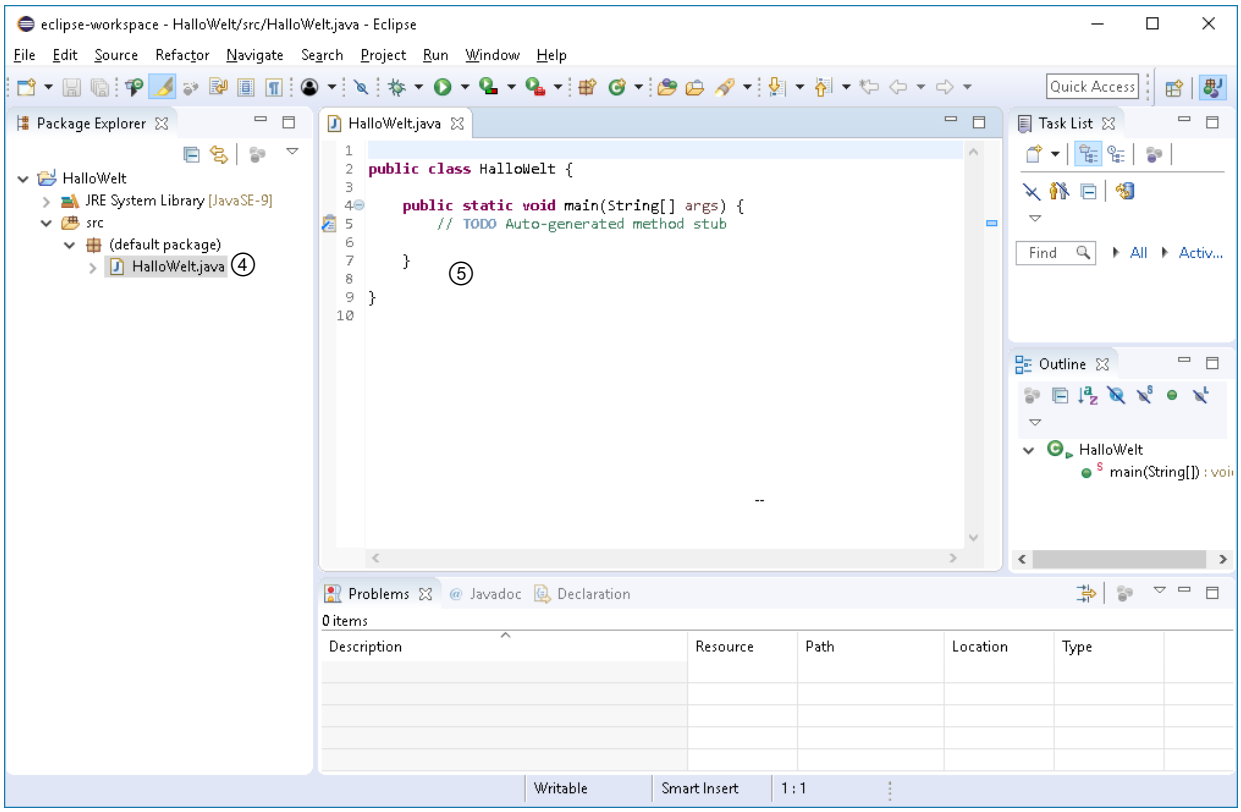
Danach bestätigen Sie den Dialog.

Die Ansicht Ihrer Arbeitsumgebung in Eclipse hat sich ein wenig geändert. Im Eclipse-Fenster stehen einige Struktur- und Statusinformationen von Ihrem neuen Programm. Auf der linken Seite im Package Explorer wurde im Verzeichnis *src* die Datei *HalloWelt.java* angelegt ④.



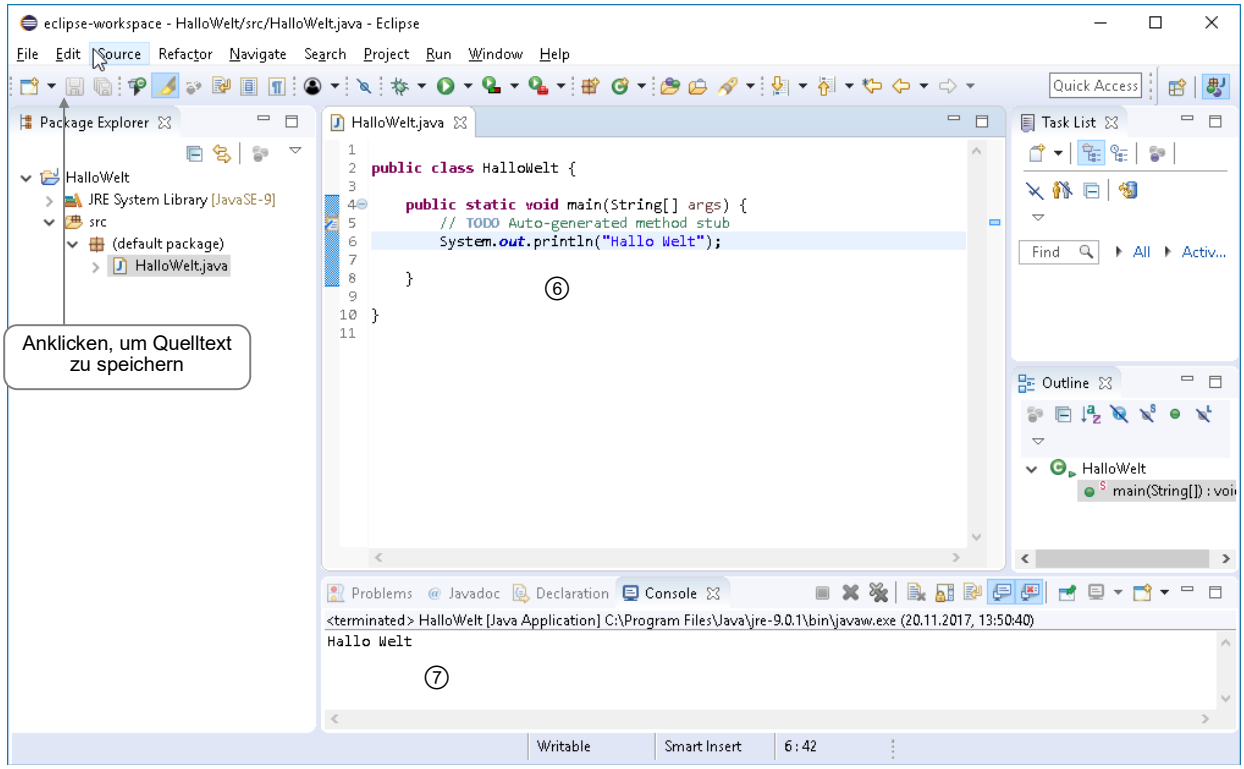
Java-Klasse in dem aktuellen Projekt anlegen

In der Mitte des Eclipse-Fensters sehen Sie den Editor, in dem Sie den Java-Quelltext eingeben ⑤. Dort hat Eclipse für Sie bereits einige Zeilen automatisch generiert. Mit einem einfachen Editor müssen Sie den Code Zeichen für Zeichen selbst eingeben.



Eclipse mit dem automatisch generierten Quellcode im integrierten Editor

Ergänzen Sie den generierten Quelltext entsprechend dem Beispiel aus Abschnitt 4.5 ⑥.



Ausführen eines Java-Programmes mit der IDE Eclipse

Speichern Sie zuerst den Quelltext. Verwenden Sie die Schaltfläche *Speichern* in der Symbolleiste oder Strg S. Der Quelltext wird unter *HalloWelt.java* gespeichert. Dieser Name wurde zuvor beim Anlegen der Java-Klasse im Projekt festgelegt.

Nächster Schritt ist die Kompilierung. Klicken Sie *Project - Build Project*. Aktivieren Sie die automatische Kompilierung über *Project - Build Automatically* , wird Eclipse immer dann, wenn es notwendig ist, automatisch das gesamte Projekt neu kompilieren.

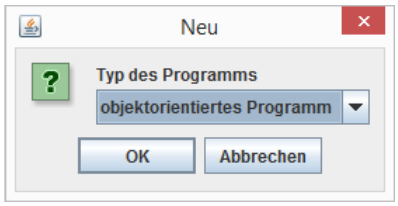
Mit *Run - Run As - Java Application* starten Sie den Java-Interpreter und übergeben die auszuführende Klasse als Parameter.

Damit führen Sie das Programm aus der IDE Eclipse heraus aus. Als Ergebnis des Aufrufs werden Sie innerhalb der IDE die Ausgabe des Programms sehen ⑦.

4.8 Ein Java-Programm Hamster-Simulator erstellen, kompilieren und ausführen

Wenn Sie den Hamster-Simulator verwenden, können Sie viele Schritte zum Erstellen, Übersetzen und Testen von Java-Quellcodes sparen, im Vergleich zu einem normalen Editor und dem JDK. Der Hamster-Simulator erstellt Ihnen auf Wunsch ein „Grundgerüst“ für eine Klasse, in der Sie dann nur den Code notieren müssen, den Sie gerade ausprobieren wollen.

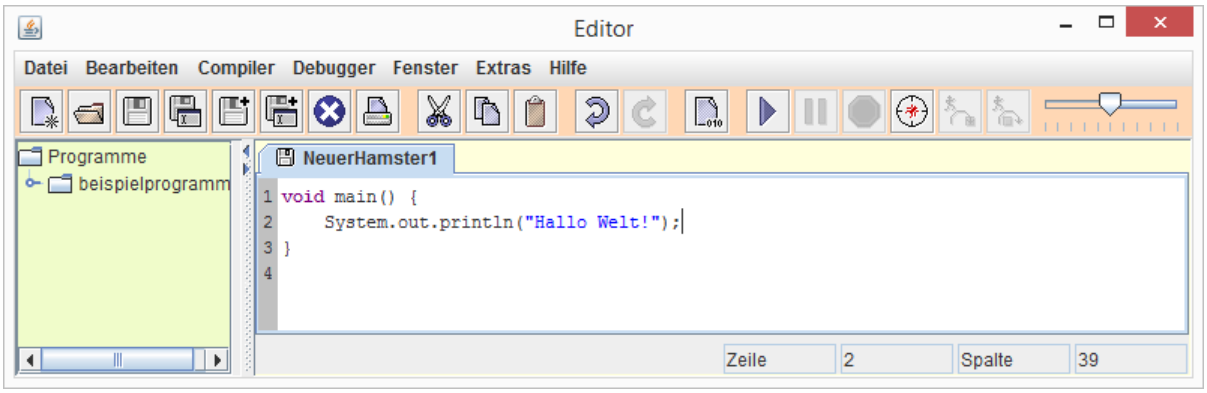
Den Hamster-Simulator können Sie direkt mit einen Doppelklick auf die Datei *hamstersimulator.bat* oder auch *hamstersimulator.jar* starten, wenn Sie die gepackte Datei extrahiert haben. Allerdings muss bereits ein JDK vorhanden sein, damit der Simulator funktioniert.



Eine Schablone für ein Java-Programm im Hamster-Simulator anlegen

Mit *Datei - Neu* legen Sie nach dem Start ein objektorientiertes Programm an.

In die generierte Schablone, von der Sie nur die `main()`-Methode sehen, ergänzen Sie den Code der main-Methode aus dem Hallo-Welt-Beispiel.



Hamster-Simulator mit ergänztem Quellcode in der main-Methode

Über die entsprechenden Menübefehle können Sie das Programm kompilieren oder debuggen.

4.9 Skripte interpretieren

Im WWW werden Sie oft mit Skripten konfrontiert. Skripte werden zur Übersetzung nicht kompiliert, sondern interpretiert. Aber auch Makros in Office-Programmen werden interpretiert. Das bedeutet, dass der Quelltext **zur Laufzeit** übersetzt wird.

Im Rahmen des WWW wird im Browser (clientseitig) fast ausschließlich **JavaScript** als Erweiterung von HTML verwendet. Nachfolgend wird beispielhaft JavaScript-Skript in eine Webseite eingebunden, interpretiert und ausgeführt.

Als Beispiel für Skripte kommt im Buch nur JavaScript zum Einsatz.

Aufbau einer Webseite

Die Basis von Webseiten ist HTML. HTML-Dateien selbst bestehen immer aus **reinem Text** (ASCII bzw. Unicode). Damit sind HTML-Dateien **plattformunabhängig** und Sie benötigen zur Erstellung nur einen einfachen Editor. Eine HTML-Datei muss jedoch im Browser interpretiert werden, um der HTML-Datei eine über reinen Text hinausgehende Bedeutung zu verleihen.

Webbrowser wurden von Anfang an so konzipiert, dass sie nach dem Prinzip der Fehlertoleranz arbeiten. Damit können auch syntaktisch fehlerhafte HTML-Dateien im Webbrowser weitgehend ausgewertet und optisch aufbereitet werden.

HTML verfügt im Gegensatz zu vollständigen Programmier- oder Skriptsprachen (wie etwa JavaScript) über keine Kontrollstrukturen, keine Variablen und keine Befehle im Sinne von Befehlsworten, die eine Aktion auslösen.

HTML-Anweisungen bestehen aus Steueranweisungen, die aus sogenannten **Tags** aufgebaut sind. Ein Tag beginnt immer mit einer geöffneten spitzen Klammer < und endet mit einer geschlossenen spitzen Klammer >. Im Inneren der beiden Klammern befindet sich der konkrete Befehl. Ein Tag sieht von der Struktur her immer so aus:

```
<Anweisung>
```

Ein Tag kann unter HTML sowohl klein als auch groß geschrieben werden. Auch Mischen von Groß- und Kleinbuchstaben ist erlaubt. Sie sollten Steueranweisungen aber ausschließlich **klein** schreiben.

In HTML gibt es zwei Formen von Tags:

- einen einleitenden Tag (Anfangs-Tag oder Beginn-Tag genannt)
- einen beendenden Tag (Abluss-Tag oder Ende-Tag genannt)

Das einleitende Tag eröffnet eine Anweisung, während das beendende Tag sie wieder ausschaltet. Beide Tags sehen fast identisch aus, außer einem vorangestellten Zeichen, mit dem der beendende Tag zusätzlich beginnt – dem Slash (Schrägstrich) /.

Wenn beide Tags angegeben werden, bilden Sie immer einen **Container**. Dies bedeutet, die im einleitenden Tag angegebene Anweisung (etwa eine Formatierung) wirkt sich auf sämtliche Dinge (Objekte) aus, die sich im Inneren des Containers befinden. Dies wird in vielen Fällen ein Text sein, es kann sich aber auch um eine Grafik oder andere Multimediaobjekte handeln. Der Ende-Tag hebt die Wirkung eines Anfangs-Tag auf.

Manche Tags sind erst dann sinnvoll einzusetzen, wenn sie genauer spezifiziert werden. **Parameter** beziehungsweise Attribute erweitern einen einleitenden HTML-Tag und spezifizieren damit genauer die Bedeutung von dem Tag.

Jede Webseite sollte eine Grundstruktur aus bestimmten Tags enthalten. Dieses **Grundgerüst** hat im Wesentlichen folgende Form:

```
① <html>
② <head>
③ <title></title>
  </head>

④ <body>
  </body>
⑤ </html>
```

① Mit dem Schlüsselwort `html` wird eine Webseite eingeleitet. Davor kann noch eine `Doctype`-Anweisung stehen, die hier nicht angegeben werden soll.

⑤ Das Abschluss-Tag bildet das Ende der Webseite.

②, ③ Der Kopf (head) einer Webseite enthält Meta-Informationen über die Webseite wie etwa den Titel. Hier werden in der Regel auch die Referenzen auf JavaScript-Dateien oder allgemein JavaScripts eingebunden.

④ Im Körper bzw. Rumpf (body) der Webseite werden die Tags und Inhalte notiert, die ein Besucher im Browser sehen soll.

JavaScript mit einer Webseite verbinden

Bei JavaScript handelt es sich um eine Interpretersprache. JavaScript-Code wird in Webseiten über spezielle Tags integriert und innerhalb des Webbrowsers von einem alternativen Interpreter interpretiert. Auch ein JavaScript-Interpreter ist Teil jedes modernen Webbrowsers. JavaScript ist auf Clientseite eine reine Erweiterung des HTML-Codes in Form von Klartext und nur als eingebundener Bestandteil eines HTML-Gerüsts zu verwenden. Dabei gibt es z. B. folgende Arten der Einbindung von JavaScript.

- Die **direkte Notation**. Im Buch wird diese Variante verwendet. Dazu werden JavaScript-Anweisungen einfach in die entsprechende HTML-Datei als Klartext eingefügt. Der Beginn eines Skripts muss dazu durch eine eigene HTML-Steueranweisung gekennzeichnet werden, die noch zu HTML gehört und die mit ihrem zugehörigen Abschluss-Tag einen Container für die Skriptanweisungen bildet. Über das `<script>`-Tag wird angegeben, dass all das, was in dem eingeschlossenen Container steht, ein Skript ist. Über den Parameter `type` können Sie festlegen, um welche Skriptsprache es sich handelt, bei JavaScript heißt die Angabe `text/javascript`.
- In der Praxis wird JavaScript meist in die Webseite in Form einer **externen Datei** eingebunden. Um eine externe JavaScript-Datei in eine Webseite einzubinden, müssen Sie das `<script>`-Tag um das Attribut `src` erweitern. Damit geben Sie die Adresse (Pfad und Dateiname) der externen JavaScript-Daten an.
- Historisch ist noch der Spezialfall von Bedeutung, bei dem Sie JavaScript-Anweisungen ohne expliziten `<script>`-Tag direkt in eine HTML-Anweisung für einen Hyperlink schreiben können. Dies ist die Inline-Referenz, die der Vollständigkeit halber erwähnt wird.

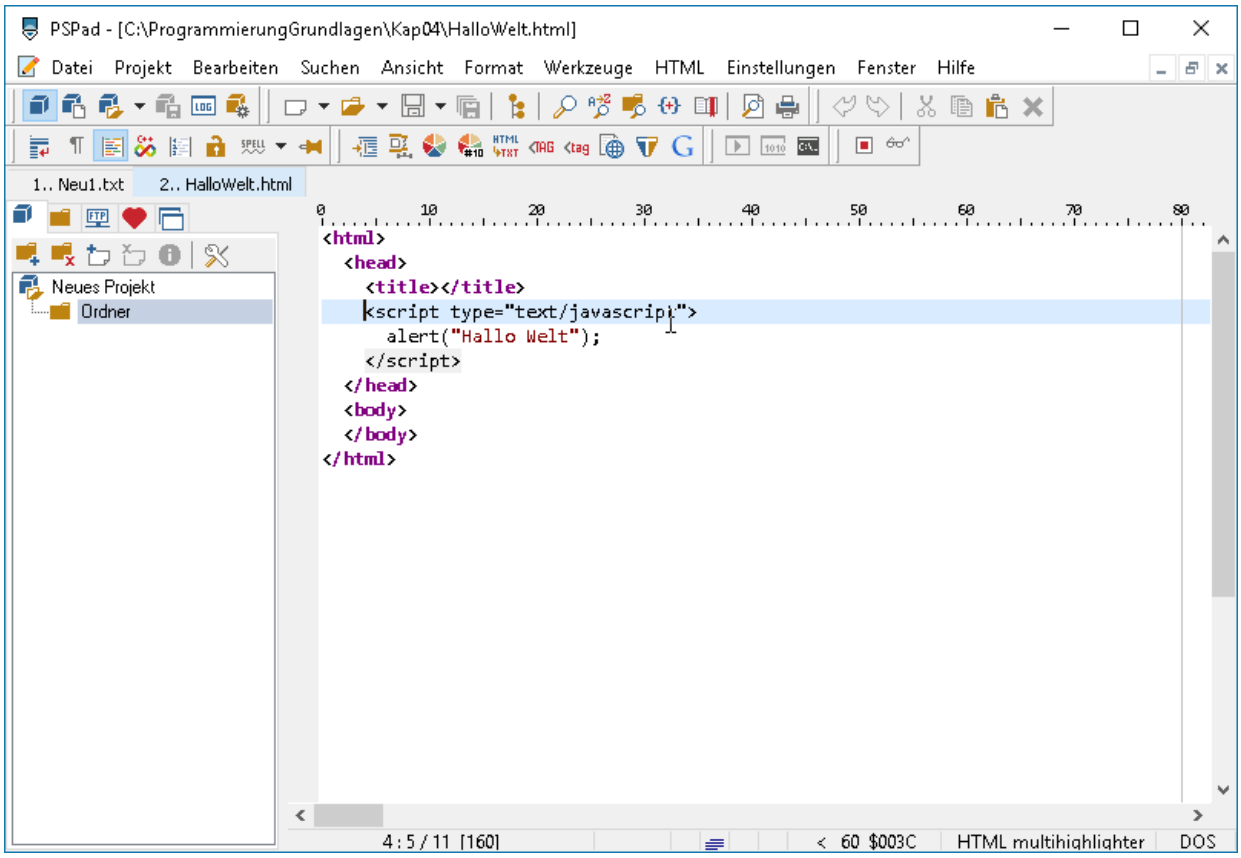
JavaScript interpretieren und ausführen

Das Hallo-Welt-Beispiel, das bereits mit Java erstellt wurde, soll nun mit JavaScript erstellt und über den Browser ausgeführt werden. Erweitern Sie dazu die Webseite wie folgt:

```
① <html>
    <head>
      <title></title>
②     <script type="text/javascript">
③       alert("Hallo Welt");
④     </script>
    </head>

⑤   <body>
    </body>
  </html>
```

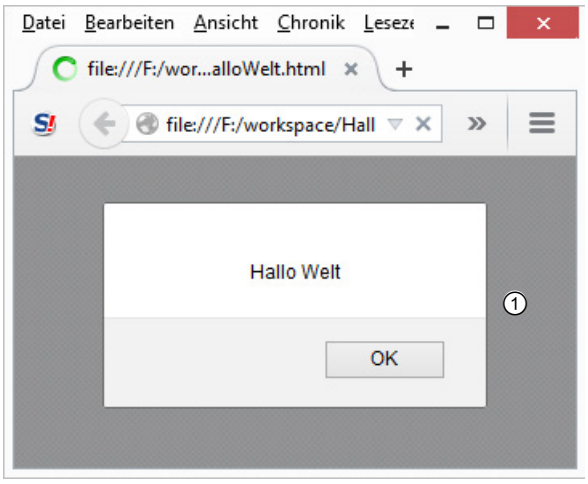
① Der Beginn der Webseite bleibt unverändert.
② Das einleitende Tag für den Skriptbereich. Ab der nächsten Zeile folgt JavaScript.
③ Eine JavaScript-Anweisung. Damit wird ein kleines Dialogfenster mit dem angegebenen Inhalt angezeigt.
④ Der Skriptbereich wird beendet. Danach folgt wieder normales HTML.
⑤ Der Abschluss der Webseite bleibt unverändert.



Der Quelltext in dem PSPad-Editor

- Speichern Sie den Quellcode unter dem Namen *HalloWelt.html*.
- Um die HTML-Datei mit dem JavaScript-Skript auszuführen öffnen Sie die Datei in einem Browser.

Die Webseite sollte in den Browser geladen und das JavaScript interpretiert werden. Sie sehen ein kleines Dialogfenster mit dem vorgegebenen Text ①.



Das Skript (JavaScript) wurde interpretiert und ausgeführt

Bei einigen Browsern bzw. Einstellungen von Browsern kann es sein, dass JavaScript nicht unmittelbar ausgeführt wird. In diesem Fall müssen Sie die Ausführung des Skripts erst gestatten, bevor Sie das Dialogfenster sehen können. JavaScript muss aktiviert sein, was bei modernen Browser in der Regel der Fall ist.

4.10 Übungen

Übung 1: Fragen zu Werkzeugen der Softwareentwicklung

Übungsdatei: -- **Ergebnisdatei: *uebung04.pdf***

1. Warum kann ein in einer höheren Programmiersprache codiertes Programm nicht sofort nach der Codierung ausgeführt werden?
2. Worin unterscheidet sich die Arbeitsweise eines Compilers von der eines Interpreters?
3. Warum ist es sinnvoll, in eigenen Programmen fertige Programmteile aus Bibliotheken zu verwenden?

Übung 2: Programm zur Ausgabe einer Textzeile erstellen

Übungsdatei: -- **Ergebnisdatei: *ErsteAusgabe.java***

1. Erstellen Sie ein Programm, das folgende Ausgabe erzeugt: `Java-Programme bestehen aus Klassen`.
2. Speichern Sie das Programm unter dem Namen *ErsteAusgabe.java*.
3. Kompilieren Sie das Programm und führen Sie das Programm anschließend aus.

5 Zahlensysteme und Zeichencodes

In diesem Kapitel erfahren Sie

- welche verschiedenen Zahlensysteme es gibt
- welche Zeichencodes es gibt
- wie Daten innerhalb des Computers dargestellt werden

Voraussetzungen

- Mathematische Grundkenntnisse

5.1 Zahlensysteme unterscheiden

Wozu werden Zahlensysteme benötigt?

Zahlensysteme werden zur Darstellung quantitativer Merkmale von Gegenständen, Vorgängen etc. verwendet. Diese Systeme basieren auf Zeichen, aus denen die Zahlen gebildet werden. Die bekanntesten Zahlensysteme sind das arabische (0, 1, 2 ...) und das römische (I, II, III, IV ...) Zahlensystem. Das arabische Zahlensystem ist in Europa das jüngere der beiden und das leistungsfähigere. Rechenoperationen sind dort sehr viel leichter möglich als mit dem römischen Zahlensystem.

Die Ziffern bzw. Zeichen, die in dem Zahlensystem vorkommen, werden **Nennwerte** genannt. Alle Zahlen des Zahlensystems setzen sich aus diesen Nennwerten zusammen. Die Anzahl der Nennwerte entspricht der **Basis** des Zahlensystems.

Dezimalsystem

Da Menschen zehn Finger zum Rechnen zur Verfügung stehen, entwickelten sie das Dezimalsystem. Die Basiszahl des Dezimalsystems ist die Zahl 10. Somit werden zehn Ziffern (Nennwerte) zur Darstellung aller Zahlen dieses Systems benötigt: **0**, **1**, **2**, **3**, **4**, **5**, **6**, **7**, **8**, **9**.

Innerhalb einer Zahl erhält jede Ziffer einen sogenannten **Stellenwert**. Der Stellenwert nimmt beim Dezimalsystem von Position zu Position um den Faktor 10 zu. Das Dezimalsystem und alle weiteren hier vorgestellten Zahlensysteme gehören im Gegensatz z. B. zum römischen Zahlensystem zu den Stellenwertsystemen.

Jede Dezimalzahl lässt sich auch als Summe von Potenzen der Basis 10 darstellen.

Beispiel: Dezimalzahl als Summe von 10er-Potenzen

234,3	=		$3 * 10^{-1}$	→	0,3	Niedrigster Stellenwert
		+	$4 * 10^{0}$	→	4	
		+	$3 * 10^{1}$	→	30	
		+	$2 * 10^{2}$	→	200	Höchster Stellenwert
				=	**234,3**	Summe ergibt Dezimalzahl

Dualsystem

Das bereits im 17. Jahrhundert entwickelte Dualsystem (auch als Binärsystem bezeichnet) arbeitet mit den beiden Nennwerten **0** und **1**. Durch Kombination dieser Ziffern lässt sich jede Dezimalzahl darstellen.

Für die EDV bot sich das Binärsystem an. Da es für elektronische Signale zwei Zustände ("ein" und "aus") gibt, lassen sich diese durch die Ziffern 1 (für "ein") und 0 (für "aus") oder umgekehrt darstellen.

Das Dualsystem ist ein Stellenwertsystem mit der Basis 2. Der Stellenwert einer Ziffer innerhalb einer Zahl des Dualsystems nimmt von Position zu Position um den Faktor 2 zu. Daher lässt sich jede Dualzahl auch als Summe von Potenzen zur Basis 2 darstellen. Das Ergebnis der Summe entspricht der Dezimalzahl dieser Dualzahl.

Beispiel: Dualzahl als Dezimalzahl

$(\mathbf{1101})_2$	=	$1 * 2^0$	→	1	Niedrigster Stellenwert
	+	$0 * 2^1$	→	0	
	+	$1 * 2^2$	→	4	
	+	$1 * 2^3$	→	8	Höchster Stellenwert
			=	$(\mathbf{13})_{10}$	Summe ergibt Dezimalzahl

Um Missverständnissen vorzubeugen bei der Frage, welchem Zahlensystem eine Zahl angehört, wird die Zahl in Klammern gesetzt und die Basis des Zahlensystems tiefergestellt, beispielsweise $(750)_{10}$ oder $(101)_2$.

Umwandlung einer Dezimalzahl in eine Dualzahl

Um eine Dezimalzahl in eine Dualzahl umzuwandeln, wird die Dezimalzahl durch 2 dividiert und der Rest notiert; das ganzzahlige Ergebnis wird wieder durch 2 dividiert und der Rest notiert, so lange, bis das ganzzahlige Ergebnis 0 beträgt.

Beispiel: Dezimalzahl als Dualzahl

$(\mathbf{22})_{10}$	=	22 / 2	→	11	Rest 0	0 Niedrigster Stellenwert
		11 / 2	→	5	Rest 1	1
		5 / 2	→	2	Rest 1	1
		2 / 2	→	1	Rest 0	0
		1 / 2	→	0	Rest 1	1 Höchster Stellenwert
			=			$(\mathbf{10110})_2$ Umwandlung in Dualzahl

Hexadezimalsystem

Da große Dualzahlen schwer lesbar sind, wurden Zahlensysteme entwickelt, die mehrere Stellen einer Dualzahl zusammenfassen, das Hexadezimal- und das Oktalsystem. Im Hexadezimalsystem werden jeweils vier Stellen einer Dualzahl codiert, wodurch sich sechzehn verschiedene Kombinationen von 0 und 1 ergeben.

Die Basiszahl des Hexadezimalsystems ist die Zahl 16. Das Hexadezimalsystem enthält die folgenden sechzehn Nennwerte: **0**, **1**, **2**, **3**, **4**, **5**, **6**, **7**, **8**, **9**, **A**, **B**, **C**, **D**, **E**, **F**.

Da das arabische Zahlensystem nur zehn Ziffern kennt, wurde vereinbart, die übrigen Ziffern mit Buchstaben zu bezeichnen: 10=A, 11=B, 12=C, 13=D, 14=E, 15=F.

Der Stellenwert einer Ziffer nimmt von Position zu Position um den Faktor 16 zu. Eine Hexadezimalzahl lässt sich daher als Summe von Potenzen der Basis 16 darstellen.

Die Umwandlung einer Dezimalzahl in eine Hexadezimalzahl erfolgt wie bei den bereits besprochenen Dualzahlen mit dem Unterschied, dass durch die Zahl 16 dividiert wird. Zur Umwandlung einer Hexadezimalzahl in eine Dualzahl wird jede Ziffer der Hexadezimalzahl als Dualzahl dargestellt.

Beispiel: Hexadezimalzahl als Dezimalzahl

(7B1) $_{16}$	=		1 *	16^0	→	1 *	1	=	1
		+	B *	16^1	→	11 *	16	=	176
		+	7 *	16^2	→	7 *	256	=	1792
								=	**(1969)** $_{10}$

Niedrigster Stellenwert

Höchster Stellenwert

Summe ergibt Dezimalzahl

Beispiel: Hexadezimalzahl als Dualzahl und umgekehrt

(7B1) $_{16}$	=	$(7)_{10}$	$(11)_{10}$	$(1)_{10}$		
	=	$(0111)_2$	$(1011)_2$	$(0001)_2$		
					=	$(0111\ 1011\ 0001)_2$

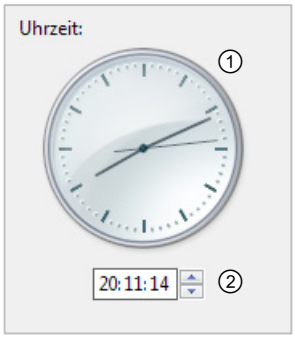
Umwandlung in Dualzahl

Oktalsystem

Dieses Zahlensystem codiert drei Stellen einer Dualzahl und ist auf der Basiszahl 8 (Nennwerte: 0, 1, 2, 3, 4, 5, 6, 7) aufgebaut. Daraus ergeben sich acht verschiedene Kombinationen aus 0 und 1, von 000 bis 111. Diese Darstellungsweise wurde früher in der Datenverarbeitung verwendet, ist jedoch mittlerweile veraltet.

5.2 Programme basieren auf Daten

Analoge und digitale Daten



Bei analogen Daten handelt es sich um Daten, die durch eine kontinuierliche Funktion, wie z. B. eine Zeigerstellung ① (Uhr) oder eine Skala (z. B. Thermometer), dargestellt werden. Der Begriff **analog** steht für entsprechend bzw. vergleichbar.

Digitale (engl. digit = Ziffer) Daten werden immer durch Ziffern ② dargestellt. Innerhalb eines Computers lassen sich Daten nur digital verarbeiten.

Probleme mit digitalen Daten können z. B. bei Divisionsrechnungen auftreten:

$$(2.0 / 3.0) * 3.0 = ?$$

Die mathematische Berechnung ergibt 2. Ein Computer rechnet in der Regel als Ergebnis z. B. 1.999999998 aus, da eine digitale Zahl im Computer nur mit einer bestimmten Anzahl von Stellen dargestellt werden kann.
Das analoge Ergebnis von 2/3 lautet 0.66666, mit unendlichen Stellen nach dem Komma. Der Speicher eines Computersystems ist jedoch begrenzt. Also wird die Zahl bei der digitalen Verarbeitung an einer bestimmten Dezimalstelle abgeschnitten. Nach einer erneuten Multiplikation wird mit dem digitalen Näherungswert gerechnet, wodurch die Abweichung des Ergebnisses zustande kommt.

Vorteile des Dualsystems für die rechnerinterne Darstellung

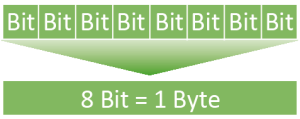
Die Verwendung eines Systems, das nur mit zwei Ziffern arbeitet, hat erhebliche Vorteile. Alle Daten, die im Dualsystem codiert sind, lassen sich problemlos und mit sehr niedriger Fehleranfälligkeit auf verschiedene Arten darstellen, weitergeben und verarbeiten. In der folgenden Übersicht finden Sie einige Beispiele der Verarbeitung und Weitergabe von digitalen Daten.

Nutzungsart	Ziffer 0 entspricht	Ziffer 1 entspricht
Speicherung mit Spannung	Keine Spannung	Ca. 6 Volt Spannung
Speicherung mit Magnetismus	Magnetisches Feld vorhanden	Veränderung der Polarität
Übertragung mit Tönen	Kurzer Ton	Langer Ton
Speicherung mit Relais	Schaltung offen	Schaltung geschlossen
Optische Speicherung auf Lochstreifen, Karte	Medium undurchsichtig (kein Loch)	Medium durchsichtig (Loch)
Optische Speicherung auf CD, DVD	Kein Übergang von oder zu einer Vertiefung	Übergang von oder zu einer Vertiefung

Bits und Bytes

Zwei wichtige Begriffe, die in der EDV immer wieder auftauchen, sind das **Bit** und das **Byte**.

- Die kleinste Speichereinheit in einem Computer oder auf einem Datenträger (z. B. Festplatte) wird als **Bit** bezeichnet. Bit ist die Abkürzung für Binary Digit, auch Binärziffer oder Dualziffer genannt.
- Zur Darstellung der internen Zeichen werden 8 Bit zu einem **Byte** zusammengefasst. Ein Byte kann genau ein Zeichen speichern. Zeichen können Ziffern, Buchstaben, Sonder- und Steuerzeichen entsprechen. Sie werden im Computer durch Zahlencodes dargestellt, z. B. $(65)_{10}$ für den Buchstaben A, $(53)_{10}$ für die Zahl 5 oder $(33)_{10}$ für das Ausrufezeichen ! beim ASCII-Code (vgl. Abschnitt 5.4).
- Eine Folge von Bits wird als **Bitmuster** bezeichnet.



Oft benötigte Größenangaben in Byte:

Abkürzungen	1 KByte / KiB	1 MByte / MiB	1GByte / GiB	1 TByte / TiB
Bezeichnung	1 Kilobyte	1 Megabyte	1 Gigabyte	1 Terabyte
Umrechnung in Byte	1.024 Byte	1.024 KByte	1.024 MByte	1.024 GByte
	2^{10} Byte	2^{20} Byte	2^{30} Byte	2^{40} Byte

Paritätsbit als Kontrollmöglichkeit

Bei der Übertragung von Bitmustern können technisch bedingte Fehler zu einer Veränderung des Bitmusters führen, indem beispielsweise ein Bit durch einen Schaltfehler umkippt, d. h., aus 0 wird 1 oder umgekehrt. Um solche Fehler zu erkennen, wird oft ein sogenanntes Prüfbit, auch Paritätsbit genannt, an das Bitmuster angehängt. Bei der geraden Parität ist vereinbarungsgemäß das Paritätsbit 0, wenn die Anzahl der Einsen gerade ist. Bei der ungeraden Parität ist es umgekehrt. Das Paritätsbit ist 0, wenn die Anzahl der Einsen ungerade ist.

Beispiel

Das erste Bit ist das zusätzliche Paritätsbit (PB), die übrigen acht Bit stellen das zu übertragende Bitmuster dar.

Der Operator modulo ermittelt den Rest bei der Division mit Ganzzahlen. 5 dividiert durch 2 ist 2, Rest 1.

```
1  10001111
PB  Bitmuster

Summe der Einsen = 5
5 modulo 2 = 1 => Das Paritätsbit ist 1.
```

5.3 Digitales Rechnen

Im Computer liegen sämtliche Daten im Dualsystem vor, egal ob es sich um Zahlen, Speicheradressen oder um Maschinenbefehle handelt. In dieser Form werden sie gespeichert und verarbeitet.

Addieren von Dualzahlen

Die Addition von dualen Zahlen ist der schriftlichen Addition von Dezimalzahlen sehr ähnlich.

- Die Ziffern werden stellenweise von rechts beginnend addiert.
- Wenn ein Übertrag entsteht, geht dieser auf die höherwertige Stelle über.
- Die Regeln sind nebenstehend dargestellt. Der Übertrag wird beim Computer in einem speziellen Bit gespeichert, dem Carrybit.

```
0 + 0 = 0
1 + 0 = 1
0 + 1 = 1
1 + 1 = 0, Übertrag 1
```

Beispiel

45	**101101**
+ 58	**+ 111010**
Übertrag: 110	*Übertrag: 1110000*
103	1100111
Schriftliches Addieren im Dezimalsystem	*Addieren im Dualsystem*

Komplementbildung

Eine sehr häufig benötigte Operation des Computers ist die Komplementbildung. Sie wird bei der Ausführung verschiedener Rechenoperationen eingesetzt, z. B. bei der Subtraktion. Es gibt zwei Arten von Komplementbildungen. Das Komplement der positiven Zahl Z mit der Basis b kann bezüglich b-1 ((b-1)-Komplement) oder bezüglich b (b-Komplement) gebildet werden. Das sind beispielsweise bei Dualzahlen (mit der Basis 2) das **1er-** und das **2er-Komplement**.

Das (b-1)-Komplement wird für gebrochene Zahlen (Zahlen mit Nachkommastellen) verwendet.

Das b-Komplement der positiven n-stelligen Zahl Z mit der Basis b wird wie folgt gebildet:

- $b^n - Z$ für Z <> 0
- 0 für Z = 0

Beispiel:

10er-Komplement der Zahl 4637	10^4	- 4637	= 5363

2er-Komplement der Zahl 11010101	2^8	- 11010101	=
	100000000	- 11010101	= 00101011

Subtraktion von Dualzahlen

Es soll der Ausdruck Z1 - Z2 berechnet werden. Das Verfahren der schriftlichen Subtraktion ist für digitale Computer ungeeignet. Es wird stattdessen eine Addition der Zahl Z1 mit dem Komplement der Zahl Z2 ausgeführt.

0 - 0 = 0
1 - 0 = 1
0 - 1 = 1, Übertrag -1
1 - 1 = 0
0 - 1 mit Übertrag -1 = 0, Übertrag -1

- Komplement der Zahl Z2 bilden
- Addition von Z1 und Komplement Z2
- Der Übertrag an der höchsten Stelle wird weggelassen.
- (Tritt an der höchsten Stelle kein Übertrag auf, wird vom Ergebnis der Addition das Komplement gebildet und mit einem negativen Vorzeichen versehen.)
- Der negative Übertrag wird beim Computer in einem speziellen Bit gespeichert, dem Borrowbit.

Beispiele

Aufgabe 1 (Dezimalsystem):	58 - 45 =
1. Komplement von 45: 2. Addieren 3. Übertrag streichen	Z1 = 58; Z2 = 45 10^2 - 45 = 100 - 45 = 55 58 + 55 = 113 ~~1~~13
→ Ergebnis	58 - 45 = 13

Aufgabe 2 (Dualsystem):	111010 - 101101 =
1. Komplement von 101101: 2. Addieren 3. Übertrag streichen	Z1 = 111010; Z2 = 101101 $(2^6)_{10}$ - 101101 = $(64)_{10}$ - 101101 = 1000000 - 101101 = 10011 111010 + 10011 = 1001101 ~~1~~001101
→ Ergebnis	111010 - 101101 = 001101

Aufgabe 3 (Dualsystem):	101001 - 110001 =
1. Komplement von 110001: 2. Addieren 3. Kein Übertrag, daher das Komplement des Ergebnisses bilden:	Z1 = 101001; Z2 = 110001 $(2^6)_{10}$ - 110001= $(64)_{10}$ - 110001= 1000000 - 110001= 1111 101001+ 1111= 111000 = 1000000 - 111000 = 1000
→ Ergebnis	101001 - 110001 = **-1000**

Multiplikation von Dualzahlen

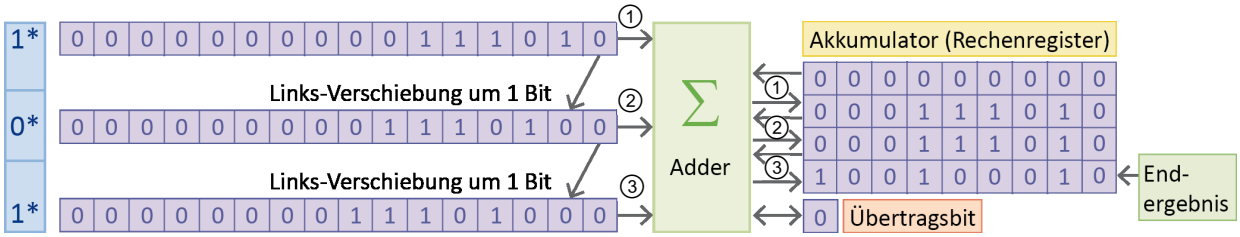
Die Multiplikation von Dualzahlen wird im Rechner durch bitweise Verschiebungen der Zahlen und Additionen realisiert. Das funktioniert ähnlich wie die schriftliche Multiplikation.

```
0 * 0 = 0
1 * 0 = 0
0 * 1 = 0
1 * 1 = 1
```

Beispiel

58 * 115	**111010 * 101**
+ 58	+ 111010
+ 58	+ 000000
+ 290	+ 111010
6670	100100010
Schriftliches Multiplizieren im Dezimalsystem	*Multiplizieren im Dualsystem*

Im Computer werden die Zwischenergebnisse der Multiplikation sofort addiert. Diese Aufgabe übernimmt der sogenannte Addierer (Adder). Die Zwischenergebnisse werden in einem Akkumulator (dem Speicher für die Rechenergebnisse) gespeichert. Das nächste Zwischenergebnis und der Inhalt des Akkumulators werden im nächsten Schritt addiert, und das Ergebnis wird im Akkumulator gespeichert. Wenn alle Zwischenergebnisse addiert wurden, steht das Endergebnis im Akkumulator. Tritt bei der Berechnung ein Übertrag auf, wird dieser in einem Übertragsbit gespeichert und fließt bei der nächsten Addition mit ein.



Multiplikation wird im Computer durch Linksverschiebung und Addition realisiert. Die Ziffern in den Kreisen kennzeichnen die Reihenfolge der Abarbeitung. Die am weitesten rechts stehende Ziffer des Multiplikators wird als Erste zur Berechnung herangezogen, gerade umgekehrt wie bei der schriftlichen Multiplikation.

5.4 Zeichencodes

Ein Zeichencode stellt eine Zuordnungsregel dar, mit deren Hilfe jedes Zeichen eines Zeichenvorrates eindeutig durch bestimmte Zeichen eines anderen Zeichenvorrates dargestellt werden kann. Zur Darstellung von Zeichen in Computern wurden verschiedene Zeichencodes entwickelt und teilweise standardisiert, z. B. ASCII-Code und Unicode.

Der ASCII-Code

ASCII steht für **A**merican **S**tandard **C**ode for **I**nformation **I**nterchange (Amerikanische Standardverschlüsselung für Datenaustausch). Der ASCII-Code wurde 1967 als 7-Bit-Zeichencode standardisiert. In der Weiterentwicklung wurde der ASCII-Code auf 8 Bit erweitert. Als Standard-ASCII-Code sind nur die 128 Zeichen des 7-Bit-Codes festgelegt. Mit 7 Bit können genau $2^7 = 128$ Zeichen dargestellt werden. Der Zeichencode umfasst druckbare Zeichen wie Buchstaben, Ziffern und Sonderzeichen und nicht druckbare Zeichen wie Übertragungs-, Format-, Geräte- und Informationssteuerungszeichen.

0	NUL	1	SOH	2	STX	3	ETX	4	EOT	5	ENQ	6	ACK	7	BEL
8	BS	9	HT	10	NL	11	VT	12	NP	13	CR	14	SO	15	SI
16	DLE	17	DC1	18	DC2	19	DC3	20	DC4	21	NAK	22	SYN	23	ETB
24	CAN	25	EM	26	SUB	27	ESC	28	FS	29	GS	30	RS	31	US
32	SP	33	!	34	"	35	#	36	$	37	%	38	&	39	'
40	(	41	)	42	*	43	+	44	,	45	-	46	.	47	/
48	0	49	1	50	2	51	3	52	4	53	5	54	6	55	7
56	8	57	9	58	:	59	;	60		61		62		63	?
64	@	65	A	66	B	67	C	68	D	69	E	70	F	71	G
72	H	73	I	74	J	75	K	76	L	77	M	78	N	79	O
80	P	81	Q	82	R	83	S	84	T	85	U	86	V	87	W
88	X	89	Y	90	Z	91	[	92	\	93	]	94	^	95	_
96	`	97	a	98	b	99	c	100	d	101	e	102	f	103	g
104	h	105	i	106	j	107	k	108	l	109	m	110	n	111	o
112	p	113	q	114	r	115	s	116	t	117	u	118	v	119	w
120	x	121	y	122	z	123	{	124	|	125	}	126	~	127	DEL

Die ASCII-Code-Tabelle (jeweils rechts der Dezimalwert des ASCII-Zeichens)

Die wichtigsten, nicht darstellbaren Zeichen sind folgende:

9	HT	horizontal tabulation – Horizontaltabulator
10	NL	new line – neue Zeile
13	CR	carriage return – Wagenrücklauf (analog zur Schreibmaschine)
127	DEL	delete – Löschen, Entfernen

Im Windows-Betriebssystem erhalten Sie das zugehörige ASCII-Zeichen, indem Sie bei gedrückter Alt-Taste den ASCII-Wert über den numerischen Tastaturblock eingeben.

EBCDI-Code

EBCDI steht für **E**xtended **B**inary-**C**oded **D**ecimal **I**nterchange. Dieser 8-Bit-Code wird vor allem auf Großrechnern verwendet, die in der kommerziellen Datenverarbeitung eingesetzt werden. Er wurde von IBM entwickelt und existiert in mehreren Varianten, um Sprachunterschiede auszugleichen. Für eine Codeumwandlung von ASCII-Code in EBCDI-Code müssen Kodierungstabellen verwendet werden.

ANSI-Code

Um Zeichen wie ä, ö, ü, æ und ê darstellen zu können, wurde der 8-Bit-ANSI-Code entwickelt. Die Abkürzung ANSI steht für **A**merican **N**ational **S**tandards **I**nstitute. Das ANSI ist das nationale Institut für Normung im Datenverarbeitungsbereich in den USA.

Der ANSI-Code ist in großen Teilen mit dem ASCII-Code identisch. Da aber z. B. die Umlaute (ä, ö, ü) im ASCII-Code nicht enthalten sind, treten bei Textkonvertierungen zwischen verschiedenen Betriebssystemen (z. B. MS-DOS (ASCII) und Microsoft Windows (ANSI)) Probleme auf.

Unicode

Unicode ist ein Zeichencode, in dem fortlaufend Zeichen, beispielsweise aller bekannten Sprachen weltweit, codiert werden. In der aktuellen Version 7.0 (Stand: Juni 2014) von Unicode sind mehr als 110 000 Zeichen codiert.

Unicode vereint Zeichencodes. So können z. B. hebräische und lateinische Buchstaben ohne Wechsel des Zeichensatzes geschrieben werden. Steuerzeichen wie Silbentrennungszeichen, erzwungene Leerzeichen oder Tabulatorzeichen sowie Zeichen mathematischer Formeln sind ebenfalls in Unicode enthalten.

Das Unicode-System ist konform zur internationalen Norm ISO/IEC 10646. Erstellt wurde es vom Unicode-Konsortium und einer ISO-Arbeitsgruppe (**ISO** – **I**nternational **S**tandard **O**rganization). Zu weiteren Informationen vgl. auch *http://www.unicode.org*.

5.5 Übung

Fragen zu Zahlensystemen und Zeichencodes

Übungsdatei: -- **Ergebnisdatei: *uebung05.pdf***

1. Erstellen Sie eine Tabelle mit vier Spalten. Tragen Sie in die erste Spalte die Dezimalzahlen von 1 bis 16 ein. In die zweite Spalte sind dann die entsprechenden Dualzahlen, in die dritte Spalte die Oktalzahlen und in die vierte Spalte die Hexadezimalzahlen einzutragen.
2. Wandeln Sie die Dezimalzahl 12345 in eine hexadezimale Zahl um.
3. Geben Sie die hexadezimale Zahl 1F binär im Dualsystem an.
4. Welchen dezimalen Wert hat die Oktalzahl 1357?
5. Rechnen Sie die Dezimalzahl 735 in das Dualsystem um.
6. Stellen Sie die Binärzahl 1101 1100 1011 1010 in hexadezimaler Form dar.
7. Was verstehen Sie unter den Begriffen Bit und Byte?
8. Addieren Sie die Dualzahlen 11011011 und 1010011001.
9. Lösen Sie folgende Aufgaben: 1101111 - 111111 und 1011000 - 1100011.
10. Multiplizieren Sie die Zahlen 1011110 und 110010.
11. Rechnen Sie um: 3600 Byte in KByte sowie 1,44 MByte in Byte.

6 Grundlegende Sprachelemente

In diesem Kapitel erfahren Sie

- was Syntax und Semantik bedeuten
- was Datentypen, Variablen und Konstanten sind
- welche Operatoren es gibt
- was unter Ausdrücken zu verstehen ist

6.1 Syntax und Semantik

Programmiersprachen sind, wie auch natürliche Sprachen, nach definierten Regeln aufgebaut. Diese Regeln legen fest, welche Zeichen verwendet werden dürfen, wie die Zeichen angeordnet sein müssen und welche Bedeutung bestimmte Zeichenfolgen haben.

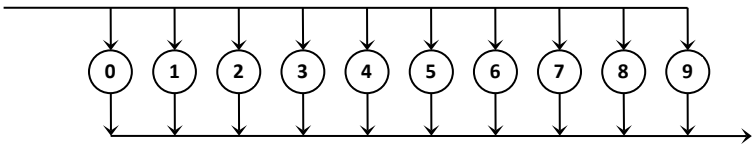
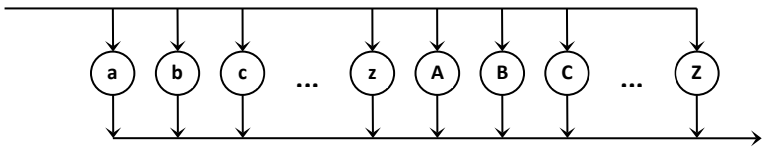
- Die **Syntax** einer Sprache bestimmt den Aufbau der Sätze. Auf Programmiersprachen bezogen legt die Syntax z. B. fest, wie Anweisungen aufgebaut sind.
- Die **Semantik** erklärt die Bedeutung der Sätze. Auf Programmiersprachen bezogen wird durch die Semantik z. B. beschrieben, was eine Anweisung bedeutet.

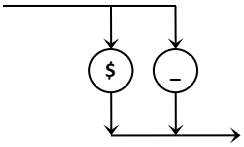
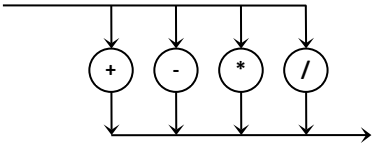
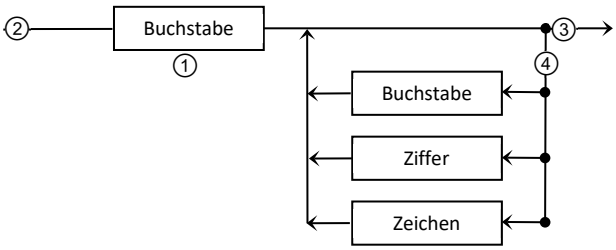
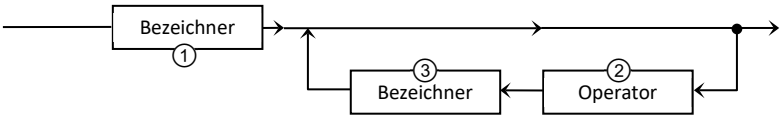
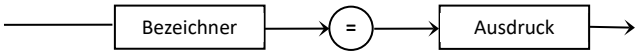
Die Syntax einer Sprache kann mithilfe von Syntaxdiagrammen oder der Backus-Naur-Form dargestellt werden. In Syntaxdiagrammen wird die Grammatik einer Sprache grafisch dargestellt und ist dadurch leichter lesbar. Die Backus-Naur-Form (BNF) verwendet eine textuelle Darstellung der Syntax und ist dadurch mit jedem Texteditor zu erfassen. In Sprachbeschreibungen wird häufig die erweiterte Backus-Naur-Form (EBNF) verwendet, die im Vergleich zur BNF mehr Möglichkeiten bietet.

Darstellung der Syntaxdiagramme

Die Syntaxdiagramme einer Sprache sind sehr umfangreich, da alle Konstrukte einer Sprache beschrieben werden. Das folgende Beispiel zeigt, wie ein numerischer Ausdruck einer Sprache dargestellt werden kann.

Beispiel: Syntaxdiagramme

Syntaxbegriff	Syntaxdiagramm und Bedeutung
<Ziffer>	 Eine Ziffer kann ein Wert von 0 bis 9 sein.
<Buchstabe>	 Ein Buchstabe kann jeder kleine und große Buchstabe des Alphabets sein.

Syntaxbegriff	Syntaxdiagramm und Bedeutung
<Zeichen>	 Ein Zeichen kann entweder $ oder _ sein.
<Operator>	 Ein Operator ist eines der Zeichen +, -, * oder /.
<Bezeichner>	 Ein Bezeichner muss mit einem Buchstaben ① beginnen. Anschließend können weitere Bestandteile folgen oder nichts. Dies wird durch die Pfeile und die Linien veranschaulicht. Wenn Sie die Linie vom Startpunkt ② aus verfolgen, kommen Sie auf jeden Fall zum Buchstaben ①. Vom Buchstaben aus gelangen Sie an einen Knotenpunkt ③. Verfolgen Sie die Linie geradeaus weiter, erreichen Sie das Ende (der Bezeichner besteht in diesem Fall aus genau einem Buchstaben). Verfolgen Sie aber die vertikale Linie ④, gelangen Sie entweder zu einem weiteren Buchstaben, einer Ziffer oder einem Zeichen. Danach kehren Sie wieder auf die ursprüngliche Linie und somit zu dem Knotenpunkt zurück. Nun können Sie den Weg beenden oder wieder den vertikalen Weg gehen.
<Ausdruck>	 Ein Ausdruck kann ein Bezeichner ① sein oder ein Bezeichner ①, gefolgt von einem Operator ② und einem Bezeichner ③. Ein Ausdruck kann aber auch mehr als zwei (beliebig viele) Bezeichner, die jeweils durch Operatoren miteinander verknüpft sind, beinhalten.
<Zuweisung>	 Eine Zuweisung besteht aus einem Bezeichner, dem Zuweisungsoperator = und einem Ausdruck.

Das Beispiel bezieht sich auf keine spezielle Sprache, sondern soll das Darstellungsprinzip verdeutlichen.

Darstellung der EBNF

Symbol	Beschreibung
`<begriff>`	Syntaxbegriff
`::=`	"ist definiert als" oder linke Seite "kann ersetzt werden durch" rechte Seite
`{}`	Der in Klammern eingeschlossene Teil kann beliebig oft wiederholt werden oder entfallen.
`[ ]`	Der in eckigen Klammern eingeschlossene Teil ist optional (kann auch entfallen).

Symbol	Beschreibung
<a>|<b>	Alternative (logisches Oder) bedeutet: <a> *oder* <b> wird verwendet.
"x"	Zeichen oder Wörter in Anführungszeichen sind Bestandteile der Sprache und müssen genau so geschrieben werden.

Beispiel: EBNF

Die oben aufgeführten Syntaxdiagramme wurden hier in die EBNF umgesetzt.

```
<ziffer>     ::= "0"|"1"|"2"|"3"|"4"|"5"|"6"|"7"|"8"|"9"
<buchstabe>  ::= "a"|"b"|"c"|...|"z"|"A"|"B"|"C"|...|"Z"
<zeichen>    ::= "$"|"_"
<operator>   ::= "+"|"-"|"*"|"/"
<bezeichner> ::= <buchstabe>{<buchstabe>|<ziffer>|<zeichen>}
<ausdruck>   ::= {<bezeichner> <operator>} <bezeichner>
<zuweisung>  ::= <bezeichner> "=" <ausdruck>
```

Die Beschreibung einer Sprache in der EBNF muss immer bis in die kleinste syntaktische Einheit (Zeichen der Sprache) auflösbar sein. Bevor ein Ausdruck, wie

```
<alternative> ::= "if (" <logischer ausdruck> ") " <anweisung> ["else"
                                                  <anweisung>]
```

angegeben werden kann, müssen zuvor die Syntaxbegriffe `<logischer ausdruck>` und `<anweisung>` erklärt sein.

Hinweis zur Schreibweise der Syntax im Buch

- Schlüsselwörter werden fett hervorgehoben.
- Optionale Angaben stehen in eckigen Klammern [ ].
- Drei Punkte (...) kennzeichnen, dass weitere Angaben folgen können.
- Sofern eckige Klammern oder drei Punkte (...) als Bestandteil des Quelltextes erforderlich sind, wird darauf in der Erläuterung explizit hingewiesen.

6.2 Grundlegende Elemente einer Sprache

Alle Sprachen werden durch ihre eigene Syntax und Semantik beschrieben. Sprachen können sich z. B. hinsichtlich des Aufbaus von Bezeichnern, der zulässigen Datentypen (vgl. Abschnitt 6.3) und der verfügbaren Operationen (vgl. Abschnitt 6.6) unterscheiden. Einige Elemente oder Mechanismen sind Bestandteil vieler Sprachen und sich sehr ähnlich. Diese werden im weiteren Verlauf des Buchs exemplarisch erklärt.

Bezeichner

Bezeichner (engl. identifier) geben Dingen in Programmen, z. B. Variablen (vgl. Abschnitt 6.5), einen Namen, über den sie im Programm angesprochen werden können. Die Anzahl der Zeichen und welche Zeichen zulässig sind, ist in den Sprachen verschieden. Einige Sprachen unterscheiden auch zwischen Groß- und Kleinschreibung (**case-sensitive**).

In Programmen können Sie als Bezeichner beispielsweise sowohl englische als auch deutsche Namen wählen. Innerhalb eines Programms sollte aber eine einheitliche Schreibweise gewählt werden. Englische Namen haben den Vorteil, dass sie oft kürzer sind, keine Umlaute enthalten, zu den reservierten Wörtern passen und, falls Programme international eingesetzt werden sollen, besser verständlich sind.

Bezeichner in Java und JavaScript

Für den Aufbau eines Bezeichners in Java als auch in JavaScript gelten folgende Regeln:

- Bezeichner müssen mit 'a' ... 'z', 'A' ... 'Z', '_' oder '$' beginnen und können dann beliebig fortgesetzt werden.
- Bezeichner beinhalten keine Leerzeichen und keine Sonderzeichen.
- Java unterscheidet zwischen Groß- und Kleinschreibung. `Name`, `NAME` und `name` sind drei unterschiedliche Bezeichner für Elemente in Java.
- Java und JavaScript verwenden den Bezeichner in seiner kompletten Länge (alle Stellen sind signifikant – bis auf technische Begrenzungen).

In Java können auch Buchstaben aus anderen Alphabeten genutzt werden, da Java den Unicode-Zeichencode verwendet. Dies ist allerdings nicht zu empfehlen, da z. B. Probleme mit Editoren und Dateisystemen auftreten können. Empfehlenswert ist es, die Zeichen 'a' ... 'z', 'A' ... 'Z', '_' und '0' ... '9' zu verwenden.

Reservierte Wörter

In jeder Sprache ist eine Anzahl von Wörtern als Schlüsselwörter definiert. Diese reservierten Wörter haben in der Programmiersprache eine spezielle Bedeutung.

Alle reservierten Wörter in Java

`abstract`	`const` ②	`final`	`int`	`public`	`throw`
`assert`	`continue`	`finally`	`interface`	`return`	`throws`
`boolean`	`default`	`float`	`long`	`short`	`transient`
`break`	`do`	`for`	`native`	`static`	`true` ①
`byte`	`double`	`goto` ③	`new`	`strictfp`	`try`
`case`	`else`	`if`	`null` ①	`super`	`void`
`catch`	`enum`	`implements`	`package`	`switch`	`volatile`
`char`	`extends`	`import`	`private`	`synchronized`	`while`
`class`	`false` ①	`instanceof`	`protected`	`this`	

- Die drei mit ① gekennzeichneten reservierten Wörter (`false`, `null`, `true`) stellen konstante vordefinierte Werte dar und werden als **Literale** bezeichnet.
- Alle anderen hier aufgeführten reservierten Wörter sind sogenannte **Schlüsselwörter**.
- Alle reservierten Wörter dürfen **nicht** als Bezeichner verwendet werden.
- `const` ② und `goto` ③ sind ebenfalls reservierte Schlüsselwörter, die allerdings in Java nicht verwendet werden. Trotzdem können keine Variablen mit diesen Namen deklariert werden.

Die reservierten Wörter in JavaScript sind weitgehend identisch mit denen in Java. In JavaScript haben jedoch noch mehr Schlüsselwörter keine Bedeutung und sind nur „vorsorglich“ reserviert.

Schlüsselwörter werden in der Darstellung von Quelltextauszügen in diesem Buch **fett** hervorgehoben.

Kommentare

Zur Dokumentation eines Quelltextes verwenden Sie Kommentare. Diese sollen Ihnen und anderen Mitarbeitern an einem Projekt helfen, den Quelltext auch nach längerer Zeit noch zu verstehen.

Kommentare sollten beispielsweise verwendet werden, um

- den Zweck von Variablen zu beschreiben,
- die Verwendung einer Methode (bzw. Funktion oder Prozedur) zu erläutern,
- einen nicht selbsterklärenden Algorithmus zu beschreiben.

Durch Kommentare wird die Größe des ausführbaren Programms nicht erhöht. Kommentare werden vom Compiler nicht berücksichtigt.

Die verschiedenen Programmiersprachen lassen unterschiedliche Arten von Kommentaren zu. Kommentare werden durch spezielle für die Programmiersprache festgelegte Zeichen eingeleitet bzw. eingeschlossen.

Beispiel: Kommentartypen in Java und JavaScript

Einzeilige Kommentare	`//`	Kommentar bis zum Ende der Zeile Alle Zeichen der Zeile hinter // werden vom Compiler überlesen.
(Mehrzeiliger) Kommentarblock	`/*` `*/`	Kommentar über mehrere Zeilen Ab der Zeichenkombination /* werden alle Zeichen im Quelltext überlesen, bis die Zeichenkombination */ auftritt.

6.3 Standarddatentypen (elementare Datentypen)

In einem Programm verarbeiten Sie Daten, die sich in ihrer Art unterscheiden:

- Zahlen – numerisch
- Zeichen – alphanumerisch
- boolesche Daten – logisch

Ein **Datentyp** ist eine Menge darstellbarer Werte. Viele Programmiersprachen besitzen vordefinierte elementare Datentypen, auch primitive Datentypen genannt. Diese Datentypen unterscheiden sich in der Art der Daten und in dem zulässigen Wertebereich. Ebenso unterscheiden sich Programmiersprachen darin, welche Datentypen sie bereitstellen und wie diese genau implementiert sind.

Numerische Datentypen

Numerische Datentypen lassen sich, wie in der Mathematik, in ganze Zahlen (Integer) und reelle Zahlen (Real) unterteilen. Dabei sind die Wertebereiche und die Genauigkeit der Zahlen durch die Festlegung des Speicherbereichs begrenzt. Numerische Datentypen werden z. B. für Berechnungen eingesetzt.

Integer-Datentypen

Integer-Datentypen sind **Ganzzahlen** und besitzen keine Nachkommastellen. Eine Programmiersprache bietet meist mehrere Integer-Datentypen an, die sich durch die Bezeichnung und die Größe des Wertebereichs unterscheiden. Die Größe des Wertebereichs ist von der Größe des Speicherplatzes abhängig, der für den Datentyp vorgesehen ist, z. B. 2, 4 oder 8 Byte.

Der kleinste bzw. größte darstellbare Wert eines **vorzeichenlosen Integer-Datentyps** kann wie folgt berechnet werden. Für x wird dabei die zulässige Speichergröße in Bit eingesetzt: $0 \ldots 2^x - 1$

Für **Integer-Datentypen mit Vorzeichen** kann folgende Formel verwendet werden: -2^{x-1} ... $2^{x-1}-1$
Ein Bit wird dabei für die Speicherung des Vorzeichens verwendet.

Beispiel: Integer-Datentypen in Java

Datentyp	Wertebereich	Speichergröße
`byte`	`-128 ... 127`	1 Byte
`short`	`-32768 ... 32767`	2 Byte
`int`	`-2.147.483.648 ... 2.147.483.647`	4 Byte
`long`	`-9.223.372.036.854.775.808 ... 9.223.372.036.854.775.807`	8 Byte

- Integer-Datentypen werden im Computer immer genau dargestellt.
- Es treten keine Rundungsfehler bei der Darstellung der Zahlen auf.

Üblicherweise werden Sie in Java für ganzzahlige Werte den Datentyp `int` verwenden, denn er bietet für die meisten Anwendungsfälle einen ausreichenden Wertebereich.

Gleitkomma-Datentypen

Für **Fließkommazahlen** (Dezimalzahlen) werden Gleitkomma-Datentypen mit Vorzeichen verwendet. Der Computer kann jedoch nicht jede Zahl genau darstellen. Dies führt auch bei einfachen Rechnungen zu Rundungsfehlern. Je nach verwendetem Typ lassen sich nur Zahlen mit einer bestimmten Genauigkeit abbilden. Für eine höhere Genauigkeit wird aber auch mehr Speicherplatz benötigt. Die meisten Programmiersprachen besitzen zwei Gleitkomma-Datentypen: Single Precision (einfache Genauigkeit) und Double Precision (doppelte Genauigkeit). Einige Sprachen bieten zusätzlich eine Extended Precision (höhere Genauigkeit).

Für die Speicherung einer Gleitkommazahl einfacher Genauigkeit stehen in Java beispielsweise 32 Bit zur Verfügung.

- Ein Bit wird für das Vorzeichen (VZ) der Zahl (0 für positiv, 1 für negativ) verwendet.
- 8 Bit beschreiben den Exponenten.
- Für die Stellen der Zahl (Mantisse) bleiben dann noch 23 Bit übrig. Damit können 7 Dezimalstellen abgelegt werden.

Gleitkommazahlen mit doppelter Genauigkeit werden in Java beispielsweise standardmäßig in 64 Bit gespeichert.

Die Bezeichnung des Datentyps ist in den jeweiligen Programmiersprachen unterschiedlich.

Beispiel: Gleitkomma-Datentypen in Java

Datentyp	Genauigkeit	Speichergröße
`float`	7 Stellen	4 Byte
`double`	15 Stellen	8 Byte

Üblicherweise werden Sie in Java für Dezimalzahlen den Datentyp `double` verwenden, denn die Computer verfügen zumeist über ausreichenden Speicherplatz, und beim Datentyp `float` machen sich Rundungsfehler deutlich bemerkbar.

Zeichen-Datentyp

Zeichen-Datentypen können beliebige Zeichen des ASCII-Zeichensatzes enthalten. Sie sind nicht auf Zahlen und Buchstaben begrenzt und können auch Sonderzeichen wie !"§$%&/ speichern. Einzelne Zeichen werden bei der Wertzuweisung durch einfache Hochkommata eingeschlossen. Es ist von der Sprache abhängig, ob Zeichen intern als ASCII- oder Unicode-Zeichen dargestellt werden.

Für Ausgaben (akustisch, auf Bildschirm oder Drucker) werden häufig Steuerzeichen (nicht druckbare Zeichen, auch Escape-Sequenzen genannt) benötigt. Dazu wird auch der Zeichen-Datentyp verwendet.

Escape-Sequenz	Beschreibung
`\a`	Klingelzeichen
`\n`	Führt einen Zeilenvorschub und Wagenrücklauf durch, d. h. springt an den Anfang der nächsten Zeile (NL – new line, LF – line feed)
`\t`	Fügt einen horizontalen Tabulator ein (HT – horizontal tabulator)
`\\, \', \?, \"`	Stellt das zweite Zeichen dar (\, ', ?, ")

Beispiel: Zeichen-Datentyp in Java

Datentyp	Wertebereich	Speichergröße
`char`	Alle Unicode-Zeichen	2 Byte

Logischer Datentyp

George Boole entwickelte im 19. Jahrhundert eine nach ihm benannte Algebra, die boolesche Algebra. Der Grundgedanke ist, dass es nur die beiden Wahrheitswerte wahr (true) und falsch (false) gibt. Eine solche Variable, die nur diese beiden Werte annehmen kann, ist vom logischen Datentyp `boolean`.

Auch wenn ein Bit im Prinzip für die Speicherung eines logischen Wertes ausreicht, wird meist ein Byte für dessen Speicherung benötigt, da ein Bit keine adressierbare Einheit im Rechner ist. Ein Byte ist die kleinste adressierbare (direkt ansprechbare) Einheit im Rechner.

- Einige Sprachen bieten zusätzlich einen Datentyp `Bit String`, der eine Sammlung von logischen Werten darstellt, die durch die einzelnen Bit-Werte repräsentiert werden.
- Nicht in allen Sprachen gibt es einen speziellen logischen Datentyp. Er wird dort meist mithilfe von ganzzahligen Werten verwaltet. Dabei steht 0 für falsch und jeder andere Wert für wahr. Einige Sprachen (etwa JavaScript, Python oder PHP) stellen einen logischen Datentyp zur Verfügung, interpretieren aber auch in Vergleichssituationen numerische Werte als wahr und falsch.

Beispiel: Logischer Datentyp in Java

Datentyp	Wertebereich	Speichergröße
`boolean`	`true, false`	1 Byte

6.4 Literale für primitive Datentypen

Werte, die Sie im Quelltext eingeben (z. B. Zahlen), werden als **Literale** bezeichnet. Für deren Schreibweise gelten entsprechende Regeln, damit der Datentyp ersichtlich ist.

Beispiel: Literale für numerische Datentypen in Java

- Für numerische Werte des Datentyps `int` können Sie beispielsweise die Vorzeichen + bzw. - und die Ziffern 0 … 9 verwenden (das Vorzeichen + kann entfallen).
- Als Dezimaltrennzeichen bei Fließkommawerten verwenden Sie einen Punkt .

Beispiel: Literale für booleschen Datentyp in Java

Für die Eingabe eines logischen Wertes existieren lediglich die beiden Literale `true` (wahr) und `false` (falsch).

Beispiel: Literale für Zeichen-Datentyp in Java

- Einzelne Zeichen werden bei der Wertzuweisung durch Apostrophe ' eingeschlossen.
- Sie können ein Zeichen auch als Unicode-Escape-Sequenz darstellen. Die Unicode-Repräsentation ist dann in Apostrophe ' zu setzen. Escape-Sequenzen beginnen mit einem Backslash \. Die nebenstehende Tabelle zeigt eine Übersicht über die Escape-Sequenzen.
- Mit der Escape-Sequenz \u können Sie den Unicode für das gewünschte Zeichen direkt angeben.

Escape-Sequenz	Bedeutung
\u… Beispiel: **\u0045**	Ein spezielles Zeichen (im Beispiel das Zeichen `E` mit dem Code 0045)
\b	Backspace
\t	Tabulator
\n	line feed
\f	form feed
\r	carriage return
\"	Anführungszeichen
\'	Hochkomma
****	Backslash

6.5 Variablen und Konstanten

In einer Programmiersprache werden **Variablen** benötigt, um Daten darin zu speichern. Variablen haben einen Namen, über den sie im Programm angesprochen werden, z. B. `Kilometerstand`, um darin die Anzahl der gefahrenen Kilometer eines Autos zu speichern.

Bei der Namensgebung der Variablen sind die Vorgaben für Bezeichner einzuhalten. Oft gibt es zusätzliche Empfehlungen für die Namensgebung, z. B. durch die Programmiersprache selbst, durch Projektabsprachen oder Firmenrichtlinien.

In vielen Programmiersprachen können in Variablen nur Daten eines bestimmten Datentyps gespeichert werden. Ein Datentyp gibt an, welche Art von Werten in der Variablen gespeichert werden dürfen; im Beispiel der Variablen `Kilometerstand` sollen Zahlen gespeichert werden. Das ist beispielsweise in Java, C oder C# der Fall. Bei anderen Sprachen wie JavaScript oder PHP kann sich der Datentyp einer Variablen ändern.

Der Wert der Variablen kann sich im Laufe des Programms ändern, er ist variabel.

Gültigkeitsbereich von Variablen

Variablen können in einem Programm verschiedene Gültigkeitsbereiche besitzen, d. h., Sie können auf diese nur in bestimmten Programmbereichen zugreifen.

Lokale Variablen	Variablen, die innerhalb von Funktionen (vgl. Kapitel 9) und Anweisungsblöcken (vgl. Abschnitt 7.1) vereinbart werden, können Sie auch nur dort ansprechen. Die Lebensdauer lokaler Variablen endet mit dem Verlassen des Blocks, in dem sie deklariert wurden. In anderen Funktionen sind diese Variablen nicht bekannt.

Statische Variablen	Auch wenn der Gültigkeitsbereich der statischen Variablen verlassen wird, behalten diese den aktuellen Wert bei. Statische Variablen haben eine Lebensdauer über die gesamte Programmausführung hinweg. Sie ähneln globalen Variablen, sind aber in objektorientierten Sprachen an Klassen gebunden.
Externe Variablen	Wenn eine Variable als extern deklariert wird, bedeutet dies, dass die Variable in einer anderen Datei oder Bibliothek schon vorhanden ist und Sie diese nutzen möchten. Externe Variablen besitzen ebenfalls eine Lebensdauer über das gesamte Programm.
Globale Variablen	Dies sind Variablen, die einmalig außerhalb von Funktionen deklariert werden. Sie können in allen Funktionen angesprochen werden und besitzen eine globale Gültigkeit und eine Lebensdauer über die gesamte Programmausführung hinweg.

Nicht alle Programmiersprachen unterstützen jeden dieser Variablentypen. So kennt Java beispielsweise keine globalen Variablen. JavaScript hingegen kennt keine statischen Variablen.

Deklaration

In vielen Programmiersprachen muss eine Variable, bevor sie in einem Programm verwendet werden kann, deklariert werden. Mit der Deklaration wird der Name der Variablen festgelegt. In Sprachen mit einer strengen Typisierung wie Java, C oder C# wird zudem der Datentyp bestimmt und über den Datentyp der Speicherplatz festgelegt, den dieser Datentyp bei der Programmausführung benötigt.

Für die Deklaration gibt es einige Regeln.

- Alle Variablennamen müssen im jeweiligen Gültigkeitsbereich eindeutig sein. Sie können z. B. in Java denselben Variablennamen in mehreren Anweisungsblöcken verwenden, da sich die Gültigkeitsbereiche von aufeinander folgenden Anweisungsblöcken nicht überschneiden.
- Globale Variablen müssen hingegen einen eindeutigen Namen besitzen, da sie für das gesamte Programm gelten.

Einige Programmiersprachen besitzen eine implizite Typvereinbarung (etwa JavaScript oder PHP). Dabei werden Variablen ohne Typangabe deklariert. Der Typ der Variablen richtet sich dann nach dem Typ der ersten Wertzuweisung oder nach dem Anfangsbuchstaben ihres Namens. Sie sollten dennoch Ihre Variablen auch in diesen Sprachen explizit mit Typangabe deklarieren, um Fehler zu vermeiden. Beim Übersetzen des Programms können die Typangaben bereits auf fehlerhafte Zuweisungen hinweisen.

Wertzuweisung und Initialisierung

Wenn Sie eine globale, statische oder externe Variable deklarieren, wird vom Programm ein bestimmter Speicherbereich bereitgestellt, dessen Größe vom entsprechenden Datentyp abhängig ist.

Bevor Sie auf den Wert einer Variablen zugreifen können, müssen Sie dieser Variablen einen Wert zuweisen. Bei der Definition einer Variablen geschieht dies nicht immer automatisch. Die erste Wertzuweisung nach der Definition einer Variablen wird **Initialisierung** genannt. Bevor Sie eine Variable auf der rechten Seite einer Anweisung (vgl. Abschnitt 7.1) nutzen, muss dieser ein gültiger Wert zugewiesen worden sein. Um einer Variablen einen Wert zuzuweisen, verwenden Sie den **Zuweisungsoperator**, der je nach Sprache durch : = oder wie in Java, JavaScript oder C durch = dargestellt wird. Einer Variablen dürfen nur Werte zugewiesen werden, die dem vereinbarten Datentyp entsprechen.

Weisen Sie jeder Variablen einen Anfangswert zu, bevor Sie diese das erste Mal benutzen, auch wenn einige Programmiersprachen Variablen automatisch initialisieren. So beugen Sie Fehlern vor und können sicher sein, dass die Variable den gewünschten Wert für weitere Berechnungen hat.

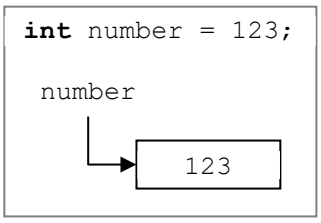
Möchten Sie einer Variablen den Wert einer anderen Variablen zuweisen, die einen anderen Datentyp besitzt, muss der Wert in den Zieldatentyp konvertiert werden. Dazu besitzen die Sprachen entsprechende Funktionen oder führen auch interne Typumwandlungen durch.

Beispiel: Variable in Java deklarieren und initialisieren

Syntax der Definition	`type identifier[, identifier1...];`
Syntax der Definition und Initialisierung	`type identifier = value;`

Es wird eine ganzzahlige Variable `number` deklariert und mit dem Wert `123` initialisiert. Dabei wird zunächst ein entsprechend großer Speicherplatz für die Variable reserviert und dann der Wert `123` dort eingetragen.

Über den Namen `number` können Sie später im Programm auf den gespeicherten Wert zugreifen.



Beispiel: *VariableDefinition.java*

```
//--richtige Definition --------
int number; //deklariert eine Variable number vom Typ int (Integer)
double price, size; // deklariert zwei Variablen (price, size) vom Typ
double
char c; // deklariert eine Variable c vom Typ char (Character)
//--fehlerhafte Deklaration --------
int &count; //Bezeichner von Variablen dürfen kein & enthalten
double a b c; //mehrere Variable müssen durch Kommata getrennt werden
```

Konstanten

Konstanten sind Variablen, die nicht verändert werden dürfen. Sobald während der Programmausführung eine Wertzuweisung erfolgt ist, ist diese endgültig. Sie können also keine zweite Wertzuweisung an eine Konstante vornehmen, nachdem diese initialisiert wurde.

Nicht alle Programmiersprachen besitzen Konstanten. So kennt JavaScript beispielsweise keine Konstanten.

Konstanten vereinfachen die Lesbarkeit und vermeiden Fehler: Der Wert einer Konstanten wird nur **einmal** eingegeben. Im weiteren Verlauf des Programms arbeiten Sie nur noch mit dem Namen der Konstanten. Sofern der Wert der Konstanten korrigiert werden muss, ist diese Änderung nur an einer einzigen Stelle im Quelltext vorzunehmen.

Wie für eine lokale Variable wird auch für jede Konstante Speicherplatz im Arbeitsspeicher Ihres Computers reserviert. Im Programm greifen Sie auf diesen Bereich über den Namen der Konstanten zu. Jede Konstante, die Sie in Ihrem Programm verwenden, muss vorher definiert werden, in Java mit dem Schlüsselwort `final`.

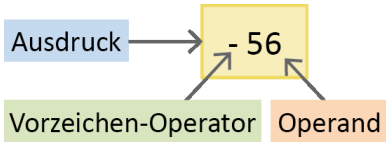
Beispiel: Konstante in Java verwenden

```
final double MWST = 0.19; // Deklaration und Initialisierung der Konstanten
MWST
... // Verwendung der Konstanten MWST in Berechnungen
```

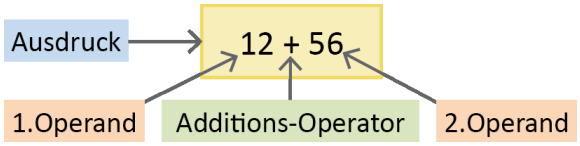
6.6 Operatoren

Operatoren sind Zeichen, die zusammen mit Operanden für die Berechnung eines Ausdrucks (engl. expression) bzw. die Ermittlung eines Wertes verwendet werden. Die meisten Sprachen besitzen unäre und binäre Operatoren. Ausdrücke lernen Sie im nachfolgenden Abschnitt kennen.

Unäre (einstellige) Operatoren werden auf nur **einen** Operanden angewendet. Der Vorzeichen-Operator '-' (Minus) ist ein unärer Operator. Er ändert das Vorzeichen des ihm folgenden Operanden. Beim Ausdruck -1 wird der Vorzeichen-Operator auf die Konstante 1 angewendet und diese so in eine negative Zahl umgewandelt.



Binäre (zweistellige) Operatoren werden auf **zwei** Operanden angewendet. Bei der Addition werden beispielsweise zwei Operanden über den Additions-Operator verknüpft.



Seltener gibt es auch **ternäre** (dreistellige) Operatoren. In den Sprachen C/C++, JavaScript oder Java existiert beispielsweise nur der Fragezeichen-Operator mit drei Operatoren. Liefert in der Anweisung `Ausdruck ? b : c` der Ausdruck den Wert wahr, wird der Wert von b zurückgegeben, anderenfalls der Wert von `c`.

Die nachfolgenden Beispiele für Operatoren verwenden explizit Java als Syntax.

Vorzeichen-Operatoren

Mit Vorzeichen-Operatoren lässt sich das Vorzeichen einer Zahl bzw. einer Variablen festlegen.

Operator	Name	Bedeutung	Initialisierung	Beispiel
+	Positives Vorzeichen	`+a` ist gleichbedeutend mit `a`.	`int a;` `a = 4;`	`a = +a;` `a` erhält den Wert `4`.
–	Negatives Vorzeichen	`-a` kehrt das Vorzeichen der Variable `a` um.	`int a;` `a = 4;`	`a = -a;` `a` erhält den Wert `-4`.

Arithmetische Operatoren

Diese Operatoren werden für mathematische Berechnungen in Ausdrücken eingesetzt. Als Operanden sind numerische Variablen oder Konstanten anzugeben. Arithmetische Operatoren liefern ein numerisches Ergebnis zurück.

Operator	Name	Bedeutung	Initialisierung	Beispiel
+	Addition	`a + b` ergibt die Summe von `a` und `b`.	`int a, b, c;` `a = 4; b = 2;`	`c = a + b;` `c` erhält den Wert `6`.
–	Subtraktion	`a - b` ergibt die Differenz von `a` und `b`.	`int a, b, c;` `a = 4; b = 2;`	`c = a - b;` `c` erhält den Wert `2`.
*	Multiplikation	`a * b` ist das Produkt aus `a` und `b`.	`int a, b, c;` `a = 4; b = 2;`	`c = a * b;` `c` erhält den Wert `8`.
/	Division	`a / b` ist der Quotient von `a` und `b`.	`int a, b, c;` `a = 4; b = 2;`	`c = a / b;` `c` erhält den Wert `2`.
% MOD	Modulo	`a % b` ist der Rest der ganzzahligen Division von `a` und `b`.	`int a, b, c;` `a = 5; b = 3;`	`c = a % b;` `c` erhält den Wert `2`.

In einigen Programmiersprachen (z. B. in C, JavaScript und Java) gibt es eine Kurzschreibweise, mit der eine Variable hoch- oder heruntergezählt werden kann.

`x = x + 1` (Inkrement) oder `x = x - 1` (Dekrement)

`++, --`	Präinkrement Prädekrement	`++a` erhöht und `--a` vermindert den Wert der Variablen `a` um `1` **vor** der weiteren Verwendung.	`int a, b, c;` `a = 5; b = 3;`	`c = ++a + b;` ① `c` erhält den Wert `9`. `a` erhält den Wert `6`. ① entspricht: 1. `a = a + 1;` 2. `c = a + b;`
	Postinkrement Postdekrement	`a++` erhöht und `a--` vermindert den Wert der Variablen `a` um `1` **nach** der Verwendung.	`int a, b, c;` `a = 5; b = 3;`	`c = a++ + b;` ② `c` erhält den Wert `8`. `a` erhält den Wert `6`. ② entspricht: 1. `c = a + b;` 2. `a = a + 1;`

Wenn Sie diese Kurzschreibweise als **einzelne Anweisung** verwenden, ersetzt sie den folgenden Programmcode (Post- und Präinkrement bzw. Post- und Prädekrement erzielen dabei jeweils das gleiche Ergebnis):

```
a++;      entspricht   ++a;      entspricht   a = a +1;
b--;      entspricht   --b;      entspricht   b = b-1;
```

Vergleichsoperatoren

Diese Operatoren, auch relationale Operatoren genannt, vergleichen Ausdrücke miteinander und liefern ein logisches Ergebnis, entweder `true` oder `false`.

Operator	Name	Bedeutung	Initialisierung	Beispiel
== (oder =)	Gleichheit	`a == b` ergibt `true`, wenn `a` und `b` gleich sind.	`int a, b;` `a = 4; b = 4;`	`a == b` liefert `true`
!= (oder <>)	Ungleichheit	`a != b` ergibt `true`, wenn `a` und `b` ungleich sind.	`int a, b;` `a = 4; b = 4;`	`a != b` liefert `false`
<	Kleiner	`a < b` ergibt `true`, wenn `a` kleiner `b` ist.	`int a, b;` `a = 3; b = 4;`	`a < b` liefert `true`
>	Größer	`a > b` ergibt `true`, wenn `a` größer `b` ist.	`int a, b;` `a = 3; b = 4;`	`a > b` liefert `false`
<=	Kleiner gleich	`a <= b` ergibt `true`, wenn `a` kleiner oder gleich `b` ist.	`int a, b;` `a = 4; b = 4;`	`a <= b` liefert `true`
>=	Größer gleich	`a >= b` ergibt `true`, wenn `a` größer oder gleich `b` ist.	`int a, b;` `a = 5; b = 4;`	`a >= b` liefert `true`

Einige Programmiersprachen kennen noch die **Identität** als Vergleichsoperator. In JavaScript werden dafür zum Beispiel drei Gleichheitszeichen (===) genutzt. Dabei werden sowohl eine Wert- als auch eine Datentypgleichheit überprüft. Die Umdrehung in JavaScript ist `!==`.

Logische Operatoren

Vergleichsoperatoren vergleichen Ausdrücke miteinander, deren Typen Zahlen oder logische Operatoren sein können. Mit logischen Operatoren werden Wahrheitswerte miteinander verknüpft. Das Ergebnis der Vergleichsoperation beider Operationen ist immer ein logischer Wert.

Die boolesche Logik basiert auf dem binären Zahlensystem und kann nur zwei Zustände annehmen: logisch 1 und logisch 0 – wahr und falsch. Abhängig von der Sprache werden logische Operatoren als Zeichen oder als Wort (Abkürzung) dargestellt. Die in der folgenden Tabelle zusammengestellten Operatoren sind nicht in allen Sprachen implementiert.

Operator	Name	Bedeutung	Beispiele
`!` (oder `not`)	Logisches NICHT Math. Zeichen: ¬	Ergibt die Umkehrung des Wahrheitswertes	`not true` → `false` `not false` → `true`

`&&` (oder `and`)	Logisches UND
	Math. Zeichen: ∧
		Ergibt `true`, wenn beide Operanden `true` sind, ansonsten `false`	`true && true` → `true`
			`true && false` → `false`
			`false && true` → `false`
			`false && false` → `false`

`||` (oder `or`)	Logisches ODER
	Math. Zeichen: ∨
		Ergibt `true`, wenn mindestens einer der beiden Operanden `true` ist	`true || true` → `true`
			`true || false` → `true`
			`false || true` → `true`
			`false || false` → `false`

`^` (oder `xor`)	Exklusives ODER	Ergibt `true`, wenn **nur einer** der beiden Operanden `true` ist	`true ^ true` → `false`
			`true ^ false` → `true`
			`false ^ true` → `true`
			`false ^ false` → `false`

`imp`	Implikation
	Math. Zeichen: →
		Aus `a` folgt `b`, d. h., wenn der erste Operand `true` ist, muss auch der zweite Operand `true` sein; wenn der erste Operand `false` ist, ist der Wert des zweiten Operanden nicht relevant.	`true imp true` → `true`
			`true imp false` → `false`
			`false imp true` → `true`
			`false imp false` → `true`

`eqv`	Äquivalenz
	Math. Zeichen: ↔
		Ergibt `true`, wenn beide Operanden übereinstimmen	`true eqv true` → `true`
			`true eqv false` → `false`
			`false eqv true` → `false`
			`false eqv false` → `true`

Die beiden logischen Operatoren `imp` und `eqv` existieren nicht in Java.

Bitweise Operatoren

Verschiedene Programmiersprachen wie C/C++ besitzen Bit-Operatoren, die eine hardwarenahe Programmierung unterstützen. Aber auch JavaScript und Java besitzen diese Operatoren, wobei diese dort nicht für hardwarenahe Programmierung verwendet werden (können). Eingesetzt werden Bit-Operationen beispielsweise auch für die Optimierung von Multiplikation und Division. Zum Beispiel kann statt einer Multiplikation mit 2 eine einfache Linksverschiebung der Bits verwendet werden, die schneller abläuft als die arithmetische Operation.

Beispiel

Im Folgenden wird die Zahl $(10)_{10}$ mit dem Wert 2 multipliziert. Die Bitfolge $(01010)_2$ wird dabei um ein Bit nach links verschoben, und es wird eine 0 angehängt => $(10100)_2$ (entspricht $(20)_{10}$).

Operator	Name	Bedeutung	Beispiel

~	Einerkomplement bitweise Negation	Invertiert alle Bits des Operanden	`int B1 = 153; 10011001`
			`B1 = ~B1; 01100110`
			B1 erhält den Wert 102.

|	ODER	a | b verknüpft die jeweiligen Bits von a und b nach der ODER-Logik.	`int B3;`
			`int B1 = 145; 10010001`
			`int B2 = 133; 10000101`
			`B3 = B1 | B2; 10010101`
			B3 erhält den Wert 149.

&	Bitweises UND	a & b verknüpft die jeweiligen Bits von a und b nach der UND-Logik.	`int B3;`
			`int B1 = 145; 10010001`
			`int B2 = 133; 10000101`
			`B3 = B1 & B2; 10000001`
			B3 erhält den Wert 129.

^	Bitweises Exklusiv-ODER	a ^ b verknüpft die jeweiligen Bits von a und b nach der XOR-Logik.	`int B3;`
			`int B1 = 145; 10010001`
			`int B2 = 129; 10000001`
			`B3 = B1 ^ B2; 00010000`
			B3 erhält den Wert 16.

>>	Rechtsschieben	a >> x verschiebt die Bits der Variablen a um x Stellen nach rechts. Dies entspricht der Division durch 2^x bei vorzeichenlosen Zahlen (Überlauf beachten!).	`int B1 = 144; 10010000`
			`B1 = B1 >> 2; 00100100`
			B1 erhält den Wert 36.

<<	Linksschieben	a << x verschiebt die Bits der Variablen a um x Stellen nach links. Dies entspricht der Multiplikation mit 2^x bei vorzeichenlosen Zahlen (Überlauf beachten!).	`int B1 = 84; 01010100`
			`B1 = B1 << 1; 10101000`
			B1 erhält den Wert 168.

>>>	Rechtsschieben ohne Vorzeichen	a >>> x verschiebt die Bits der Variablen a um x Stellen nach rechts, wobei das niederwertigste Bit mit 0 gefüllt wird.	`int B1 = 145; 10010001`
			`B1 = B1 >>> 1;01001000`
			B1 erhält den Wert 72.

Beachten Sie, dass die Auswertung eines Ausdrucks, der logische Operatoren verwendet, in einigen Sprachen abgebrochen wird, wenn sich am Gesamtergebnis nichts mehr ändern kann. Dies ist insbesondere dann zu beachten, wenn innerhalb der Ausdrücke Methoden ausgeführt werden.

Zuweisungsoperatoren

Um einer Variablen einen Wert zuzuweisen, wird der Zuweisungsoperator verwendet. Der Wert, der einer Variablen zugewiesen wird, ergibt sich meist aus einem Ausdruck. Sie können einer Variablen in der Regel nur Werte des Datentyps zuweisen, den sie selbst besitzt.

Operator	Bedeutung	Beispiele

= (oder :=)	Der Variablen auf der linken Seite dieser Anweisung wird der Wert des Ausdrucks der rechten Seite zugewiesen.	`int zahl;`
		`boolean richtig;`
		`char zeichen;`
		`zahl = 123 * 5;`
		`richtig = zahl > 100;`
		`zeichen = 'b';`

Bei einigen Programmiersprachen werden Zuweisungsoperatoren mit anderen Operatoren gekoppelt, um den Schreibaufwand zu verringern.

So kann z. B. in C, JavaScript und Java statt `x = x + 72;` die Kurzschreibweise `x += 72;` verwendet werden.

Operatoren	Bedeutung	Initialisierung	Beispiel

`+=` `-=` `*=`
`/=` `&=` `%=`
`|=` `^=`
`<<=` `>>=` `>>>=`
	`a *= b` weist der Variablen `a` den Wert von `a * b` zu und liefert `a * b` als Rückgabewert. Dies ist gleichzusetzen mit der Funktion `a = a * b`.
	Diese Funktionsweise ist bei all diesen Operatoren gleich.
		`int a;`
		`int b;`
		`a = 4;`
		`b = 5;`
			`a *= b;`
			
			`a` erhält den Wert `20`.

6.7 Ausdrücke

In den meisten Programmiersprachen gehören Ausdrücke zu den kleinsten ausführbaren Einheiten eines Programms. Sie werden verwendet, um Variablen einen Wert zuzuweisen, numerische Berechnungen durchzuführen oder logische Bedingungen zu formulieren.

Ein Ausdruck ist eine gültige Kombination von Operanden und Operatoren, die ein Ergebnis liefert. Bei der Anwendung der Operatoren gilt wie in der Mathematik die Regel „Punktrechnung geht vor Strichrechnung“, geklammerte Ausdrücke werden zuerst ausgewertet.

Beispiele

`a`	Auch eine Variable ist ein Ausdruck.
`a*(b+c)`	Zuerst wird die Klammer berechnet und danach das Ergebnis mit dem Wert `a` multipliziert.
`((a>100)&&(b<1000))||(c>200)`	Zuerst werden die Ausdrücke `a>100` und `b<1000` mit dem logischen Operator `&&` (UND) verknüpft. Dieses Ergebnis wird dann mit dem Ergebnis des Ausdrucks `c>200` über den `||`-Operator (ODER) verknüpft.

Rangfolge der Operatoren

Oft ist es notwendig, mehr als zwei Operanden miteinander zu vergleichen, z. B.:

```
(amount1 > 500) || (amount2 < 801) && !(amount3 == 400)
```

Die Abarbeitungsreihenfolge für diese Anweisung kann in verschiedenen Programmiersprachen unterschiedliche Ergebnisse liefern. Das liegt daran, dass der Ausdruck nach bestimmten Vorrangregeln für die Operatoren abgearbeitet wird. Einige Sprachen arbeiten die Ausdrücke prinzipiell von links nach rechts ab.

Wollen Sie sichergehen, dass die Ausdrücke in der richtigen Reihenfolge abgearbeitet werden, setzen Sie Klammern. Wie in der Mathematik werden die Bedingungen in Klammern **immer** zuerst ausgewertet. Auch die Lesbarkeit Ihres Programms wird durch das Setzen von Klammern erhöht.

6.8 Übungen

Übung 1: Arbeiten mit grundlegenden Sprachelementen

Übungsdatei: -- **Ergebnisdatei: *uebung06.pdf***

1. Was verstehen Sie unter einer Variablen, und welche Gültigkeitsbereiche gibt es?
2. Welche Zahlenart kann eine mit int (integer) deklarierte Variable in Java darstellen?
3. Welche Zuweisungen sind nicht korrekt?

Initialisierungen	Zuweisungen

`int i = 0;`
`int j = 0;`
`short k = 0;`
`char b = 'a';`
	`i = 256;`
	`j = -42314;`
	`k = j;`
	`b = "Hallo";`

4. Zwei Variablen, var1 und var2, vom Datentyp char (character) sind gegeben. Die Werte der beiden Variablen sollen vertauscht werden. Beschreiben Sie den Algorithmus im Pseudocode.
5. Stellen Sie die arithmetischen, logischen und Vergleichsoperatoren in einem Syntaxdiagramm und in der erweiterten BNF dar. Fassen Sie dabei unäre und binäre Operatoren in je einer Darstellung zusammen. Verwenden Sie jeweils eines der möglichen Zeichen pro Operator. Erzeugen Sie das Syntaxdiagramm und die erweiterte BNF für einen Ausdruck. Verwenden Sie dabei unäre und binäre Operatoren.
6. Werten Sie folgende Bedingungen aus:
   - (value1 > 500) || (value2 < 800) && !(value3 == 400)
   - (value1 > 500) || ((value2 < 800) && !(value3 == 400))

   Die Variablen sollen folgende Werte enthalten:

   value1 = 501; value2 = 799; value3 = 400

   In den Bedingungen wird angenommen, dass die Abarbeitung von links nach rechts erfolgt.

Übung 2: Zwei Variablen vertauschen

Übungsdatei: -- **Ergebnisdatei: *ChangeValues.java***

1. Setzen Sie den Pseudocode zum Vertauschen der zwei Variablen, var1 und var2, vom Datentyp char (character) in Java um.

   Mit dem Befehl System.out.println (var1); können Sie den Wert einer Variablen ausgeben. Um verschiedene Werte zusammen mit Text auszugeben, verwenden Sie den Befehl wie folgt: System.out.println("Wert der ersten Variablen" + var1 + "hier kann noch mehr Ausgabetext stehen." + var2);

7 Kontrollstrukturen

In diesem Kapitel erfahren Sie

- wie Sie das Ausführen von Anweisungen von Bedingungen abhängig machen
- welche Möglichkeiten der Fallauswahl es gibt
- wie Sie Anweisungen wiederholt ausführen können

Voraussetzungen

- Arbeiten mit Variablen, Operatoren
- Verwendung der Datentypen

7.1 Anweisungen und Folgen

Was sind Anweisungen?

Ein Programm setzt sich aus einer Menge von Anweisungen zusammen, um eine bestimme Aufgabe zu lösen. Anweisungen arbeiten dabei mit den im Programm festgelegten Variablen und führen damit Operationen aus. Die einzelnen Programmiersprachen stellen unterschiedliche Anweisungen zur Verfügung; dabei finden sich jedoch folgende Grundstrukturen wieder:

- Einfache Anweisungen, z. B. Zuweisung
- Verzweigungen
- Schleifen

Einfache Anweisungen

Eine Anweisung (engl. statement) ist die kleinste ausführbare Einheit eines Programms. In Java, JavaScript und vielen anderen Programmiersprachen wird eine Anweisung mit einem Strichpunkt (Semikolon) abgeschlossen. In einigen Sprachen wird pro Zeile eine Anweisung geschrieben. Ist eine Anweisung so lang, dass sie nicht in eine Zeile passt, wird sie in der nächsten Zeile fortgesetzt. Einige Programmiersprachen verlangen, dass diese Fortsetzungszeile dann mit einem speziellen Fortsetzungszeichen beginnt.

Die folgenden Beispiele zeigen die Programmierung unter Verwendung von Java oder JavaScript, d. h. dass das Anweisungsende entsprechend mit einem Semikolon abgeschlossen wird.

Die leere Anweisung

Die einfachste Form der Anweisung ist die **leere Anweisung**.

Syntax	Bedeutung
`;`	Eine leere Anweisung besteht in Java oder JavaScript nur aus dem Semikolon und hat keinerlei Auswirkung auf das laufende Programm. Eine leere Anweisung kann dort verwendet werden, wo syntaktisch eine Anweisung erforderlich ist, aber von der Programmlogik her nichts zu tun ist.

Anweisungsblöcke

Bei einer Reihe von syntaktischen Sprachkonstrukten darf nicht mehr als eine Anweisung angegeben werden. Die Logik des Programms kann es jedoch erfordern, dass eine Folge von zwei oder mehr Anweisungen ausgeführt werden muss. In diesen Fällen muss ein **Anweisungsblock** (Verbundanweisung) verwendet werden.

In den nachfolgenden Beschreibungen der Syntax wird jeweils nur der Begriff `statement` verwendet. Dies kann dann eine einzelne Anweisung oder ein Anweisungsblock sein. Da eine einzelne Anweisung das Semikolon beinhaltet, wird hinter dem Begriff `statement` **kein** Semikolon geschrieben.

Syntax	Bedeutung

`{`
`  statement1`
`  statement2`
`}`
	Ein Anweisungsblock ist eine Zusammenfassung von Anweisungen, die nacheinander ausgeführt werden. Alle im Block zusammengefassten Anweisungen gelten in ihrer Gesamtheit als eine einzelne Anweisung. Ein Block wird in Schlüsselwörter oder in definierte Zeichen eingeschlossen. In Java wird ein Block in geschweifte Klammern { } eingeschlossen. Einige Sprachen verwenden die Worte `begin` und `end`. Andere Sprachen wie Python nutzen Einrückungen zur Markierung eines Blocks; die Funktion eines Blocks ist jedoch identisch.

Wenn Sie eine Berechnung durchführen oder eine Eingabe eines Anwenders speichern möchten, erfolgt dies über die Zuweisungsanweisung.

Syntax	Bedeutung
`i = i + 1;`	Einer Variablen wird ein berechneter Wert des Ausdrucks `i + 1` zugewiesen.

Die Ausgabe von Werten beispielsweise auf dem Bildschirm können Sie über Ausgabeanweisungen realisieren.

Syntax	Bedeutung
`System.out.println("text");`	Die Ausgabe eines Wertes oder Textes erfolgt über eine von der Sprache vorgegebene Ausgabe-Routine. (Zur Eingabe werden entsprechende Eingabeanweisungen verwendet.)

Diese Arten von Anweisungen werden auch **einfache Anweisungen** genannt.

Folgen

Eine **Folge**, auch **Sequenz** genannt, ist eine Liste von Anweisungen, die nacheinander (sequenziell) abgearbeitet wird.

Beispiel: Programm zur Berechnung des Bruttopreises – *Amount.java*

Der Nettopreis wird im Beispiel vorgegeben. Daraus wird der Bruttopreis berechnet und ausgegeben.

```
{
  double netAmount = 7.84;  //Nettopreis
  final double VAT = 0.19;  //Mehrwertsteuer
  double totalAmount = netAmount * (1 + VAT);
  System.out.println("Berechneter Bruttopreis: " + totalAmount);
}
```

Ausschnitt aus dem Java-Beispielprogramm

7.2 Bedingungen und Kontrollstrukturen

Was sind Bedingungen?

Durch eine Bedingung (engl. condition) wird der Ablauf eines Programms beeinflusst. Es wird ein logischer Ausdruck ausgewertet. Für die Formulierung solcher Bedingungen stehen die Vergleichsoperatoren und die logischen Operatoren zur Verfügung. Eine Bedingung kann entweder mit "Ja" beantwortet werden (`true` bzw. wahr) oder mit "Nein" (`false` bzw. falsch). Vom Ergebnis der Bedingung ist abhängig, welche Anweisungen des Programms danach abgearbeitet werden und welche nicht.

Beispiele

Es wird mithilfe des Modulo-Operators geprüft, ob eine Zahl gerade ist:	`(Zahl % 2) == 0`
Es wird geprüft, ob eine Zahl zwischen 100 und 500 liegt (ohne die Werte 100 und 500):	`(Zahl > 100) && (Zahl < 500)`
Es wird geprüft, ob ein Zeichen eine Ziffer ist:	`(Zeichen >= '0') && (Zeichen <='9')`
Es wird geprüft, ob der Mitarbeiter "Meier" seine 30 Urlaubstage bereits angebrochen hat:	`(Mitarbeiter == "Meier") && (Urlaubstage < 30)`

Kontrollstrukturen im Überblick

Sollen Programmteile mehrmals oder gar nicht ausgeführt werden, werden Sprachelemente benötigt, mit denen der Programmablauf gesteuert werden kann, sogenannte **Kontrollstrukturen**. Die Entscheidung, nach welchen Kriterien der Ablauf gesteuert wird, wird in **Bedingungen** formuliert.

Es werden zwei Gruppen von Kontrollstrukturen unterschieden:

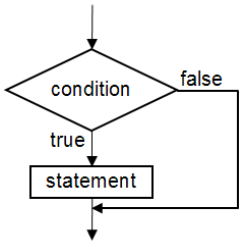
Verzweigungen	Es werden alternative Programmteile angeboten, in die – abhängig von einer Bedingung – beim Programmablauf verzweigt wird.
Schleifen (Wiederholung)	Ein Programmteil kann – abhängig von einer Bedingung – mehrmals durchlaufen werden (Schleife = engl. loop) oder gar nicht.

7.3 Grundlagen zu Verzweigungen

Eine Verzweigung kann eine der folgenden Ausprägungen besitzen:

Bedingte Anweisung („Einseitige“ Verzweigung)	Je nachdem, ob eine **Bedingung** erfüllt ist oder nicht, wird ein Programmteil ausgeführt oder übersprungen.
Verzweigung	In Abhängigkeit vom Ergebnis einer **Bedingung** wird ein Programmteil oder ein anderer Programmteil ausgeführt.
Geschachtelte Verzweigung	Innerhalb der alternativen Programmteile einer bedingten Anweisung oder einer Verzweigung wird eine weitere Verzweigung integriert. Mit einer solchen geschachtelten Verzweigung lassen sich mehrere Alternativen bereitstellen.
Mehrfache Verzweigung	Es wird ein **Ausdruck** ausgewertet. Je nachdem, welchem Wert der Ausdruck entspricht, verzweigt das Programm fallweise (**case**) in einen entsprechenden Programmteil. Eine solche Kontrollstruktur kann somit ebenfalls mehrere Alternativen bereitstellen.

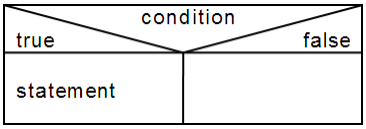
7.4 Bedingte Anweisung



PAP

Die bedingte Anweisung ist dadurch gekennzeichnet, dass nach einer Bedingungsabfrage durch einen Ausdruck eine Anweisung oder ein Anweisungsblock (mehrere Anweisungen) ausgeführt wird oder nicht.

Wenn die Bedingung mit "Ja" (`true`) beantwortet wird, **dann** wird die Anweisung (statement) ausgeführt. Wird die Bedingung mit "Nein" beantwortet (`false`), erfolgt keine Aktion des Programms. In beiden Fällen wird nach dem Ende der Alternative mit der nächsten Anweisung fortgesetzt. Die bedingte Anweisung entspricht in den meisten Programmiersprachen einer `if`-Anweisung mit alternativem Zweig.



Struktogramm

Syntax zur bedingten Anweisung in Java

- Eine bedingte Anweisung wird mit dem Schlüsselwort `if` eingeleitet.
- Nach dem Schlüsselwort `if` wird in Klammern ( ) als Bedingung ein logischer Ausdruck angegeben, dessen Auswertung `true` oder `false` sein muss.
- Ist die Bedingung erfüllt (`true`), wird die folgende Anweisung ① bzw. der folgende in geschweiften Klammern eingeschlossene Anweisungsblock ② ausgeführt.
- Die Anweisungen innerhalb des Blocks werden eingerückt, um die Programmstruktur besser zu verdeutlichen.

```
if (condition)
  statement ①
```

```
if (condition)
{
  statement ②
  ...
}
```

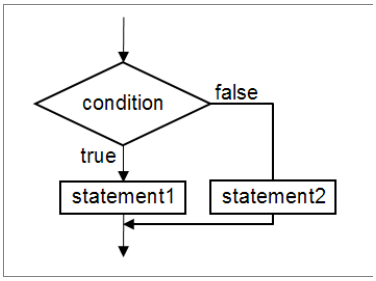
Beispiel: *Discount.java*

Wenn (`if`) ein Kunde einen Auftrag höher als 1000,- EUR erteilt, bekommt er 3 % Rabatt und der neue Preis wird berechnet. Bei Aufträgen bis 1000,- EUR wird kein Rabatt gewährt. Die Überprüfung der Bedingung kann nur das Ergebnis "ja" bzw. "nein" ergeben, in Java entsprechend `true` oder `false`.

```
...
double price = 1497.56;
if (price > 1000)
  price = price * (1 - 0.03);
...
```

Java

7.5 Verzweigung

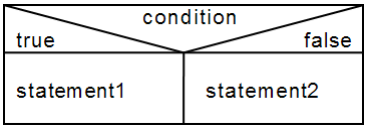


PAP

Bei einer sogenannten Verzweigung (auch Alternative genannt) wird einer von zwei möglichen Anweisungsblöcken in Abhängigkeit von einer Bedingung ausgeführt.

Wenn die Bedingung erfüllt ist, **dann** wird Anweisungsblock `A` (`statement1`) ausgeführt, **sonst** wird Anweisungsblock `B (statement2)` ausgeführt. Die Verzweigung entspricht in den meisten Programmiersprachen der `if`-Anweisung mit `else`-Zweig.

Sie können die Verzweigung verwenden, wenn es genau **zwei** Auswahlmöglichkeiten gibt.



Struktogramm

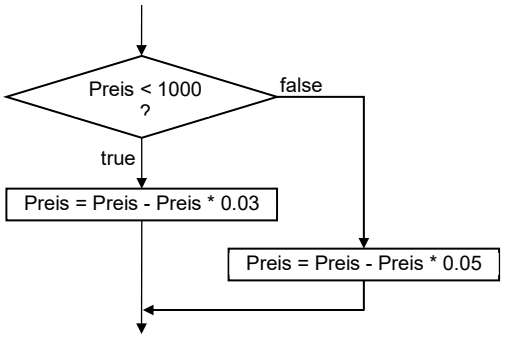
Syntax zur Verzweigung in Java

- Nach der Bedingung in der `if`-Anweisung und der Anweisung ① folgt das Schlüsselwort `else`.
- Nach `else` schließt sich die Anweisung ② an, die ausgeführt wird, wenn die Bedingung in der `if`-Anweisung nicht zutrifft (Ausdruck liefert `false`).

```
if (condition)
  statement1 ①
else
  statement2 ②
```

Beispiel: *Discount2.java*

Bei einem Einkaufswert von weniger als 1000 EUR wird ein Rabatt von 3 % auf den Einkaufspreis gewährt. Liegt der Einkaufspreis bei 1000 EUR oder mehr, erhält der Kunde 5 % Rabatt. Der neue Preis wird berechnet.



PAP

```
...
if (price < 1000)
{
  price = price - price * 0.03;
}
else
{
  price = price - price * 0.05;
}
...
```

Java

7.6 Geschachtelte Verzweigung

Innerhalb eines `if`-Blocks oder eines `else`-Blocks dürfen wiederum `if`-Anweisungen stehen, d. h., sie werden geschachtelt. Damit kann die Ausführung von Anweisungen von mehreren Bedingungen abhängig gemacht werden. Zur Vereinfachung geschachtelter `if`-Anweisungen bieten Programmiersprachen oft den `else-if`-Block.

Ist die erste Bedingung wahr, wird kontrolliert, ob auch die nächste Bedingung wahr ist, usw.

```
if (condition1)
{
  statement1
  if (condition2)
    statement2
  else
    statement3
}
else
  statement4
```

```
if (condition1)
  statement1
else if (condition2)
       statement2
     else
       statement3
}
```

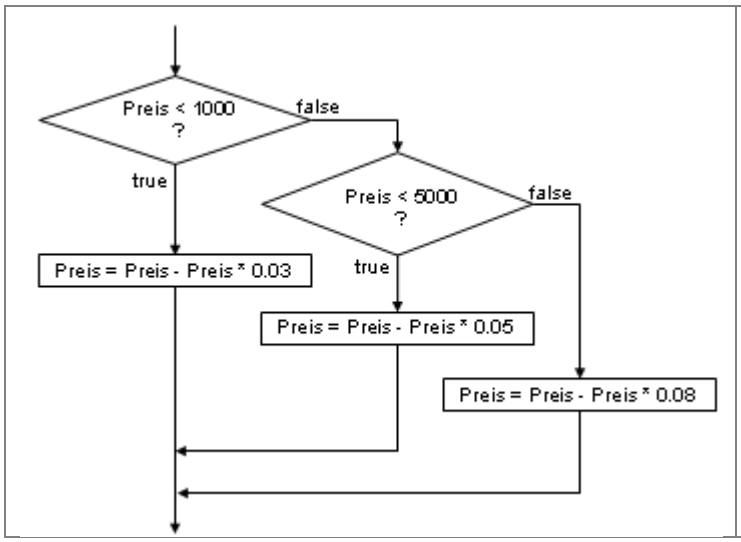
```
if (condition1)
  statement1
else
{
  statement2
  if (condition2)
    statement3
  else
    statement4
}
```

Möglichkeiten für geschachtelte Verzweigungen in Java

Für die geschachtelten `if`-Anweisungen gelten die gleichen Regeln wie für eine einfache `if`-Anweisung.

Beispiel: *Discount3.java*

Sie können die mehrfache Verzweigung verwenden, wenn es **mehr als zwei** Auswahlmöglichkeiten gibt.
Das Beispiel zur Berechnung des Rabattes wird dazu noch etwas erweitert. Bei einem Einkaufswert von weniger als 1000 EUR wird ein Rabatt von 3 % auf den Einkaufspreis gewährt. Liegt der Einkaufspreis bei 1000 EUR oder mehr, erhält der Kunde 5 % Rabatt. Ab 5000 EUR erhält er 8 % Preisnachlass.



PAP

```
...
if (price < 1000)
{
  price = price - price * 0.03;
}
else if (price < 5000)
  {
    price = price - price * 0.05;
  }
  else
  {
    price = price - price * 0.08;
  }
...
```

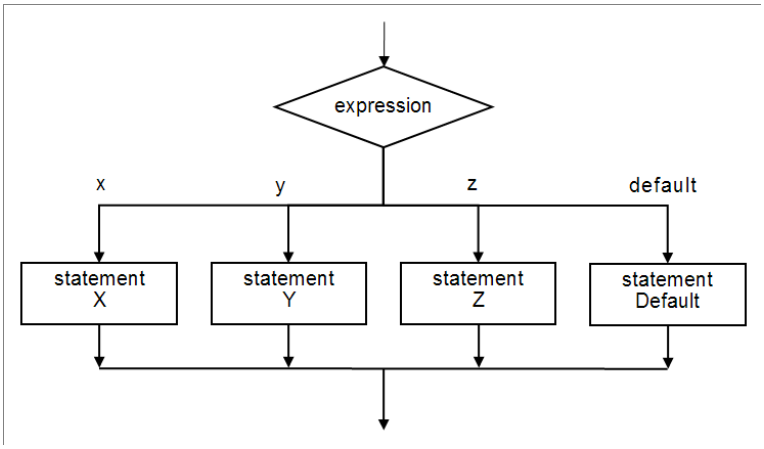
Java

- Kennzeichnen Sie eine geschachtelte Verzweigung immer durch eine Einrückung, damit die Struktur auch optisch erkennbar ist.
- Vermeiden Sie, wenn möglich, geschachtelte Verzweigungen, die mehr als drei Stufen umfassen. Ihr Programm wird sonst zu unübersichtlich und ist schlecht lesbar.

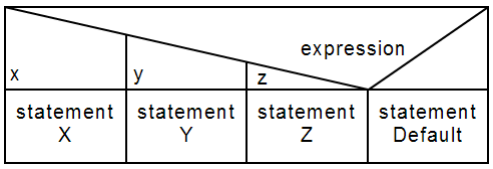
7.7 Mehrfache Verzweigung (Fallauswahl)

Bei einer mehrfachen Verzweigung wird der Wert einer Variablen ausgewertet und in Abhängigkeit von diesem Wert eine Anweisung bzw. ein Anweisungsblock ausgeführt. Die Variable, deren Inhalt geprüft werden soll, wird auch als Selektor bezeichnet. In der Programmierung wird auch von einer **Fallauswahl** (oder Selektion) gesprochen.

Ist der Wert der auszuwertenden Variablen gleich dem ersten Wert (in der Abbildung: `x`), wird die zugehörige Anweisung (hier: `statementX`) ausgeführt. Entspricht der Variablenwert dem zweiten Wert (`y`), wird Anweisung (`statementY`) ausgeführt usw. Ist keiner der Werte gleich dem Wert der Variablen, wird der Default-Block `statementDefault` abgearbeitet.



PAP



Struktogramm

Syntax zur mehrfachen Verzweigung in Java oder JavaScript

- Die mehrfache Fallauswahl wird mit dem Befehl `switch` eingeleitet.
- Danach folgt in Klammern der Selektor, ein Ausdruck (expression), der ordinal (abzählbar) sein muss, z. B. vom Typ `int`. Der Wert des Selektors bestimmt, welcher Auswahlblock ausgeführt wird.
- Jeder Auswahlblock beginnt in Java mit `case` und einem konstanten Wert, der den gleichen Typ hat wie der Wert des Ausdrucks im Selektor.
- Stimmt der Wert des Selektors mit einem der aufgeführten Auswahlwerte überein, wird der Anweisungsblock hinter dem Wert ausgeführt. Damit in Java bzw. JavaScript nicht auch die nachfolgenden Auswahlblöcke ausgeführt werden, schließen Sie einen Auswahlblock mit `break` ab.

```
switch (expression)
{
  case constantValue1:
statement1

break;
  case constantValue2:
statement2

break;
  ...
  [default: statement]
}
```

- Die Überprüfung des Auswahlwertes wird nur einmal durchgeführt.
- Stimmt der Wert des Selektors mit keinem Auswahlwert überein, werden die Anweisungen des `default`-Zweigs ausgeführt. Wird dieser Defaultblock weggelassen, führt das Programm die nächste Anweisung nach dem Ende der Fallauswahl durch.

Bei der Fallauswahl sind in einigen Programmiersprachen Abweichungen von dieser Syntax zu beachten.

Beispiel: *Grading.java*

Ein Programm soll jeweils für eine Schulnote (grade) eine entsprechende Beurteilung als Text ausgeben.

Eine gut strukturierte Fallauswahl ist dadurch gekennzeichnet, dass

- die Fälle nach ihrer Häufigkeit angeordnet sind. Der am häufigsten vorkommende Fall steht an erster Stelle, der seltenste Fall am Ende. Dadurch wird die Fallauswahl schneller abgearbeitet;
- alle Auswahlfälle, die nicht in ihrer Häufigkeit vergleichbar sind, alphabetisch oder numerisch geordnet sind;
- der letzte `default`-Zweig (auch otherwise-Zweig genannt) für unvorhergesehene Fälle und Fehlermeldungen verwendet wird. Trifft keine der vorherigen Auswahlmöglichkeiten zu, kann ein Fehler über den `default`-Zweig abgefangen werden.
- die Anweisungen innerhalb von Auswahlblöcken möglichst kurz und einfach gehalten sind. So bleibt die Struktur leichter lesbar und verständlich.

```
...
int grade = 4; //Note
String text = "";
switch (grade)
{
  case 1:  text = "sehr gut"; break;
  case 2:  text = "gut"; break;
  case 3:  text = "befriedigend"; break;
  case 4:  text = "ausreichend"; break;
  case 5:  text = "mangelhaft"; break;
  case 6:  text = "ungenügend"; break;
  default: text = "FEHLER";
}
System.out.println(grade + " entspricht " + text);
...
```

Java

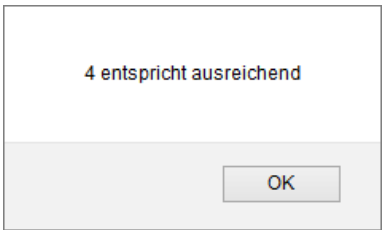
Beispiel: *Grading.html*

Das Beispiel in JavaScript sieht wie folgt aus und zeigt die große syntaktische Ähnlichkeit zwischen Java und JavaScript (oder allgemein Programmiersprachen mit Ursprung in C). In diesem Ausschnitt eines Quelltextes sind nur die Deklaration der Variablen (in JavaScript ohne Typangabe) und die Ausgabeanweisung unterschiedlich.

```
...
    <script type="text/javascript">
      grade = 4; //Note
      text = "";
      switch (grade)
      {
        case 1:  text = "sehr gut"; break;
        case 2:  text = "gut"; break;
        case 3:  text = "befriedigend"; break;
        case 4:  text = "ausreichend"; break;
        case 5:  text = "mangelhaft"; break;
        case 6:  text = "ungenügend"; break;
        default: text = "FEHLER";
      }
      alert(grade + " entspricht " + text);
    </script>
...
```

JavaScript

Ein Dialog zeigt im Browser das Ergebnis des JavaScripts an:



Mehrfache oder geschachtelte Verzweigung?

Die mehrfache Verzweigung ist dann sinnvoll, wenn Sie eine einzige Variable auf unterschiedliche Werte abfragen. Die mehrfache Verzweigung ist in diesem Fall übersichtlicher als eine geschachtelte Verzweigung und vermeidet so Programmierfehler. Die Anwendung der mehrfachen Verzweigung ist aber nicht immer möglich, da für den Selektor nur ordinale Datentypen, z. B. Integer, zulässig sind. Reelle Zahlen und Zeichenketten sind für den Selektor nicht erlaubt. Sie können aber alle Verzweigungskonstrukte ineinander verschachteln oder kombinieren, wenn es die Aufgabenstellung erfordert.

Beispiel: *Ticket.java*

Ein Fahrkartenautomat gibt Einzel-, Wochen- und Monatskarten für die Tarifzonen 1 – Kurzstrecke und 2 – Normal aus. Das folgende Programm simuliert solch einen Automaten und berechnet die Preise. Es wird die folgende Vorgehensweise vorgeschlagen:

① Die Tarifzone und die Fahrkartenart werden eingegeben.

② Es muss geprüft werden, ob die Tarifzone 1 oder 2 ist, denn nur das sind gültige Zonen. Ist die Zone ungültig, wird eine Fehlermeldung ausgegeben und das Programm verlassen.

③ Ist die Tarifzone gültig, wird für Tarifzone 1 der Grundpreis von 2.00 Euro festgelegt und bei Tarifzone 2 ein Grundpreis von 2.80 EUR.

④ Nun wird die Fahrkartenart überprüft.

1: Normalfahrschein – es wird der Grundpreis ausgegeben.

2: Wochenkarte – der Grundpreis wird mit 7 multipliziert und ausgegeben.

3: Monatskarte – der Grundpreis wird mit 30 multipliziert und ausgegeben.

Stimmt die Fahrkartenart nicht mit einem der drei Werte überein, erfolgt eine Fehlermeldung.

```
...
① int region = 1;
int art = 2;
double price = 0.0;
② if ((region == 1) || (region == 2))
{
③   if (region == 1)
    {
      price = 2.00;
    }
    else
    {
      price = 2.80;
    }
④   switch (art)
    {
      case 1:  System.out.println(price);
       break;
      case 2:  System.out.println(price * 7);
           break;
      case 3:  System.out.println(price * 30);
               break;
      default: System.out.println("falsche Fahrkartenart" );
    }
}
else
{
  System.out.println("falsche Tarifzone");
}
```

Java

```
...
① region = 1;
art = 2;
price = 0.0;
② if ((region == 1) || (region == 2))
{
③   if (region == 1)
    {
      price = 2.00;
    }
    else
    {
      price = 2.80;
    }
```

```
④   switch (art)
    {
      case 1:  alert(price);
       break;
      case 2:  alert(price * 7);
           break;
      case 3:  alert(price * 30);
               break;
      default: alert("falsche Fahrkartenart" );
    }
}
else
{
  alert("falsche Tarifzone");
}
...
```

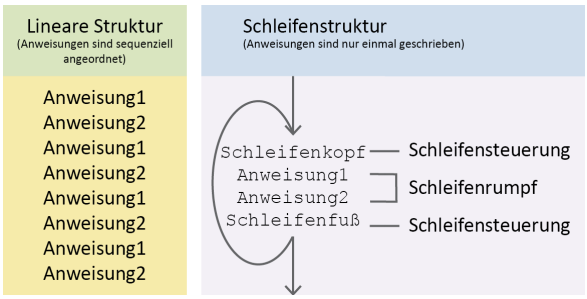
JavaScript

7.8 Schleifen

Zur Umsetzung von Algorithmen müssen die gleichen Anweisungen meist mehrmals wiederholt werden. Häufig ist es dabei auch nicht vorhersehbar, wie oft diese Anweisungen ausgeführt werden sollen. In jeder Programmiersprache gibt es verschiedene Strukturen, die eine wiederholte Ausführung von Anweisungen ermöglichen. Diese Kontrollstrukturen werden als **Schleifen** bezeichnet.

Eine Schleife besteht aus einer **Schleifensteuerung** und einem **Schleifenrumpf**. Die Schleifensteuerung gibt an, wie oft oder unter welcher Bedingung die Anweisungen abgearbeitet werden. Innerhalb der Schleifensteuerung befindet sich der Schleifenrumpf, der die zu wiederholenden Anweisungen enthält.

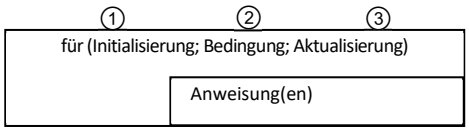
Drei Arten von Schleifenstrukturen werden unterschieden:



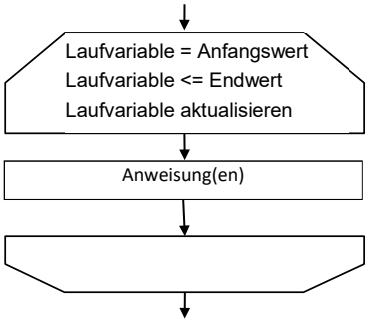
Zählergesteuerte Schleife	Die Schleife wird über einen Zähler gesteuert. Ein Zähler, dessen Startwert vorgegeben ist, wird für jeden Schleifendurchlauf um einen bestimmten Wert erhöht. **Vor** der Ausführung des eingeschlossenen Programmteils wird überprüft, ob der festgelegte Endwert erreicht ist. Ist der Endwert erreicht, wird das Programm hinter der Struktur fortgesetzt.
Bedingte Schleife	**Kopfgesteuerte Schleife:** (auch **vorprüfende** oder **abweisende** Schleife genannt) Solange eine Bedingung erfüllt ist, wird der in der Struktur eingeschlossene Programmteil ausgeführt. Anschließend wird das Programm hinter der Struktur fortgesetzt. Die Überprüfung der Bedingung erfolgt **vor** der Ausführung des eingeschlossenen Programmteils.
	Fußgesteuerte Schleife: (auch **nachprüfende** oder **annehmende** Schleife genannt) Ein Programmteil wird ausgeführt. Anschließend wird die Bedingung geprüft. Ist die Bedingung erfüllt, wird der Programmteil erneut ausgeführt. Erst wenn die Bedingung nicht mehr erfüllt ist, wird das Programm hinter der Struktur fortgesetzt.

7.9 Zählergesteuerte Schleife (Iteration)

Die zählergesteuerte Schleife (auch Iteration, Zählschleife oder Laufanweisung genannt) ist dadurch gekennzeichnet, dass die Anzahl der Schleifendurchläufe durch einen Anfangs- und Endwert festgelegt ist.



Struktogramm



PAP

- Die Variable, die die Schleifendurchläufe zählt, wird als Laufvariable oder Zähler bezeichnet.
- Die Laufvariable wird beim Eintritt in die Schleife mit dem Anfangswert initialisiert ①.
- Der Endwert kann durch eine Konstante, eine Variable oder einen Ausdruck festgelegt werden.
- Bei jedem Schleifendurchlauf wird in der Bedingung ② im Schleifenkopf geprüft, ob die Laufvariable den Endwert erreicht hat oder nicht. Ist die Bedingung erfüllt, d. h., der Endwert ist noch nicht erreicht, werden die Anweisungen im Schleifenrumpf durchgeführt.
- Der Wert, um den die Laufvariable in jedem Schritt verändert wird, heißt Schrittweite. Bei jedem Durchlauf des Schleifenkopfs wird der Wert der Laufvariablen automatisch aktualisiert ③, indem zu der Laufvariablen die festgelegte Schrittweite addiert wird (oder subtrahiert bei negativer Schrittweite).
- Wird beim Prüfen der Bedingung festgestellt, dass die Laufvariable den Endwert erreicht hat, wird die Schleife verlassen.
- Laufvariable und Endwert müssen vom selben Datentyp sein.

Sie können die zählergesteuerte Schleife einsetzen, wenn z. B. die Anzahl der Schleifendurchläufe bekannt ist bzw. genau ermittelt werden kann.

Syntax zur zählergesteuerten Schleife in Java

```
for (initStatement; condition; nextStatement)
 statement
```

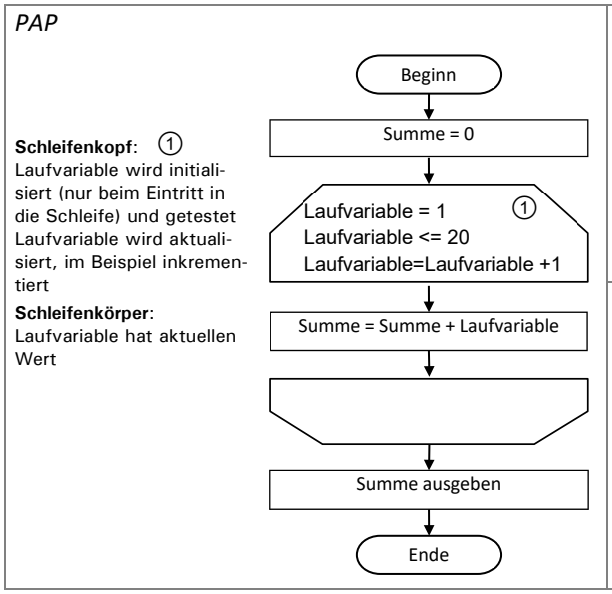
- Das Schlüsselwort `for` leitet die Zählschleife ein.
- Danach folgt die Schleifensteuerung, die in runde Klammern eingeschlossen wird.
- Die Schleifensteuerung besteht aus drei Teilen, die jeweils mit einem Semikolon getrennt werden:
  dem **Initialisierungsteil** (`initStatement`),
  dem **Bedingungsausdruck** (`condition`) und
  dem **Aktualisierungsteil** (`nextStatement`).
- Im Initialisierungsteil (`initStatement`) wird die Laufvariable definiert und mit dem Anfangswert initialisiert.
- Die Bedingung (`condition`) legt das Abbruchkriterium für die Schleife fest. Ist eine Bedingung erfüllt (`true`), wird der Schleifenrumpf ausgeführt. Ist die Bedingung nicht erfüllt, wird das Programm hinter der Schleife fortgesetzt.
- Die Bedingungen werden vor einem Schleifendurchlauf geprüft, es handelt sich daher um eine kopfgesteuerte Schleife.
- Am Ende eines Schleifendurchlaufs wird die Anweisung im Aktualisierungsteil ausgeführt. Dies ist üblicherweise das Erhöhen oder Verringern der Laufvariablen. Der Schleifenrumpf kann aus einer einzelnen Anweisung oder einem Anweisungsblock bestehen, der in geschweiften Klammern eingeschlossen wird.

Einige Programmiersprachen bieten die Möglichkeit, für das Aktualisieren der Laufvariablen eine Schrittweite anzugeben. Damit kann die Laufvariable beispielsweise in Fünfer-Schritten inkrementiert bzw. dekrementiert werden. Steht diese Möglichkeit nicht zur Verfügung, kann im Bedarfsfall eine zusätzliche Variable mit der gewünschten Schrittweite initialisiert und verwendet werden. Durch Verringern der Laufvariablen haben Sie die Möglichkeit, eine Schleife zu erstellen, bei der Sie die Laufvariable herunterzählen. In Java können Sie im Aktualisierungsteil eine beliebige Anweisung integrieren.

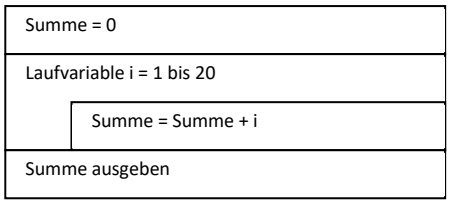
Beispiel: *AddNumbers.java*

Es wird ein Programm benötigt, das die Summe der Zahlen von 1 bis 20 berechnet.

PAP



Struktogramm



Java

```
...
int total = 0;
for (int i = 1; i <= 20; i++)
{
  total = total + i;
}
System.out.println("Summe:" + total);
...
```

Mit der Formel n(n+1)/2 für die Summe der natürlichen Zahlen können Sie das Ergebnis schnell überprüfen.

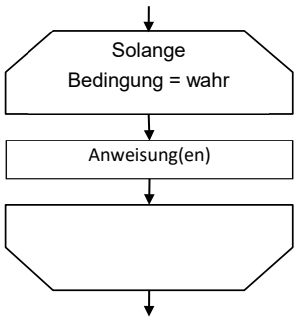
Fehler entstehen bei `for`-Schleifen oft durch falsche Initialisierung oder falsche Angabe für die Abbruchbedingung. Um zu sehen, wie oft eine Schleife tatsächlich durchlaufen wird oder um Zwischenergebnisse anzuzeigen, können Sie im Beispiel *AddNumbers.java* folgende Anweisungen in den Schleifenkörper aufnehmen:

```
System.out.println(i + ". Schleifendurchlauf");
System.out.println("Summe:" + total);  // zeigt Zwischenergebnisse an
```

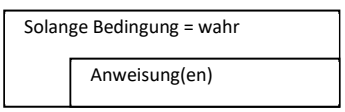
7.10 Kopfgesteuerte bedingte Schleife

Bei einer **kopfgesteuerten** bedingten Schleife wird eine Bedingung ausgewertet, **bevor** die Anweisungsfolge innerhalb des Schleifenrumpfs ausgeführt wird. Wenn das Ergebnis der Bedingung falsch ist, bevor die Schleife das erste Mal durchlaufen werden soll, werden die darin enthaltenen Anweisungen **nie** ausgeführt (abweisende Schleife).

Wenn das Ergebnis der Bedingung wahr ist, wird die Anweisungsfolge so lange ausgeführt, bis die Bedingung nicht mehr zutrifft.

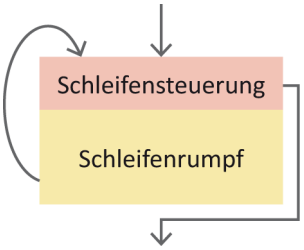


PAP



Struktogramm

Eine kopfgesteuerte Schleife wird auch abweisende Schleife genannt. Auch die Zählschleife ist eine kopfgesteuerte Schleife. Wenn die Laufvariable den Endwert bereits bei der Initialisierung überschreitet, werden die Anweisungen der Schleife nie ausgeführt.



Sie sollten kopfgesteuerte Schleifen einsetzen, wenn das Programm nur dann die Anweisungen der Schleife abarbeiten soll, wenn eine bestimmte Bedingung zutrifft.

Syntax zur kopfgesteuerten bedingten Schleife in Java

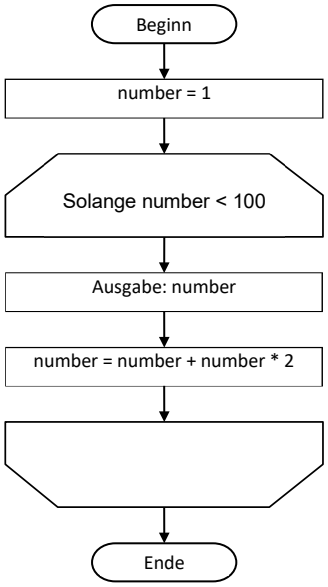
- Die kopfgesteuerte bedingte Schleife wird durch das Schlüsselwort `while` eingeleitet. Danach folgt eine Bedingung (`condition`), die **am Anfang** der Schleife geprüft wird.
- **Solange** die Bedingung (`condition`) im Schleifenkopf erfüllt ist, wird die Schleife ausgeführt.
- Falls die Bedingung nicht erfüllt ist, wird die Schleife beendet.
- `statement` kann eine einzelne Anweisung oder ein Anweisungsblock sein.

```
while (condition)
  statement
```

Beispiel: *PlusTwice.java*

Solange eine Zahl, beginnend bei 1, kleiner als 100 ist, soll das Doppelte der Zahl zu der Zahl addiert werden.

PAP



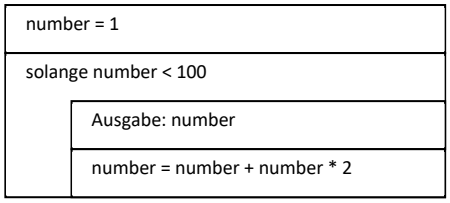
Schleifenkopf:
Es wird vor der Ausführung geprüft, ob `number` < 100 ist.

Schleifenkörper:
Die Zahl `number` wird ausgegeben.

Es wird ein neuer Wert für `number` berechnet.

Schleifenfuß:
kennzeichnet das Schleifenende, hat sonst keine Aufgaben

Struktogramm



Java

```
...
int number = 1;
while (number < 100)
{
  System.out.println(number);
  number = number + number * 2;
}
...
```

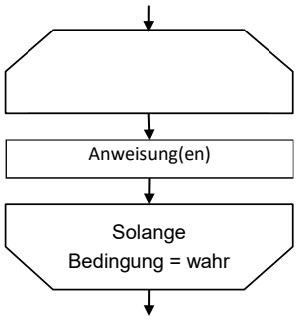
Die Abbruchbedingung muss innerhalb des Schleifenkörpers irgendwann erfüllt sein, damit die Bedingung im Schleifenkopf irgendwann nicht mehr erfüllt ist und die Schleife stoppt.

7.11 Fußgesteuerte bedingte Schleife

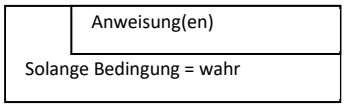
Fußgesteuerte Schleife

Bei der **fußgesteuerten** Schleife findet die Bedingungsprüfung **am Ende** der Schleife statt.

Erst **nach** Abarbeitung der Anweisungen wird am Ende der Schleife die Bedingung geprüft, ob der Schleifenrumpf noch einmal durchlaufen werden soll oder nicht (annehmende Schleife).

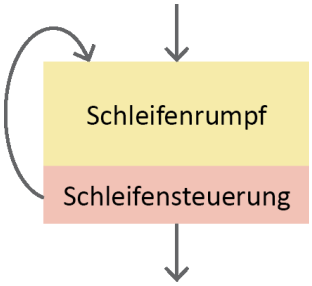


PAP



Struktogramm

Setzen Sie fußgesteuerte Schleifen ein, wenn die Anweisungen innerhalb des Schleifenrumpfs **mindestens einmal** ausgeführt werden sollen. Erst danach entscheidet eine Bedingungsprüfung, ob die Schleife erneut ausgeführt wird.



Syntax zur fußgesteuerten bedingten Schleife in Java

- Die fußgesteuerte bedingte Schleife wird mit dem Schlüsselwort `do` eingeleitet.
- Danach folgt direkt der Schleifenrumpf (`statement`). `statement` kann eine einzelne Anweisung oder ein Anweisungsblock sein.
- Die Anweisungen innerhalb des Schleifenrumpfs werden **mindestens einmal** ausgeführt.
- Der Schleifenfuß enthält die Schleifensteuerung. Sie beginnt mit dem Schlüsselwort `while`.
- Anschließend folgt die Bedingung, die in runde Klammern gesetzt wird. Der Bedingungsausdruck muss einen logischen Wert (`true` oder `false`) liefern.
- Die Schleife wird ausgeführt, **solange** die Bedingung zutrifft.

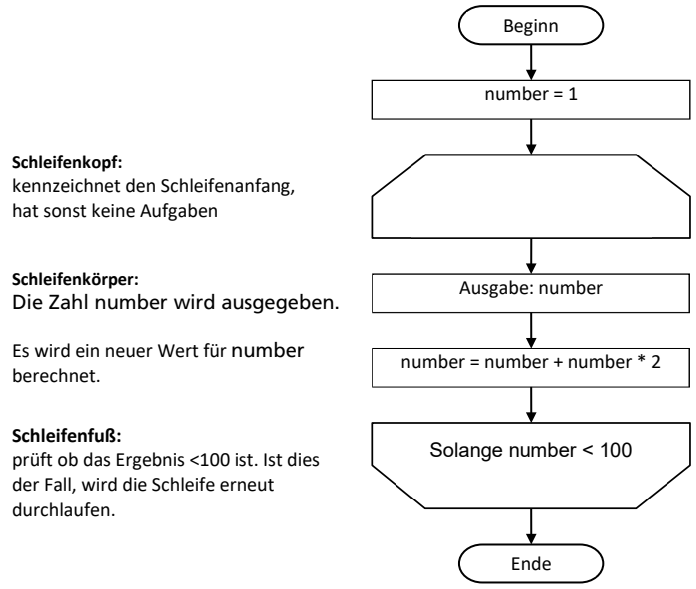
```
do
  statement
while (condition)
```

Beachten Sie beim Erlernen einer Programmiersprache, dass in einigen Sprachen die Schleife solange ausgeführt wird, bis die Bedingung den Wert `true` hat. Dies betrifft z. B. in Pascal die `repeat-until`-Anweisung. Die Schleife wird erneut ausgeführt, wenn die Bedingung den Wert `false` hat.

Beispiel: *PlusTwice2.java*

Zu einer Zahl soll, beginnend bei 1, immer wieder das Doppelte der Zahl addiert werden, solange sie kleiner als 100 ist.

PAP



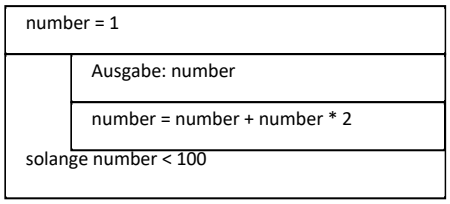
Schleifenkopf:
kennzeichnet den Schleifenanfang, hat sonst keine Aufgaben

Schleifenkörper:
Die Zahl number wird ausgegeben.

Es wird ein neuer Wert für number berechnet.

Schleifenfuß:
prüft ob das Ergebnis <100 ist. Ist dies der Fall, wird die Schleife erneut durchlaufen.

Struktogramm



Java

```
...
int number = 1;
do
{
  System.out.println(number);
  number = number + number * 2;
}
while (number < 100);
...
```

Schleifen können wie die Verzweigungsstrukturen ineinander verschachtelt werden. Genauso können sie auch die verschiedenen Kontrollstrukturen miteinander verbinden, wenn es die Aufgabenstellung erfordert.

7.12 Schnellübersicht

Was bedeutet …?	
Anweisung	Dies ist die kleinste ausführbare Einheit eines Programms.
Folge (Sequenz)	Die Anweisungen werden nacheinander, jede **einmal**, abgearbeitet.
Bedingung	Eine Bedingung beeinflusst den Ablauf eines Programms durch eine Ja-Nein-Entscheidung.
bedingte Anweisung	In Abhängigkeit von einer oder mehreren **Bedingungen** wird ein Anweisungsblock **einmal oder gar nicht** ausgeführt. WENN...DANN... `if (...) {...}`
Verzweigung	Zwei alternative Anweisungsblöcke stehen zur Auswahl. In Abhängigkeit von einer oder mehreren **Bedingungen** wird der **eine oder** der **andere** Anweisungsblock **einmal** ausgeführt. WENN...DANN...SONST... in Java: `if (...) {...} else {...}`
geschachtelte Verzweigung	In Abhängigkeit von einer oder mehreren **Bedingungen** wird von **mehreren** Anweisungsblöcken **einer** ausgeführt. WENN...DANN...SONST WENN...DANN... in Java: `if (...) {... else if (...) {...} else {...} ...}`

Was bedeutet …?	
mehrfache Verzweigung (Fallauswahl)	In Abhängigkeit von einem Variablenwert wird **einer von mehreren** möglichen Auswahlblöcken **einmal** ausgeführt. FALLAUSWAHL...FALL A...FALL B... SONST FALL... in Java: `switch (...) {case a: ... break; case b: ... break; ... default ...}`
zählergesteuerte Schleife	Es ist im Voraus bekannt oder berechenbar, wie oft ein Anweisungsblock wiederholt werden soll. Eine Zählvariable wird automatisch erhöht (oder vermindert). FÜR...BIS...(MIT DER SCHRITTWEITE...) in Java: `for (...;...;...) {...}`
kopfgesteuerte Schleife	Die Anweisungen in der Schleife werden in Abhängigkeit von einer oder mehreren **Bedingungen 0...n mal** ausgeführt. Die Bedingung wird **am Anfang** der Schleife geprüft. SOLANGE... DURCHLAUFE SCHLEIFE in Java: `while (...) {...}`
fußgesteuerte Schleife	Die Anweisungen in der Schleife werden in Abhängigkeit von einer oder mehreren **Bedingungen 1...n mal** abgearbeitet. Die Bedingungsprüfung findet **am Ende** der Schleife statt. DURCHLAUFE SCHLEIFE...SOLANGE... in Java: `do {...} while (...);`

7.13 Übungen

Übung 1: Verzweigung

Übungsdatei: -- **Ergebnisdatei: *uebung07.pdf, Maximum.java***

1. Erstellen Sie ein Struktogramm, um die Zahlen 543 und 246 zu vergleichen und die größere von beiden auszugeben.
2. Setzen Sie das Struktogramm in Java-Code um.

Übung 2: Geschachtelte Verzweigung

Übungsdatei: -- **Ergebnisdatei: *uebung07.pdf, Letter.java***

1. Für eine Nachricht soll von einem Programm automatisch die Anrede formuliert werden. Folgende Variablen existieren:
   - name: Name
   - sex: Geschlecht
   - currentHour: Uhrzeit (Stundenangabe)

   Die Anrede soll je nach Tageszeit mit "Guten Morgen" (0 – 9 Uhr), "Guten Tag" (10 – 17), "Guten Abend" (18 – 0 Uhr) beginnen und anschließend mit "Herr xxx" bzw. "Frau xxx" fortgesetzt werden. Für xxx soll der entsprechende Name eingesetzt werden.

   Schreiben Sie ein entsprechendes Struktogramm.
2. Erstellen Sie aus dem Struktogramm ein Java-Programm. Für Zeichenketten stellt Java den Datentyp String zur Verfügung.

Übung 3: Kopfgesteuerte bedingte Schleife

Übungsdatei: ***DivideBy5.java*** **Ergebnisdateien:** ***uebung07.pdf, DivideBy5.java***

1. Schreiben Sie ein Programm, das zu einer eingelesenen Integer-Zahl ermittelt, wie oft mit dieser Zahl eine Division durch 5 durchgeführt werden kann. Die ermittelte Anzahl wird ausgegeben. Initialisieren Sie die einzulesende Integer-Zahl wie folgt:

   int number = readInt("Zahl: ");

   Dazu können Sie die Funktion readInt()der Übungsdatei DivideBy5.java verwenden, die eine Zahl über die Standardeingabe einliest.
2. Warum ist die Verwendung einer fußgesteuerten Schleife in Aufgabe 1 ungünstig?
3. Für folgendes Problem ist ein Algorithmus aufzustellen. Eine Firma hat mehrere Filialen, und in jeder Filiale werden Waren aus verschiedenen Warengruppen verkauft. Die Firma verwaltet die Daten in einer Datenbank. Ein Angestellter der Firma gibt die Daten (Filialnummer, Warengruppe, Stueck und Verkaufsdatum) ein, die dann in der Datenbank gespeichert werden. Während der Eingabe soll das Programm ermitteln, wie viele Produkte der Warengruppe 7 in der Filiale 9 verkauft wurden.

Übung 4: Verzweigungen und Schleifen

Übungsdatei: -- **Ergebnisdateien:** ***Days.java***

1. Erstellen Sie ein Programm, das mit dem gegebenen letzten Tag des Vorjahres alle Sonntage ausgibt, die auf den ersten Tag eines Monats fallen. Verwenden Sie Zahlen von 1 (Montag) bis 7 (Sonntag) für die Tage und 1 (Januar) bis 12 (Dezember) für die Monate.

8 Elementare Datenstrukturen

In diesem Kapitel erfahren Sie

- was Arrays und Records sind
- wie Sie Arrays definieren und einsetzen
- was Zeiger sind

Voraussetzungen

- Deklaration von Variablen und Konstanten
- Kontrollstrukturen

8.1 Warum werden Datenstrukturen benötigt?

Mithilfe der ganzzahligen Datentypen wie `integer`, `long`, `short` und den Gleitkommazahlen in Form von `float` und `double` können Sie in Java Zahlen darstellen. Mit dem Datentyp `char` können Sie Variablen definieren, die Zeichen aufnehmen können. Andere Sprachen – auch ohne explizite Typfestlegung wie JavaScript oder PHP – stellen ähnliche Konstruktionen (teils implizit) zur Verfügung.

Häufig werden in der Programmierung aber auch **zusammengehörige** Daten verwendet, wie beispielsweise verschiedene Zahlen, die zu einem Vektor bzw. einer Matrix gehören, oder Adressen mit ihren Bestandteilen. Viele Programmiersprachen besitzen dafür **elementare Datenstrukturen**.

- Ein **Array**, auch Feld genannt, wird in manchen Sprachen für Elemente **gleichen** Datentyps verwendet, z. B. für einen Vektor. Andere Sprachen lassen auch Elemente unterschiedlichen Datentyps zu, und damit gibt es dort keine Unterscheidung zu einem Record.
- Ein **Record**, auch Verbund oder Struktur (Struct) genannt, wird für eine Anzahl Elemente **unterschiedlichen** Datentyps verwendet, z. B. für eine Adresse. Verbundstrukturen werden syntaktisch oft unterschiedlich umgesetzt.

Nahezu alle Programmier- und Skriptsprachen unterstützen Arrays. Records bzw. allgemein Strukturen mit Elementen verschiedenen Datentyps gibt es jedoch nicht in allen Programmiersprachen bzw. Versionen einer Programmiersprache. Java beispielsweise kennt grundsätzlich keine Records, aber in Klassen können Daten unterschiedlichen Typs zusammengefasst werden.

Die Datenstrukturen Array und Record lernen Sie nachfolgend kennen. Informationen zu speziellen Datenstrukturen wie Stapel (engl. stack), Schlage (engl. queue) und Liste (engl. list) können Sie auf der Website des HERDT-Verlages unter *www.herdt.com* herunterladen (vgl. Abschnitt 1.2).

8.2 Arrays

Arrays nutzen

Stellen Sie sich eine Berechnung vor, die 20 Ergebnisse liefert. Die Ausgabe der 20 Werte soll erfolgen, wenn alle Berechnungen abgeschlossen wurden. Auf jedes Ergebnis soll der Zugriff möglich sein. Sie könnten zum Speichern der Ergebnisse 20 Variablen definieren (`x1`, `x2`, `x3`, `x4` ...). Bei der Berechnung müssten Sie dann jeder Variablen einzeln einen Wert zuweisen und bei der Ausgabe jede Variable einzeln ansprechen – das sind 40 verschiedene Anweisungen.

Durch die Verwendung eines Arrays können Sie **alle** Ergebnisse unter **einem** Bezeichner speichern. Der Zugriff auf die einzelnen Ergebnisse erfolgt über einen sogenannten **Index**. Der Index entspricht dabei der aktuellen Position eines Ergebnisses im Array. Zum Beispiel befindet sich das Ergebnis der 13. Berechnung auf Position 12 des Arrays (Index startet meist mit 0).

Der Aufwand für obige Problemlösung reduziert sich dann mithilfe einer Schleife auf wenige Anweisungen, unabhängig davon, wie viele Elemente das Array besitzt.

Merkmale von Arrays

- Ein Array hat einen Namen.
- Arrays bestehen in einigen Sprachen (z. B. Java) immer aus Elementen **desselben** Datentyps. In nicht streng typisierten Sprachen wie JavaScript oder PHP müssen die Elemente explizit **nicht** vom selben Datentyp sein, weshalb dort die Grenzen zu einem Record aufgehoben sind.
- Ein Element eines Arrays wird über einen **Index** angesprochen; die einzelnen Elemente haben keinen eigenen Namen.
- In den meisten Programmiersprachen (z. B. C, C#, Java, JavaScript) hat das erste Element eines Arrays den Index 0, das zweite Element hat den Index 1, das dritte Element hat den Index 2 usw. Man nennt das **nullindiziert**. Der Index eines Arrays mit n Elementen reicht dann von 0 bis n-1. Es gibt allerdings auch Sprachen, bei denen der erste Index mit 1 beginnt.
- Wird die Anzahl der Elemente bei der Definition des Arrays festgelegt, wird ein statisches Array erzeugt. Wird die Anzahl nicht angegeben, wird ein dynamisches Array erzeugt. Nicht alle Programmiersprachen erlauben dynamische Arrays. In Java beispielsweise sind (vollständig) dynamische Arrays nicht erlaubt. Auch statische Arrays sind nicht in allen Programmiersprachen erlaubt. In JavaScript sind Arrays beispielsweise **immer** dynamisch.

Die Schreibweise für Arrays kann in verschiedenen Programmiersprachen unterschiedlich sein.

Ein- und mehrdimensionale Arrays

Einfache Arrays bestehen aus *n* Elementen, die über einen Index angesprochen werden können. Das Array kann beispielsweise einen Vektor speichern. Das Array repräsentiert dann eine Dimension. Besteht ein Array-Element wiederum aus einem Array, z. B. um eine Tabelle zu speichern, handelt es sich um ein zweidimensionales Array. Ein zweidimensionales Array besteht aus *m x n* Elementen, die über zwei Indizes angesprochen werden. Ein Array kann auch mehr als zwei Dimensionen besitzen. Für jede weitere Dimension wird ein weiterer Index für den Zugriff auf die Elemente benötigt.

8.3 Eindimensionale Arrays

Syntax für die Definition von Arrays in Java

Sie definieren eine Array-Variable, indem Sie dem Datentyp ① ein Klammerpaar [ ] anfügen.

```
type[] identifier;
①
```

Der Name des Arrays und der Datentyp seiner Elemente sind somit bekannt. Unbekannt ist noch, wie viele Elemente das Array enthalten soll. Bevor ein Array verwendet werden kann, muss es erzeugt werden.

Syntax für die Erzeugung von Arrays in Java

- Um ein Array zu erzeugen, verwenden Sie den Operator `new` ②. Die Array-Größe (die Anzahl der Array-Elemente) wird dazu in eckigen Klammern hinter dem Datentyp angegeben. Sie können in Java die Größe später **nicht** mehr ändern.

```
identifier = new type[size];   ②
```

- Die Größe muss vom Datentyp `int` sein. Sie können sie mit einer Zahl, mit einer Konstanten oder einem Ausdruck angeben. Der Ausdruck wird zur Laufzeit ausgewertet, und somit wird auch die Größe erst zur Laufzeit festgelegt.
- Bei der Erzeugung des Arrays mit `new` werden die Elemente automatisch mit Default-Werten für den jeweiligen Datentyp initialisiert. So erhalten z. B. Elemente vom Typ `int` den Wert `0`.
- Sie können die Definition und das Erzeugen des Arrays in einer Anweisung durchführen ③.

```
type[] identifier1 = new type[size1];③
```

Die Array-Variable `identifier` enthält nicht das Array selbst, sondern einen sogenannten **Zeiger** auf das erzeugte Array. Zeiger, auch Referenz genannt, lernen Sie im weiteren Verlauf dieses Kapitels kennen.

Syntax für die Erzeugung und Initialisierung von Arrays in Java

- Bereits bei der Erzeugung des Arrays können Sie die Elemente mit Daten initialisieren.

```
type[] identifier1 = {value0,
value1,...valueN};
```

- Geben Sie auf der rechten Seite des Gleichheitszeichens in geschweiften Klammern { } die gewünschten Werte ein. Trennen Sie dabei die einzelnen Werte durch Kommata.
- Durch die Anzahl der Werte, die Sie innerhalb der geschweiften Klammern angeben, wird gleichzeitig die Größe des Arrays festgelegt.

Beispiele für die Definition, Erzeugung und Initialisierung von Arrays

```
①  double[] field1;
   field1 = new double[15];        //Array mit 15 Elementen vom Typ double
②  int[] field2 = new int[20];     //Array mit 20 Elementen vom Typ int
③  double[] field3 = {1.2, 42.3, 27.0, 12.567};   //Elemente mit Werten
                                                    initialisieren
④  int a = 1;
   int b = 6;
   int c = 4;
   int[] field4 = {a, b, c, a + b + c};           //Ausdruecke verwenden
```

① Definition und Erzeugung eines Arrays in zwei separaten Anweisungen

② Definition und Erzeugung eines Arrays in einer einzigen Anweisung

③ – ④ Definition, Erzeugung und Initialisierung eines Arrays mit individuellen Werten

Mit Arrays arbeiten

Wie z. B. Variablen primitiver Datentypen können Sie auch Array-Elementen Werte zuweisen und die Inhalte auslesen.

Syntax für Wertzuweisungen und das Auslesen von Array-Elementen

- Der Zugriff auf Array-Elemente erfolgt durch den Index, der bei 0 beginnt und entsprechend für das letzte Array-Element den Wert Array-Größe - 1 besitzt.

```
identifier[intExpression] = expression;  ①
type identifier1 = identifier[intExpression1];②
```

- Der Index kann ein beliebiger Ausdruck sein, der ein Ergebnis vom Typ `int` liefert.
- Sie können einem über den Index bestimmten Array-Element einen Wert zuweisen ①.
- Über den Index (`intExpression1`) können Sie den Wert des entsprechenden Array-Elements auslesen ② und beispielsweise in einer anderen Variablen speichern.

Der Wertebereich für den Index erstreckt sich von 0 für das erste Element bis zur Array-Größe - 1 für das letzte Element. Wenn Sie einen ungültigen Index angeben, wird dieser Fehler vom Java-Laufzeitsystem erkannt. Falls Sie z. B. bei einem Array mit drei Elementen auf das vierte Element zugreifen wollen, gehört dieser Speicherbereich nicht mehr zum Array. Eine entsprechende Fehlermeldung wird erzeugt und das Programm beendet.

Die Anzahl der Array-Elemente ermitteln

Mit `arrayIdentifier.length` können Sie in Java die Größe bzw. Länge des Arrays, d. h. die Anzahl der Array-Elemente, bestimmen.

Arrays zuweisen

Einer Array-Variablen darf eine andere Array-Variable zugewiesen werden, vorausgesetzt, die Elemente sind vom gleichen bzw. passenden Datentyp. Somit können z. B. mehrere Array-Variablen auf dasselbe Array zeigen.

Beispiel für den Einsatz eines eindimensionalen Arrays: *Dimension1Array.java*

Im folgenden Beispiel wird ein Array mit zehn Elementen verwendet, um die Quadratzahlen von 1 bis 10 zu speichern.

```
...
① int[] squareNumber = new int[10];
② for (int i = 0; i < squareNumber.length; i++)
{
③   squareNumber[i] = (i + 1) * (i + 1);
④   System.out.print("Index: " + i + " - ");
    System.out.println("Die Quadratzahl von "+(i+1)+" ist
    "+squareNumber[i]);
}
...
```

Enthält die Länge des Arrays, hier 10.

① Definition und Erzeugung eines Arrays mit 10 Elementen

② Wertzuweisung an die einzelnen Elemente mit der `for`-Schleife. Im Initialisierungsteil wird der Index auf `0` und somit auf das erste Array-Element gesetzt. Im Bedingungsausdruck wird die Laufvariable `i` mit der Anzahl der Array-Elemenete verglichen. Diese Anzahl wird über das Attribut `length` ermittelt.

③ Berechnung der Quadratzahl von `i+1` für das Element mit dem Index i

④ Mit der Anweisung `System.out.print();` können Sie Zeichen auf dem Bildschirm ausgeben, ohne anschließend einen Zeilenumbruch zu erzeugen. Folgt eine weitere Anweisung zur Ausgabe, kann die Ausgabe in derselben Zeile fortgesetzt werden.

```
Index 0: Die Quadratzahl von 1 ist 1
Index 1: Die Quadratzahl von 2 ist 4
Index 2: Die Quadratzahl von 3 ist 9
Index 3: Die Quadratzahl von 4 ist 16
Index 4: Die Quadratzahl von 5 ist 25
Index 5: Die Quadratzahl von 6 ist 36
Index 6: Die Quadratzahl von 7 ist 49
Index 7: Die Quadratzahl von 8 ist 64
Index 8: Die Quadratzahl von 9 ist 81
Index 9: Die Quadratzahl von 10 ist 100
```

Die Ausgabe des Programms

8.4 Zwei- und mehrdimensionale Arrays

Der Aufbau mehrdimensionaler Arrays

Mehrdimensionale Arrays sind als geschachtelte eindimensionale Arrays aufgebaut. Bei mehrdimensionalen Arrays geben Sie in Java für jede Dimension ein Klammerpaar [ ] an. Ein zweidimensionales Array ist ein Array aus Zeilen, wobei jede Zeile wiederum aus einem Array besteht, den Spalten. Für den Zugriff auf die einzelnen Elemente eines zweidimensionalen Arrays werden zwei Indizes gebraucht: ein Index zur Bestimmung der Zeile und ein Index zur Bestimmung der Spalte.

Beispiel für den Einsatz eines zweidimensionalen Arrays: *Dimension2Array.java*

In einem zweidimensionalen Array sollen alle Produkte der Zahlen 1 bis 10 gespeichert werden. Dazu wird ein 10 x 10 Elemente großes Integer-Array benötigt.

```
...
① int[][] product = new int[10][10];      bestimmt die Anzahl der Zeilen
② for (int i = 0; i < product.length; i++)
{                                         bestimmt die Anzahl der Spalten
③   for (int j = 0; j < product[i].length; j++)
    {
      product[i][j] = (i + 1) * (j + 1); //Produkt berechnen
      System.out.print("Index " + i + "/ " + j + ": ");
④     System.out.println("Das Produkt von "+(i+1)+" und "+(j+1)+" ist "
      +product[i][j]);
    }
}
...
```

① Es wird ein Array `product` mit 10 x 10 Elementen (10 Zeilen mit je 10 Spalten) angelegt.

②,③ Um jedes Element des Arrays anzusprechen, müssen zwei Schleifen ineinander verschachtelt werden. Die äußere Schleife ② durchläuft pro Durchlauf eine Zeile des Arrays, während die innere Schleife ③ pro Durchlauf alle Spalten einer Zeile durchläuft.

④ Die Werte für das Array `product` ergeben sich aus dem Produkt der Laufvariablen `i` und `j` jeweils erhöht um 1, da die Indizes innerhalb der Arrays mit 0 beginnen, die Berechnung aber für Werte ab der Zahl 1 erfolgen soll.

```
Index 0/ 0: Das Produkt von 1 mal 1 ist 1
Index 0/ 1: Das Produkt von 1 mal 2 ist 2
Index 0/ 2: Das Produkt von 1 mal 3 ist 3
...
Index 1/ 0: Das Produkt von 2 mal 1 ist 2
Index 1/ 1: Das Produkt von 2 mal 2 ist 4
Index 1/ 2: Das Produkt von 2 mal 3 ist 6
...
Index 9/ 7: Das Produkt von 10 mal 8 ist 80
Index 9/ 8: Das Produkt von 10 mal 9 ist 90
Index 9/ 9: Das Produkt von 10 mal 10 ist 100
...
```

Auszug der Ausgabe des Programms

Spalten

i \ j	0	1	2	3	4	5	6	7	8	9
0	1	2	3	4	...					
1	2	4	6	...						
2	3	6	...							
3	4	...								
4	...									
5										...
6									...	70
7								...	72	80
8							...	72	81	90
9						...	70	80	90	100

product[4][5]

Zeilen

Beispiel für ein zweidimensionales Array mit den Produkten der Zahlen 1 bis 10

8.5 Zeichenketten und Records

Grundlagen zu Zeichenketten

Zeichenketten, auch **Strings** genannt, sind eine Folge von Zeichen. Die maximale Länge einer Zeichenkette ist von der Sprache abhängig. Zeichenketten werden in den verschiedenen Sprachen unterschiedlich gespeichert und gehandhabt.

- Die meisten Sprachen besitzen einen Datentyp `String`.
- In einigen Sprachen wird eine Zeichenkette als Array des Typs `character` angelegt.
- In einigen Sprachen muss bei der Definition der Zeichenkette deren maximale Länge angegeben werden, in anderen Sprachen ist die Länge variabel bzw. wird dynamisch festgelegt.

- Meist werden Zeichenketten in doppelten Hochkommas dargestellt (im Unterschied zu Zeichen, die in einfache Hochkommas geschrieben werden).
- In einigen Sprachen wird die Zeichenkette mit 0 abgeschlossen, um deren Ende zu kennzeichnen (Null-terminierte Zeichenkette).

Informationen zur Arbeit mit Zeichenketten

Bearbeitet werden Zeichenketten immer über Funktionen bzw. Methoden, da Zeichenketten in der Regel nicht zu den elementaren Datentypen zählen. Die meisten Sprachen bieten Funktionen bzw. Methoden, welche

- Zeichenketten-Variablen einen Wert (eine Zeichenkette) zuweisen.
- die Länge einer Zeichenkette bestimmen.
- Teilzeichenketten ermitteln.
- Zeichen in die Zeichenkette einfügen bzw. daraus entfernen.
- Zeichenketten bzw. Teilzeichenketten kopieren.
- Zeichenketten vergleichen.

Merkmale von Records

- In einem Record lassen sich Elemente zusammenfassen, die im Gegensatz zu Arrays in manchen Sprachen explizit aus **verschiedenen** Datentypen bestehen können.
- Jedes Element eines Records hat einen Namen und einen Datentyp. Der Zugriff auf die Elemente des Records erfolgt über die Namen.

Hinweise zur Verwendung von Records

Viele Programmiersprachen bieten die Datenstruktur Record an. Allerdings kann sich beispielsweise die Schreibweise für Records in den verschiedenen Programmiersprachen unterscheiden. In objektorientierten Programmiersprachen, wie z. B. Java, gibt es keine Datenstruktur Record – dafür stehen Klassen zur Verfügung.
In Sprachen ohne strenge Typüberwachung wie JavaScript sind Records und Arrays meist identisch.

8.6 Zeiger (Referenz)

Was sind Zeiger?

Die bisher eingeführten Variablen werden verwendet, um Werte verschiedener Datentypen zu speichern. Durch die Deklaration einer Variablen und die Angabe des Datentyps teilen Sie dem Compiler mit, wie viel Speicherplatz er für die Variable reservieren muss. Über den Variablennamen greifen Sie dann auf diesen Speicherbereich, der den Wert der Variablen enthält, zu.

Für verschiedene Anwendungsgebiete, wie z. B. die Programmierung von verketteten Listen (vgl. Abschnitt 8.10), werden Variablen gebraucht, die die **Adressen** von Speicherbereichen beinhalten können. Die Adresse eines Speicherbereichs zeigt (referenziert) einen Speicherbereich, der wiederum den Wert eines bestimmten Datentyps beinhaltet. Variablen, deren Wert Adressen von Speicherbereichen sind, werden **Zeiger** (engl. **Pointer**) genannt.

Programmiersprachen, die direkt mit Zeigern arbeiten

- Die Arbeit mit explizit vereinbarten Zeigern erlauben nur einige Programmiersprachen, wie z. B. C/C++ oder Pascal.
- In einigen Programmiersprachen sind Zeiger an Datentypen gebunden. Ein Zeiger auf eine Variable vom Typ Integer darf nur Adressen von Speicherbereichen enthalten, an denen Werte vom Typ Integer gespeichert sind.

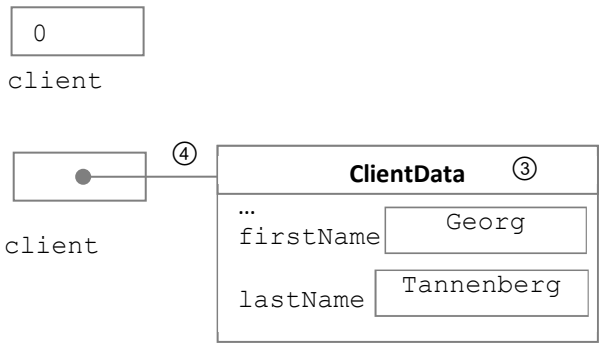
Programmiersprachen, die indirekt mit Zeigern arbeiten

In Java oder auch JavaScript gibt es keine expliziten Zeiger oder einen speziellen Datentyp `Zeiger`. Variablen, die zum Zugriff auf Objekte verwendet werden, heißen Referenzvariablen und werden automatisch als „Zeiger“ auf das Objekt interpretiert.

Beispiel für die Erzeugung eines Objekts einer Klasse in Java

```
...
① ClientData client;

② client = new
ClientData("Georg",
"Tannenberg", "Hamburg", 33);
...
```



① Bei der Definition einer Variablen vom Typ der Klasse `ClientData` wird eine sogenannte **Referenzvariable** erzeugt, die später eine Referenz (einen Verweis) auf das eigentliche Objekt darstellt. Das Objekt selbst existiert zunächst noch nicht. Speicherplatz wird auch nur für die Aufnahme der Referenz belegt.

② Erst mit dem Operator `new` wird ein Objekt ③ der Klasse erzeugt. Dabei wird auch entsprechend Speicherplatz für das Objekt bereitgestellt.

④ Die Referenzvariable verweist nun auf dieses neue Objekt.

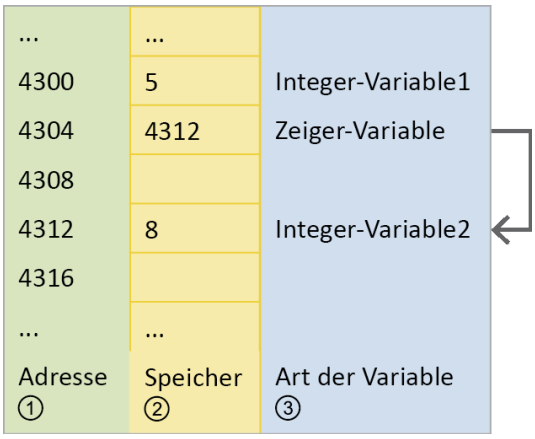
Zeiger im Pseudocode

Im Pseudocode wird die folgende Syntax verwendet.

- Ein Zeiger wird durch den Datentyp `pointer` definiert.
- Die Adresse einer Variablen wird geliefert, wenn dem Variablennamen das Zeichen `@` vorangestellt wird.
- Den Wert, auf den ein Zeiger verweist, erhalten Sie durch das nachgestellte Zeichen `^`.

Vergleich zwischen Variable und Zeiger

- Spalte ① zeigt einen Ausschnitt von Adressen des Speicherbereichs.
- Die Belegung des Speichers ist in Spalte ② dargestellt.
- Spalte ③ bezeichnet die Art der Variablen, die mit dem jeweiligen belegten Speicherbereich verbunden ist.



Wo werden Zeiger angewendet?

Zeiger werden beispielsweise zur Programmierung folgender Anwendungen eingesetzt:

- Dynamische Speicherverwaltung
- Parameterübergabe in Funktionen
- Zeichenkettenverarbeitung

Vorteile von Zeigern

- Durch die Verwendung von Zeigern kann zur Laufzeit ein Programm so viel Speicher anfordern, wie es die jeweilige Aufgabenstellung erfordert. Bei der Verwendung von Arrays muss beim Programmstart die maximal mögliche Array-Länge bekannt sein, und dafür wird Speicher angefordert.
- Bei der Parameterübergabe eignen sich Zeiger, wenn Sie die übergebenen Parameter ändern möchten. Sie haben über einen Zeiger direkt Zugriff auf die Daten über deren Adresse und zur Parameterübergabe selbst (vgl. Kapitel 9).

Plus **Ergänzende Lerninhalte:** *Spezielle Datenstrukturen.pdf*

Für besondere Anwendungen gibt es spezielle Datenstrukturen, die eine spezialisierte Art eines bequemen Datenzugriffs (etwa der Reihe nach oder in Form eines Stapels) gestatten. Mehr Informationen dazu finden Sie im oben angegebenen BuchPlus-Dokument.

8.7 Übungen

Übung 1: Arrays

Übungsdatei: -- **Ergebnisdateien: *uebung08.pdf, Searchnumbers.java***

1. Welchen Index hat das n-te Element eines Arrays? Begründen Sie Ihre Aussage.
2. Schreiben Sie ein Java-Programm, um in einem Array `int[] numbers = {0, 10, 12, 4, 7, 20, 21, 13};` nach der Zahl 13 zu suchen. Das Ergebnis der Suche soll ausgegeben werden.

Übung 2: Zweidimensionale Arrays

Übungsdatei: -- **Ergebnisdateien: *uebung08.pdf, ReadMultiNumbers.java, MultiplyScalar.java, FindPrimes.java***

1. Schreiben Sie einen Algorithmus, der fünf Wertepaare in ein zweidimensionales Array einliest. Nach der Eingabe sollen die Produkte der eingegebenen Wertepaare berechnet und ausgegeben werden.
2. Setzen Sie den Algorithmus aus Aufgabe ① in Java-Code um. Verwenden Sie zum Einlesen der Wertepaare die Funktion `readInt()`. Speichern Sie das Programm unter *ReadMultiNumbers.java*.
3. Erstellen Sie einen Algorithmus, in dem eine Matrix mit einem Skalar multipliziert wird. Die Zahlen der Matrix und das Skalar werden vor der Berechnung eingelesen und danach ausgegeben.
4. Schreiben Sie für den Algorithmus aus Aufgabe ③ ein Java-Programm. Verwenden Sie die Funktion `readInt()`, um die Zahlen einzulesen. Speichern Sie das Programm unter *MultiplyScalar.java*.
5. Primzahlen sind Zahlen aus dem Zahlenbereich der natürlichen Zahlen größer oder gleich 2, die außer sich selbst und der Zahl 1 keine weiteren Teiler besitzen. Alle natürlichen Zahlen lassen sich in Faktoren von Primzahlen zerlegen. Erstellen Sie einen Algorithmus, der die Primzahlen von 2 bis n mithilfe des Siebs des Eratosthenes ermittelt. Bei dieser Methode werden alle Zahlen gestrichen, die sich aus anderen Zahlen zusammensetzen.
   - Schreiben Sie die Zahlen von 1 bis zur gewünschten Zahl der Größe nach geordnet in ein Array.
   - Sie starten mit der kleinsten natürlichen Primzahl, der Zahl 2.
   - Setzen Sie alle Array-Inhalte, die ein Vielfaches der Zahl 2 sind, auf 0.
   - Ist das gesamte Array durchlaufen, wird von der Zahl 2 aus die nächstgrößere Zahl gesucht (3).
   - Alle Array-Inhalte, die ein Vielfaches dieser Zahl sind, werden anschließend auf 0 gesetzt.
   - Führen Sie diese Schritte so lange durch, bis das Quadrat der aktuellen Zahl größer als die letzte Zahl des Arrays ist.
   - In allen Array-Elementen größer 1, deren Inhalt nicht auf 0 gesetzt wurde, befinden sich die gesuchten Primzahlen.

Siebzahl 2: alle Vielfache von 2 entfallen

alt	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	...
neu	2	3	~~4~~	5	~~6~~	7	~~8~~	9	~~10~~	11	~~12~~	13	~~14~~	15	~~16~~	17	~~18~~	19	~~20~~	21	~~22~~	23	~~24~~	25	...

Siebzahl 3: zusätzlich entfallen alle Vielfache von 3

alt	2	3	~~4~~	5	~~6~~	7	~~8~~	~~9~~	~~10~~	11	~~12~~	13	~~14~~	~~15~~	~~16~~	17	~~18~~	19	~~20~~	~~21~~	~~22~~	23	~~24~~	25	...
neu	2	3	4	5	~~6~~	7	8	~~9~~	10	11	~~12~~	13	14	~~15~~	16	17	~~18~~	19	20	~~21~~	22	23	~~24~~	25	...

usw.

6. Primzahlen sind Zahlen aus dem Zahlenbereich der natürlichen Zahlen größer oder gleich 2, die außer sich Setzen Sie den Algorithmus aus Aufgabe 5 in Java-Code um. Verwenden Sie zum Einlesen der Zahl die Funktion `readInt()`. Speichern Sie das Programm unter *FindPrimes.java*.

9 Methoden, Prozeduren und Funktionen

In diesem Kapitel erfahren Sie

- wie Sie Methoden, Prozeduren und Funktionen schreiben
- welche Möglichkeiten der Parameterübergabe existieren

Voraussetzungen

- Deklarieren von Variablen und Konstanten
- Verwenden von Datentypen
- Zeiger und Arrays

9.1 Unterprogramme

Warum Methoden, Prozeduren und Funktionen verwendet werden

Code mehrfach nutzen

Häufig werden in einem Programm bestimmte Anweisungen mehrmals benötigt. Dazu werden Anweisungen, die eine bestimmte Funktion erfüllen, zusammengefasst und erhalten einen Namen. Diese funktionalen Einheiten werden bei objektorientierten Programmiersprachen **Methoden**, in prozeduralen Sprachen **Funktionen** oder **Prozeduren** genannt. Diese funktionalen Einheiten werden auch mit **Unterprogramm** oder **Subroutine** bezeichnet. Unterprogramme können beliebig oft in einem Programm verwendet werden, indem sie über ihren Namen aufgerufen werden. Die Unterprogramme können über Bibliotheken auch für andere Programme zur Verfügung gestellt werden (**Mehrfachverwendbarkeit**). Unterprogramme aus Bibliotheken sind bereits getestet. Somit wird durch die Verwendung dieser Unterprogramme die Korrektheit von Programmen erhöht, da sie meist weniger Fehler enthalten als entsprechend neu erstellte Programme.

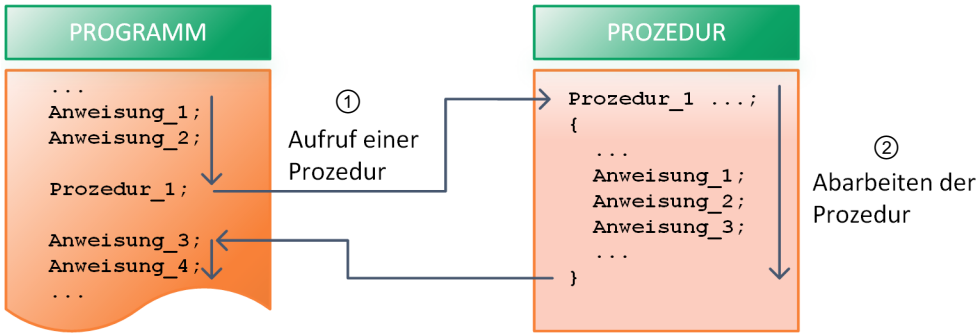
Code strukturieren

Unterprogramme sind gemäß dem Modularitätsprinzip überschaubare Einheiten eines Programms. Ohne diese Strukturierung könnten große Programme nicht verstanden und gewartet werden.

Mit Prozeduren arbeiten

Unterprogramme, die in der prozeduralen Programmierung **keinen Rückgabewert** liefern, heißen **Prozeduren**. Soll eine Prozedur bei jedem Aufruf mit anderen Daten arbeiten, benötigt eine Prozedur **Parameter**. Parameter, die eine Prozedur zur Verfügung stellt, heißen **formale Parameter**. Beim Aufruf einer Prozedur über den Prozedurnamen können dann entsprechende Werte an die formalen Parameter übergeben werden. Die beim Prozeduraufruf übergebenen Werte werden **aktuelle Parameter** genannt. Soll eine Prozedur z. B. verschiedene Texte ausgeben, kann der Text als Parameter übergeben werden.

Wird beim Programmablauf ein Prozeduraufruf ① erreicht, kommt es zur Unterbrechung des Programmflusses an dieser Stelle, und es wird in die zugehörige Prozedur verzweigt. Die Anweisungen der Prozedur werden ausgeführt ②, anschließend wird die dem Prozeduraufruf folgende Anweisung, im Beispiel Anweisung_3, ausgeführt.



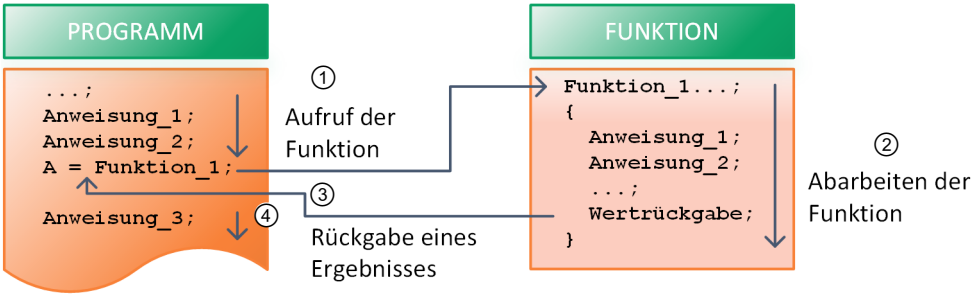
Aufruf und Abarbeiten einer Prozedur

Die meisten Programmiersprachen bieten Standardprozeduren. Sie erledigen häufig benötigte Standardaufgaben, z. B. Prozeduren für Ausgaben auf den Bildschirm.

Mit Funktionen arbeiten

Funktionen liefern in der prozeduralen Programmierung im Unterschied zu Prozeduren **einen Rückgabewert** an das aufrufende Programm zurück. Dadurch können Sie Funktionen auch in Ausdrücken angeben.

Funktionen werden, analog zu Prozeduren, mit dem Funktionsnamen aufgerufen. Auch Funktionen können Sie bei Bedarf Parameter übergeben. Wird eine Funktion aufgerufen ①, werden die Anweisungen der Funktion abgearbeitet ② und ein Rückgabewert ③ wird an das aufrufende Programm übergeben. Dieser Rückgabewert kann danach z. B. einer Variablen zugewiesen oder in einer Berechnungsformel verwendet werden. Anschließend wird die dem Funktionsaufruf folgende Anweisung ④, im Beispiel Anweisung_3, ausgeführt.



Aufruf und Abarbeiten einer Funktion

Aufbau von Prozeduren und Funktionen

Prozeduren und Funktionen bestehen aus einem Kopf und einem Rumpf.

- Im Kopf wird der Name der Prozedur bzw. Funktion zusammen mit der optionalen Parameterliste festgelegt. Der Kopf mit der Parameterliste ist die **Schnittstelle** der Prozedur bzw. Funktion. Die Schnittstelle, auch **Signatur** genannt, legt fest, wie die Prozedur bzw. Funktion benutzt werden kann.
- Der Rumpf enthält die einzelnen Anweisungen der Prozedur bzw. Funktion.
- Prozeduren und Funktionen können in ihrem Rumpf ebenfalls Prozeduren oder Funktionen aufrufen.

Wichtige Unterschiede zwischen Prozeduren und Funktionen

Aktion	Funktion	Prozedur
Ergebnisrückgabe	ja	nein
Verwendung in einer Wertzuweisung	ja	nein
Verwendung in einem Ausdruck	ja	nein
Verwendung als Vergleichsoperator in Bedingungsausdrücken	ja	nein

Mit Methoden arbeiten

Unterprogramme, die in der objektorientierten Programmierung verwendet werden, sind an Objekte oder Klassen gebunden. Um sie deutlich von prozeduralen Unterprogrammen abzugrenzen, nennt man sie **Methoden**. Dabei wird nicht zwischen Methoden, die keinen Rückgabewert liefern und solchen, die einen Rückgabewert liefern, unterschieden. Auch an Methoden können Sie bei Bedarf Parameter übergeben.

Gelegentlich taucht auch bei der objektorientierten Programmierung der Begriff der Funktion oder Prozedur auf, obwohl eine Methode gemeint ist. Mit der nötigen Vorsicht kann man das machen, obgleich es eine etwas unsaubere Formulierung darstellt. Sie sollten bei der objektorientierten Programmierung die korrekte Bezeichnung Methode verwenden.

Syntax für die Beschreibung einer einfachen Methode in Java

- Die Beschreibung einer Methode erfolgt durch den Methodenkopf ① und den Methodenrumpf ②.

```
[modifiers] void identifier() ①
{
   ...          ②
}
```

Der Methodenkopf besteht aus dem Namen der Methode, der sich an die Konventionen von Bezeichnern hält, und runden Klammern. Das Schlüsselwort `void` kennzeichnet eine

- Methode, die keinen Rückgabewert liefert.
- Anschließend folgt ein Block mit Anweisungen, der in geschweifte Klammern { } gesetzt wird.
- Der Methodendefinition können Sie sogenannte Modifizierer (modifiers) voranstellen. Mit Modifizierern haben Sie die Möglichkeit, beispielsweise die Zugriffsrechte auf die Methode festzulegen.

Methodennamen in Java sind gewöhnlich aus einem Verb und einem Substantiv zusammengesetzt, wobei das Verb die Funktion beschreibt, z. B. `printReport`. Das Verb beginnt mit einem Kleinbuchstaben; weitere Wörter innerhalb des Namens sollten mit einem Großbuchstaben beginnen, damit der Methodenname gut lesbar ist.

Syntax für die Beschreibung einer Methode mit Parametern in Java

```
[modifiers] void identifier(type identifier1[, type identifier2,...])
{                                                    ①
   ...
}
```

- Die runden Klammern der Methode beinhalten die Beschreibung der Parameter.
- Die Namen der Parameter halten sich an die Konventionen von Bezeichnern.
- Mehrere Parameter werden durch Kommata getrennt.
- Wie bei einer Variablendefinition werden für jeden Parameter der Datentyp und der Name angegeben.

- Der Datentyp kann ein primitiver Datentyp oder ein Referenztyp (eine Klasse) sein.
- Der Teil des Methodenkopfs, der den Methodennamen und die in Klammern eingeschlossenen Parameter umfasst, wird wie bei Funktionen oder Prozeduren **Signatur** ① der Methode genannt.

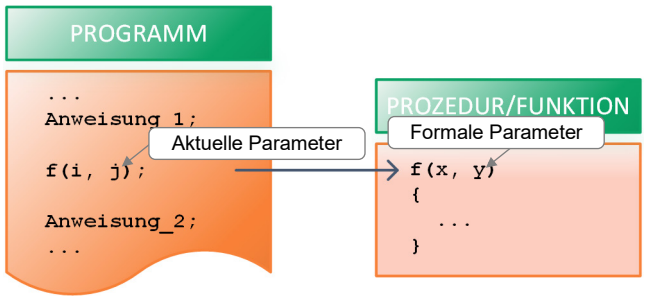
Sowohl die Parameter als auch die in einer Methode deklarierten Variablen sind lokale Variablen. Sobald die Ausführung der Methode beendet ist, verlieren sie ihre Gültigkeit.

In allen Programmiersprachen gibt es vordefinierte Funktionen (Standardfunktionen) oder Methoden (Standardmethoden), die innerhalb einer Standardbibliothek vorliegen. Dazu zählen beispielsweise die mathematischen Funktionen zur Sinus- (`sin`) oder Wurzelberechnung (`sqrt` – Square root).

9.2 Parameterübergabe

Aktuelle und formale Parameter

Oft ist es notwendig, Werte an Unterprogramme zu übergeben, damit die Methode, Prozedur oder Funktion mit den aktuellen Werten des aufrufenden Programms arbeiten kann. Diese Werte werden **aktuelle Parameter** oder auch **Argumente** genannt.



Bei der Definition der Methode, Prozedur oder Funktion wird in vielen Programmiersprachen genau festgelegt, welche Parameter für die Berechnung benötigt werden. Diese Parameterwerden auch **formale Parameter** genannt.

Die Liste mit allen formalen Parametern wird rechts vom Namen des Unterprogramms angegeben.

Bedingungen für die Übergabe der aktuellen Parameter

Beim Aufruf des Unterprogramms mit den aktuellen Parametern müssen Sie beachten, dass in vielen Sprachen aktuelle und formale Parameter in den folgenden Punkten genau übereinstimmen. In anderen Sprachen wie JavaScript oder PHP muss das allerdings nicht sein.

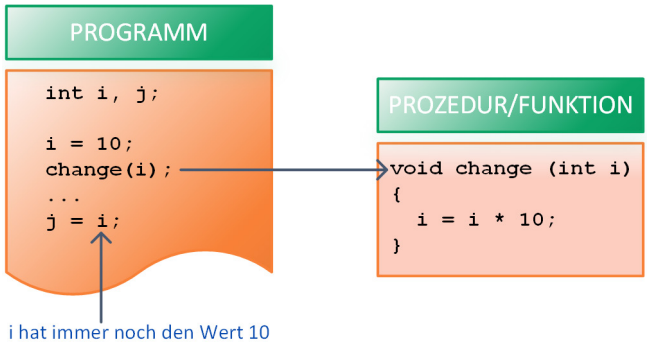
Anzahl der Parameter	Die Anzahl der aktuellen Parameter muss der Anzahl der formalen Parameter entsprechen, wenn die Sprache (etwa Java) das fordert. Bei weniger strengen Sprachen wie JavaScript oder PHP muss das nicht sein.
Reihenfolge der Parameter	Der erste aktuelle Parameter wird an den ersten formalen Parameter übergeben, der zweite aktuelle Parameter an den zweiten formalen Parameter usw.
Datentypen der Parameter	Es hängt an der Art der Sprache, ob die Datentypen der Parameter von Bedeutung sind. In Sprachen ohne strenge Typbindung (JavaScript, PHP, etc.) ist der Datentyp vollkommen irrelevant. In strengen Sprachen wie Java hingegen ist der Datentyp wichtig. Der Datentyp des ersten aktuellen Parameters muss dort mit dem Datentyp des ersten formalen Parameters übereinstimmen, der Datentyp des zweiten aktuellen Parameters mit dem Datentyp des zweiten formalen Parameters usw. Jeder Parameter kann einen anderen Datentyp haben.

- Bei Entwurf von Programmen mit PAP, Struktogramm oder Pseudocode finden meist nur die ersten beiden Punkte Beachtung. Der letzte Punkt – die Übereinstimmung der Datentypen – ist bei der Umsetzung in die Programmiersprache zu beachten.
- Im Pseudocode können Prozeduren beispielsweise durch das Wort `procedure` gekennzeichnet werden, dem der Name der Prozedur folgt. Nach dem Prozedurnamen kann in Klammern eine Liste mit Parametern angegeben werden, die 1 bis n Parameter, durch Komma getrennt, enthalten kann. Bei Funktionen wird entsprechend das Wort `function` zur Kennzeichnung verwendet.

9.3 Parameterübergabe als Wert

Einen Parameter als Wert übergeben

Der Wert (engl. value) einer übergebenen Variablen kann in dem Unterprogramm bei der Parameterübergabe als Wert, auch **call by value** genannt, nicht verändert werden. Bei der Parameterübergabe wird nur eine **Kopie** des Wertes an den formalen Parameter des Unterprogramms übergeben.



Parameterübergabe als Wert

Einen Parameter als Wert in Java übergeben

Bei der Parameterübergabe als Wert in Java können aktuelle Parameter entweder

- primitive Datentypen, z. B. boolean, oder
- Referenztypen sein.

Beispiel für Parameterübergabe: *ChangeNumbers.java*

Im nachfolgenden Beispiel werden die Werte zweier Variablen in einer Methode miteinander vertauscht. Außerhalb der Methode bleiben die Werte der beiden Variablen unverändert.

```
   ...
③    static void change(int n1, int n2)
     {
④      System.out.println("Zahl 1: "+n1+" - Zahl 2: "+n2);
⑤      int temp = n1;
⑥      n1= n2;
⑦      n2= temp;
④      System.out.println("Zahl 1: "+n1+" - Zahl 2: "+n2);
     }
     public static void main(String args[])
     {
①      int n1 = 1234;
       int n2 = 5678;
②      change(n1, n2);
```

```
⑧     System.out.println("Zahl 1: "+n1+" - Zahl 2: "+n2);
    }
...
```

① Die Variablen `n1` und `n2` werden deklariert und initialisiert.

② Diese Variablen werden als aktuelle Parameter beim Aufruf der Methode `change()` übergeben.

③ Die Methode `change()` wird mit zwei Parametern deklariert. Die Namen der aktuellen und formalen Parameter sind in diesem Beispiel gleich. Da sie nicht im selben Gültigkeitsbereich existieren, können sie auch unterschiedliche Namen haben. Da die Methode keinen Rückgabewert liefern soll, muss das in Java mit dem Schlüsselwort `void` gekennzeichnet werden.

④ Die Zahlen werden vor und nach dem Vertauschen ausgegeben.

⑤ – ⑦ Die Werte der Variablen werden vertauscht. Dazu wird eine temporäre Variable `temp` benötigt, in der der Wert von `n1` zwischengespeichert wird. Der Wert der Variablen `n2` wird nun auf `n1` übertragen, und `n2` wird der zwischengespeicherte Wert von `temp` zugewiesen.

⑧ Bei der Ausgabe im Hauptprogramm, haben die Variablen noch ihre ursprünglichen Werte, da die Parameterübergabe als Wert erfolgte.

Die Ausgabe des Programms lautet:

`Zahl 1: 1234 - Zahl 2: 5678` → erste Ausgabe von `change()`
`Zahl 1: 5678 - Zahl 2: 1234` → zweite Ausgabe von `change()`
`Zahl 1: 1234 - Zahl 2: 5678` → Ausgabe von `main()`

Beispiel für Parameterübergabe: *ChangeNumbers.html*

Der nachfolgende Code zeigt die JavaScript-Version von dem letzten Beispiel. Sie sehen hier sowohl, wie in der prozeduralen Programmierung Funktionen eingesetzt werden, als auch wie Sie ohne explizite Berücksichtigung der Datentypen Unterprogramme programmieren.

```
...
③   function change(n1, n2)
    {
④     alert ("Zahl 1: "+n1+" - Zahl 2: "+n2);
⑤     var temp = n1;
⑥     n1= n2;
⑦     n2= temp;
④     alert("Zahl 1: "+n1+" - Zahl 2: "+n2);
    }
①     var n1 = 1234;
      var n2 = 5678;
②     change(n1, n2);
⑧     alert("Zahl 1: "+n1+" - Zahl 2: "+n2);
    ...
```

① Die Variablen `n1` und `n2` werden als globale Variablen deklariert und dabei gleich initialisiert. Dabei wird explizit kein Datentyp festgelegt. In JavaScript wird bei der Deklaration von globalen Variablen entweder gänzlich auf ein Schlüsselwort verzichtet oder das neutrale Schlüsselwort `var` hingeschrieben, wenn man eine Variable deklarieren möchte.

② Diese Variablen werden wie in Java als aktuelle Parameter beim Aufruf der Funktion `change()` übergeben.

③ Die Funktion `change()` wird mit zwei Parametern deklariert. Beachten Sie, dass auch hier keine Datentypen spezifiziert werden – weder bei den Parametern noch beim Rückgabetyp. Dafür verwendet man in JavaScript das Schlüsselwort `function`, um die Deklaration eines Unterprogramms zu beginnen.

④ Die Zahlen werden vor und nach dem Vertauschen ausgegeben.

⑤ – ⑦ Die Werte der Variablen werden vertauscht. Dazu wird erneut eine temporäre Variable `temp` deklariert. Hier wird auch wieder kein Datentyp angegeben. Die Verwendung von `var` innerhalb der Funktion bewirkt allerdings, dass die Variable als lokal verstanden wird.

⑧ Bei der Ausgabe im Hauptprogramm haben die Variablen wieder ihre ursprünglichen Werte, da die Parameterübergabe als Wert erfolgte.

Parameterübergabe als Wert anwenden

Die Parameterübergabe als Wert kann z. B. eingesetzt werden,

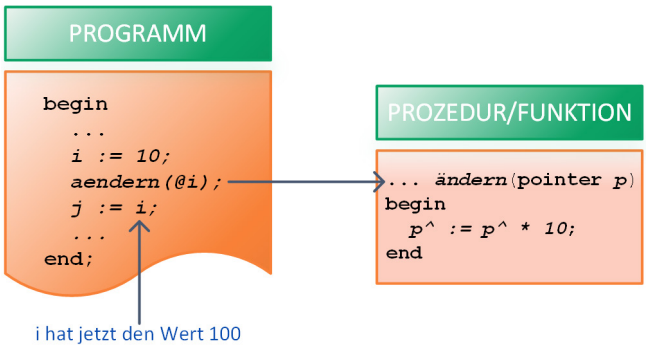
- wenn im Unterprogramm die Werte der Parameter nicht verändert werden
- wenn eine Änderung der Parameterwerte im Unterprogramm auf die als Parameter übergebenen Variablen des Hauptprogramms keine Auswirkungen haben soll

9.4 Parameterübergabe über Referenzen

Einen Parameter als Referenz übergeben

Bisher wurden die aktuellen Parameter an ein Unterprogramm nur als Kopien der Werte der aktuellen Variablen übergeben. Die Änderung des Wertes eines Parameters innerhalb des Unterprogramms hatte keine Auswirkung auf den Wert der Variablen, die im aufrufenden Programm als aktueller Parameter übergeben wurde.

Die Parameterübergabe über Referenzen, auch **call by reference** genannt, stellt eine Möglichkeit dar, Werte der Variablen, die als aktuelle Parameter übergeben werden, innerhalb der Methode, Prozedur bzw. Funktion zu ändern, sodass sie auch nach dem Verlassen des Unterprogramms den neuen Wert behalten. Beim Aufruf des Unterprogramms wird nun statt des Wertes die Adresse (Referenz) der Variablen übergeben, die diesen Wert beinhaltet. Diese verweist dann auf dieselbe Speicherstelle wie die Variable des aufrufenden Programms. Ändern Sie den Wert der Variablen nun in der Methode, Prozedur bzw. Funktion, wird die Änderung auch im aufrufenden Programm sichtbar.



Parameterübergabe als Referenz (Pseudocode)

Die Möglichkeiten der Parameterübergabe und deren Darstellung sind in den einzelnen Sprachen recht unterschiedlich. In Sprachen, in denen Sie mit Zeigern arbeiten können, ist eine Unterscheidung der Übergabeart möglich, z. B. in Pascal oder C++. Java stellt keine Parameterübergabe über Referenzen zur Verfügung.

9.5 Rückgabewerte von Funktionen oder Methoden

Die Rückgabewerte von Funktionen oder Methoden können z. B. primitive Datentypen oder Referenzen sein.

Syntax für Methoden mit Rückgabewert in Java

```
[modifiers] type identifier([type identifier1,...])
{
  ...
  return expression;
}
```

- Vor dem Namen der Methode geben Sie den Datentyp an, den der zurückgegebene Wert der Methode erhalten soll. Dies kann ein primitiver Datentyp sein oder aber auch ein Referenztyp.
- Bevor im Programmablauf die Methode wieder verlassen wird, muss mit dem Schlüsselwort `return` der Ergebniswert festgelegt werden.

Wie Sie oben bereits gesehen haben, geben Sie in JavaScript (oder auch in einigen anderen Sprachen wie PHP) nicht an, ob es einen Rückgabewert gibt und welchen Typ des Rückgabewerts ein Unterprogramm liefert.

Beispiel für einen primitiven Datentyp als Rückgabewert: *FindMax.java*

Im folgenden Programm soll der größere von zwei Variablenwerten ermittelt werden. Der Vergleich wird von einer Funktion ausgeführt. Als Ergebnis ist der größere der beiden Werte zurückzugeben.

```
...
③   static int returnMax(int a, int b)
    {
④     if (a > b)

⑤       return a;
      else
⑥       return b;
    }
    public static void main(String args[])
    {
①     int n1 = 1234;
      int n2 = 5678;
      int max = 0;
②     max = returnMax(n1, n2);   Aufruf einer Funktion in einer Wertzuweisung
⑦     System.out.println("Maximum: "+max);
    }
...
```

① Die Variablen `n1` und `n2` werden initialisiert.

② Die Methode `returnMax()` wird aufgerufen, um den größeren der beiden Werte zu ermitteln. Als Parameter werden die beiden Variablen `n1` und `n2` übergeben.

③ Die Methode ist so definiert, dass sie mit zwei Parametern aufgerufen werden muss und einen Rückgabewert vom Typ `int` zurückliefert.

④ Die Werte der beiden Parametervariablen werden verglichen.

⑤, ⑥ Ist der Wert der Variablen `a` größer, wird `a` als Rückgabewert an das Hauptprogramm geliefert, sonst `b`.

⑦ Das Maximum wird nun ausgegeben. Die Ausgabe lautet: `Maximum: 5678`.

Zeiger als Rückgabewert von Funktionen oder Methoden

Sie können aus einer Funktion oder Methode auch Zeiger zurückgeben, wenn eine Sprache das unterstützt. Dadurch ist der Rückgabewert der Funktion oder Methode eine Adresse, die auf einen bestimmten Wert verweist. Diese Form der Rückgabe ist nützlich, wenn Sie nicht der Wert, sondern die Speicherstelle eines Wertes interessiert. Allerdings dürfen Sie keinen Zeiger zurückgeben, der auf eine lokale Variable der Funktion oder Methode zeigt. Diese ist nach dem Verlassen des Unterprogramms ungültig und in diesem Fall auch der Zeiger. In Java gibt es keine expliziten Zeiger oder einen speziellen Datentyp `Zeiger`. Rückgabewerte können allerdings Referenztypen sein.

Beispiel

Sie möchten in einer Zeichenkette das erste Auftreten eines Leerzeichens ermitteln. Dazu übergeben Sie die Adresse der Zeichenkette an eine Funktion, die ihrerseits die Adresse des gefundenen Leerzeichens zurückgibt.

9.6 Übungen

Übung 1: Prozeduren bzw. Methoden ohne Rückgabewert

Übungsdatei: -- **Ergebnisdateien: *uebung09.pdf, MoveRight.java***

1. Schreiben Sie eine Prozedur in Pseudocode, die alle Elemente eines Feldes um jeweils eine Position nach rechts verschiebt. Das letzte Element soll das erste Element werden.
2. Setzen Sie den Pseudocode aus Aufgabe 1 in Java-Code um, wobei Sie dann eine Methode ohne Rückgabewert statt einer Prozedur erstellen.
3. Erstellen Sie eine Prozedur in Pseudocode, der als Parameter der Radius einer Kugel übergeben wird. In der Prozedur sind der Umfang der Kugel, die Mantelfläche und das Volumen zu berechnen. Die Werte sollen anschließend dem aufrufenden Programm zur Verfügung stehen.

Umfang einer Kugel (Kreis):	2*Π*R	Π entspricht der Konstanten Pi = 3.14
Mantelfläche der Kugel:	4*Π*R2	R entspricht dem Radius
Volumen einer Kugel:	4 / 3*Π*R3	

Übung 2: Funktionen bzw. Methoden mit Rückgabewert

Übungsdatei: -- **Ergebnisdateien: *uebung09.pdf, FindMin.java, UsePower.java***

1. Schreiben Sie eine Methode in Pseudocode, die das Minimum dreier verschiedener Werte bestimmt und an das Hauptprogramm zurückgibt. Lösen Sie dieses Problem mithilfe der Parameterübergabe call by value.
2. Lösen Sie die Aufgabe 1 mit einem Java-Programm. Speichern Sie das Programm unter *FindMin.java*.
3. Erstellen Sie eine Funktion in Pseudocode, die die n-te Potenz einer Zahl `x` berechnet ($y = x^n$). Die Zahlen x und n sollen der Funktion als Parameter übergeben werden, wobei `x` und `n` natürliche Zahlen (0, 1, 2, 3 …) sind. Das Ergebnis soll im Hauptprogramm ausgegeben werden.
4. Erstellen Sie für die Aufgabe 3 ein Java-Programm unter Verwendung passender Methoden. Die Methode soll mit `x = 5` und `n = 4` getestet werden. Speichern Sie das Programm unter *UsePower.java*.

Übung 3: Mit Parametern arbeiten

Übungsdateien: *SingleLinkedList.java*, *UseSingleLinkedList.java* **Ergebnisdateien: *SingleLinkedList.java*, *UseSingleLinkedList.java***

1. Erweitern Sie die Methode `insertNode()` im Programm *SingleLinkedList.java* um einen Parameter, mit dem für den einzufügenden Knoten die gewünschte Einfügeposition angegeben werden kann.
2. In der Methode `deleteNode()` soll der zu löschende Knoten anhand seines Indexes bestimmt werden. Passen Sie die Methode entsprechend an.
3. Ändern Sie die Argumente der Methodenaufrufe im Programm *UseSingleLinkedList.java* entsprechend den bearbeiteten Parameterlisten.

10 Einführung in die objektorientierte Programmierung (OOP)

In diesem Kapitel erfahren Sie

- welche theoretischen Aspekte die objektorientierte Programmierung hat
- welche Prinzipien der objektorientierte Programmierung zu Grunde liegen
- was Klassen, Objekte, Attribute und Methoden sind
- was unter Vererbung und Polymorphie zu verstehen ist

Voraussetzungen

- Grundlegende Kenntnisse in der Programmierung

10.1 Kennzeichen der objektorientierten Programmierung

Viele moderne und leistungsfähige Programmiersprachen sind ganz oder teilweise objektorientiert. Insbesondere Java ist streng objektorientiert, auch C++ oder C# sind objektorientiert. JavaScript hingegen ist objektbasierend, nutzt aber damit auch wesentliche Techniken der objektorientierten Programmierung. Mit Disziplin können Sie auch in JavaScript objektorientiert programmieren. Andere Sprachen wie PHP sind hybrid und können damit sowohl prozedural, als auch objektorientiert eingesetzt werden. Um solche Sprachen effektiv nutzen zu können, müssen Sie die Konzepte der objektorientierten Programmierung kennen.

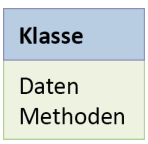
Was bedeutet objektorientiertes Programmieren?

Die Weiterentwicklung der Erkenntnisse und Erfahrungen der strukturierten Analyse führten zur objektorientierten Programmierung. Ihr Kernstück sind sogenannte Klassen. Klassen sind Datentypen, die sich an der realen Welt orientieren. In einem Programm, das beispielsweise eine Tabellenkalkulation realisiert, gibt es dementsprechend Klassen, mit denen z. B. ein Tabellenblatt, eine Zeile, eine Spalte oder ein Diagramm beschrieben werden kann.

Um die entscheidenden Ziele bei der Entwicklung großer Softwaresysteme, wie z. B. Produktivitäts- und Qualitätssteigerung, zu realisieren, werden bei objektorientierten Programmiersprachen die Wiederverwendbarkeit, die Erweiterbarkeit und die Kompatibilität in den Vordergrund gestellt. Objektorientierte Programmiersprachen bieten Konzepte wie Abstraktion, Datenkapselung, Modularität und Hierarchieprinzipien, um diese Ziele zu erreichen.

Klassen und Objekte

In Klassen werden Daten und Methoden zusammengefasst. Methoden dienen dem Zugriff auf bzw. Verändern von Daten in der Klasse.



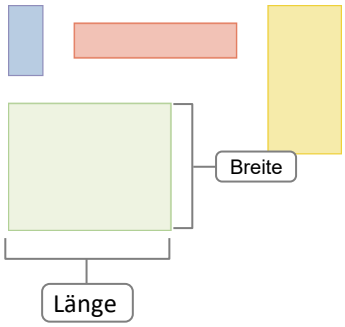
Aus Klassen können **Objekte** erzeugt werden. Objekte sind sogenannte **Instanzen** einer Klasse. Der Aufbau und die Funktionsweise eines Objekts werden durch die Daten und Methoden der Klasse festgelegt.

- Die **Daten** einer Klasse entsprechen den Daten, die ein Objekt aufnehmen kann. Sie sind von einem bestimmten Datentyp, der elementar, strukturiert oder wiederum ein Objekt sein kann (Referenztyp). Wird eine Instanz der Klasse gebildet, besitzt jedes Objekt diese Daten. Die Werte können aber bei jedem Objekt unterschiedlich sein. Auf die Daten kann in der Regel von außen nicht direkt zugegriffen werden, sondern nur über die entsprechenden Methoden (**Datenkapselung**). Daten beschreiben die Eigenschaften von Klassen bzw. Objekten. Diese Daten werden auch **Attribute** genannt.
- Eine **Methode** (Operation) ist wie eine Prozedur oder Funktion aufgebaut. Sie wird innerhalb einer Klasse definiert. Methoden können auf die Daten eines Objekts dieser Klasse zugreifen. Methoden beschreiben das Verhalten von Klassen bzw. Objekten und stehen nur über die Klasse bzw. ein Objekt der Klasse zur Verfügung.

Beispiel für das Zuordnen von Daten zu einer Klasse:

Aus der Geometrie kennen Sie Formen, die vier Eckpunkte besitzen und deren benachbarte Kanten im rechten Winkel (90°) zueinander stehen.

Alle diese Formen gehören zur Gruppe (Klasse) der Rechtecke. Das Rechteck mit der Breite 3 cm und der Länge 5 cm ist ein bestimmtes Exemplar (Objekt) dieser Klasse.



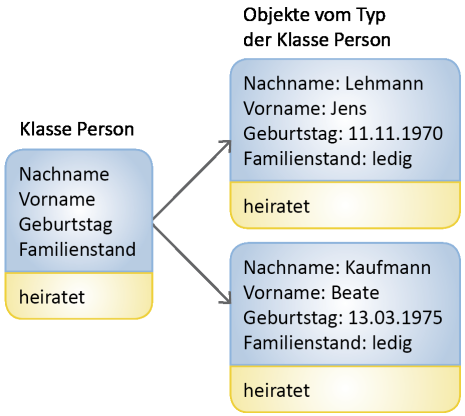
Rechtecke können verschieden lang sein, aber sie besitzen alle eine Länge. Auf diese Weise lassen sich gemeinsame Daten finden, die die Klasse `Rechteck` auszeichnen:

- Breite
- Länge

Beispiel für Objekte aus einer Klasse:

In einer Klasse `Person` werden die Daten `Nachname`, `Vorname`, `Geburtstag` und `Familienstand` gespeichert. Wenn eine Person heiratet, ändern sich der Familienstand und gegebenenfalls der Nachname. Diese Änderungen werden durch die Methode `heiratet` durchgeführt.

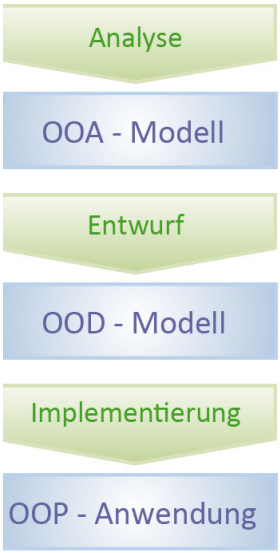
Von der Klasse `Person` werden hier zwei Objekte erzeugt.



Objekte können einen spezifischen **Zustand** besitzen, der die Verfügbarkeit prinzipiell vorhandener Methoden beeinflussen kann. So kann eine Person etwa nicht heiraten, wenn sie bereits verheiratet ist.

10.2 Stufen der OOP

Der Weg vom Problem zur Softwarelösung verläuft beim objektorientierten Ansatz in verschiedenen Phasen (Analyse, Entwurf und Implementierung). Für eine Lösung können abhängig vom Umfang auch mehr oder weniger Phasen benötigt werden.



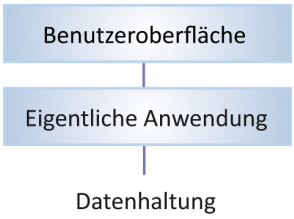
Objektorientierte Analyse (OOA)

Der zu modellierende Prozess wird aus der Sicht des Anwenders und seiner Tätigkeiten analysiert. Das Ziel der Analyse ist es, ein Modell des betrachteten Systems zu erzeugen, welches das System mit der verwendeten Analysemethode dokumentiert. Dieses Modell wird auch fachliches Modell genannt und ist eine Schnittstelle zwischen allen beteiligten Personen im Prozess der Softwareerstellung: Kunde (Manager, Nutzer) und Auftragnehmer (Projektleiter, Entwickler). Für die Notation des Modells kann die UML (Unified Modeling Language) verwendet werden.

Ergebnis: In der objektorientierten Analyse wird ein fachliches Modell des Anwendungsbereiches erstellt. Es wird das „WAS“ des Systems modelliert.

Objektorientierter Entwurf (OOD)

Im objektorientierten Entwurf, kurz OOD (Object Oriented Design), wird das OOA-Modell aus der Sicht von Datenstrukturen und deren Kommunikation untereinander betrachtet. Ziel des Entwurfs ist es, eine objektorientierte Strukturierung des Softwaresystems vorzunehmen. Beim OOD geht es darum, ein Problem objektorientiert zu zerlegen und diese Zerlegung von der jeweiligen Programmiersprache losgelöst zu beschreiben. Das Ergebnis sind logische Modelle auf der Ebene von Klassen und Objekten als Beschreibung der Struktur und die Beschreibung des Verhaltens des Systems durch die Dokumentation der Interaktionen von Objekten. Im Mittelpunkt des Designs steht die Bildung von Klassen, also die Vereinigung von Daten und Methoden.



Häufig wird die Anwendung in die Schichten grafische Benutzeroberfläche (GUI), eigentliche Anwendung und Datenhaltung unterteilt.

Ergebnis: Im objektorientierten Entwurf wird aus dem fachlichen Modell ein technisches Modell für die Software erstellt. Es wird das Wie der Umsetzung geklärt.

Objektorientierte Programmierung (OOP)

Die Umsetzung des OO-Entwurfs erfolgt mithilfe einer objektorientierten Programmiersprache. Die im OOD-Modell definierten Datenstrukturen und Beziehungen werden in den Anweisungen einer oder mehrerer konkreter objektorientierter Programmiersprachen übersetzt.

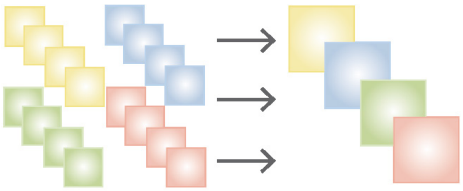
Objektorientierte Programme sind Sammlungen von Klassen, die in einer gegliederten Rangordnung und einer spezifizierten Interaktionsfolge zueinander stehen. Während der Programmausführung werden konkrete Objekte als Instanzen der Klassen erzeugt, die den Programmablauf durch den Austausch von Nachrichten steuern.

Ergebnis: In der objektorientierten Programmierung wird das technische Modell in Quelltexte einer objektorientierten Programmiersprache umgesetzt. Es entsteht die Software.

10.3 Prinzipien der OOP

Abstraktion

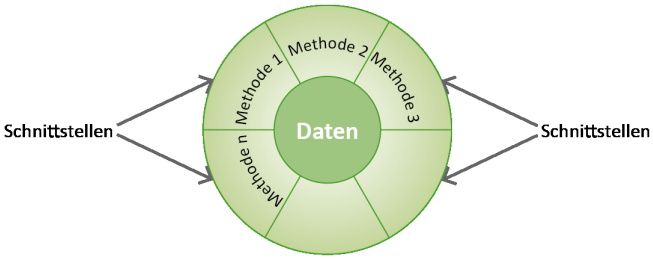
Beim Vorgang der Abstraktion werden gemeinsame Daten und Methoden ermittelt, die für eine Problemstellung relevant sind und in einer Klasse zusammengefasst werden.



Gleiche Daten und Methoden finden und abstrahieren

Datenkapselung

Als Datenkapselung wird die Beschränkung des Zugriffs von außen auf Daten einer Klasse bezeichnet. Auf die Daten einer Klasse darf nur indirekt über Methoden der Klasse zugegriffen werden. Die Methoden einer Klasse bilden eine definierte Schnittstelle nach außen. Auf diese Weise lassen sich die Daten einer Klasse verbergen. Somit können gegebenenfalls Änderungen an der inneren Struktur der Klasse durchgeführt werden, ohne dass die Software, die diese Klasse verwendet, geändert werden muss, solange die Schnittstelle gleich bleibt. Durch die Kapselung von Daten und den sie verändernden Methoden werden die Änderungsmöglichkeit und die Wartbarkeit einer Software verbessert.



Darstellung der Datenkapselung

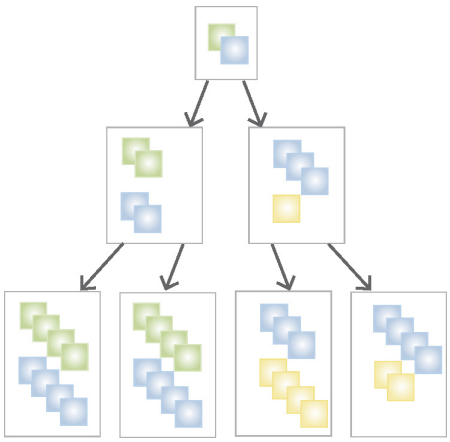
Beachten Sie, dass der Begriff der **Schnittstelle** hier nicht im Sinn einer Java-Schnittstelle verwendet wird.

Modularität

Das Modularitätsprinzip bedeutet allgemein, dass Programme in einzelne Module unterteilt werden. Dies bedeutet in der objektorientierten Programmierung, dass Sie Klassen als einzelne Module entwickeln. Damit bei größeren Programmen keine unüberschaubare Anzahl von Klassen entsteht, werden die Klassen, die in einem logischen Zusammenhang stehen, in einem Modul zusammengefasst. Sollten später Änderungen erforderlich sein, beschränken sich diese auf einige wenige Module und betreffen nicht das gesamte Programmsystem.

Hierarchie und Vererbung

Klassen enthalten oft ähnliche Daten und Methoden. Bei der Bildung von Klassen wird geprüft, ob sich vorhandene Daten und Methoden generalisieren lassen. Ist dies der Fall, wird eine übergeordnete Klasse angelegt, welche die gemeinsamen Daten oder Methoden enthält. Eine übergeordnete Klasse (Superklasse) **vererbt** ihre Daten und Methoden an die untergeordneten Klassen (Subklassen). Dies bedeutet, dass die Daten und Methoden automatisch den untergeordneten Klassen zur Verfügung stehen. Außer den geerbten Daten und Methoden verfügt eine untergeordnete Klasse über eigene Daten oder Methoden. Eine untergeordnete Klasse kann wiederum einer weiteren Klasse übergeordnet sein usw., sodass sich eine Klassenhierarchie ergibt. Aus dem Blickwinkel der Abstraktion stellt eine übergeordnete Klasse eine abstraktere Darstellung ihrer untergeordneten Klassen dar.



Klassenhierarchie

Typisierung

Bei stark typisierten Sprachen ist schon zur Zeit der Kompilierung bekannt, von welchem Datentyp eine bestimmte Variable ist. Der Datentyp ist nicht nur eine Angabe über den benötigten Speicherbedarf, sondern vor allem auch eine Festlegung, für welche Funktionen diese Variable eingesetzt werden kann. Durch die Festlegung auf einen bestimmten Typ kann schon während der Kompilierung überprüft werden, ob bestimmte Funktionen auf den entsprechenden Datentyp anwendbar sind. Bei einer objektorientierten Sprache wird zusätzlich überprüft, ob eine Methode auf ein übergebenes Objekt angewandt werden kann.

10.4 Klassen

Um mit Objekten arbeiten zu können, werden Klassen benötigt. Klassen können entweder fertig in Bibliotheken der objektorientierten Programmiersprache zur Verfügung stehen oder sie werden vom Programmierer selbst erstellt.

Syntax für die grundlegende Klassendefinition in Java

- Die Definition einer Klasse beginnt mit dem Schlüsselwort `class` (Klasse), gefolgt von dem Namen der Klasse.
- Alle weiteren Beschreibungen innerhalb der Klasse werden in geschweifte Klammern { } eingeschlossen.
- Die Klassendefinition sollte in einer eigenständigen Textdatei gespeichert werden. Der Name dieser Datei entspricht dem Namen der Klasse und erhält die Erweiterung *.java*.
- Der Klassendefinition können Sie bei Bedarf sogenannte Modifizierer (modifiers) voranstellen. Mit Modifizierern haben Sie die Möglichkeit, beispielsweise die Zugriffsrechte auf die Klasse festzulegen.

```
[modifiers] class identifier
{
   ...
}
```

Der Name einer Klasse sollte immer mit **Großbuchstaben** beginnen. Es wird empfohlen, Substantive zu verwenden, die dann jeweils ohne Leerzeichen angehängt werden und ebenfalls mit einem Großbuchstaben beginnen. Sinnvolle Klassennamen sind beispielsweise *ProductList* oder *CustomerData*.

Beispiel für die Definition einer Klasse: *Rectangle.java*

- Der nebenstehende Quelltextauszug zeigt die Definition einer Klasse Rechteck (`Rectangle`).
- Diese Klassendefinition schreiben Sie vorzugsweise in eine neue Datei, deren Name dem Klassennamen entspricht (*Rectangle.java*).

```
class Rectangle
{
   ...
}
```

10.5 Daten (Attribute)

Eine Klasse gibt die Daten und Methoden für die Objekte vor. Alle Objekte einer Klasse besitzen die gleichen Daten, aber unterschiedliche Datenwerte. Der Name der Daten muss im Kontext der Klasse eindeutig sein und sollte auf den Inhalt schließen lassen. Wenn Sie bei einer Klasse eine Datenkapselung vornehmen, können deren Daten in der Regel nur noch indirekt mithilfe von Methoden verändert und gelesen werden. Nach außen hin sind die Daten dann geschützt, sodass sie für andere Objekte nicht direkt sichtbar sind. Vielfach setzt man dann für den Zugriff auf die Daten Zugriffsmethoden ein, eine Lese- und eine Schreiboperation (`get`- und `set`-Methoden). Setzen Sie nur eine Zugriffsmethode ein, können Sie Nur-Lese- bzw. Nur-Schreib-Eigenschaften realisieren.

Syntax für die Definition von Daten in Java

Die Daten in einer Klasse werden in Java auch **Attribute** oder **Felder** genannt.

```
[modifiers] type identifier[,
identifier1...];
```

Attribute werden **außerhalb** einer Methode, aber noch in der Klassenbeschreibung definiert.

- Die Definition eines Attributs beginnt mit dem Schlüsselwort für den Datentyp, gefolgt vom Namen des Attributs (`identifier`).
- Um dem Attribut einen definierten Wert zu geben, sollten Sie diesen mit einer Wertzuweisung festlegen.
- Die Definition eines Attributs ist eine Anweisung und wird daher mit einem Semikolon abgeschlossen.
- Der Definition können Modifizierer vorangestellt werden, über die Sie wie bei der Definition einer Klasse beispielsweise die Zugriffsrechte festlegen können.
- In einer Anweisung können Sie mehrere Attribute des gleichen Typs definieren. Dazu geben Sie, durch Kommata getrennt, die Namen der Attribute an.

Der Name eines Attributs sollte in Java immer mit einem Kleinbuchstaben beginnen. Sinnvolle Attributnamen sind beispielsweise *numberArticles* oder *maxTextLength*. Beachten Sie, dass in anderen Namenskonventionen (etwa oft bei Microsoft) Attributnamen auch mit einem Großbuchstaben beginnen können. Wichtig ist nur die Konsistenz.

Das Beispiel zeigt die Klasse `Rectangle`, die zwei Attribute vom Datentyp `int` enthält.

```
class Rectangle
{
  int width = 0;
  int length = 0;
  ...
}
```

10.6 Objekte

Ein Objekt ist die konkrete Ausprägung (Instanz) einer Klasse. Damit ist ein Objekt mit seinen eigenen Daten etwas Einmaliges. Die Methoden und Daten einer Klasse können dagegen von allen Objekten dieser Klasse gemeinsam genutzt werden.

Es gibt zwei verschiedene Arten von Objekten. Aktive Objekte steuern den Ablauf eines Systems, und passive Objekte reagieren nur dann, wenn sie von anderen angesprochen werden. Je nach Programmiersprache werden Objekte unterschiedlich erzeugt.

Syntax für die Erzeugung von Objekten mit `new` in Java

Beim Erzeugen eines Objekts einer Klasse werden meist zwei Schritte durchgeführt:

① Zunächst definieren Sie eine Variable vom Typ der gewünschten Klasse.

Die Variable wird als Referenzvariable bezeichnet und ist später nur eine Referenz (Verweis) auf das eigentliche Objekt. Es wird nur Speicher für die Referenz reserviert. Das Objekt selbst existiert noch nicht.

② Anschließend erzeugen Sie mit dem Operator `new` ein Objekt der Klasse und weisen die Referenz darauf der vorher erzeugten Variable zu. Nach `new` geben Sie den Klassennamen und runde Klammern ( ) ein. Dies bezeichnet den sogenannten Konstruktor (vgl. Abschnitt 10.8).

```
① ClassName identifier;

② identifier = new ClassName();
```

Beispiel für die Erzeugung eines Objekts

```
...
①    Rectangle specialRect;
②    specialRect = new Rectangle();
③    Rectangle anotherRect = new Rectangle();
...
④    specialRect.width = 5;.
...
```

① Die Variable `specialRect` vom Typ `Rectangle` wird definiert. Bei der Definition einer Variablen vom Typ der Klasse `Rectangle` wird eine sogenannte **Referenzvariable** erzeugt, die später eine Referenz (einen Verweis) auf das eigentliche Objekt darstellt. Das Objekt selbst existiert zunächst noch nicht. Speicherplatz wird auch nur für die Aufnahme der Referenz belegt.

② Mit dem Operator `new` wird ein Objekt der Klasse `Rectangle` erzeugt und die Referenz auf dieses Objekt der Referenzvariablen `specialRect` zugewiesen. Dabei wird auch entsprechend Speicherplatz für das Objekt bereitgestellt. Sofern den Attributen in der Klassendefinition spezielle Werte zugewiesen wurden, werden diese Werte als Eigenschaften der Objekte vorgegeben. Anderenfalls wird beispielsweise bei Attributen vom Typ `int` automatisch der Wert `0` verwendet.

③ Objekte können auch in einer Anweisung definiert und erzeugt werden.

④ Die mit `new` erzeugten Objekte besitzen die Attribute der entsprechenden Klasse. Sie können auf die Attribute der Objekte zugreifen, indem Sie den Namen des Objekts mit einem Punkt abgetrennt vor den Attributnamen schreiben, z. B. `specialRect.width = 5;.`

Wenn Sie ein Objekt nur genau einmal benötigen, können Sie es auch anonym erzeugen. Das bedeutet, dass Sie nur mit dem Operator `new` ein Objekt der Klasse erzeugen und direkt mit einem nachfolgenden Punkt eine Methode oder Eigenschaft verwenden. Die Zuweisung zu einer Referenzvariable kann unterbleiben, aber das Objekt steht dann auch nicht mehr für weitere Aktionen zur Verfügung.

10.7 Methoden

Als **Methoden** oder auch Operationen werden die Aktionen bezeichnet, die auf ein Objekt angewendet werden können. Methoden sind das objektorientierte Äquivalent von Funktionen oder Prozeduren (vgl. Kapitel 9) und werden innerhalb einer Klasse definiert. Sie arbeiten in der Regel auch nur mit den Attributen der Objekte dieser Klasse, auf die sie direkt zugreifen können.

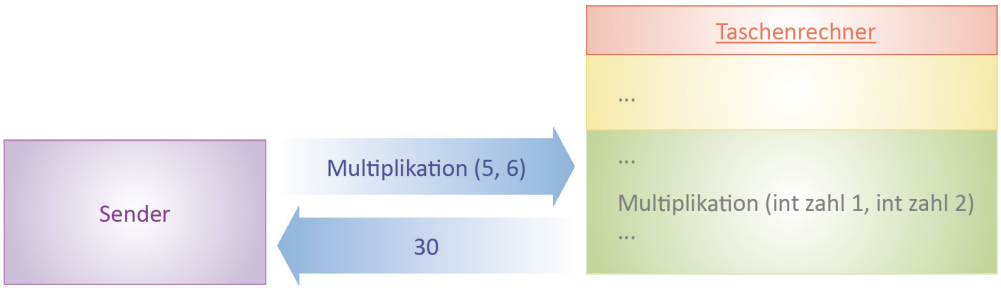
Methode als Botschaft (Nachricht) ausführen

Jede Methode, die für ein Objekt aufgerufen wird, wird als Botschaft an das Objekt interpretiert. Das Objekt führt die gleichnamige Methode aus und gibt die entsprechenden Daten an den Sender der Botschaft zurück.

Beachten Sie, dass eine Botschaft vom Objekt auch ignoriert werden kann und damit die gewünschte Methode nicht ausgeführt wird. Das bedeutet in der Regel nicht, dass die Methode grundsätzlich nicht ausgeführt werden kann sondern nur, dass sie in einer anderen Situation (etwa erst zu einem späteren Zeitpunkt) verfügbar ist. Der Zustand eines Objekts kann also in gewissen Situationen die Ausführung einer Methode verhindern, was als Ignorieren der Botschaft verstanden werden kann.

Beispiel:

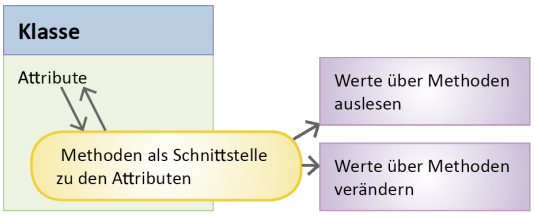
In einer Klasse `Taschenrechner` gibt es eine Methode `Multiplikation`, die mit zwei Integer-Werten aufgerufen werden kann und das Produkt der Zahlen zurückliefert.



Methode ausführen

Methoden für den kontrollierten Zugriff auf Daten

Bei der Datenkapselung benötigt eine Klasse in der Regel spezielle Methoden, um die Daten der Klasse zur Verfügung zu stellen oder zu ändern.



Kontrollierter Zugriff auf gekapselte Attribute

Um in Java die Daten einer Klasse zu kapseln, werden die Attribute mit dem Zugriffsmodifizierer `private` geschützt. Die so gekennzeichneten Attribute sind nur innerhalb der Klasse selbst sichtbar, und der Zugriff ist so weit eingeschränkt, dass nur Methoden der Klasse selbst darauf zugreifen können:

- Die sogenannten Getter-Methoden liefern bei Bedarf den Attributwert nach außen.
- Die sogenannten Setter-Methoden ändern bei Bedarf Attributwerte.

Der Name der Getter-Methoden wird gewöhnlich aus dem Wort *get* (get=erhalten) und dem Namen des Attributs gebildet, der dann mit einem Großbuchstaben beginnt. Eine Ausnahme bilden Attribute des Datentyps `boolean`. Den Namen für Getter-Methoden auf diese Attribute stellen Sie in der Regel das Wort *is* voran. Die Methode heißt beispielsweise `isFilled`.

Ähnlich den Getter-Methoden setzt sich der Name einer Setter-Methode üblicherweise aus dem Wort *set* (set = setzen) und dem Namen des Attributs, der dann mit einem Großbuchstaben beginnt, zusammen.

Eine Setter-Methode besitzt einen Parameter, mit dem der künftige Wert des Attributs mitgeteilt wird. Als Name des Parameters wird gewöhnlich ebenfalls der Attributname verwendet. So heißt in der Methode `setLength` der Parameter ebenfalls `length`. Da die Methode aber zwischen den beiden Variablen mit dem Namen `length` unterscheiden muss, wird die Referenzvariable `this` verwendet. Sie besagt, dass das Attribut gemeint ist.

Mit jedem Objekt wird vom Compiler automatisch eine Referenzvariable auf das eigene Objekt erzeugt, die den Namen `this` trägt. Diese Referenzvariable kann in allen Methoden des Objekts eingesetzt werden. `this` ist ein Schlüsselwort und kann deshalb nicht in Ihren eigenen Programmen als Bezeichner eingesetzt werden.

Die Referenzvariable `this` existiert (nur) für jedes erzeugte Objekt. Da statische Methoden, die Sie im weiteren Verlauf des Kapitels kennenlernen, nicht an Objekte gebunden sind, kann in statischen Methoden die Referenzvariable `this` nicht verwendet werden.

Beispiel der Getter- und Setter-Methoden in Java: *Rectangle.java*

```
   class Rectangle
   {
①    private int width = 0;
     private int length = 0;
     ...
②    int getWidth()
     {
       return width;
     }
     ...
③    void setWidth(int width)
     {                    ⑤
④      this.width = width;
     }
     ...
   }
```

① Definiert und initialisiert die Attribute `width` und `length` vom Datentyp `int` als geschützte Attribute innerhalb der Klasse `Rectangle`.

② Die Getter-Methode `getWidth()` liefert den Attributwert von `width`.

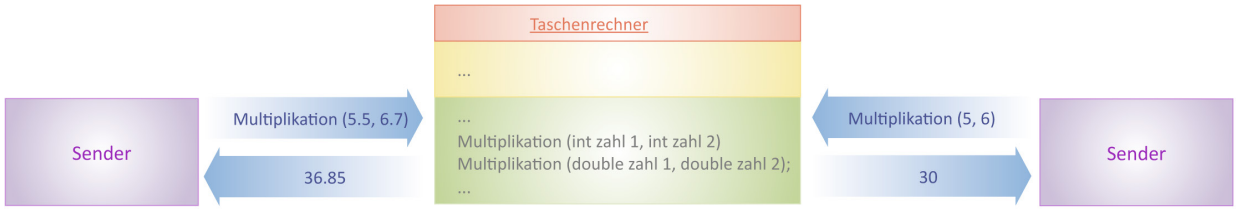
③ Der neue Attributwert von `width` ergibt sich aus dem in der Setter-Methode `setWidth()` übergebenen Parameter `width` ⑤. Da sowohl der Parameter als auch das Attribut denselben Namen haben, muss innerhalb der Setter-Methode `setWidth()` auf das Attribut `width` ① mit der Referenzvariablen `this` ④ zugegriffen werden.

Überladen von Methoden

Eine Methode wird in der OOP bei vielen Sprachen nicht nur durch ihren Namen identifiziert, sondern auch durch die Anzahl und die Typen der Parameter. Dadurch ist es möglich, dass in solchen Sprachen verschiedene Methoden denselben Namen besitzen, wenn sie unterschiedliche Parameterlisten haben. Dies wird als Überladen von Methoden (Overloading) bezeichnet.

Beispiel

In einer Klasse `Taschenrechner` gibt es eine Methode `Multiplikation`, die einmal mit zwei Integer-Werten aufgerufen werden kann und einmal mit zwei Real-Werten. Je nachdem, von welchem Typ die Übergabeparameter sind, arbeitet diese Methode dann mit ganzen Zahlen oder mit Gleitpunktzahlen. Übergeben Sie aber einen `int`- und einen `double`-Wert, kommt es zum Fehler, da es eine Methode mit einer solchen Parameterliste nicht gibt.



Methoden überladen

Was sind statische Variablen und statische Methoden?

Statische Variablen und Methoden sind nicht an die Existenz eines Objekts einer Klasse gebunden. Objektvariablen gehören zu einem Objekt vom Datentyp der Klasse und stellen die Eigenschaften des Objekts dar. Sie existieren somit erst dann, wenn das Objekt existiert, d. h. nach der Initialisierung der Instanzvariablen. Demgegenüber gehören statische Variablen zu der Klasse selbst. Sie werden daher auch als **Klassenvariablen** bezeichnet. Klassenvariablen existieren genau einmal, und es ist nicht erforderlich, dass Objekte vom Datentyp der Klasse existieren.

Ebenso ist die Festlegung von statischen Methoden möglich.

Einsatzgebiete statischer Variablen und Methoden sind Klassen, deren Funktionalitäten unabhängig von konkreten Objekten sind. Beispielsweise sind in Java alle Methoden der Klasse `java.lang.Math` statisch und stellen mathematische Grundfunktionen bereit.

Statische Variablen und Methoden verwenden

Sie können jederzeit über den Klassennamen (ohne dass ein Objekt der Klasse existiert) auf statische Variablen und Methoden der Klasse zugreifen.

Syntax statischer Variablen und Methoden in Java

```
[modifiers] static type identifier; ①

[modifiers] static type identifier1([type identifier2,...]) ②
{
  return expression;
}
```

① Um eine statische Variable oder Methode zu erzeugen, beginnen Sie die Definition mit dem Schlüsselwort `static`. Bei Bedarf stellen Sie Sichtbarkeitsmodifizierer voran.

② Statische Variablen können Sie wie andere Variablen sofort initialisieren.

Statische Methoden können direkt nur mit statischen Variablen und Methoden arbeiten, da diese unabhängig von einem Objekt der Klasse immer vorhanden sind. Allerdings können Sie in statischen Methoden jederzeit ein Objekt einer verfügbaren Klasse erzeugen und darüber dessen Objektmethoden und -eigenschaften verwenden.

Die statische Methode `main` in Java

Um eine normale Java-Anwendung zu schreiben, benötigen Sie mindestens eine Klasse, in der Sie die Methode `main()`, das sogenannte Hauptprogramm, notieren. Von der Klasse steht beim Start jedoch noch kein Objekt zur Verfügung. Daher muss die Methode `main()` statisch sein, damit sie dennoch vom Interpreter ausgeführt werden kann.

Syntax der Methode `main`

- Benennen Sie die Klasse mit dem Programmnamen ①.
- Speichern Sie die Klasse in einer gleichnamigen Datei mit der Erweiterung *.java*.
- Innerhalb dieser Klasse schreiben Sie eine Methode `main()`, die beim Programmstart ausgeführt wird.
- Der Anweisungsblock der Methode `main()` enthält Anweisungen, d. h. die Definition von Variablen, Wertzuweisungen und Methodenaufrufe.

```
class ProgramName①
{
  ...
/*********************
* die Methode main, das Hauptprogramm
*********************/
  public static void main(String[] args)
  {
    //statements
    //  - Variablendefinitionen
    //  - Zuweisungen
    //  - Methodenaufrufe
  }
  ...
}
```

Die Methode `main()` verwendet Argumente. Auf diese Weise lassen sich beim Programmstart Parameter übergeben.

10.8 Konstruktoren

Um Objekte bei der Deklaration mit bestimmten Werten zu initialisieren, können in Klassen spezielle Methoden definiert werden. Diese Methoden heißen **Konstruktoren**. Sie haben gegenüber Standardmethoden folgende zusätzliche Eigenschaften:

- Die Methode wird als Konstruktor gekennzeichnet, z. B. durch ein spezielles Schlüsselwort. In einigen Sprachen wird ein Konstruktor durch ein Schlüsselwort wie `constructor` oder `create` definiert, in anderen (etwa Java oder C#) muss er den gleichen Namen wie die Klasse besitzen.
- Ein Konstruktor liefert in der Regel keinen Rückgabewert.
- Ein Konstruktor wird automatisch beim Erzeugen eines Objekts aufgerufen.
- Ein Konstruktor kann bzw. sollte nicht wie eine normale Methode aufgerufen werden. Der Aufruf erfolgt automatisch beim Anlegen einer Instanz der Klasse.
- Eine Klasse kann in den meisten Sprachen mehrere Konstruktoren besitzen.

Beispiel für Konstruktoren in Java: *Rectangle.java, Geometry.java*

Java hält standardmäßig für jede Klasse einen Standardkonstruktor bereit. Sobald Sie einen eigenen Konstruktor erstellen, existiert der Standardkonstruktor nicht mehr.

Standardkonstruktor

```
Rectangle specialRect =
  new Rectangle();
```

```
...
Rectangle()  Konstruktor ohne Parameter
{
  setWidth(2);
  setLength(2);
}
Rectangle(int width, int length)  Konstruktor mit zwei Parametern
{
  setWidth(width);
  setLength(length);
}
...
```

Konstruktoren definieren

```
class Geometry
{
  ...
  // ein Rechteck mit der Breite 2 und der Länge 2 erzeugen
  Rectangle specialRect = new Rectangle();   Konstruktor ohne Parameter aufrufen

  // ein Rechteck mit der Breite 3 und der Länge 5 erzeugen
  Rectangle anotherRect = new Rectangle(3, 5);   Konstruktor mit zwei Parametern aufrufen
  ...
}
```

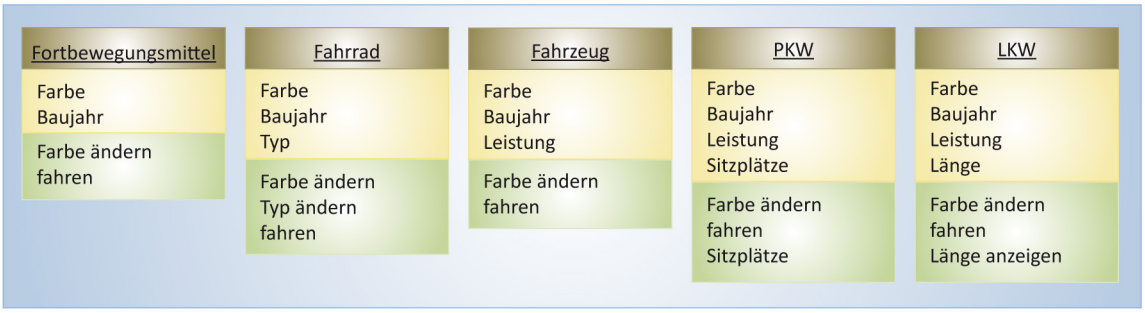
Konstruktoren anwenden

Ein Konstruktor mit einem Parameter ist in der Klasse Rectangle nicht definiert. Daher würde eine Anweisung wie beispielsweise `Rectangle thirdRect = new Rectangle(4);` eine Fehlermeldung hervorrufen.

10.9 Vererbung

Unter **Vererbung** wird die Fähigkeit einer Klasse verstanden, Eigenschaften und Methoden automatisch von einer anderen Klasse zu übernehmen. Die vererbende Klasse wird als Basisklasse, Superklasse oder Oberklasse bezeichnet, die erbende Klasse als abgeleitete Klasse, Subklasse oder Unterklasse.

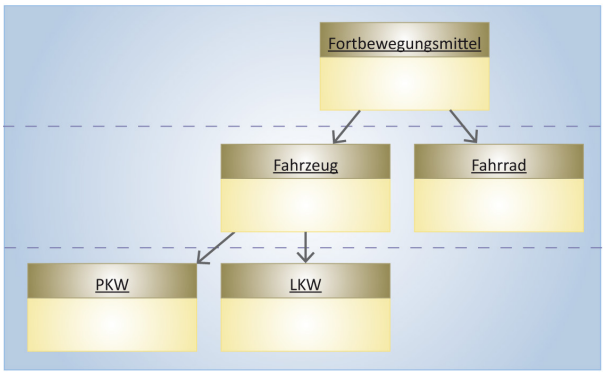
Beispiel



Entwurf verschiedener Klassen

Sie möchten die fünf Klassen, wie in der obigen Abbildung, entwerfen. Diese Klassen haben keine direkten Beziehungen untereinander. In der realen Welt sind aber Fahrräder und Fahrzeuge Fortbewegungsmittel, so wie LKW und PKW Fahrzeuge sind. Deshalb besitzen auch die Klassen `Fahrrad` und `Fahrzeug` die Attribute und Methoden der Klasse `Fortbewegungsmittel`. Ebenso besitzen die Klassen `LKW` und `PKW` die Attribute und Methoden der Klasse `Fahrzeug`. So sind die Methoden und Attribute der Klasse `Fortbewegungsmittel` in allen dargestellten Klassen vorhanden. Die gemeinsamen Methoden und Attribute sollen bei jeder Klasse nicht neu definiert werden. Eine Änderung in der Klasse `Fortbewegungsmittel`, z. B. das Hinzufügen des Attributes `Anschaffungskosten`, hätte die Änderung aller anderen Klassen zur Folge. Die Lösung des Problems ist die **Vererbung**.

Die Klassen `Fahrrad` und `Fahrzeug` erben die Attribute und Methoden von der Klasse `Fortbewegungsmittel`. Da PKW und LKW zu der Kategorie der Fahrzeuge gehören, erben sie von der Klasse `Fahrzeug`. Somit erben auch die Klassen `PKW` und `LKW` indirekt die Attribute und Methoden von der Klasse `Fortbewegungsmittel`.



Der Vorteil liegt darin, dass die Basisklasse `Fortbewegungsmittel` nur einmal definiert wird und alle Unterklassen die Eigenschaften von ihr erben. Sie besitzen damit automatisch die Funktionalität und Struktur der Basisklasse und müssen nur noch erweitert werden. Änderungen in der Basisklasse werden somit ebenfalls in allen davon abgeleiteten Klassen wirksam.

Bei der Festlegung einer Vererbungsstruktur können Sie einerseits Gemeinsamkeiten existierender Klassen in einer Basisklasse abstrahieren (Bottom-up – Generalisierung), andererseits aber auch Klassen einer höheren Abstraktionsebene in Unterklassen spezialisieren (Top-down – Spezialisierung).

Auf die vererbten privaten Teile einer Basis(Ober)klasse können Sie in einer abgeleiteten Klasse nicht zugreifen.

Vorteile der Vererbung

In der objektorientierten Programmentwicklung gilt die Vererbung als Schlüsseltechnik zur Wiederverwendung von Programmbausteinen. Von schon existierenden Klassen werden Unterklassen (abgeleitete Klassen) gebildet, die bereits die Attribute und Methoden der übergeordneten Klasse besitzen, ohne dass sie neu definiert werden müssen. Der Unterklasse können Sie jetzt Erweiterungen hinzufügen oder auch die Funktionalität der Basisklasse überschreiben.

Methoden überschreiben

Mitunter soll in der Unterklasse eine von der Basisklasse geerbte Methode anders arbeiten als in der Basisklasse. In diesem Fall können Sie diese Methode neu implementieren. Die Methoden müssen die gleichen Parameter besitzen. Wird eine Methode für ein Objekt der Klasse aufgerufen, in der die geerbte Methode überschrieben wurde, wird immer diese Version verwendet.

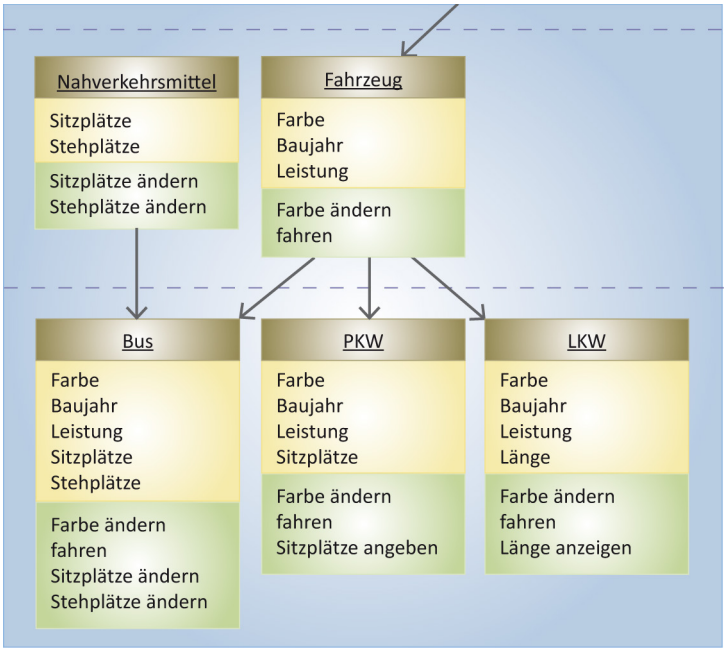
Mehrfachvererbung

Besitzt eine abgeleitete Klasse mehr als eine direkte Basisklasse, wird von Mehrfachvererbung gesprochen (Multiple Inheritance).

- Die Verwendung von Mehrfachvererbung ist nicht in jeder objektorientierten Programmiersprache möglich. Ein wichtiger Vertreter ist C++.
- In fast allen modernen Programmiersprachen wie Java oder C# gibt es keine Mehrfachvererbung: Es kann immer nur eine Klasse als unmittelbare Basisklasse dienen.

Beispiel

In der nebenstehenden Abbildung ist eine Klasse `Bus` dargestellt, die von zwei Klassen erbt, von der Klasse `Fahrzeug` und von der Klasse `Nahverkehrsmittel`. Die Klasse `Bus` erbt somit alle Attribute und Methoden beider Klassen. Die Attribute `Farbe`, `Baujahr` und `Leistung` erbt sie von der Klasse `Fahrzeug` und die Attribute `Sitzplätze` und `Stehplätze` von der Klasse `Nahverkehrsmittel`. Ebenso werden die Methoden `Farbe ändern` und `fahren` von der Klasse `Fahrzeug` und `Sitzplätze ändern` und `Stehplätze ändern` von der Klasse `Nahverkehrsmittel` vererbt. In der Klasse `Bus` können Sie auch noch weitere Attribute und Methoden hinzufügen.



Syntax für das Ableiten von Klassen in Java

```
[modifiers] class identifier extends superClassName
{                                   ①          ②
  statements
}
```

Syntax für das Ableiten einer Klasse

- Über das Schlüsselwort `extends` ① geben Sie an, dass Sie die Klasse `identifier` von der Basisklasse `superClassName` ableiten wollen ②.
- Die abgeleitete Klasse erbt alle Elemente dieser Klasse, die **nicht** mit dem Zugriffsmodifizierer `private` deklariert sind.
- Über Vererbung werden nur Methoden und Attribute vererbt, aber keine Konstruktoren. Es steht somit zunächst nur der Standardkonstruktor zur Verfügung. Sie müssen in den abgeleiteten Klassen eigene Konstruktoren definieren, um die gewünschten Initialisierungen vorzunehmen.
- Der Aufruf eines Konstruktors der direkten Basisklasse erfolgt über das Schlüsselwort `super`.

Durch den Zugriffsmodifizierer `private` sind Attribute bzw. Methoden gekapselt und ein Zugriff darauf ist nur innerhalb der Klasse erlaubt. Um den direkten Zugriff auf diese Attribute bzw. Methoden aus einer abgeleiteten Klasse zu ermöglichen, kann der Zugriffsschutz für die gekapselten Attribute bzw. Methoden gelockert werden, indem statt `private` der Zugriffsmodifizierer `protected` eingesetzt wird.

Beispiel für abgeleitete Klassen und deren Nutzung: *Rectangle.java, Cuboid.java, UsingCuboid.java*

```
class Cuboid extends Rectangle
{
①   private int height;
    ...
②   Cuboid(int width, int length, int height)
    {
③     super(width, length);           //Konstruktor der Basisklasse nutzen
④     setHeight(height);
    }
    ...
⑤   int computeBase()
    {
⑥     return super.computeArea();     //Methode der Basisklasse nutzen
    }
⑦   public int computeVolume()
    {
      return getWidth() * getLength() * getHeight();
    }
⑧   int computeArea()
    {
      return 2 * (getWidth() * getLength()
                + getWidth() * getHeight()
                + getHeight() * getLength());
    }
...
```

Ausschnitt der abgeleiteten Klasse `Cuboid`

① Die Klasse besitzt ein zusätzliches Attribut `height`. Der Zugriff auf die Attribute erfolgt mit den entsprechenden Getter- und Setter-Methoden.

② Der Konstruktor mit Parametern ruft den Konstruktor der Basisklasse auf ③ und initialisiert anschließend das Attribut `height` (Höhe) ④, da dies in der Basisklasse nicht enthalten ist.

⑤ Die Klasse `Cuboid` ermittelt mit der Methode `computeBase()` die Grundfläche des Quaders. Dabei wird über das Schlüsselwort `super` die Methode `computeArea()` der Basisklasse ⑥ verwendet.

⑦ Für die Berechnung des Volumens nutzt die Methode `computeVolume()` die Methoden `getWidth()` und `getLength()` der Basisklasse und die Methode `getHeight()` der abgeleiteten Klasse selbst.

⑥ Abgeleitete Klassen können Methoden der Basisklasse überschreiben (redefinieren– Overriding). Die Klasse `Cuboid` redefiniert die Methode `computeArea()` ⑧, um die Fläche eines Quaders zu berechnen. Über das Schlüsselwort `super` haben Sie weiterhin die Möglichkeit, die entsprechende Methode der Basisklasse aufzurufen. Der Compiler kann so zwischen der Methode der Basisklasse und der Methode der Klasse selbst unterscheiden.

- Die Zugriffsrechte dürfen Sie beim Überschreiben einer Methode erweitern. Beispielsweise können Sie eine Methode, die in der Basisklasse als `protected` gekennzeichnet wurde, beim Überschreiben mit dem Zugriffsmodifizierer `public` als öffentlich kennzeichnen. Eine Einschränkung der Zugriffsrechte ist nicht erlaubt.
- Private und statische Methoden einer Basisklasse können nicht überschrieben werden. Private Methoden sind in der abgeleiteten Klasse nicht sichtbar (es ist kein Aufruf von `super.methode()` möglich). Statische Methoden dürfen in abgeleiteten Klassen nicht überschrieben werden. Sie erhalten in diesem Fall einen Compiler-Fehler.

```
...
                          Konstruktor mit Parametern aufrufen
Cuboid oneCuboid = new Cuboid(3, 5, 7);
                                        Methode der Basisklasse aufrufen
System.out.println("Breite des Quaders: " + oneCuboid.getWidth());
                                      Methode der abgeleiteten Klasse aufrufen
System.out.println("Oberflaeche: " + oneCuboid.computeArea());
...
```

Abgeleitete Klasse aufrufen (Ausschnitt der Klasse `UsingCuboid`)

10.10 Polymorphie

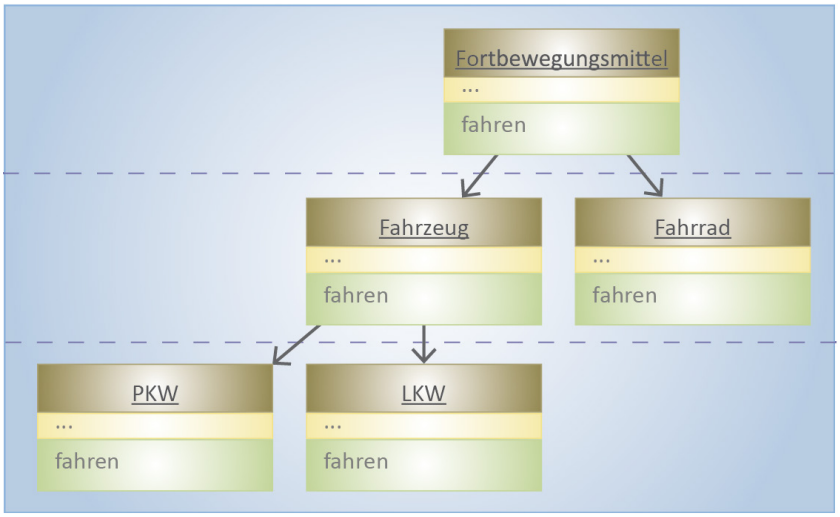
Polymorphie (Vielgestaltigkeit) ist ein Konzept in Programmiersprachen, das die Fähigkeit eines Bezeichners beschreibt, abhängig von seiner Verwendung unterschiedliche Verhaltensweisen anzunehmen. So kann etwa in einigen Sprachen ein Operator polymorph sein. Das Pluszeichen kann beispielsweise für die mathematische Addition, aber auch die Verkettung von Zeichenketten verwendet werden.

In der OOP gehören die Verfahren des Overloadings als auch Overridings zu den polymorphen Prinzipien. Bei überladenen Methoden entscheidet die spezifische Angabe von Parametern, welche Methode ausgewählt wird – ein vielgestaltiges Verhalten. In Hinsicht auf die Vererbung und das Überschreiben beschreibt Polymorphie die Fähigkeit, eine gleichartige Operation mit dem gleichen Namen auf Objekten verschiedener Klassen auszuführen. Beim Aufruf dieser Operation muss nur bekannt sein, dass ein Objekt dieses Verhalten aufweist, aber nicht von welcher Klasse es ist. Dabei kann ein Objekt einer Unterklasse einer Objektvariablen der Oberklasse zugewiesen werden. Auf der Objektvariablen der Oberklasse können Sie dann polymorphe Operationen ausführen. Die Unterschiede der Objekte gehen dabei nicht verloren.

Beispiel

Alle Fortbewegungsmittel können fahren. So fahren Fahrräder ebenso wie PKW und Motorboote, wenn auch auf unterschiedliche Art und Weise.

Von der Oberklasse `Fortbewegungsmittel` werden zwei verschiedene Fortbewegungsmittel abgeleitet (`Fahrzeug` und `Fahrrad`). `Fahrzeug` ist wiederum die Oberklasse für verschiedene motorisierte Fortbewegungsmittel, wie z. B. `PKW` oder `LKW`. Fortbewegungsmittel können sich von einem bestimmten Ort zu einem anderen Ort bewegen. Deshalb besitzen alle Klassen die Methode `fahren`, die sie von der Klasse `Fortbewegungsmittel` erben.



In den Unterklassen werden diese Methoden überschrieben, da jedes Fortbewegungsmittel auf eine andere Art fährt. Wird die Methode `fahren` beispielsweise für ein Objekt der Klasse `PKW` aufgerufen, wird die Methode `fahren()` der Klasse `PKW` ausgeführt. Im Programm können Sie die Methode `fahren()` auch auf ein Objekt der Oberklasse `Fortbewegungsmittel` anwenden, ungeachtet dessen, welches Fortbewegungsmittel nun tatsächlich fährt. Erst zur Laufzeit wird entschieden, welche Methode nun tatsächlich aufgerufen wird. Das wird auch als späte oder dynamische Bindung bezeichnet.

Sie schreiben nur eine Anweisung, z. B.:

```
Fortbewegungsmittel Fortbewegungsmittel_X;      → Objektvariable definieren
...
Fortbewegungsmittel_X.fahren();                 → Polymorphe Methode fahren aufrufen
```

Bei der Übersetzung kann noch nicht bestimmt werden, für welches Objekt welcher Klasse die Methode `fahren()` aufgerufen werden soll. Es wird nur festgelegt, dass es ein Objekt einer Unterklasse von `Fortbewegungsmittel` sein wird. Erst wenn zur Laufzeit der Objektvariablen `Fortbewegungsmittel_X` beispielsweise ein Objekt der Klasse `PKW` zugewiesen wird, steht fest, dass die Methode `fahren` der Klasse `PKW` abgearbeitet wird. Wurde der Objektvariablen `Fortbewegungsmittel_X` ein Objekt der Klasse `Fahrrad` zugewiesen, wird die Methode `fahren()` der Klasse `Fahrrad` aufgerufen usw.

```
Fortbewegungsmittel Fortbewegungsmittel_X;   → Objektvariable definieren
PKW MeinWagen = new PKW();                   → Objekt der Klasse PKW definieren
Fahrrad MeinFahrrad = new Fahrrad();         → Objekt der Klasse Fahrrad definieren
...
Fortbewegungsmittel_X = MeinWagen;           → Der Objektvariablen wird ein Objekt der
                                               Klasse PKW zugewiesen
Fortbewegungsmittel_X.fahren();              → Methode fahren der Klasse PKW wird
...                                            aufgerufen
 Fortbewegungsmittel_X = MeinFahrrad;        → Der Objektvariablen wird ein Objekt der
                                               Klasse Fahrrad zugewiesen
Fortbewegungsmittel_X.fahren();              → Methode fahren der Klasse Fahrrad wird
...                                            aufgerufen
```

So lassen sich z. B. umfangreiche `case`-Anweisungen sparen, in denen getestet wird, ob das Objekt nun ein Fahrrad oder ein PKW oder ein LKW usw. ist, um die richtige Methode aufzurufen.

10.11 Schnellübersicht

Was bedeutet …?	
Klasse	Eine Klasse beschreibt die Attribute und Methoden von gleichartigen Objekten und definiert dadurch einen neuen Typ.
Objekt, Instanz	Objekte/Instanzen sind reale Ausprägungen einer Klassendefinition und somit eigenständige Datenobjekte. Sie besitzen die in der Klasse definierten Attribute. Auf Objekte/Instanzen können die in der Klasse definierten Methoden angewendet werden.
Attribut	Attribute sind die Eigenschaften eines Objekts. Sie enthalten die Daten des Objekts. In der Regel sind sie für andere Objekte nur durch die Methoden der Klasse sichtbar.
Methode	Die Methoden sind die in einer Klasse definierten Operationen. Sie bestimmen das Verhalten des Objekts. Über sie kann auf die Attribute des Objekts zugegriffen werden.
Datenkapselung	Dies ist das Verbergen von speziellen Eigenschaften einer Klasse. Nur über eine definierte indirekte Schnittstelle (ausgewählte Methoden) kann auf die gekapselten Daten zugegriffen werden.
Konstruktor	Der Konstruktor ist eine spezielle Methode einer Klasse, die beim Erzeugen eines Objekts ausgeführt wird und Initialisierungen durchführt.
Vererbung	Eine neu definierte Klasse erhält automatisch (erbt) die freigegebenen Attribute und Methoden einer Basisklasse (Oberklasse). Sie kann um zusätzliche Attribute und Methoden erweitert werden.
Polymorphie	Eine gleichnamige Operation kann unterschiedliche Ergebnisse liefern bzw. Aktionen ausführen.

10.12 Übungen

Übung 1: Klassenstruktur festlegen

Übungsdatei: -- **Ergebnisdatei: *uebung10.pdf***

1. Überlegen Sie sich eine Struktur von Klassen, die zum Zeichnen grafischer Objekte (z. B. Rechteck, Ellipse, Dreieck, n-Eck) dienen sollen. Folgende Fragen sollen beantwortet werden:
   - Welche gemeinsamen Eigenschaften haben die Klassen? Welche Eigenschaften sind spezifisch für die jeweilige Klasse? Fassen Sie gemeinsame Eigenschaften in einer Oberklasse zusammen.
   - Welche Operationen (Methoden) besitzen die Klassen?
   - Welche Operationen können bereits in der Oberklasse definiert werden?
   - Welche Operationen können als polymorph definiert werden?
2. Erstellen Sie eine Klassenstruktur mit Basis- und abgeleiteten Klassen, welche die Personalverwaltung innerhalb einer großen Firma widerspiegelt. Verwenden Sie die Fragestellungen aus Aufgabe ①.

Übung 2: Getter- und Setter-Methoden und Konstruktor festlegen

Übungsdateien: *SingleLinkedList.java*, *UseSingleLinkedList.java* **Ergebnisdatei: *SingleLinkedList.java***

1. Erweitern Sie die Klasse `Node` im Programm *SingleLinkedList.java* um Getter- und Setter-Methoden für die Attribute `next` und `data`.
2. Stellen Sie für die Klasse `Node` einen Konstruktor mit geeigneten Parametern zur Initialisierung der Attribute zur Verfügung.
3. Ändern Sie die Argumente der Methodenaufrufe im Programm *UseSingleLinkedList.java* entsprechend den bearbeiteten Signaturen.
4. Passen Sie die durch die Erweiterungen betroffenen Anweisungen in den Methoden `insertNode()`, `deleteNode()` und `printList()` an. Verwenden Sie zum Testen das Programm *UseSingleLinkedList.java*.

Übung 3: Vererbung und Polymorphie

Übungsdatei: -- **Ergebnisdateien: *Rectangle.java, Cuboid.java, Pyramid.java, Polymorphism.java***

1. Erstellen Sie in Java eine Klasse `Pyramid` mit der Basisklasse `Rectangle` aus den Beispieldateien mit folgenden Daten:
   - Attribut `height`
   - Getter- und Setter-Methode für das Attribut `height`
   - Konstruktor mit den Parametern `width`, `length`, `height`
   - Methoden zur Berechnung der Grundfläche, des Volumens und der Oberfläche. Die Berechnung der Oberfläche soll hierbei der Einfachheit halber entfallen und den Testwert `-1` zurückliefern.
2. Im Hauptprogramm (Klasse `Polymorphism`) sollen drei Objekte vom Typ `Rectangle` erstellt werden. Testen Sie die Methode `computeArea()` und geben Sie das Ergebnis für alle drei Objekte aus.

```
Das Rechteck hat die Oberflaeche 15.
Der Quader hat die Oberflaeche 94.
Die Pyramide hat die Oberflaeche -1.
```

Ausgabe des Programms „Polymorphism.java“

11 Algorithmen

In diesem Kapitel erfahren Sie

- welche verschiedenen Arten von Algorithmen es gibt
- den Unterschied zwischen einem iterativen und rekursiven Algorithmus

Voraussetzungen

- Methoden, Prozeduren und Funktionen

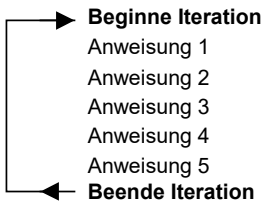
11.1 Eigenschaften eines Algorithmus

Damit ein Algorithmus zur Lösung einer Aufgabe in Programmen eingesetzt werden kann, muss er bestimmte Eigenschaften besitzen.

endlich	Der Algorithmus muss vollständig mit einer endlichen Anzahl von Anweisungen beschrieben werden können.
allgemein	Der Algorithmus muss für die gleichen Eingabedaten immer das gleiche Ergebnis liefern.
eindeutig	Jeder Schritt des Algorithmus muss eindeutig ausgeführt werden können.
terminiert	Der Algorithmus muss nach einer endlichen Anzahl von Schritten zum Ende kommen.

Algorithmen können nach verschiedenen Kriterien unterschieden werden, z. B. nach der Art der Abarbeitung oder dem Umgang mit Datentypen.

11.2 Iterativer Algorithmus



Ein **iterativer Algorithmus** löst eine Aufgabe mit Schleifendurchläufen.

Beispiel für einen iterativen Algorithmus: *Factorial.java*

Die Fakultät einer Zahl `x`, geschrieben `x!`, ist das Produkt aller ganzen positiven Zahlen von 1 bis `x`. Die Fakultät von 3 berechnet sich beispielsweise wie folgt: `3! = 1 * 2 * 3 = 6`.

```
...
①  int compFact(int x)
   {
②    int result = 1;
③    while (x > 1)
     {
④      result = result * x;
⑤      x = x - 1;
     }
⑥    return result;
   }
...
```

① Die Methode besitzt den Namen `compFact`. Der formale Parameter `x` ist vom Typ `int`.

② Die Variable `result` (Ergebnis) wird deklariert und gleich mit dem Wert `1` initialisiert.

③ Die Schleife wird so lange durchlaufen, bis die Variable `x` kleiner oder gleich `1` ist. Die letzte Multiplikation (mit dem Wert `1`) kann entfallen, da eine Multiplikation mit 1 das Ergebnis nicht verändert.

④ Die Berechnung der Fakultät wird durchgeführt. Das bisherige Resultat in der Variablen `result` wird mit der Variablen `x` multipliziert.

⑤ Die Variable `x` wird um den Wert `1` verringert (dekrementiert).

⑥ Das Endergebnis `result` wird mit der `return`-Anweisung zurückgegeben.

Die Iteration Schritt für Schritt

Zur Verdeutlichung des iterativen Algorithmus werden die Variablenwerte bei den einzelnen Schleifendurchläufen für den aktuellen Parameter `4` betrachtet.

Schleifendurchlauf	Programmschritte	Erklärung
	`compFact(4);`	Methodenaufruf
	`result = 1;`	Initialisierung der Variablen

1. Iteration	`4 > 1 ?`
	`result = 1 * 4 = 4`
	`x = 4 - 1 = 3`
		Wahr → 1. Schleifendurchlauf

2. Iteration	`3 > 1 ?`
	`result = 4 * 3 = 12`
	`x = 3 - 1 = 2`
		Wahr → 2. Schleifendurchlauf

3. Iteration	`2 > 1 ?`
	`result = 12 * 2 = 24`
	`x = 2 - 1 = 1`
		Wahr → 3. Schleifendurchlauf

4. Iteration	`1 > 1 ?` `result = 24`	Falsch → Schleifendurchlauf wird beendet
	`return result`	Rückgabe des Ergebnisses der Methode

Beispiel für den iterativen Algorithmus mit JavaScript: *Factorial.html*

Das letzte Beispiel sieht in JavaScript wie folgt aus:

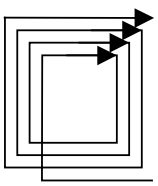
```
...
①  function compFact(x)
   {
②    var result = 1;
③    while (x > 1)
     {
       result = result * x;
       x = x - 1;
     }
     return result;
   }
   alert(compFact(3));
...
```

① Die Funktion wird mit dem Namen `compFact` deklariert. Es gibt bei JavaScript beim formalen Parameter `x` keinen Datentyp.

② Die Variable `result` (Ergebnis) wird deklariert und mit dem Wert `1` initialisiert. Es wird kein Datentyp angegeben. Das Schlüsselwort `var` kennzeichnet die Variable als lokal.

③ Die Schleife ist unverändert. Die Ausgabe erfolgt in JavaScript mit `alert()`.

11.3 Rekursiver Algorithmus



BeginneRekursion
Anweisung 1
Anweisung 2
Anweisung 3
Anweisung 4
Gehezu **BeginneRekursion**
BeendeRekursion

Die Idee des rekursiven Algorithmus ist es, einen Algorithmus durch sich selbst zu beschreiben. Die Rekursion muss jedoch an einer definierten Stelle beendet werden, sonst ist der Algorithmus nicht endlich.

Bei rekursiven Algorithmen werden Funktionen oder Methoden bzw. deren Zustand vor der konkreten Abarbeitung auf einem Stapel (Stack) abgelegt. Die Ausführung erfolgt, indem der Stapel von oben nach unten abgearbeitet wird. Der letzte rekursive Schritt wird also als Erstes ausgeführt.

Fakultätsberechnung mit rekursivem Algorithmus

Die Fakultät einer ganzen Zahl x berechnet sich

- für alle x > 0: `x! = x * (x-1)!`
- für x = 0: 0! = 1

Das Beispiel zur Berechnung von `4!` ergibt nach Auflösung aller Fakultäten:

```
4! = 4 * (4-1)! = 4 * 3 * (3-1)! = 4 * 3 * 2 * (2-1)! = 4 * 3 * 2 * 1 *
(1-1)! = 24.
```

Beispiel für einen rekursiven Algorithmus: *FactorialRec.java*

Mit dem rekursiven Algorithmus x! = x * (x-1)! und dem Abbruchkriterium der Rekursion 0! = 1 kann die Fakultät in Java wie folgt berechnet werden:

```
   ...
   int compFact(int x)
     {
①     if (x > 0)
②       return x * compFact(x-1);
      else
③       return 1;
     }
   ...
```

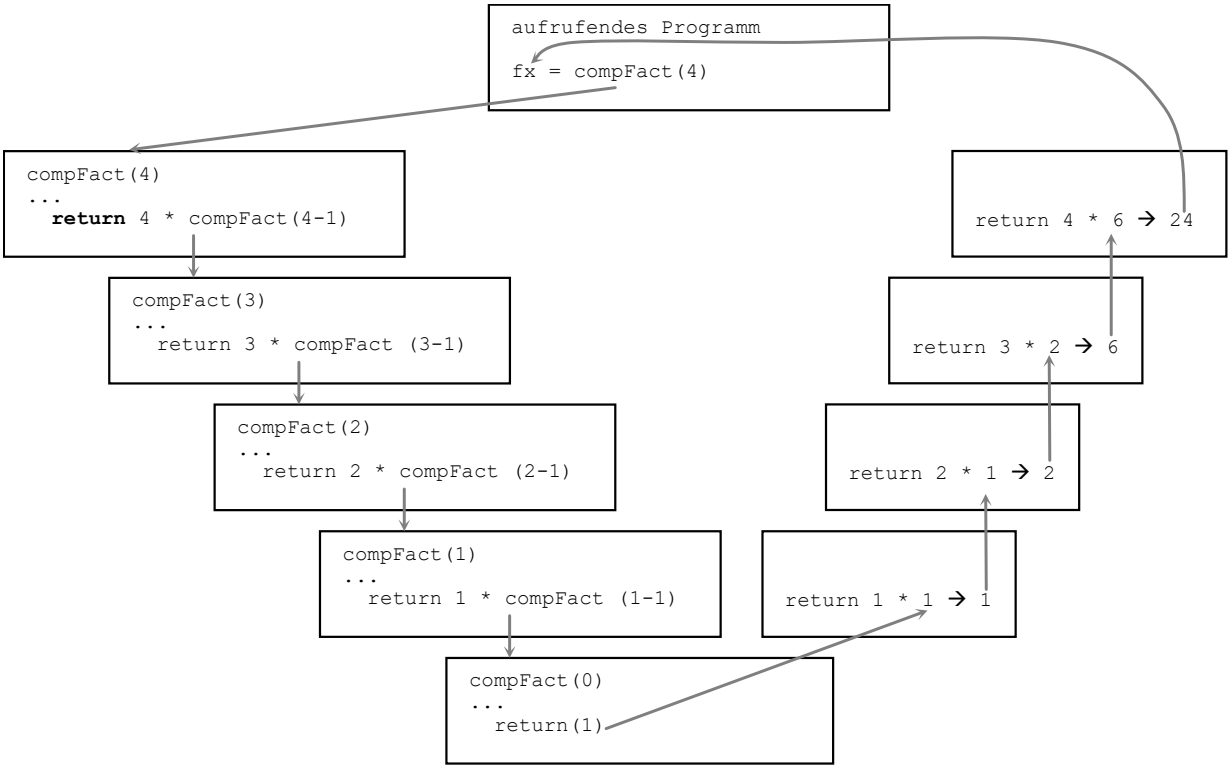
① Die Abbruchbedingung wird dadurch definiert, dass die Rekursion nur so lange durchgeführt wird, wie der Parameter `x` einen Wert größer als null hat.

② Die Berechnung der Fakultät wird durchgeführt. Dabei ruft sich die Methode selbst mit dem neuen Parameter `(x-1)` auf. Erst wenn der Methodenaufruf zurückkehrt und das Methodenergebnis liefert, kann die weitere Berechnung erfolgen.

③ Hat `x` den Wert `0`, wird der Rückgabewert `1` zurückgeliefert, da gilt: `0! = 1`. Es findet kein neuer Methodenaufruf statt. Die Rekursion wird beendet.

Die Rekursion Schritt für Schritt

Zur Verdeutlichung des rekursiven Algorithmus wird die Methode anhand der übergebenen Daten betrachtet.



Der Aufruf der Methode `compFact(4)` erfolgt mit dem Wert `4` als Argument. Nach der Regel `x*(x-1)!` ist die folgende Aktion `x * compFact (3)`. Die Methode `compFact` wird diesmal mit dem Wert `3` als Argument aufgerufen (1. Rekursion). Dies wird so lange wiederholt, bis in der 4. Rekursion festgestellt wird, dass das Argument 0 ist. Die Rekursion endet hier.

Die einzelnen Resultate werden an die übergeordneten Rekursionen zurückgegeben. Nach dem Auflösen aller Rekursionen wird das Endergebnis der Methode zurückgeliefert:

```
4! = 24.
```

Beispiel für den gleichen rekursiven Algorithmus mit JavaScript: *FactorialRec.html*

Auch in JavaScript wird Rekursion unterstützt. Die Berechnung der Fakultät in JavaScript ist wieder fast identisch wie in Java. Die wenigen entscheidenden Unterschiede sollten Sie mittlerweile kennen:

```
...
function compFact(x)
  {
   if (x > 0)
     return x * compFact(x-1);
   else
     return 1;
  }
...
```

① Die Funktion wird wieder mit dem einleitenden Schlüsselwort `function` und ohne Angabe von einem Datentyp bei den Parametern und dem Rückgabewert notiert.

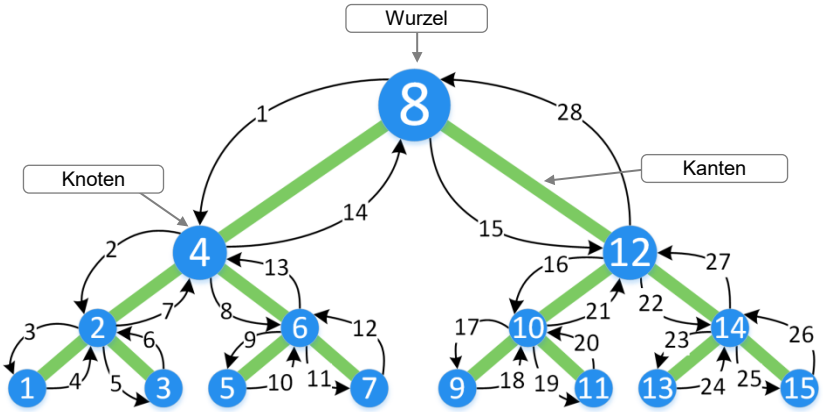
② Der Rest der Syntax ist identisch wie beim Java-Beispiel.

Backtracking-Technik mit rekursivem Algorithmus

Eine weitere Möglichkeit einer Problemlösung mithilfe einer Rekursion ist das Backtracking-Verfahren (Rückverfolgung). Dies ist ein Verfahren, um alle Möglichkeiten einer Situation durchzuspielen.

Zum Abfragen jedes Knotens eines binären Baumes, der im hier verwendeten Beispiel sortierte Zahlen enthält, gehen Sie nach folgenden Anweisungen vor:

- Startpunkt und Ziel ist die Wurzel des Baumes (8).
- Steigen Sie im Baum so lange ab, bis die linke Spitze des Baumes erreicht ist (1).
- Laufen Sie zurück zum letzten Knoten (2), und versuchen Sie, in der zweiten Asthälfte des Knotens abzusteigen (3).
- Ist auch hier die Spitze erreicht, laufen Sie abermals zurück, bis ein Absteigen in noch nicht durchlaufene Teile des Baumes wieder möglich ist (4).
- Alle Knoten sind mindestens einmal passiert worden, wenn Sie am Start (Wurzel) wieder angekommen sind.



Bäume sind häufig verwendete Strukturen in Algorithmen. Sie bestehen aus **Knoten** und **Kanten**, ausgehend von einer sogenannten **Wurzel**. Gehen von jedem Knoten höchstens zwei Kanten aus, heißt ein Baum **Binärbaum**. Kanten und Knoten eines Baums unterhalb eines bestimmten Knotens, z. B. Knoten (4) in der Abbildung, heißen Teilbaum. Enthalten die Knoten aller Teilbäume rechts von der Wurzel einen kleineren Wert als in der Wurzel und die Knoten links von der Wurzel einen größeren Wert als in der Wurzel, heißen die Binärbäume **Suchbäume**. Suchbäume eignen sich besonders gut für die Suche in großen Datenmengen.

Suchbäume zu programmieren ist dann effektiv, wenn große Datenmengen in einer relativ kurzen Zeit zu durchsuchen oder zu sortieren sind. Dabei sollte die Ausgangsdatenmenge möglichst unsortiert sein. Sie können in einem binären Baum z. B. auch Lösungsmöglichkeiten für ein mathematisches Problem speichern und durch entsprechendes Durchsuchen die richtige Lösung oder eine der günstigsten Lösungen finden.

Beim Schachspiel wäre es z. B. denkbar, die Bewegungsmöglichkeiten der Spielfiguren zu errechnen und in einem Baum als Inhalt der Knoten zu speichern, um anschließend einen geeigneten Spielzug zu suchen. Aufgrund der rekursiven Funktionen sind die Quelltexte erheblich kürzer als mit allen anderen Steuerstrukturen, wenn diese Ähnliches leisten sollen.

11.4 Iterativ oder rekursiv?

Die iterativen und rekursiven Algorithmen spielen in der Programmierung eine wesentliche Rolle, da viele Probleme je nach Anforderung entweder durch Iteration oder Rekursion gelöst werden können.

Vor- und Nachteile iterativer und rekursiver Algorithmen

	Iterativ	Rekursiv
Vorteil	✔ Keine hohen Systemanforderungen, da lokale Variablen benutzt werden	✔ Einfacher und kurzer Quellcode durch Reduzierung des Problems auf seinen einfachsten Fall ✔ Die Lösung ist schneller entwickelt. ✔ Passt sich an die Komplexität der Lösungsmöglichkeiten an (Sie müssen z. B. nicht die Anzahl der Schritte vorher berechnen und sich keine Zwischenschritte merken.)
Nachteil	✔ Bei größeren Aufgaben kann ein komplexer Algorithmus entstehen. ✔ Zu langer und unübersichtlicher Quellcode	✔ Je nach Rekursionstiefe besteht ein hoher Ressourcenbedarf, z. B. an Hauptspeicher. ✔ Durch das ständige Merken der rekursiven „Sprünge“ in ein und dieselbe Funktion auf dem Stack kann es bei komplexen Aufgaben zu einem Speichermangel im System kommen (Stapelüberlauf). Die Folge ist Programmabsturz.

Welches Verfahren Sie verwenden, hängt von den allgemeinen Rahmenbedingungen ab. Rekursive Lösungen können schneller entwickelt werden, da der Algorithmus nicht komplex wird und somit übersichtlicher ist. Iterative Lösungen enthalten in den meisten Fällen einen komplexeren Algorithmus, können jedoch leichter optimiert werden.

11.5 Generischer Algorithmus

„Generisch“ bedeutet „die Gattung betreffend“. In der Programmierung bezieht sich diese Aussage auf die entsprechenden Datentypen. Üblicherweise sind Algorithmen in Programmen für spezielle Datentypen ausgelegt. Dies bedeutet, der Algorithmus kann entweder nur Daten vom Typ `Integer` oder vom Typ `String` bearbeiten. Falls sich der Algorithmus selbst für verschiedene Datentypen nicht unterscheidet, liegt es nahe, diese Anweisungen für alle Datentypen nur einmal zu implementieren. Die datentypabhängigen Funktionsteile werden herausgelöst und für die verschiedenen Datentypen jeweils separat bereitgestellt. Dadurch kann ein generischer Algorithmus mit verschiedenen Datentypen arbeiten.

Generische Algorithmen werden beispielsweise in den Standardbibliotheken verschiedener Programmiersprachen zur Verfügung gestellt. Sie dienen z. B. dem Sortieren von Feldern und Verwalten von Listen. Ein typischer Vertreter dieser Gattung ist ein „Sortier-Algorithmus“, der oft als Funktion `sort()` implementiert ist. Er sortiert auf- oder absteigend eine Menge von Ganzzahlen (`Integer`), Fließkommazahlen (`Real`) sowie Zeichenketten (`String`) und liefert das Ergebnis an das Programm zurück. Damit der Algorithmus Daten eines beliebigen Typs sortieren kann, müssen Sie eine separate Vergleichsfunktion (`compare`) für die einzelnen Datentypen entwickeln. Dieser werden zwei Datenelemente des betreffenden Typs als Parameter übergeben. Für jeden Datentyp wird die Vergleichsfunktion definiert, z. B. können Namen nach dem Alphabet sortiert werden und Adressen nach der Postleitzahl.

11.6 Übung

Iterative und rekursive Algorithmen

Übungsdatei: -- **Ergebnisdateien: *uebung11.pdf, AddRec.java, AddIter.java, KingChess.java, GreatestCommonDivisor.java***

1. Die Multiplikation der ganzen Zahlen x und y kann auch als x-maliges Addieren der Zahl y gelöst werden. Zum Beispiel ist 3 * 4 = 4 + 4 + 4 = 12.

   Erstellen Sie eine rekursive Funktion für diese Operation.
2. Lösen Sie die Übung 1 mithilfe eines iterativen Algorithmus.
3. Erstellen Sie für die Aufgaben 1 und 2 jeweils ein Java-Programm. Speichern Sie die Programme unter AddRec.java und AddIter.java.
4. Berechnen Sie mit einem rekursiven Algorithmus die Anzahl der Weizenkörner, ausgehend von einem Schachspiel. Das erste Feld soll mit einem Korn belegt sein, das zweite Feld mit zwei Körnern und so fort. Die Zahl der Körner soll auf jedem Feld verdoppelt werden.
5. Lösen Sie die Aufgabe 4 mit einem Java-Programm. Speichern Sie das Programm unter KingChess.java.
6. Erstellen Sie einen iterativen Algorithmus, der den größten gemeinsamen Teiler von zwei vorgegebenen ganzen Zahlen ermittelt. Verwenden Sie dazu den Algorithmus von Euklid, der die folgenden Schritte umfasst.
   - Ausgangspunkt sind zwei ganze Zahlen, z. B. 75 und 54.
   - Bilden Sie durch die ganzzahlige Division den Quotienten aus der größeren Zahl (Dividend) und der kleineren Zahl (Divisor), z. B. 75/54.
   - Ist der Rest der ganzzahligen Division verschieden von 0, wird ein neues Zahlenpaar aus der kleineren Zahl und dem Rest der ganzzahligen Division gebildet, z. B. 54 und 21.

Zahlenpaar		Quotient	Rest	
75	54	1	21	>0
54	21	2	12	>0
21	12	1	9	>0
12	9	1	3	>0
9	3	3	0	=0

   größter gemeinsamer Teiler

   - Bilden Sie davon analog zum ersten Zahlenpaar den Quotienten, z. B. 54/21.
   - Wiederholen Sie den Vorgang so lange, bis der Rest der ganzzahligen Division 0 ist.
   - Der größte gemeinsame Teiler ist der Divisor der letzten Division.
7. Lösen Sie die Aufgabe 6 mit einem Java-Programm. Speichern Sie das Programm unter GreatestCommonDivisor.java.
8. Die Verzeichnisse und Dateien in einem Betriebssystem sind in der Regel innerhalb einer Baumstruktur angeordnet. Um eine bestimmte Datei zu suchen, werden die Ordner und Unterordner nacheinander durchsucht, bis die Datei gefunden wird. Wie kann ein Algorithmus zum Suchen einer Datei in einer solchen Baumstruktur aussehen? Geben Sie eine iterative und eine rekursive Lösung an.

12 Spezielle Algorithmen

In diesem Kapitel erfahren Sie

- wie Sie Arrays durchsuchen können
- wie Sie Arrays auf verschiedene Art sortieren können
- wie Sie mit sequenziellen Dateien arbeiten können

Voraussetzungen

- Kontrollstrukturen
- Arrays
- Unterprogramme

12.1 Suchalgorithmen

Problembeschreibung

In der Programmierpraxis wird häufig in einem Array ein Element mit bestimmten Eigenschaften gesucht. Dazu gibt es verschiedene Suchalgorithmen, die sich bezüglich Aufbau und Effizienz unterscheiden. Bei Suchalgorithmen wird die Effizienz an der Anzahl der Vergleiche gemessen, die zur Lösung notwendig sind.

Arten von Suchalgorithmen

Im Folgenden werden

- die (einfache) **lineare Suche** und
- die **binäre Suche** vorgestellt.

12.2 Lineare Suche

Wie arbeitet die lineare Suche?

Bei der linearen Suche wird eine Menge von Elementen, z. B. ein Feld, nach einem bestimmten Kriterium durchsucht, z. B. nach dem Minimum der Werte. Die lineare Suche ist sehr langsam, da jedes Element des Arrays verglichen werden muss. Bei der linearen Suche steigt die Anzahl der Vergleiche linear mit der Anzahl der Elemente.

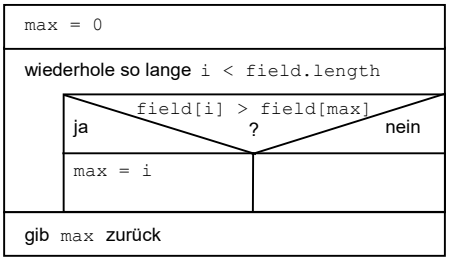
Die Implementierungsbeispiele verdeutlichen das Prinzip der Algorithmen. Sie können diese sowohl auf Arrays mit Zahlen oder Zeichenketten als auch auf verschachtelte Arrays anwenden. Sie müssen in diesem Fall das Vergleichskriterium entsprechend anpassen.

Beispiel: Maximum suchen: *LinearSearch1.java*

Die lineare Suche zur Ermittlung der Position des größten Wertes (Maximums) innerhalb des Arrays wird in der folgenden Methode implementiert.

```
①  int findMax(int[] field)
   {
②    int max = 0;
③    for (int i = 1; i < field.length; i++)
     {
④      if (field[i] > field[max])
⑤      max = i;
     }
⑥    return max;
   }
```

Funktion: `findMax`
Parameter:
`field` → Feld mit n Elementen
Rückgabewert:
`max` → Position des Maximums



① Der Methode `findMax()` wird als Parameter das zu durchsuchende Feld übergeben.

② Zu Beginn wird das erste Element (mit dem Index 0) als das aktuelle Maximum angenommen. Daher wird die Variable `max`, die den Index des Maximums liefert, mit 0 initialisiert.

③ Die Schleife durchläuft alle Elemente des Arrays, bis auf das erste, da dieses bereits mit der Anweisung ② als Startwert für das Maximum angenommen wurde. Die Arraylänge steht in Java über `arrayIdentifier.length` zur Verfügung, wobei `arrayIdentifier` der Bezeichner für das Array ist.

④ Ist der Wert des Arrays an der Position `i` größer als der an der Position `max`, wird `i` als neue Position für das Maximum in der Variablen `max` ⑤ gespeichert.

⑥ Die Position, an der sich das Maximum befindet, wird von der Funktion zurückgegeben. Tritt zweimal der gleiche Maximumwert im Array auf, wird die Position des ersten Auftretens übergeben.

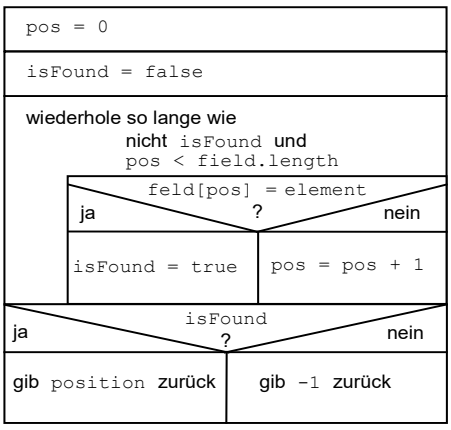
Wenn Sie in Zeile ④ den logischen Operator [>] in den Operator [<] ändern, sucht die Methode die Position des Minimums des Feldes.

Beispiel: Bestimmtes Element suchen: *LinearSearch2.java*

Im nächsten Beispiel wird ein bestimmtes Element eines Arrays gesucht. Die Methode gibt die Position des gesuchten Elements im Array zurück oder -1, wenn das Element nicht gefunden wurde.

```
①  int findElement(int[] field, int element)
   {
②    int pos = 0;
③    boolean isFound = false;
④    while (!isFound && (pos < field.length))
     {
⑤      if (field[pos] == element)
⑥        isFound = true;
       else
⑦        pos++;
     }
⑧    if (isFound)
⑨      return pos;
     else
⑩      return -1;
   }
```

Funktion: `findElement`
Parameter:
`field` → Feld mit n Elementen
`element` → Element, dessen Position ermittelt werden soll
Rückgabewert:
`pos` → Position des Elements



① Der Methode `findElement()` werden als Parameter das Array, das durchsucht werden soll, und das Element, das gesucht werden soll, übergeben.
② Die Variable `pos` dient zur Speicherung des aktuellen Index und wird mit `0` initialisiert.
③ Die Variable `isFound` gibt an, ob das Element gefunden wurde. Sie wird mit dem Wert `false` vorbelegt, da das Element noch nicht gefunden wurde.
④ Die Schleife wird so lange durchlaufen, bis das Element gefunden wurde oder bis das Ende des Arrays erreicht ist.
⑤ Das aktuelle Element des Arrays wird mit dem gesuchten Element verglichen.
⑥ Wurde das gesuchte Element gefunden, wird die Variable `isFound` mit dem Wert `true` belegt. Dadurch wird die Schleife beendet.
⑦ Wurde das gesuchte Element nicht gefunden, muss das nächste Element überprüft werden. Die aktuelle Position wird um 1 erhöht.
⑧ Es wird geprüft, ob das Element gefunden wurde.
⑨ Die Methode liefert die Position des gesuchten Elementes, falls das Element gefunden wurde.
⑩ Wird das gesuchte Element nicht im Feld gefunden, gibt die Methode den Wert -1 zurück.

12.3 Binäre Suche

Wie arbeitet die binäre Suche?

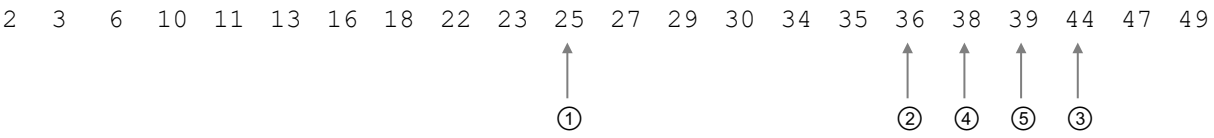
Liegt ein Array **sortiert** vor, können Sie die binäre Suche verwenden. Dieser Algorithmus liefert wesentlich schneller ein Ergebnis als die lineare Suche. Die Anzahl der Vergleiche steigt bei der binären Suche logarithmisch an.

Die binäre Suche funktioniert nach dem folgenden Prinzip:

- Es wird zuerst das mittlere Element in der sortierten Datenmenge untersucht. Ist es größer als das gesuchte Element, muss nur noch in der unteren Hälfte gesucht werden, anderenfalls in der oberen.
- Nun wird die Suche auf die neue halbierte Datenmenge angewendet. Das mittlere Element wird mit dem gesuchtem Element verglichen. Ist das mittlere Element größer, wird in der unteren Hälfte weitergesucht, sonst in der oberen usw.
- Somit halbiert sich bei jedem Schritt die Anzahl der Elemente, die noch untersucht werden müssen.
- Der Algorithmus ist zu Ende, wenn das Element gefunden wurde oder wenn zum Schluss nur noch ein Element übrig ist. Entweder ist dieses letzte Element das gesuchte, oder das gesuchte Element kommt nicht vor.

Beispiel: Bestimmtes Element suchen: *BinarySearch.java*

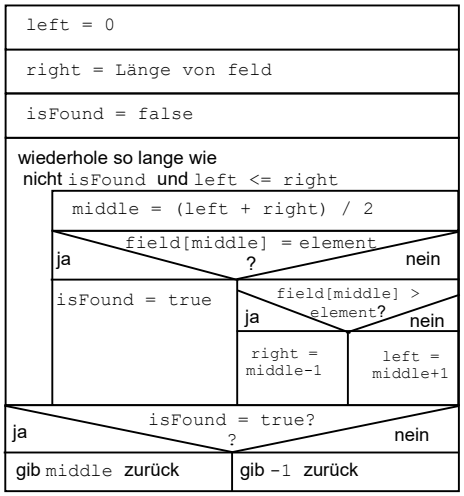
Es soll der Wert 39 in einem bereits sortierten Array gesucht werden.



Schritt ①	Da 25 < 39, wird das mittlere Element in der rechten Seite ermittelt.
Schritt ②	Da 36 < 39, wird das mittlere Element in der rechten Seite ermittelt.
Schritt ③	Da 44 > 39, wird das mittlere Element in der linken Seite ermittelt.
Schritt ④	Da 38 < 39, wird das mittlere Element in der rechten Seite ermittelt.
Schritt ⑤	Da 39 = 39, wurde das Element gefunden.

```
①  int findElement(int[] field, int element)
   {
②    int left = 0;
③    int right = field.length - 1;
     int middle = 0;
④    boolean isFound = false;
⑤    while (!isFound && (left <= right))
     {
⑥      middle = (left + right) / 2;
⑦      if (field[middle] == element)
         isFound = true;
⑧      else
       {
⑨        if (field[middle] > element)
           right = middle -1;
⑩        else
           left = middle + 1;
       }
     }
⑪    if (isFound)
⑫      return middle;
     else
⑬      return -1;
   }
```

Unterprogramm: `findElement`
Parameter:
`field` → sortiertes Feld mit n Elementen
`element` → Element, dessen Position ermittelt werden soll
Rückgabewert:
`position` → Position des Elements



① Der Methode `findElement()` werden als Parameter das Array, das durchsucht werden soll, und das Element, das gesucht werden soll, übergeben.

② Die linke Grenze für den Suchbereich wird auf `0` festgelegt.

③ Die rechte Grenze des Suchbereichs ist zu Beginn der höchste Indexwert.

④ Die Variable `isFound` gibt an, ob das Element gefunden wurde. Sie wird mit dem Wert `false` vorbelegt, da das Element noch nicht gefunden wurde.

⑤ Die Schleife wird so lange durchlaufen, bis das Element gefunden wurde oder bis die linke Grenze des zu durchsuchenden Bereichs größer wird als die rechte Grenze.

⑥ Die Mitte des zu durchsuchenden Bereichs wird errechnet.

⑦ Das aktuelle (mittlere) Element wird mit dem gesuchten Element verglichen. Wurde das gesuchte Element gefunden, wird die Variable `isFound` mit dem Wert `true` belegt. Dadurch wird die Schleife beendet.

⑧ Wurde das gesuchte Element nicht gefunden, muss der Bereich, in dem gesucht wird, festgelegt werden.

⑨ Ist das gesuchte Element kleiner als das mittlere, wird im linken Bereich weitergesucht. Die neue rechte Grenze wird um 1 vor die aktuelle Mitte gesetzt, da das gesuchte Element mit dem Element der aktuellen Mitte bereits verglichen wurde ⑦.

⑩ Ist das gesuchte Element größer als das mittlere, wird im rechten Bereich weitergesucht. Die neue linke Grenze ergibt sich aus der aktuellen Mitte um 1 erhöht.

⑪ Es wird geprüft, ob das Element gefunden wurde.

⑫ Die Methode liefert die Position des gesuchten Elementes, falls das Element gefunden wurde.

⑬ Wird das gesuchte Element nicht im Feld gefunden, gibt die Methode den Wert -1 zurück.

Für die Elementsuche in sortierten Feldern ist die binäre Suche vorzuziehen, da sie schneller als die lineare Suche ist. Je mehr Elemente das Array hat, umso größer werden die Unterschiede in der Ausführungsgeschwindigkeit sein.

12.4 Sortieralgorithmen

Problembeschreibung

Das Ziel von Sortieralgorithmen ist es, die Elemente einer Menge (z. B. eines Arrays) nach einem bestimmten Kriterium zu sortieren. Nach dem Sortieren liegen die Elemente in aufsteigender oder absteigender Reihenfolge vor. Voraussetzung für eine Sortierung ist, dass die Elemente nach irgendeinem Kriterium vergleichbar sind. Das kann z. B. ein ganz einfacher Zahlenvergleich sein oder ein komplizierter Vergleich von Strukturelementen. Die Art des Vergleichs ist für den Sortieralgorithmus selbst nicht wichtig. Die verschiedenen Sortieralgorithmen besitzen unterschiedliche Ausführungsgeschwindigkeiten. Die Ausführungsgeschwindigkeit hängt davon ab, wie viele Vergleiche und wie viele Vertauschungen stattfinden müssen.

Arten von Sortieralgorithmen

Im Folgenden werden verschiedene Sortieralgorithmen vorgestellt:

- Bubble-Sort
- Insertion-Sort
- Shell-Sort
- Quick-Sort

12.5 Bubble-Sort

Wie arbeitet Bubble-Sort (Blasensortierung)?

Beim Bubble-Sort werden jeweils zwei benachbarte Elemente verglichen. Wenn das linke Element größer ist als das rechte, werden sie vertauscht.

① Begonnen wird der Vergleich mit dem ersten und dem zweiten Element. Dann werden das zweite und das dritte Element verglichen, danach das dritte und vierte Element usw., bis ans Ende des Arrays.
② So wandert das größte Element ans Ende des Arrays.
③ Nun werden die Schritte ① bis ② bis zum vorletzten Element wiederholt. Das letzte Element ist bereits sortiert, denn es ist entsprechend Schritt ② das größte Element des Arrays.
④ Danach steht das zweitgrößte Element auf der vorletzten Position. Die letzten zwei Elemente des Arrays sind sortiert.
⑤ Die Schritte ① bis ② werden so lange wiederholt, bis die zwei kleinsten Elemente verglichen werden. Dabei entfällt bei jeder Wiederholung das jeweils letzte Element, da es bereits sortiert ist.

Die kleinen (leichten) Elemente steigen wie Blasen in einer Flüssigkeit nach oben (an den Beginn des Arrays), deshalb heißt der Algorithmus Bubble-Sort (Sortieren mit Blasen).

In der nebenstehenden Grafik ist das Prinzip an einem Feld mit 5 Elementen dargestellt.

- Die mit einem Pfeil (↑) gekennzeichneten Elemente werden verglichen.
- Der Doppelpfeil (↔) steht zwischen Elementen, die vertauscht werden.
- Bereits sortierte Elemente sind farblich hinterlegt.

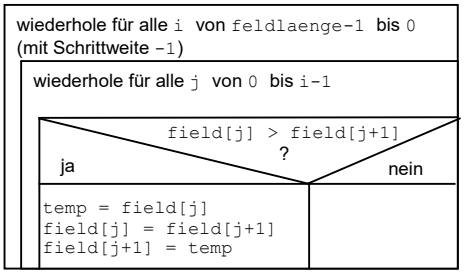
1. Schleifendurchlauf für alle n Elemente	1. Vergleich	3	6	2	4	1
	3 < 6 ? ja	↑	↑			
	2. Vergleich	3	6 ↔	2	4	1
	6 < 2 ? nein		↑	↑		
	3. Vergleich	3	2	6 ↔	4	1
	6 < 4 ? nein			↑	↑	
	4. Vergleich	3	2	4	6 ↔	1
	6 < 1 ? nein				↑	↑
2. Schleifendurchlauf für n-1 Elemente	1. Vergleich	3 ↔	2	4	1	6
	3 < 2 ? nein	↑	↑			
	2. Vergleich	2	3	4	1	6
	3 < 4 ? ja		↑	↑		
	3. Vergleich	2	3	4 ↔	1	6
	4 < 1 ? nein			↑	↑	
3. Schleifendurchlauf für n-2 Elemente	1. Vergleich	2	3	1	4	6
	2 < 3 ? ja	↑	↑			
	2. Vergleich	2	3 ↔	1	4	6
	3 < 1 ? nein		↑	↑		
4. Schleifendurchlauf für 2 Elemente	1. Vergleich	2 ↔	1	3	4	6
	2 < 1 ? nein	↑	↑			
Endergebnis		1	2	3	4	6

Beispiel: Array sortieren mit Bubble-Sort: *BubbleSort.java*

Die Feldelemente eines Arrays werden mit dem Sortieralgorithmus `bubbleSort` sortiert.

```
①  void bubbleSort(int[] field)
   {
②    int temp = 0;
③    for (int i = field.length - 1; i>= 0; i--)
     {
④      for (int j = 0; j <= i-1; j++)
       {
⑤        if (field[j] > field[j + 1])
         {
⑥          temp = field[j];
⑦          field[j] = field[j + 1];
⑧          field[j + 1] = temp;
         }
       }
     }
```

Prozedur: `bubbleSort`
Parameter: `field[ ]` → Feld mit n Elementen
Rückgabewert: keiner



① Der Methode `bubbleSort()` wird als Parameter das Feld, das sortiert werden soll, übergeben.

② Die Variable `temp` wird zum Zwischenspeichern eines Feldelementes verwendet.

③, ④ Es werden zwei Schleifen ineinander geschachtelt. Die äußere Schleife wird rückwärts für alle `i` von `Feldlänge-1` bis `0` durchlaufen. Damit wird jeweils die richtige Sortierung des i-ten Elements bewirkt. In der inneren Schleife werden bei jedem Schleifendurchlauf zwei Elemente miteinander verglichen, beginnend mit den ersten beiden Elementen bis zu den letzten beiden Elementen des (Teil-) Arrays.

⑤ Es wird geprüft, ob das aktuelle Element kleiner als das nachfolgende ist.

⑥ Die Elemente werden vertauscht. Dazu wird der Wert des aktuellen Elementes in der Variablen `temp` zwischengespeichert.

⑦ Der Wert des aktuellen Elementes ergibt sich aus dem Wert des nachfolgenden Elementes.

⑧ Dem nachfolgenden Element wird der zwischengespeicherte Wert von `temp` zugewiesen.

12.6 Insertion-Sort

Wie arbeitet Insertion-Sort?

Beim Insertion-Sort werden in eine bereits sortierte Menge, die zu Beginn auch leer sein kann, weitere Elemente sofort an der richtigen Position eingefügt, vergleichbar mit dem Einsortieren von Spielkarten beim Aufnehmen. Bei der Implementierung von Insertion-Sort wird häufig nur mit einem Array von Elementen gearbeitet. In diesem Fall haben Sie alle Karten auf einmal aufgenommen und sortieren sie durch Umstecken an die richtige Position (von einer Seite beginnend). Dadurch teilen Sie die Gesamtmenge aller Karten in eine sortierte und eine unsortierte Teilmenge.

① Die Sortierung wird zu Beginn mit dem kleinsten Teilarray mit zwei Elementen begonnen.
② Für das letzte Element des Teilarrays (in der unten stehenden Abbildung grau hinterlegt) wird jeweils die richtige Position gesucht.
③ Das letzte Element des Teilarrays wird mit den Elementen verglichen, die sich vor ihm im Array befinden.
④ Ist das letzte Element kleiner als ein Vorgängerelement, rutscht dieses Vorgängerelement um eine Position nach hinten.
⑤ Nach und nach rutschen alle Elemente des Arrays, die größer sind als das letzte Element, um eine Position nach hinten.
⑥ Wird ein Element gefunden, das kleiner ist als das letzte, wird das letzte Element an der Position hinter dem kleineren Element eingefügt (daher der Name Insertion-Sort).
⑦ Nun wird das Teilarray um das nächste Element erweitert, und die Schritte ③ bis ⑥ werden wiederholt.
⑧ Wenn das Teilarray die Größe des gesamten Arrays annimmt, ist der Algorithmus zu Ende.

In der nachfolgenden Grafik wird ein Array mit 5 Elementen (3, 6, 2, 4, 1) nach diesem Algorithmus sortiert.

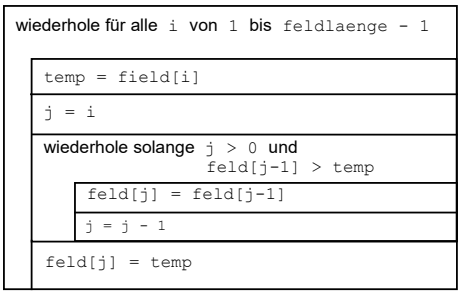
1. Schleifendurchlauf		temp = 6					
	1. Vergleich	3>6? nein	3	6	2	4	1
	innere Schleife beenden		3	6	2	4	1
Ergebnis 1. Durchlauf:			3	6	2	4	1
2. Schleifendurchlauf		temp = 2					
	1. Vergleich	6>2? ja	3	6	2	4	1
	2. Vergleich	3>2? ja	3	6	6	4	1
			3	3	6	4	1
Ergebnis 2. Durchlauf:			2	3	6	4	1
3. Schleifendurchlauf		temp = 4					
	1. Vergleich	6 > 4? ja	2	3	6	4	1
	2. Vergleich	3>4? nein	2	3	6	6	1
	innere Schleife beenden		2	3	6	6	1
Ergebnis 3. Durchlauf:			2	3	4	6	1
4. Schleifendurchlauf		temp = 1					
	1. Vergleich	6 >1? ja	2	3	4	6	1
	2. Vergleich	4 >1? ja	2	3	4	6	6
	3. Vergleich	3 >1? ja	2	3	4	4	6
	4. Vergleich	2 >1? ja	2	3	3	4	6
			2	2	3	4	6
Ergebnis 4. Durchlauf = Endergebnis			1	2	3	4	6

Beispiel: Feld sortieren mit Insertion-Sort: *InsertionSort.java*

Mithilfe der äußeren Schleife wird die Grenze zwischen dem sortierten Bereich von Element zu Element nach rechts verschoben. Begonnen wird beim zweiten Element (Index 1), da ein Element immer richtig sortiert ist (das erste Element kann noch nicht mit einem anderen verglichen werden). Über die innere Schleife werden im sortierten Bereich die Elemente, die größer als das aktuelle Element sind (`temp`), um eine Position nach rechts verschoben. In diese entstandene Lücke wird das Element eingefügt. Dabei wird die Reihenfolge der bereits einsortierten Elemente nicht geändert.

```
①  void insertionSort(int[] field)
   {
②    int temp = 0;
③    int j = 0;
④    for (int i = 1; i <= field.length - 1; i++)
⑤    {
⑥      temp = field[i];
⑦      j = i;
       while ((j > 0) && (field[j - 1] > temp))
⑧      {
⑨        field[j] = field[j - 1];
         j = j - 1;
⑩      }
       field[j] = temp;
     }
   }
```

Prozedur: `insertionSort`
Parameter:
`field[ ]` → unsortiertes Feld mit n Elementen
Rückgabewert: keiner



① Der Methode `insertionSort()` wird als Parameter das Feld, das sortiert werden soll, übergeben.

② Die Variable `temp` wird zum Zwischenspeichern eines Feldelementes verwendet.

③ In der Variablen `j` kann ein Integerwert zwischengespeichert werden.

④, ⑦ Es werden zwei Schleifen ineinander geschachtelt. Die äußere Schleife wird für alle `i` von `1` bis `Feldlänge - 1` durchlaufen. Damit wird jeweils ein Teilfeld mit `i+1` Elementen geordnet.

⑤ Das letzte Element des Teilfeldes wird in der Variablen `temp` zwischengespeichert.

⑥ Für die innere Schleife wird die aktuelle Länge des Teilfeldes in `j` gespeichert.

⑦ In der inneren Schleife wird bei jedem Schleifendurchlauf das letzte Element (`temp`) mit einem der Vorgänger verglichen. Die Schleife wird so lange durchlaufen, bis ein Vorgängerelement gefunden wird, das kleiner ist als das letzte Element, oder bis keines mehr vorhanden ist. Bei manchen Programmiersprachen muss an dieser Stelle das bedingte logische Und verwendet werden.

⑧ Ist das Vorgängerelement größer als das letzte Element, rutscht es auf die nachfolgende Position.

⑨ Damit das vorhergehende Element untersucht werden kann, wird der Wert der Variablen `j` um 1 verringert.

⑩ Das letzte Element, welches in `temp` zwischengespeichert ist, wird an der gefundenen Position eingefügt.

12.7 Shell-Sort

Wie arbeitet Shell-Sort?

Der Shell-Sort ist eine Erweiterung des Insertion-Sort. Beim Shell-Sort werden Elemente vertauscht, die weiter voneinander entfernt sind. Dadurch wird die Geschwindigkeit der Sortierung gegenüber dem Insertion-Sort erhöht.

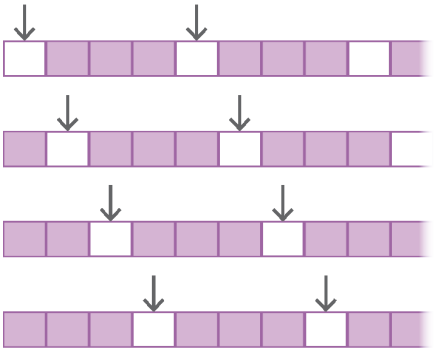
Das Prinzip beruht darauf, dass eine Vorsortierung des Arrays mit großen Schrittweiten erfolgt. Die Schrittweite wird im Laufe des Algorithmus verkleinert, bis sie den Wert 1 annimmt. Die Elemente des Arrays werden erst mit der großen Schrittweite sortiert, sodass die Elemente näher an ihre Endposition gebracht werden. Dadurch müssen sie bei einer Schrittweite von 1 (entspricht Insertion-Sort) nur noch um wenige Positionen verschoben werden.

Von der Folge der Schrittweiten hängt die Effizienz der Sortierung ab. Mit der folgenden Formel können Sie geeignete Schrittweiten berechnen: $h_{i+1} = 3*h_i+1$.

Gehen Sie vom Wert $h_i=1$ aus. Notieren Sie das Ergebnis (h_{i+1}), und setzen Sie den berechneten Wert anschließend für h_i in die Formel ein. Wiederholen Sie die Berechnungen, bis das Ergebnis die Hälfte der Anzahl der zu sortierenden Elemente erreicht hat. Sie erhalten eine Zahlenfolge, die die Schrittweiten in umgekehrter Reihenfolge enthält.

Somit ergibt sich eine Schrittfolge von 1, 4, 13, 40, 121, 364, 1093 ..., die Sie in umgekehrter Reihenfolge beim Sortiervorgang verwenden können. Die Anzahl der Vorsortierungen wird von der Anzahl der Elemente des Arrays bestimmt.

Bei der ersten Sortierung wird z. B. nur jedes 4 Element betrachtet
→ Schrittweite h = 4.



Es liegen nun vier sortierte Teilfolgen vor.

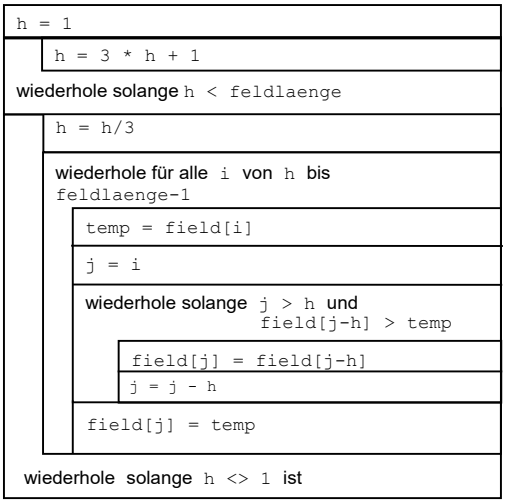
Danach wird das Feld mit der Schrittweite 1 nach dem Insertion-Sort-Algorithmus als eine Teilfolge sortiert.

Beispiel: Feld sortieren mit Shell-Sort: *ShellSort.java*

Zunächst wird die größtmögliche Schrittweite mit der oben beschriebenen Formel berechnet. Elemente mit dem Abstand der Schrittweite werden entsprechend ihrer Größe verglichen und wenn notwendig gegeneinander vertauscht. Anschließend wird mit den Nachfolgern der Zahl ebenso verfahren. Ist das Ende der Elementeliste erreicht, wird der Abstand (Schrittweite) auf ein Drittel der eben verwendeten reduziert (`h/3`) und die Elemente werden erneut durchlaufen. Die letzte angewendete Schrittweite ist 1. Danach liegt die Menge der Elemente sortiert vor.

```
①  void shellSort(int[] field)
   {
②    int temp = 0;
③    int h = 1;
④    int j = 0;
     do
⑤      h = 3 * h + 1;
⑥    while (h < field.length);
⑦    do
     {
⑧      h = h / 3;
⑨      for (int i = h; i <= field.length - 1;
            i++)
       {
         temp = field[i];
         j = i;
⑩        while ((j >= h) &&
                (field[j - h] > temp))
         {
           field[j] = field[j - h];
           j = j - h;
         }
         field[j] = temp;
⑪      }
     }
⑫    while (h != 1);
   }
```

Prozedur: `shellSort`
Parameter:
`field[ ]` → unsortiertes Feld mit n Elementen
Rückgabewert: keiner



① Der Methode `shellSort()` wird als Parameter das Array, das sortiert werden soll, übergeben.

② Die Variable `temp` wird zum Zwischenspeichern eines Elementes des Arrays verwendet.

③ Der Variablen `h` wird der Wert `1` zugewiesen, der der kleinsten Schrittweite entspricht.

④ In der Variablen `j` kann ein Integerwert zwischengespeichert werden.

⑤, ⑥ Der Wert der Variablen `h` wird so lange mithilfe der angegebenen Formel erhöht, bis er die Anzahl der Elemente erreicht bzw. übersteigt.

⑦, ⑫ Bei diesem Sortierverfahren werden drei Schleifen ineinander geschachtelt. Die beiden inneren Schleifen ⑨, ⑩ entsprechen dem Insertion-Sort. Die äußere Schleife dient der Veränderung der Schrittweite, die in jedem Schleifendurchlauf durch 3 geteilt wird.

⑧ Die neue Schrittweite wird berechnet. Da die Schrittweite eine ganze natürliche Zahl sein muss, ist bei der Implementierung innerhalb einer konkreten Sprache eine Rechenoperation anzuwenden (hier die ganzzahlige Division), die ein ganzzahliges Ergebnis liefert.

⑨ – ⑪ Nun wird der Algorithmus des Insertion-Sort mit der Schrittweite `h` auf das jeweilige Teilarray angewendet.

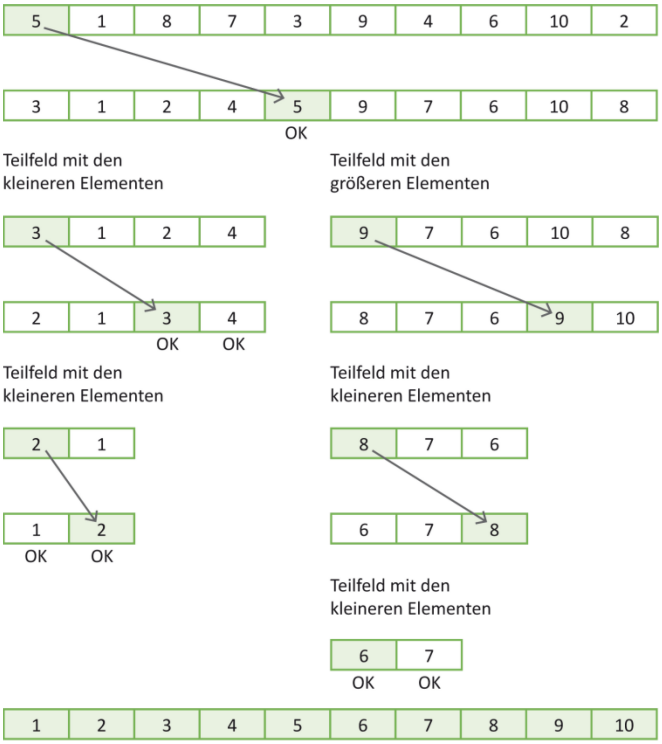
12.8 Quick-Sort

Wie arbeitet Quick-Sort?

Der wohl am häufigsten verwendete Sortieralgorithmus ist Quick-Sort.

Ein Array wird dabei in zwei Teile zerlegt, die vorsortiert sind.

- Dazu wird ein Element als Vergleichselement genutzt, das an die richtige Stelle im Array verschoben wird.
- Alle Elemente, die kleiner sind als dieses Element, werden in das linke Teilarray verschoben.
- Die größeren Elemente kommen in das rechte Teilarray.
- Das einsortierte Element steht nun an der richtigen Stelle und braucht nicht mehr beachtet zu werden.
- Die beiden Teilarrays links und rechts von diesem Element werden dann nach dem gleichen Prinzip weiterverarbeitet wie das gesamte Array.

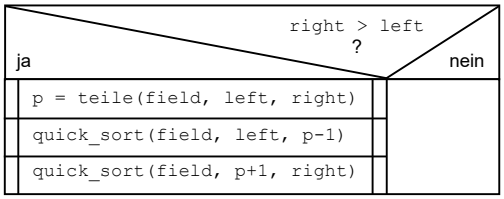


Beispiel: Feld sortieren mit Quick-Sort: *QuickSort.java*

Die Methode im folgenden Beispiel ist rekursiv programmiert. Sie ruft die Methode `divideAndSort()` auf, die die Unterteilung des Arrays in zwei Teilarrays übernimmt. Danach wird die Methode `quickSort()` mit beiden Teilfeldern aufgerufen.

```
①  void quickSort(int[] field, int left,
                  int right)
   {
②    if (right > left)
     {
③      int p = divideAndSort(field, left,
                             right);
④      quickSort(field, left, p -1);
⑤      quickSort(field, p + 1, right);
     }
   }
```

Prozedur: `quickSort`
Parameter:
`field[ ]` → unsortiertes Feld mit n Elementen
`left` → linker Rand des (Teil-)Feldes
`right` → rechter Rand des (Teil-)Feldes
Rückgabewert: keiner



① Als Parameter für die Methode `quickSort()` müssen ein Feld sowie die Unter- und die Obergrenze des Arrays bzw. Teilarrays (für rekursiven Aufruf) übergeben werden.

② Solange die linke Grenze (Untergrenze) größer ist als die rechte Grenze (Obergrenze), hat das Array mindestens 2 Elemente und muss sortiert werden.

③ Das Teilen und Umsortieren wird durch die Methode `divideAndSort()` realisiert. Der Methode werden als Parameter das Array sowie die Unter- und die Obergrenze des Arrays übergeben. Sie gibt die Position des Elements zurück, welches sie einsortiert hat. Links von diesem Element stehen nur noch Werte, die kleiner sind als der einsortierte Wert, und rechts nur noch Werte, die größer sind. Diese beiden Teilarray müssen jetzt sortiert werden.

④ Die Methode `quickSort()` wird nun mit dem Teilarray aufgerufen, das die kleineren Werte enthält.

⑤ Das Teilarray mit den größeren Werten wird anschließend als Parameter der Methode `quickSort()` übergeben.

Das Zerlegen des Arrays ist der kompliziertere Teil des Algorithmus. In diesem Beispiel funktioniert er wie folgt:

① Das erste Element des Arrays wird als das Element angenommen, welches an die richtige Position gebracht werden soll.

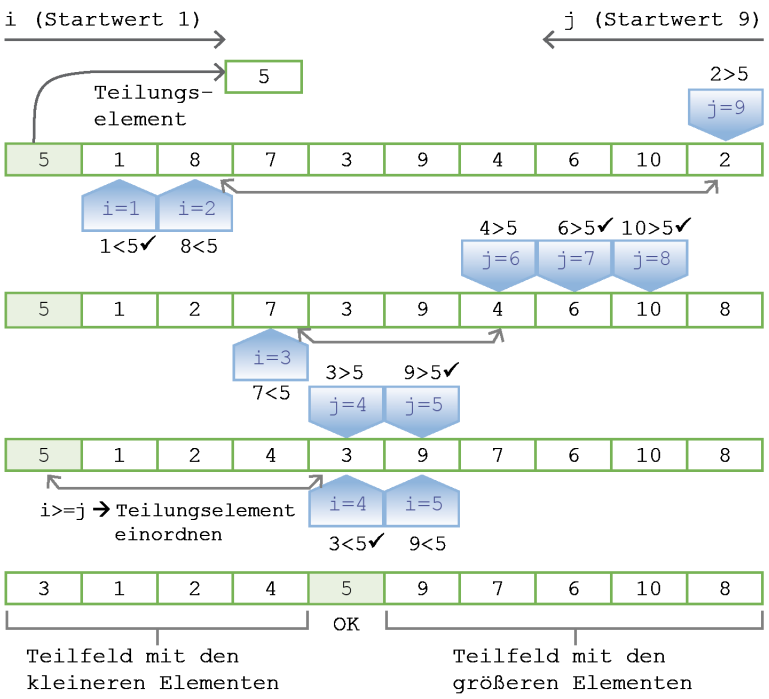
② Bevor das geschehen kann, müssen die anderen Elemente entsprechend umgeordnet werden.

③ Dazu werden zwei Zeiger verwendet. Der Zeiger `i` läuft vom ersten zum letzten Element, der Zeiger `j` vom letzten zum ersten.

④ Mithilfe des Zeigers `i` werden die Elemente so lange untersucht, bis eines gefunden wird, das größer ist als das Teilungselement.

⑤ Mithilfe des Zeigers `j` werden von rechts beginnend die Elemente durchsucht, bis eines gefunden wird, das kleiner ist als das Teilungselement.

⑥ Das i-te und das j-te Element werden vertauscht.



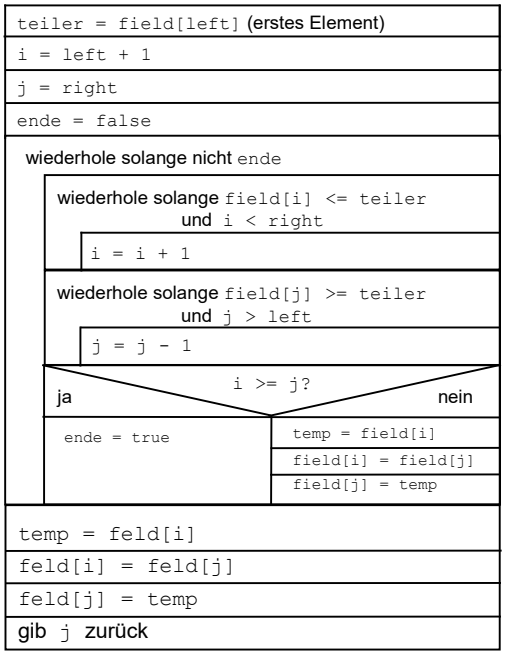
⑦ Die Punkte ④ bis ⑥ werden so lange wiederholt ausgeführt, bis der Zeiger `i` größer oder gleich dem Zeiger `j` ist. Dann ist die Position gefunden, an der das Teilungselement eingefügt werden muss – nämlich an die Position, auf die der Zeiger `j` verweist.

⑧ Dazu tauschen das erste und das j-te Element die Plätze.

⑨ Es entstehen zwei Teilarrays. Ein Teilarray, welches die kleineren Elemente enthält, und eines, das die größeren Werte enthält.

```
①  int divideAndSort(int[] field, int left,
                     int right)
   {
②    int divider = field[left];
③    int i = left + 1;
④    int j = right;
⑤    boolean isFinished = false;
⑥    int temp = 0;
⑦    while (!isFinished)
     {
⑧      while ((field[i] <= divider) &&
              (i < right))
         i++;
⑨      while ((field[j] >= divider) &&
              (j > left))
         j--;
⑩      if (i >= j)
         isFinished =true;
       else
       {
⑪       temp = field[i];
         field[i] = field[j];
         field[j] = temp;
       }
     }
⑫    temp = field[j];
     field[j] = field[left];
     field[left] = temp;
⑬    return j;
   }
```

Funktion: `divideAndSort`
Parameter:
`field[ ]` → unsortiertes Feld mit n Elementen
`left` → linker Rand des (Teil-)Feldes
`right` → rechter Rand des (Teil-)Feldes
Rückgabewert:
`j` → Position des einsortierten Elements



① Als Parameter für die Methode `divideAndSort()` muss ein Feld übergeben werden sowie die Unter- und die Obergrenze des Arrays.

② Das erste Element des Arrays wird in der Variablen `divider` abgelegt.

③ Die Variable `i` wird als Zeiger auf die aktuelle Position im vorderen Teil des Arrays verwendet. Da das erste Element einsortiert werden soll, zeigt `i` zu Beginn auf das zweite Element.

④ Die Variable `j` wird als Zeiger auf die aktuelle Position im hinteren Teil des Arrays verwendet.

⑤, ⑦ Mithilfe der Variable `isFinished` wird die folgende Schleife (⑦) gesteuert. Solange die Variable `isFinished` den Wert `false` besitzt, wird die Schleife immer wieder durchlaufen.

⑥ Die Variable `temp` wird zum Zwischenspeichern eines Arrayelementes verwendet.

⑧ Die Schleife endet, wenn ein Element gefunden wurde, welches größer ist als das einzuordnende Element (`divider`), oder wenn `i` die obere Grenze erreicht hat.

⑨ Die Schleife wird so lange durchlaufen, bis ein Element gefunden wird, welches kleiner ist als das einzuordnende Element (`divider`), oder bis der Wert von `j` die untere Grenze erreicht hat.

⑩ Ist `i` größer oder gleich `j`, wird die Schleife beendet. Dieser Fall tritt ein, wenn die richtige Position für das einzuordnende Element gefunden ist. Der Wert der Variablen `isFinished` wird dann auf `true` gesetzt.

⑪ Wurden in den Schleifen ⑧ und ⑨ zwei Elemente gefunden, sodass `i < j` gilt, werden die Elemente getauscht.

⑫ Das einzuordnende Element wird an seine endgültige Position gesetzt, sodass es richtig einsortiert ist. Alle Elemente links von ihm sind kleiner, und alle Elemente rechts von ihm sind größer.

⑬ Die Funktion gibt die Position zurück, an der das Element eingeordnet wurde.

12.9 Vergleich der Sortierverfahren

Die Sortieralgorithmen unterscheiden sich in ihrer Leistungsfähigkeit, die von der Anzahl der Vergleiche und der Anzahl der Austauschoperationen beeinflusst wird. Auch die Reihenfolge vor der Sortierung hat Einfluss auf die Geschwindigkeit des Algorithmus. In der folgenden Tabelle sind einige Sortieralgorithmen und die durchschnittliche Anzahl der Vergleiche beim Sortieren von n Elementen aufgeführt.

Sortieralgorithmus	Anzahl Vergleiche im Durchschnitt	Beispiel für 1000 Elemente
Quick-Sort	$2n \ln n$	13816
Shell-Sort (für Schrittweite h = 3 * h+1)	$n^{3/2}$	31623
Insertion-Sort	$n^2/4$	250000
Bubble-Sort	$n^2/2$	500000

An den berechneten Beispielwerten können Sie erkennen, dass Quick-Sort der schnellste Algorithmus zum Sortieren ist. Das macht sich bei einer großen Zahl von Elementen noch deutlicher bemerkbar.

12.10 Mit Daten in Dateien arbeiten

Als Datei (engl. file) wird eine Sammlung von logisch zusammengehörigen Daten bezeichnet. Daten in Dateien dienen Programmen zur Eingabe (Daten lesen) oder werden von Programmen für die Ausgabe genutzt (Dateien schreiben).

Dateien werden meist auf Datenträgern wie Festplatte, Magnetbändern, USB-Stick oder DVD gespeichert. Auf diesen Datenträgern können Daten über längere Zeit aufbewahrt werden.

Die Inhalte der Dateien können verschieden sein und z. B. Texte, Grafiken, Videos oder Programme enthalten. Von Programmen mit betrieblicher Anwendung werden häufig Dateien für die Speicherung von Datensätzen mit einer festgelegten Datenstruktur (wie z. B. bei Strukturen oder Feldern) verwendet.

Zugriffsarten

Auf Dateien gibt es verschiedene Zugriffsarten, die auch durch das Speichermedium bedingt sind.

- **Sequenzielle Datei**

  Können die Daten, wie in der folgenden Abbildung, nur nacheinander gelesen werden, wird die Datei als **sequenzielle Datei** bezeichnet. Dateien auf Magnetbändern z. B. sind sequenzielle Dateien.

Beispiel für den Aufbau einer sequenziellen Datei

Die Laufzeitumgebung von Programmen stellt meist eine Standardeingabe über die Tastatur und eine Standardausgabe über den Bildschirm oder Drucker als sequenzielle Datei zur Verfügung.

- **Index-sequenzielle Datei**

  Über einen Index, der in einer zusätzlichen Index-Datei gespeichert wird, kann direkt auf Daten innerhalb der sequenziellen Datei zugegriffen werden.

- **Direktzugriffsdatei**

  In einer **Direktzugriffsdatei** (Random-Access-Files) können Sie auf einen bestimmten Datensatz bzw. bestimmte Elemente direkt zugreifen. Direktzugriffsdateien befinden sich z. B. im Hauptspeicher oder der Festplatte. Dadurch kann auf die Daten über die Speicheradresse bzw. über die Sektor- und Zylindernummer zugegriffen werden.

Arbeiten mit Dateien

Soll eine Datei in einem Programm verarbeitet werden, muss der Aufbau der Datei klar definiert sein. Sie müssen wissen, von welchem Datentyp die einzelnen Daten sind, ob und welche Trennzeichen zwischen Daten bzw. Datensätzen existieren und wie die Datei organisiert ist.

Für die Arbeit mit Dateien im Programm sind folgende Arbeitsschritte erforderlich.

- **Vereinbaren der Dateivariablen**

  Jede Programmiersprache besitzt einen Datentyp zur Vereinbarung von Dateivariablen, die auf eine Datei verweist. Häufig kann dabei auch angegeben werden, von welchem Datentyp die Daten der Datei sind.

- **Öffnen der Datei**

  Damit auf die Datei zugegriffen werden kann, muss sie geöffnet werden. Zum Öffnen der Datei besitzen die Programmiersprachen verschiedene Modi, die oft auch kombiniert werden können. Eine Datei kann zum *Lesen* (read), *Schreiben* (write), *Lesen und Schreiben* und zum *Anfügen* (append) von Daten geöffnet werden. Dateien besitzen einen Dateizeiger, der auf die aktuelle Lese- bzw. Schreibposition verweist. Beim Öffnen der Datei wird der Datensatzzeiger initialisiert. Er verweist dann immer auf das erste Zeichen der Datei.

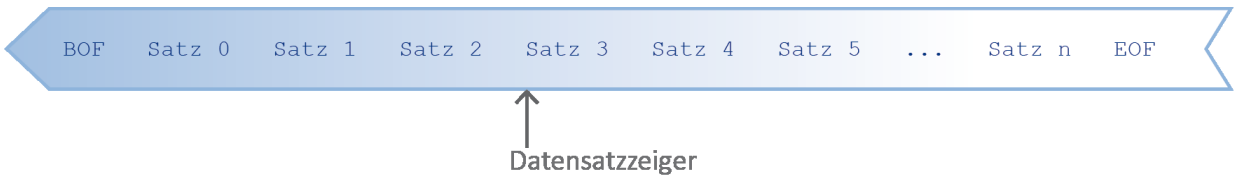
- **Zugreifen auf die Daten**

  Nur mit einer geöffneten Datei kann gearbeitet werden. Je nachdem, unter welchem Modus die Datei geöffnet wurde, kann lesend und/oder schreibend darauf zugegriffen werden.

  Bei der Verarbeitung sequenzieller Dateien wird bei jedem Lese- bzw. Schreibvorgang der Datensatzzeiger um die Anzahl der gelesenen bzw. geschriebenen Zeichen weitergerückt. Bei einer Direktzugriffsdatei ist dies nicht der Fall. Dazu gibt es bei der Verarbeitung von Direktzugriffsdateien einen Befehl.

  Hierbei ist Vorsicht geboten, da der Datensatzzeiger nicht vor das erste (bei negativen x-Werten) bzw. hinter das letzte Element positioniert werden darf. Meist gibt es noch Funktionen, mit denen die Anzahl der Elemente in der Datei und die aktuelle Position des Datensatzzeigers ermittelt werden können. Diese können bei der Positionierung helfen.

  Jede sequenzielle Datei besitzt ein Endekennzeichen (EOF – End of File) und in einigen Sprachen auch ein Anfangskennzeichen (BOF – Begin of File), welches abgefragt werden kann. Bevor Sie z. B. in einer Datei lesen, sollten Sie testen, ob sich der Datensatzzeiger nicht schon am Ende der Datei, also auf EOF befindet. Versuchen Sie, in einer Datei zu lesen, die auf EOF steht, tritt ein Fehler auf.



Dateiaufbau und Datensatzzeiger

- **Schließen der Datei**

  Ist die Dateiarbeit beendet, muss die Datei wieder geschlossen werden.

In objektorientierten Programmiersprachen stehen für die Dateieingabe und -ausgabe zusätzlich zu den beschriebenen Methoden noch weitere Klassen zur Verfügung, die diese Aufgabe übernehmen. Die Arbeit mit diesen Klassen ist zum einen komfortabler, zum anderen steht mehr Funktionalität zur Verfügung. Das Prinzip ist aber das gleiche wie bei der konventionellen Methode.

Beispiel: Mit einer Direktzugriffsdatei arbeiten: *RandomAccess.java*

Im folgenden Beispiel werden Daten in eine Direktzugriffsdatei gespeichert und wieder ausgelesen.

```
① try
②   {
      RandomAccessFile file = new RandomAccessFile("testData.dat", "rw");
③     file.writeBytes("Testdaten\nfuer\ndie\nDirektzugriffsdatei!");
④     file.seek(0);
      System.out.println("Ausgabe der Testdaten:");
      for (int i = 1; i <= 4; i++)
⑤       System.out.println(file.readLine());
⑥     file.seek(10);
⑦     System.out.println("Ausgabe ab 12.Zeichen:"+file.readLine());
⑧     file.close();
    }
⑨ catch ...
```

①, ⑨ Die Laufzeitfehler, die beim Arbeiten mit Random-Access-Dateien auftreten können, müssen in `try-catch`-Blöcken abgefangen werden. Laufzeitfehler erzeugen eine sogenannte Exception (Ausnahme), die behandelt werden muss. Im `try`-Block stehen die regulären Anweisungen, die auch Ausnahmen erzeugen können. Im `catch`-Block wird auf diese Ausnahmen reagiert.

② Die Datei *testData.dat* wird zum Lesen und Schreiben (`rw`) geöffnet. Wenn sie noch nicht existiert, wird sie angelegt.

③ Die Methode `writeBytes()` schreibt die übergebene Zeichenkette in die Datei. Die Zeichenkette besteht aus vier Zeilen mit jeweils unterschiedlich vielen Buchstaben, die mit `\n` (Zeilenendekennzeichen) abgeschlossen sind.

④ Der Dateizeiger wird auf den Dateianfang positioniert. Das erste Zeichen hat die Position 0.

⑤ Um eine Zeile (bis zum Zeilenendekennzeichen `\n`) aus der Datei zu lesen, wird die Methode `readLine()` verwendet. Die eingelesene Zeichenkette wird auf dem Bildschirm angezeigt.

⑥ Der Dateizeiger wird auf den Anfang des zweiten Wortes der Testdaten positioniert. Das erste Zeichen des zweiten Wortes hat die Position 11, da auch die Zeilenendezeichen mitgezählt werden müssen.

⑦ Ab der Position, auf die der Positionszeiger zeigt, wird eine Zeile aus der Datei gelesen. Die gelesene Zeile wird auf dem Bildschirm angezeigt.

⑧ Die Methode `close()` schließt die Datei.

```
Ausgabe der Testdaten:
Testdaten
fuer
die
Direktzugriffsdatei!
Ausgabe ab 12.Zeichen:fuer
```

Die Ausgabe des Programms

12.11 Übung

Sortieralgorithmen

Übungsdatei: -- **Ergebnisdateien:** ***uebung12.pdf, MeasuredValues.java, MeasuredValuesInput.java,MeasuredValuesOutput.java, MeasuredValues2.java***

1. Unter welchen Bedingungen würden Sie die Sortieralgorithmen Bubble-Sort, Insertion-Sort, Shell-Sort und Quick-Sort einsetzen?
2. Erstellen Sie einen Algorithmus für folgendes Problem: Vom Benutzer eines Programms werden Messwerte (ganzzahlig) eingegeben. Diese sollen zunächst in einem Feld gespeichert werden. Nach Beendigung der Eingabe werden die Messwerte sortiert (durch einen geeigneten Sortieralgorithmus) und anschließend in eine Datei Messwerte.dat geschrieben.
3. Erstellen Sie für die Aufgabe 2 ein Java-Programm. Speichern Sie den Programmcode für die Eingabe unter MeasuredValuesInput.java, für die Ausgabe unter MeasuredValuesOutput. java und für das Hauptprogramm sowie die Sortierung unter MeasuredValues.java.
4. Die Daten aus der in Aufgabe 2 erstellten Datei Messwerte.dat sollen in ein Feld gelesen werden. Danach ist der Mittelwert zu berechnen und der minimale und maximale Wert zu ermitteln. Geben Sie die Ergebnisse aus.

   Der Mittelwert ergibt sich aus der Summe der Werte aller Feldelemente, geteilt durch deren Anzahl. Zum Beispiel kann der Mittelwert der Zahlen 3, 4, 5 durch (3 + 4 + 5) / 3 = 4 berechnet werden.
5. Lösen Sie die Aufgabe 4 mit Java-Programmen. Für die Ein- und Ausgabe wird der in Aufgabe 3 genannte Programmcode verwendet. Speichern Sie das Hauptprogramm und den Programmcode zur Berechnung des Mittelwertes etc. unter MeasuredValues2.java.

13 Grundlagen der Softwareentwicklung

In diesem Kapitel erfahren Sie

- wie Sie konzeptionell beim Entwurf einer Software vorgehen können
- was das Phasenmodell des Software-Lebenszyklus ist
- nach welchen Vorgehensmodellen Software entwickelt werden kann
- welche Anforderungen an ein Programm gestellt werden

13.1 Software entwickeln

Vorgehensmodelle einsetzen

Abhängig von der zu erstellenden Software werden bei der professionellen Programmierung **Vorgehensmodelle** gewählt, die festlegen, wie die Software entwickelt wird. Ein Vorgehensmodell beschreibt, welche Prinzipien, Methoden und Darstellungsmittel eingesetzt werden. Vorgehensmodelle sind in großen Projekten mit mehreren beteiligten Personen oder Gruppen/Abteilungen notwendig, um die Entwicklung der Software in verwaltbare und kontrollierbare Teile zu gliedern.

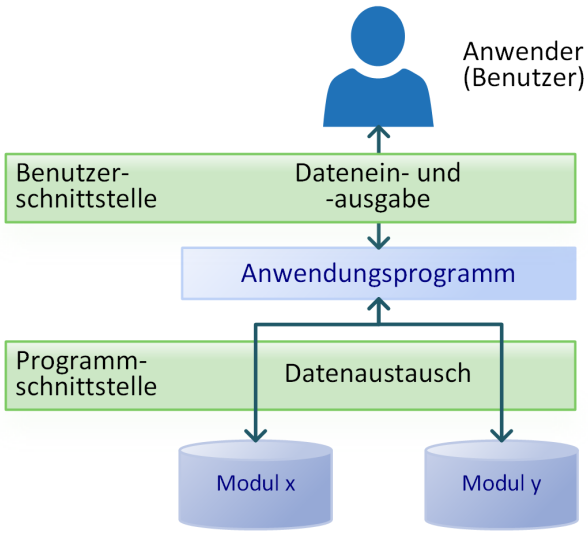
Prinzipien lassen sich aus Erfahrungen und Erkenntnissen herleiten. Prinzipien sind nicht speziell auf ein Anwendungsgebiet begrenzt. So lässt sich beispielsweise das Abstraktionsprinzip in der Informatik ebenso anwenden wie in der Mathematik oder Philosophie. Von grundlegender Bedeutung für die Entwicklung von Software sind das Modularitäts- und das Abstraktionsprinzip.

Wie eine große Aufgabe überschaubar wird

Das Prinzip der **Modularität** besagt, dass eine Gesamtaufgabe in Teilaufgaben (**Module**) zerlegt wird. Dadurch erhöht sich die Verständlichkeit und reduziert sich die Komplexität.

Durch die Modularisierung z. B. eines Programms können die Module einzeln programmiert und getestet werden.

Module besitzen eine definierte Schnittstelle zu deren Verwendung. Als **Schnittstelle** (engl. interface) wird die Verbindungsstelle zwischen Modulen selbst oder zwischen Programm und Anwender bezeichnet, über die ein Daten- bzw. Informationsaustausch stattfindet.



Vorteile der Modularisierung

Beschreibung	Vorteil
Bei der Programmierung von Software, die aus einzelnen Modulen besteht, können diese unabhängig voneinander programmiert werden. Die Module können somit in Teams entwickelt werden.	verkürzte Entwicklungszeit, bessere Verteilung von Arbeiten und Ressourcen
Jedes Modul kann einzeln auf seine Korrektheit getestet werden. Fehler, die nach dem Zusammenfügen der einzelnen Module auftreten, lassen sich relativ leicht finden, da sie bei Modulen, die selbst korrekt arbeiten, dann meist nur an den Schnittstellen auftreten. Die definierten Schnittstellen erlauben es, ein Modul durch ein anderes Modul mit identischer Schnittstelle auszutauschen.	verbesserte Wartbarkeit
Das Modul selbst kann in verschiedenen Programmen verwendet werden, in denen ein Modul mit der entsprechenden Funktionalität benötigt wird.	Wiederverwendbarkeit
Die Datenkapselung (vgl. Abschnitt 10.3) in Modulen stellt einen wichtigen Aspekt bezüglich Sicherheit und Zuverlässigkeit eines Programms dar. Unkontrollierte Zugriffe auf ein Modul sind bei einer sauberen Umsetzung (weitgehend) ausgeschlossen, Manipulationen sind nicht möglich.	Sicherheit und Zuverlässigkeit

Die Wiederverwendbarkeit von Modulen bzw. Softwarekomponenten spielt eine große Rolle bei der Einsparung von Entwicklungszeit und -kosten.

Beispiel: Modularisierung

Weltweit arbeiten viele Entwickler an sogenannter OpenSource-Software mit. Das ist Software, deren Quellcode für jedermann zugänglich ist, z. B. Linux oder Gimp. Die Entwicklung dieser Software ist nur durch Aufteilung in Module möglich, die dann von einzelnen Entwicklern bearbeitet werden können. Diese Module werden dann zu einem größeren System zusammengefasst und bilden damit erst als zusammengefügte Einheit das konkrete Programm.

Reduktion auf das Wesentliche durch Abstraktion

Das **Abstraktionsprinzip** ist wichtig, um unwesentliche Informationen von Aufgabenstellungen auszuschließen. Somit liegen nach der Abstraktion nur die zur Lösung einer vorgegebenen Aufgabe notwendigen Informationen vor. Eine Abstraktion ist damit eine Auswahl relevanter Informationen aus einer größeren Menge von verfügbaren Informationen.

Beispiel: Abstraktion

Das Durchschnittsalter von Schülern einer Klasse soll mithilfe eines Programms automatisch berechnet werden. Bekannt sind die Geburtsdaten der Schüler, deren Anzahl und Namen, das Geschlecht sowie das Datum des ersten Schultags. Die Namen der Schüler sowie deren Geschlecht und das Datum des ersten Schultags sind für die Berechnung des Durchschnittsalters unwesentlich. Durch die Abstraktion werden sie bei der Lösung der Aufgabe nicht berücksichtigt. Als notwendige Information für die Berechnung des Durchschnittsalters bleiben die Geburtsdaten und die Anzahl der Schüler. Die Geburtsdaten und die Anzahl sind die für den konkreten Fall gewünschte Abstraktion.

13.2 Methoden

Eine **Methode** ist eine systematische Vorgehensweise, um bestimmte Aufgaben im Rahmen festgelegter Prinzipien zu lösen. Ausgehend von den gegebenen Bedingungen wird ein Ziel mithilfe definierter Arbeitsschritte erreicht.

Beachten Sie, dass hier nicht die Definition der Methode gemeint ist, wie sie in der objektorientierten Programmierung Verwendung findet (vgl. Abschnitt 10.7).

Schrittweise Verfeinerung (stepwise refinement)

Zunächst wird auf einer hohen Abstraktionsebene die Aufgabenstellung des Problems formuliert. Damit wird festgelegt, **was** gemacht werden soll.

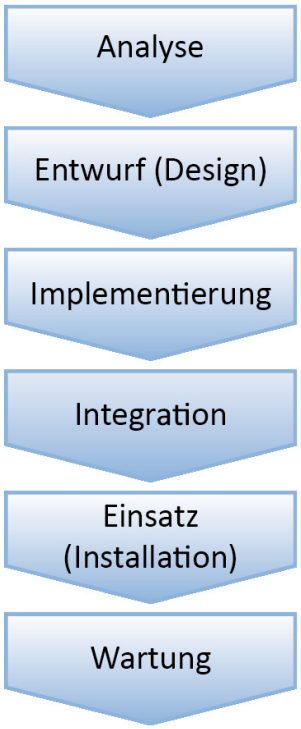
Mit der nachfolgenden schrittweisen Verfeinerung der Aufgabenstellung wird die Frage beantwortet: **Wie** ist die Aufgabenstellung zu lösen?

Bleibt noch die Frage: **Womit** ist die Aufgabenstellung zu lösen, das heißt, mit welcher Programmiersprache, welchen Algorithmen und anderen Softwarekomponenten kann die Aufgabenstellung realisiert werden?
Es existieren verschiedene Methoden des strukturierten Programmentwurfs:

Top-down-Methode	Ausgangspunkt ist die Gesamtaufgabe, die in Teilaufgaben zerlegt wird. Bei komplexen Aufgabenstellungen können die Teilaufgaben in weitere Teilaufgaben aufgeteilt werden. Bei der Entwicklung eines einzelnen Programms ist diese Methode oft vorzuziehen.
Bottom-up-Methode	Diese Methode ist für Vorgehensweisen interessant, bei denen eine Aufgabenstellung nicht exakt beschrieben ist bzw. noch Änderungen erwartet werden. Einzeln entwickelte Module werden später zu einem großen Modul zusammengesetzt. Die Bottom-up-Methode ist meist beim Entwurf großer Programmsysteme sinnvoll. Bei dieser Methode können Probleme beim Zusammensetzen der Module zum Gesamtsystem auftreten, da zu Beginn das Gesamtsystem nicht genau festgelegt war. Der Vorteil besteht darin, dass bereits entwickelte Module eingesetzt werden können.
Up-down-Methode (Middle-Out, Gegenstromverfahren)	Bei dieser Strukturierungsmethode wird die Gesamtaufgabe durch Top-down-Methoden verfeinert, und es werden Teilaufgaben bottom-up abstrahiert. Auf diese Weise können kritische Teilaufgaben zuerst getestet werden.

13.3 Der Software-Lebenszyklus

Professionelle Software wird zunehmend komplexer. Um die Erstellung trotzdem planen und kontrollieren zu können, wurden strukturierte Vorgehensmodelle entwickelt. Grundlage dieser Modelle ist die Einteilung des Software-Lebenszyklus in sogenannte **Phasen**.



In jeder Phase werden bestimmte Aufgaben erledigt, getestet und dokumentiert. Tests in den jeweiligen Phasen dienen dazu, Fehler bei der Entwicklung der Software zu einem möglichst frühen Zeitpunkt zu erkennen.

Je später im Entwicklungsprozess ein Fehler erkannt wird, umso höher ist der Aufwand, z. B. Kosten oder Zeit, zu seiner Beseitigung.

Eine Phase sollte beendet sein, bevor die nächste begonnen wird. In der Praxis kann die strikte Abfolge und Trennung der Phasen meist nicht eingehalten werden. Es kann vorkommen, dass bestimmte Aufgaben verschiedenen Phasen zugeordnet werden können. Ebenso sind nicht alle Schritte einer jeweiligen Phase immer zwingend notwendig.

Sowohl für die Entwicklung neuer Software als auch für die Wartung und Pflege bereits existierender Software ist das Phasenmodell verwendbar. Der Software-Lebenszyklus ist dadurch charakterisiert, dass einzelne Phasen aufgrund von Entscheidungen, die aus nachfolgenden Phasen resultieren, zyklisch wiederholt zu durchlaufen sind.

Analysephase

Aufgabe	**Was** soll die Software tun? Folgende Anforderungen werden in dieser Phase definiert: ✔ Funktionsumfang und Qualitätsmerkmale des Programms ✔ Art der Benutzeroberfläche ✔ Schnittstelle der Systemumgebung ✔ Eingesetzte Hardware während der Programmierung ✔ Software, die zum Erstellen des Programms eingesetzt wird (entfällt beim programmiersprachenunabhängigen Entwurf) ✔ Umfang der Dokumentation ✔ Ein-/Ausgabe- und Testdaten festlegen Außerdem können Wirtschaftlichkeitsbetrachtungen angestellt, Verantwortlichkeiten bestimmt und Termine für die Fertigstellung der einzelnen Phasen festgelegt werden.
Test	Anforderungstest Bei großen Softwaresystemen werden die Anforderungen in einer formalisierten Sprache erstellt und können so mit geeigneten Testverfahren automatisch kontrolliert werden. Als formale Sprache kommt im Fall der objektorientierten Programmierung beispielsweise UML (Unified Modeling Language) zum Einsatz, die Methoden für verschiedene Phasen zur Verfügung stellt.
Ergebnis	Pflichtenheft mit eindeutig beschriebenen Anforderungen

Beispiel: Analysephase

Ein Busunternehmen besitzt mehrere Busse und Taxen. Die Fahrzeuge können für beliebige Fahrten bestellt werden. Das Busunternehmen stellt Fahrer und Fahrzeug für die gewünschte Zeit. Das Unternehmen ist stark gewachsen, und der Inhaber möchte für die Planung, Verwaltung und Rechnungslegung eine neue Software einsetzen.

In der Analysephase werden im Beispiel folgende Fragen gestellt und beantwortet:

Wozu soll das Programm dienen?	Zur Planung und Verwaltung der Fahrzeuge einschließlich Rechnungslegung
Welche Informationen sind vorgegeben? Wovon wird ausgegangen?	Anzahl der Busse und Taxen, Anzahl der Fahrer, Mietpreise für einzelnes Fahrzeug inklusive Fahrer, Kundenanfragen
Welche Informationen soll das Programm ausgeben?	Planungstabellen, Angebote, Verträge, Rechnungen, Mahnungen
Wie können diese Informationen ermittelt werden?	Kundenanfragen bzw. Kundendaten müssen erfasst werden. Die Daten für Busse, Taxen und Fahrer enthält eine Datenbank.

Entwurfsphase

Aufgabe	**Wie** ist die Software zu realisieren? ✔ Strukturierter Softwareentwurf, z. B. mithilfe der Top-down-Methode ✔ Beschreibung der Algorithmen mithilfe eines dafür geeigneten Darstellungsmittels (vgl. Kapitel 4) ✔ Festlegung der Programmiersprache für die Implementierung ✔ Festlegung von Programmierstrukturen ✔ Anlegen der entsprechenden Dokumentation
Test	✔ Entwurfstest der Dokumente und des Designs ✔ Funktionstest des Prototyps, falls vorhanden
Ergebnis	Beschreibung des Entwurfs bzw. Softwarespezifikation

Der Schwerpunkt beim Entwurf liegt auf der Bestimmung der Aufgabe der einzelnen Module sowie auf dem Zusammenspiel und der Kommunikation der Module untereinander. Je nach festgelegter Programmiersprache kann sich die Vorgehensweise bei der Fortführung des Entwurfs ändern.

Den Entwurf kann man in drei verschiedene Phasen unterteilen:

- ✔ Grobentwurf
- ✔ Feinentwurf
- ✔ Implementationsentwurf

Ein guter Entwurf zahlt sich langfristig aus und verringert die Entwicklungszeit.

Beim **Grobentwurf** werden die Hauptmodule eines Programms festgelegt und ihre Beziehungen untereinander bestimmt und dokumentiert. Dabei ist es wichtig, dass die einzelnen Module bereits so ausgearbeitet werden, dass ihre Schnittstellen zur Wiederverwendung im Verlauf der weiteren Entwicklung nicht mehr geändert werden müssen.

Im **Feinentwurf** werden die Module des Grobentwurfs weiter untergliedert. Die Zerlegung wird so lange vorangetrieben, bis die Komplexität und die Größe aller Module so weit reduziert sind, dass sie präzise und ohne überflüssige Redundanz (mehrfache Speicherung von Informationen) definiert werden können.

Weiterhin werden beim Feinentwurf die Struktur der Daten und die benötigten Anweisungen präzisiert, und der Aspekt der späteren Wiederverwendung wird mit einbezogen. Das Ergebnis des Feinentwurfs wird in der implementationsunabhängigen Softwarespezifikation festgehalten.

Mit dem Ergebnis des Implementationsentwurfs liegt die Softwarespezifikation vor. In der nachfolgenden Implementierung werden die Module in eine Programmiersprache umgesetzt.

Beispiel: Entwurfsphase

Wie könnte der Grobentwurf für das Busunternehmen aussehen?

Welche Informationen sind vorgegeben?	Kundendaten, Mietpreis, verfügbare Busse, Taxen und Fahrer
Was soll ausgegeben werden?	Planungstabellen, Angebote, Verträge, Rechnungen, Mahnungen
Welche Funktionen sind dafür nötig?	✔ Eingabemaske für die Erfassung der Kundendaten ✔ Eingabemaske für die Busse und Taxen ✔ Eingabemaske für die Fahrer ✔ Anzeigen der verfügbaren (freien) sowie der ausgebuchten Fahrzeuge und Fahrer ✔ Anzeigen der Kundendaten ✔ Planungsfunktion: Zuordnen von Fahrer, Fahrzeug und Kunde für entsprechende Zeit ✔ Abrechnungsfunktion für durchgeführte Einsätze ✔ Druckfunktion für Kundendaten, Angebote, Verträge, Fahrzeuge und Fahrer (Fahrauftrag), Rechnungen, Mahnungen
Welche zusätzlichen Funktionen sollen eingebaut werden?	✔ Druckfunktion für Grußkarten zu Weihnachten und Ostern an die Kunden ✔ Fehlermeldung für ungültige Eingaben (z. B. unvollständige Adressen)

Die Aufgaben können nun auf mehrere Programmierer verteilt werden. So kann beispielsweise ein Mitarbeiter die Eingabemasken erstellen und ein weiterer Mitarbeiter sich um die Ausgaben kümmern.

Implementierungsphase

Aufgabe	✔ Programmcode in einer Programmiersprache erstellen und dokumentieren ✔ Programmcode testen und dokumentieren
Test	Funktionstest/Modultest
Ergebnis	✔ Programmcode ✔ Dokumentation zum Programmcode ✔ Dokumentation zum Test des Programmcodes

Diese Phase wird **Codierung** genannt. In dieser Phase wird der Programmcode, auch Quelltext oder Quellcode (engl. source code) genannt, erstellt. Dazu werden die Module der Entwurfsphase in einer bestimmten Programmiersprache implementiert.

Bei den Tests wird zwischen sogenannten Whitebox- und Blackbox-Tests unterschieden.

- Whitebox-Tests prüfen die innere Funktionsweise von Komponenten, z. B. Modulen.
- Im Gegensatz dazu wird das Innere einer Komponente beim Blackbox-Test nicht betrachtet, sondern das Zusammenspiel der einzelnen Komponenten gemäß der Spezifikation.

Entgegen vielen Erwartungen beansprucht die Phase der Implementierung in der Regel den geringsten Anteil der Gesamtentwicklungszeit.

Integrationsphase

Aufgabe	✔ Zusammenfügen der Einzelaufgaben zur Gesamtaufgabe ✔ Dokumentation
Test	✔ Integrationstest, testet das Zusammenspiel der Einzelteile ✔ Systemtest, der gewährleisten soll, dass das Endprodukt die festgelegten Anforderungen des Pflichtenhefts erfüllt
Ergebnis	Komplettes System und Dokumentation für den Benutzer

Treten während der Integrationsphase Fehler auf, wird wieder die Implementierungsphase mit nochmaligen Tests durchlaufen. Dieser „Kreislauf“ wird so lange wiederholt, bis alle Tests fehlerfrei durchgeführt werden konnten und somit gravierende Fehler der Software ausgeschlossen werden können.

Einsatz und Wartung

Diese Phasen gehören nicht mehr zum eigentlichen Entwicklungsprozess, sondern umfassen den Zeitraum von der ersten Installation beim Auftraggeber bis zum Einsatzende der Software.

Einsatzphase (Installation)

Aufgabe	✔ Software für die Auslieferung fertigstellen ✔ Beenden der Dokumentation
Test	Akzeptanztest/Abnahmetest – Software wird durch den Kunden geprüft
Ergebnis	Software im Einsatz

Wartungsphase

Aufgabe	✔ Korrektur auftretender Fehler ✔ Anpassung an die Systemumgebung ✔ Änderung oder Erweiterung von Funktionen
Test	Konfigurationstest, prüft z. B. Software nach einer Anpassung
Ergebnis	Änderung oder Erweiterung der Funktionen

Bei fehlender Dokumentation und ungenügender Weitsicht für die Einsatzdauer eines Programms können die Kosten für die Wartung einer Software schnell ansteigen.

13.4 Vorgehensmodelle im Überblick

Für die Darstellung des Software-Lebenszyklus gibt es verschiedene Vorgehensmodelle. Sie unterscheiden sich in:

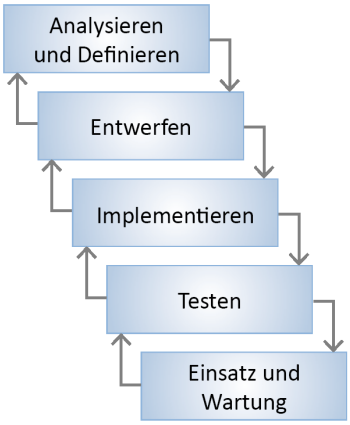
- den Beziehungen zwischen den einzelnen Phasen,
- der Anordnung der einzelnen Phasen,
- der Art und dem Inhalt der einzelnen Phasen,
- dem betrachteten Projektumfang.

Nachfolgend lernen Sie einige Vorgehensmodelle kennen:

- das Wasserfallmodell
- das V-Modell
- das Prototyping-Modell
- das Spiralmodell
- agile Modelle

Das Wasserfallmodell

Das Wasserfallmodell ist ein lineares Modell. Die Phasen sind stufenartig angeordnet. Am Ende jeder Phase steht ein Teilprodukt, das in der nächsten Phase weiterentwickelt wird. Das Modell ist streng sequenziell. Jede Phase besitzt nur eine Rückkopplung zur vorherigen Phase. Diese dient dazu, die bei der Weiterentwicklung gefundenen Schwachstellen aus den vorangegangenen Phasen zu beseitigen. Der Name „Wasserfallmodell“ beruht darauf, dass die Ergebnisse jeder Stufe wasserfallartig auf die nächste Stufe „fallen“.

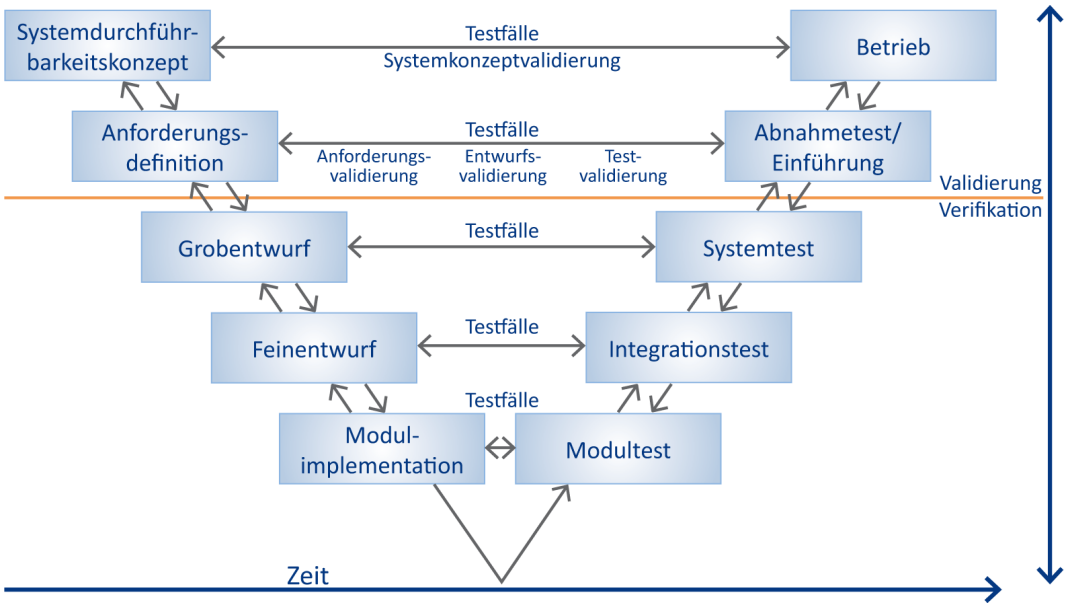


Die im Wasserfallmodell beschriebene Struktur wurde früher in vielen Projekten verwendet, hat aber durch die mangelnde Flexibilität mittlerweile nicht mehr die Bedeutung vergangener Zeiten. Dennoch gibt es die Vorgehensweise gerade in großen Firmen noch häufiger.

Vorteile	Nachteile
✔ Geringer Managementaufwand ✔ Leicht verständlich	✔ Späte Änderungen sind nur mit hohem Aufwand realisierbar ✔ Sehr unflexibel durch die strenge Sequenz

Das V-Modell

Das V-Modell (nach Boehm) kann als eine Weiterentwicklung des Wasserfallmodells betrachtet werden. Die V-Form entsteht dadurch, dass die Phase, die jeweils weiter ins Detail geht, treppenartig versetzt unter der vorhergehenden Phase angeordnet wird. Die Testphase verläuft vom Test des Details hin zum gesamten System – also wieder nach oben. Die Phase und die zugehörige Testphase können als eine Hierarchieebene betrachtet werden. In das Modell sind Validierung (Gültigkeitsprüfung) und Verifikation (Nachweis der Richtigkeit/Korrektheit) integriert.



Das V-Modell besteht aus den folgenden Submodellen (die aber nicht Bestandteil dieses Buchs sind):

- Systemerstellung
- Qualitätssicherung
- Konfigurationsmanagement
- Projektmanagement

Für die Anwendung des V-Modells werden CASE-Werkzeuge (vgl. Abschnitt 13.5) eingesetzt, da es sonst aufgrund seines Umfangs nur schwer zu handhaben ist.

Vorteile	Nachteile
✔ Gesamtmodell (umfassendes Modell) ✔ Gut geeignet für große Projekte ✔ Qualitätssicherung steht im Vordergrund ✔ Kann angepasst und erweitert werden	✔ Vorgehensweisen sind sehr allgemein ✔ Für kleine Projekte ungeeignet (zu viele Zwischen-produkte) ✔ Sehr aufwändige Verwaltung ✔ CASE-Werkzeuge für die Entwicklung erforderlich

Das Prototyping-Modell

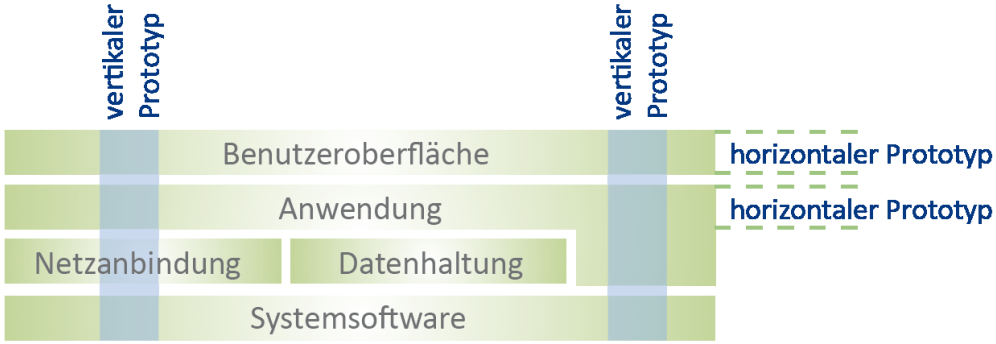
Prototypen sind ablauffähige, aber nicht voll funktionstüchtige Modelle. Sie lassen sich relativ schnell entwickeln und ermöglichen somit eine Überprüfung der Entwürfe. Auch zum Experimentieren lassen sich Prototypen verwenden. Ein für gut befundener Prototyp kann als frühe Produktversion angesehen und inkrementell (zur nächsten Version) weiterentwickelt werden. Das Prototyping-Modell wird angewendet, um folgende Ziele zu erreichen:

- Klare Definition der Anforderungen
- Klärung von Problemen vor der Erstellung eines Softwareprodukts
- Einbeziehung der Benutzer in die Entwicklung
- Überprüfung und Sicherung der Realisierbarkeit des Projekts
- Aufzeigen verschiedener Lösungsmöglichkeiten

Es gibt verschiedenen Arten von Prototypen:

- Demonstrations-Prototyp: wird meist für die Akquisition von Kunden entwickelt, die darüber bereits einen ersten Eindruck von dem späteren Produkt gewinnen (Rapid Prototyping)
- Labor-Prototyp: wird zur Klärung konstruktionsbezogener und technischer Fragen entwickelt
- Pilot-Prototyp: bildet den Kern eines Systems, das zum Produkt weiterentwickelt werden kann

Bei der Entwicklung von Prototypen wird unterschieden, ob es sich um einen vertikalen oder einen horizontalen Prototyp handelt. Bei horizontalen Prototypen wird eine Ebene vollständig implementiert, aber ohne Funktionalität. Ein vertikaler Prototyp besitzt die vollständige Funktionalität ausgewählter Teile des Systems.



Vorteile	Nachteile
✔ Entwicklungsrisiko wird reduziert ✔ Prototypen lassen sich in andere Vorgehensmodelle integrieren ✔ Lösungsalternativen werden aufgezeigt ✔ Einbeziehung des Benutzers	✔ Mitunter zusätzliche Entwicklung und damit erhöhter Entwicklungsaufwand ✔ Häufig keine klaren Beschränkungen für Prototypen

Das Spiralmodell

Das Spiralmodell vereint das klassische lineare Modell (Wasserfallmodell) und evolutionäre Modelle (z. B. Prototyping). Dieses Modell findet heutzutage in sehr vielen Projekten Verwendung.

Die Softwareentwicklung mithilfe des Spiralmodells läuft in vier Schritten ab, die so lange wiederholt werden, bis das Softwareprodukt fertiggestellt ist.

① Definition von Zielen, Alternativen und Nebenbedingungen des neuen Spiralprozesses

- Zu Beginn jeder Phase (jeder neuen Windung der Spirale) werden inhaltliche Vorgaben für das Teilprodukt definiert.
- Es werden alternative Vorgehensweisen ausgearbeitet.
- Die Randbedingungen (Nebenbedingungen), wie z. B. Personal, Finanzen und Zeit, sind festzulegen.

② Evaluieren der Ziele und Alternativen, Risiken erkennen und reduzieren

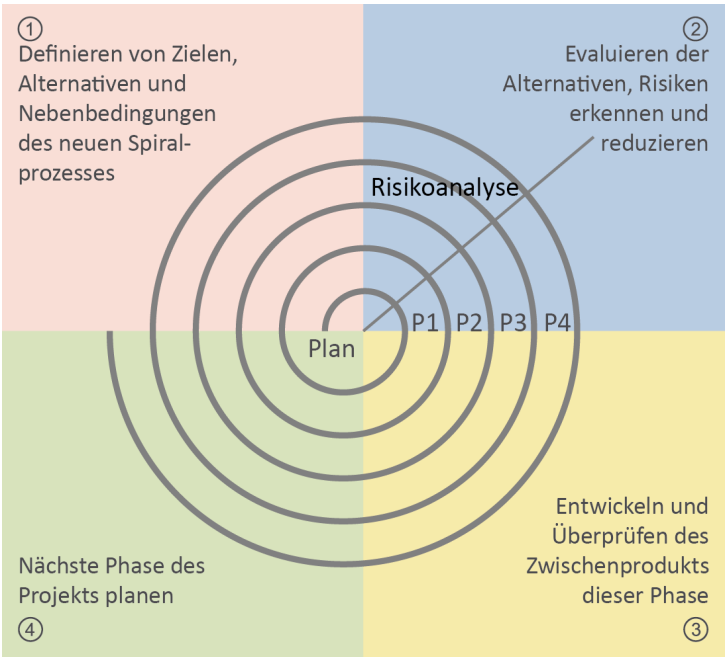
- Zu Beginn des zweiten Schrittes werden die Alternativen unter Berücksichtigung der Ziele und Randbedingungen evaluiert.
- Risikofaktoren werden benannt und so weit wie möglich reduziert.
- Es können dazu unterschiedliche Techniken verwendet werden (z. B. Entwicklung von Prototypen). Das Ergebnis ist der Prototyp (P1, P2 ...).

③ Entwickeln und Überprüfen des Zwischenprodukts dieser Phase

- In diesem Schritt wird das Teilprodukt (z. B. einzelne Module oder in der letzten Phase die komplette Software) unter Einhaltung des Ziels und der geplanten Ressourcen realisiert und getestet.
- Die Methode für die Realisierung kann flexibel unter Beachtung des Restrisikos gewählt werden.
- Die fertig getesteten Module sind beispielsweise Ergebnis dieses Schrittes.

④ Nächste Phase des Projekts planen

- Als letzter Schritt einer Phase wird die nächste Phase inhaltlich und organisatorisch geplant.
- Diese Planung kann mehrere Phasen umfassen, sodass unabhängige Teilprojekte entstehen. Diese werden zu einem späteren Zeitpunkt wieder zusammengeführt.
- In einem **Review** (Prüfung) werden die Schritte 1 bis 3 analysiert, und es werden Schlussfolgerungen für die weitere Entwicklung gezogen. Sind die technischen oder wirtschaftlichen Risiken einer Projektfortsetzung zu hoch, kann das Projekt abgebrochen werden.
- Am Ende der Spirale liegt die fertige Software vor, die bezüglich der Anfangshypothese getestet wird.



Das Spiralmodell

Vorteile	Nachteile
✔ Fehler werden frühzeitig erkannt. ✔ Zwischenprodukte werden regelmäßig überprüft. ✔ Flexibel (Änderungen sind leicht möglich.) ✔ Alternativen werden betrachtet. ✔ Risiken werden minimiert.	✔ Hoher Managementaufwand ✔ Mitarbeiter benötigen bei Anwendung einer Risikoabschätzung Kenntnisse der Risikoanalyse.

Agile Modelle

Klassische Vorgehensmodelle zeigen in verschiedenen Situationen im Praxiseinsatz auch Schwächen: falls zu Beginn der Softwareerstellung nicht genau bekannt ist, was zu erstellen ist, z. B. wegen der Ungewissheit unternehmenspolitischer Entscheidungen, oder falls sich während der Erstellung der Software Änderungen ergeben, z. B. Markteinbrüche.

Um diese Probleme zu umgehen, wurden sogenannte **agile Modelle** entwickelt. Diese Vorgehensmodelle versuchen mit flexiblen Prozessen, Software in kurzer Zeit in kleinen Teams unter Einbeziehung des Kunden zu entwickeln. Die Software kann bei Bedarf nach und nach ausgebaut werden.

Agile Programmierung stellt folgende Leitlinien in den Fokus:

- Individuen und Interaktion bzw. Kommunikation sind wichtiger als Prozesse und Werkzeuge.
- Lauffähige Software ist wichtiger als umfangreiche Dokumentation.
- Teamwork und Zusammenarbeit mit dem Kunden ist wichtiger als Vertragsverhandlungen.
- Auf Änderungen reagieren ist wichtiger, als einem vorher festgelegten Plan zu folgen.

Positive Eigenschaften agiler Methoden sind folgende Kriterien:

- Einfach einzuführen und zu kontrollieren, wenig Administration, wenig Dokumente
- Schnelle Auslieferung von Funktionen, Analyse- und Designphase im Lauf des Projekts
- Informelle Kommunikation und gemeinsame Entscheidungen

Es gibt aber auch einige Nachteile:

- Die Ergebnisse hängen stark von individuellen Fähigkeiten und Motivationen sowie der Disziplin der Mitarbeiter ab.
- Durch fehlende Strukturen kann es bei großen, sicherheitskritischen Projekten Probleme geben.
- Das Einbeziehen externer Ressourcen und Mitarbeiter ist schwierig, da wenige formale Rahmen vorhanden sind.
- Es besteht durch die fehlenden formalen Rahmen die Gefahr von Mehraufwand durch missverständliche Kommunikation oder verspätete Entscheidungen.

13.5 Computergestützte Softwareentwicklung (CASE)

Was ist CASE?

CASE ist die Abkürzung für Computer Aided Software Engineering – computergestützte Softwareentwicklung. Bei der Entwicklung von Software werden computergestützte Hilfsmittel eingesetzt mit dem Ziel, die Produktivität und die Qualität der Software zu verbessern und das Management zu unterstützen. Die computergestützten Hilfsmittel – die CASE-Plattform – bestehen aus einer CASE-Umgebung und CASE-Werkzeugen, die in die CASE-Umgebung integriert sind. CASE-Werkzeuge sind z. B.

- grafische Editoren, in denen Sie Ihre Softwareprojekte mithilfe verschiedener Vorgehensmodelle gliedern und bearbeiten können;
- Testfunktionen, um erzeugte Modelle auf Richtigkeit zu überprüfen;
- die automatische Quelltexterstellung in verschiedenen Programmiersprachen.

Beispiele für CASE-Werkzeuge:

- Eclipse (Eclipse Foundation)
- VisualStudio (Microsoft)
- PsPad
- Modellierungsprogramme zum Erstellen von Diagrammen

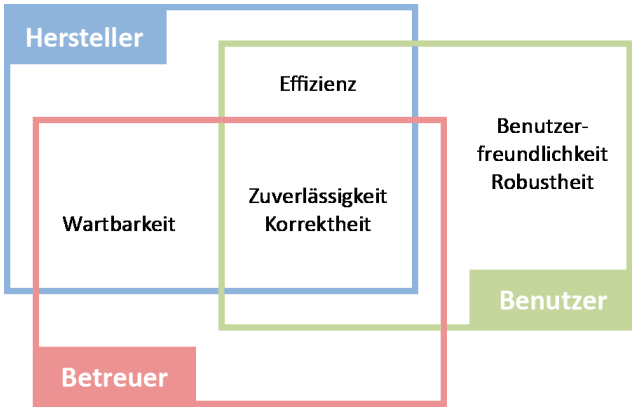
13.6 Qualitätskriterien

An Software werden hohe Qualitätsansprüche gestellt. Die Software muss dabei den Anforderungen zweier Interessengruppen entsprechen – denen der Hersteller und denen der Kunden. Zur Erfüllung von Qualitätsanforderungen werden Qualitätskriterien festgelegt.

Der Kunde achtet bei der Software z. B. auf Benutzerfreundlichkeit und auf den Kundendienst. Für ihn ist es wichtig, dass die Software zum bestellten Termin und zum vereinbarten Preis ausgeliefert wird. Außerdem spielt der Einarbeitungsaufwand eine große Rolle.

Dem Hersteller dagegen ist es wichtig, bei der Entwicklung und der nachfolgenden Wartung rentabel zu arbeiten. Sobald das Produkt ausgeliefert ist, geht die Verantwortung vom Hersteller auf den Betreuer der Software, meistens den Hersteller selbst, über. Dieser übernimmt die Wartung der Software, z. B. in Form des Kundendienstes.

Die nebenstehende Abbildung zeigt die Verbindung von Qualitätskriterien.



Korrektheit, Robustheit, Zuverlässigkeit

Die **Korrektheit** stellt bei allen Programmen ein Kriterium dar. Ein Programm ist dann korrekt, wenn es ein vorgegebenes Problem fehlerfrei löst. Die Beurteilung der Korrektheit basiert auf Anforderungen an das Programm, der Produktdefinition (oder dem Pflichtenheft). Die Problematik innerhalb dieses Kriteriums besteht darin, dass Sie nicht entscheiden können, ob ein Programm 100-prozentig korrekt ist. Der vollständige Test aller möglichen Programmzustände ist schon für kleine Programme nur mit erheblichem Aufwand durchführbar. Beim Programmieren sehr umfangreicher Programme scheitert jedoch dieses Verfahren.

Mathematisch gesehen können Sie meist entscheiden, ob ein Programm (die verwendeten Algorithmen etc.) 100-prozentig korrekt ist. Da die Ausführung von Software aber auch noch von anderen Bedingungen abhängt (Hardware, Betriebssystem), kann nicht davon ausgegangen werden, dass sie in Bezug auf diese Bedingungen korrekt ist (Anpassungen sind notwendig). Korrektheit ist somit eher ein theoretisches Maß.

Eine Software ist **robust**, wenn sie beispielsweise bei falschen Eingaben sinnvoll reagiert und bei Eingabefehlern nicht abstürzt. Dies kann von einer einfachen Fehlermeldung bis zu einer automatischen Fehlerkorrektur reichen.

Ein robustes Programm stürzt nicht ab, sondern reagiert auf mögliche Fehler durch eine geeignete Fehlerbehandlung.

Die **Zuverlässigkeit** ist das wichtigste Kriterium von Software bezüglich ihrer Fehlerhaftigkeit. Es kann davon ausgegangen werden, dass jede Software Fehler enthält und diese auch nicht auszuschließen sind. Um zu entscheiden, wie zuverlässig ein Programm ist, müssten die folgenden vier Fragen beantwortet werden:

- Wie oft tritt ein und derselbe Fehler auf?
- Wie viele unterschiedliche Fehler gibt es?
- Sind es Fehler, die bei der Programmierung hätten entdeckt werden müssen? Oder sind es selten auftretende Fehler, die nur schwer zu lokalisieren sind?
- Führt der Fehler zum Systemabsturz, Verlust von Daten oder zu einem falschen Ergebnis?

Software ist zuverlässig, wenn selten Fehler auftreten und diese nur geringe Auswirkungen haben. Zuverlässige Programme sind auch stets robust.

Benutzerfreundlichkeit

Ist eine Software sowohl von erfahrenen als auch unerfahrenen Benutzern einfach zu bedienen, ist sie **benutzerfreundlich**. Wesentliche Aspekte dabei sind die Entwicklung von Benutzeroberflächen nach **softwareergonomischen Kriterien**, wie beispielsweise der übersichtliche Aufbau und der Inhalt der Bildschirmfenster. Hinweise zur Software-Ergonomie finden Sie in zahlreichen Publikationen. Die ständig verfügbare Hilfefunktion und die Möglichkeit, den Umfang von Funktionen an den Kenntnisstand des Nutzers anzupassen, sind weitere wichtige Gesichtspunkte.

Effizienz

Bei der **Effizienz** (Wirksamkeit und Wirtschaftlichkeit) wird zwischen der Laufzeit- und der Speicherplatzeffizienz unterschieden. Die Grundlage für effiziente Programme sind leistungsfähige Algorithmen und deren Verwendung in der jeweiligen Anwendung. Konzentrieren Sie sich während der Programmierung jedoch zu sehr auf diesen Gesichtspunkt, kann dies zulasten der Übersichtlichkeit gehen. Deshalb sollten Sie bereits eine Vorauswahl für effiziente Algorithmen treffen und diese so implementieren, dass sie einfach durch effizientere austauschbar sind.

Bei der Softwareentwicklung orientieren Sie sich zunächst meist an den anderen Qualitätskriterien. Erst wenn bei Tests der Software eine mangelnde Effizienz festzustellen ist, wird versucht, durch gezieltes Austauschen der Algorithmen eine Verbesserung zu erreichen.

Wartbarkeit

Immens wichtig für die Hersteller von Software ist die **Wartbarkeit**. Wartbarkeit umfasst den Aufwand bei der Fehlersuche, der Fehlerkorrektur und bei funktionalen Erweiterungen, der sich in den anfallenden Kosten widerspiegelt. Für den Benutzer spielt dieses Kriterium in Bezug auf die Softwarequalität jedoch keine direkte Rolle.

Dokumentation

Softwareprogramme sollten gut dokumentiert werden. Dabei gibt es verschiedene Faktoren, die zu erfassen sind:

- Benutzerschnittstelle zum Anwender
- Systemschnittstelle zu anderen Produkten
- Kommentierter Quelltext für den Hersteller
- **Dokumentation** für die Anwender und den Hersteller

Bei der Dokumentation für eine Software werden zwei Arten unterschieden:

- Systemdokumentation
- Benutzerdokumentation

Systemdokumentation

Sie beschreibt die Eigenschaften der Software vom Beginn der Entwicklung bis zur letzten Version. Die Systemdokumentation enthält die Definition des Produkts, die Spezifikation der Software, die exakten Beschreibungen der Strukturen. Diese Dokumentation ist sehr detailliert beschrieben und bildet die Grundlage für die Wartung der entsprechenden Software.

Benutzerdokumentation

Hier findet der Anwender ein Handbuch zur Einführung in die Softwarenutzung sowie ein komplettes Nachschlagewerk zu allen angebotenen Funktionen. Sollte es notwendig sein, muss ein gesondertes Handbuch für den Systemadministrator, beispielsweise für die Systeminstallation, in der Benutzerdokumentation integriert sein. Die Nutzungsbedingungen sind ebenso einzufügen wie die Angaben zu den Systemvoraussetzungen, unter denen das Programm effizient arbeitet. Angaben zu den Autoren der Dokumentation bzw. dem Hersteller der Software sind ebenfalls Bestandteil einer guten Dokumentation.

Eine Dokumentation muss klar verständlich sein, der aktuellen Programmversion entsprechen, einen logischen Aufbau besitzen und sachlich präzise geschrieben sein. Ungebräuchliche Bezeichnungen sind in einer Dokumentation zu vermeiden bzw. sollten bei Gebrauch näher erläutert werden.

Zunehmende Qualitätsanforderungen

Software wird heute in vielen Bereichen eingesetzt, in denen ein Ausfall kritisch ist. Softwarefehler können verheerende Folgen haben. In sensiblen Bereichen steht die Forderung nach der Null-Fehler-Software. Trotz Anwendung wissenschaftlicher Ergebnisse bei der Qualitätssicherung können diese Anforderungen (noch) nicht erfüllt werden. Das ergibt sich auch aus der Tatsache, dass umfassende Tests neu entwickelter Software mitunter sehr schwierig und teuer sind.

13.7 Schnellübersicht

Was bedeutet …?	
Modul/Komponente	Dies ist ein abgeschlossener Baustein einer Software. Eine Komponente kann auch ein eigenständiges Programm sein.
Software-Lebenszyklus	Gliedert die Entwicklung der Software in einzelne Phasen. In jeder Phase wird eine bestimmte Aufgabe erledigt, getestet und dokumentiert.
Vorgehensmodell	Modell für den Planungs- und Entwicklungsprozess von Software
CASE	**C**omputer **A**ided **S**oftware **E**ngineering, computergestützter Softwareentwurf mit CASE-Werkzeugen

13.8 Übung

Fragen zur Softwareentwicklung

Übungsdatei: -- **Ergebnisdatei: *uebung13.pdf***

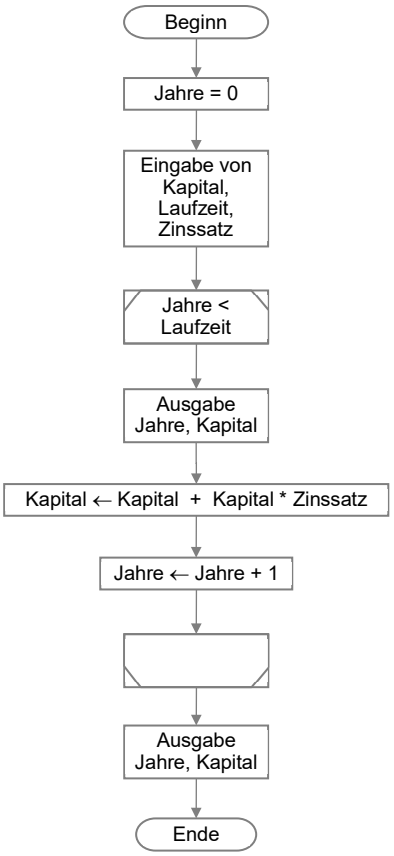
1. Benennen Sie die einzelnen Teilphasen des Software-Lebenszyklus.
2. Welche Angaben sind in einem Pflichtenheft festzuhalten?
3. Welche Vorgehensmodelle kennen Sie? Wodurch unterscheiden sie sich?
4. Welche Arten von Dokumentationen für Software gibt es? Warum ist die Dokumentation von so großer Bedeutung?
5. Was bedeutet CASE, und wozu dient es?

Anhang A: PAP, Struktogramm und Pseudocode

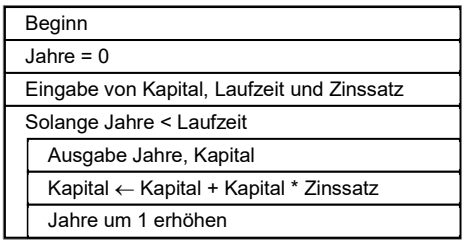
A.1 Beispiel Zinsberechnung

Bei Algorithmen wie der Zinsberechnung werden Schleifen eingesetzt. Das Beispiel zeigt die Zinsberechnung als PAP, Struktogramm und im Pseudocode.

Programmablaufplan



Struktogramm



Pseudocode

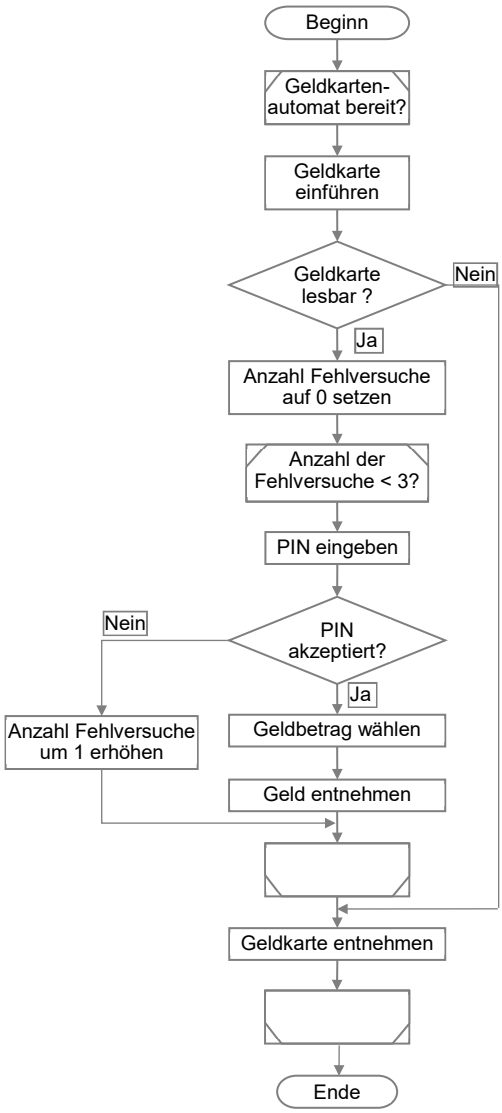
```
begin Zinsberechnung
  Jahre := 0;
  Eingabe(Kapital, Laufzeit, Zinssatz);
  while Jahre < Laufzeit
  do
    Ausgabe(Jahre, Kapital);
    Kapital := Kapital + Kapital * Zinssatz;
    Jahre := Jahre + 1;
  end do
  Ausgabe(Jahre, Kapital);
end Zinsberechnung
```

Die Variable `Jahre`, die im Ablauf des Programms für die Zählung der Jahre benötigt wird, erhält den Initialwert `0`. Der Benutzer gibt das Anfangskapital, die Laufzeit und den Zinssatz ein. Diese Größen werden für die Berechnung benötigt. In einer Schleife erfolgt die Ausgabe des aktuellen Jahres und Kapitals. Das Kapital wird neu berechnet. Es ergibt sich aus dem bisherigen Kapital zuzüglich der Zinsen, die sich aus Kapital mal Zinssatz errechnen. Danach wird das Jahr um 1 weitergezählt. Für jedes Jahr erfolgt ein Schleifendurchlauf. Die Schleife wird so lange immer wieder abgearbeitet, bis die Anzahl der Jahre die Laufzeit erreicht hat. Danach erfolgt die Ausgabe der endgültigen Ergebnisse der Berechnung, und das Programm wird beendet.

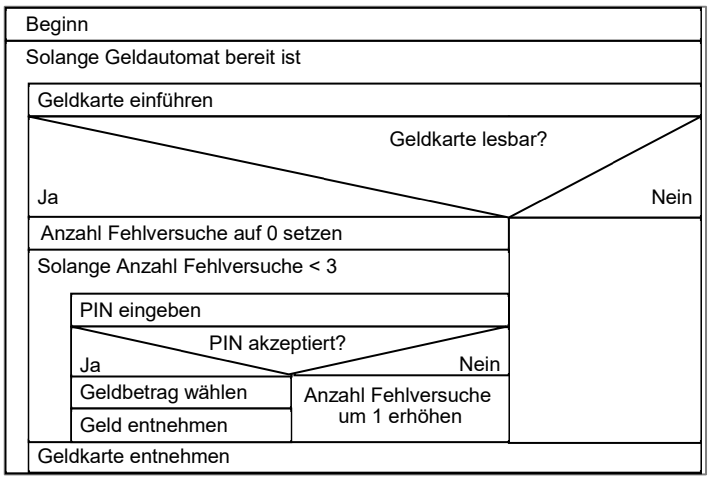
A.2 Beispiel Geldautomat

Der Vorgang zum Benutzen eines Geldautomaten (stark vereinfacht) ist im Folgenden als PAP, Struktogramm und im Pseudocode dargestellt.

Programmablaufplan



Struktogramm



Pseudocode

```
begin Geldautomat
  while Geldautomat bereit
  do
    Karte einführen;
    if Karte lesbar
    then
      Fehlversuche := 0;
      while Fehlversuche < 3
        Einlesen(PIN);
        if PIN = PIN der Karte then
          Eingabe(Geldbetrag);
          Geld auswerfen;
        else
          Fehlversuche := Fehlversuche + 1;
        end if
      end do
    end if
    Geldkarte auswerfen;
  end do
end Geldautomat
```

Wenn der Geldautomat eingeschaltet ist, kann er immer wieder Geldkarten einlesen. Ist die Geldkarte nicht lesbar oder wurde bereits dreimal die falsche PIN eingegeben, wird sie wieder ausgeworfen. Sie können die Karte entnehmen und es wiederholt versuchen, solange der Geldkartenautomat bereit ist. Akzeptiert der Automat die Karte, wird die Anzahl der Fehlversuche auf 0 gesetzt, dann geben Sie die PIN ein. Ist die PIN richtig, können Sie den Geldbetrag wählen und das Geld entnehmen. War die PIN falsch, wird die Anzahl der Fehlversuche um 1 erhöht. Nach drei Fehlversuchen können Sie die Geldkarte entnehmen und es erneut versuchen. Dieses Programm läuft so lange, bis der Geldkartenautomat nicht mehr bereit ist.

Anhang B: Installationen und Quellangaben

B.1 Den Editor PSPad installieren und konfigurieren

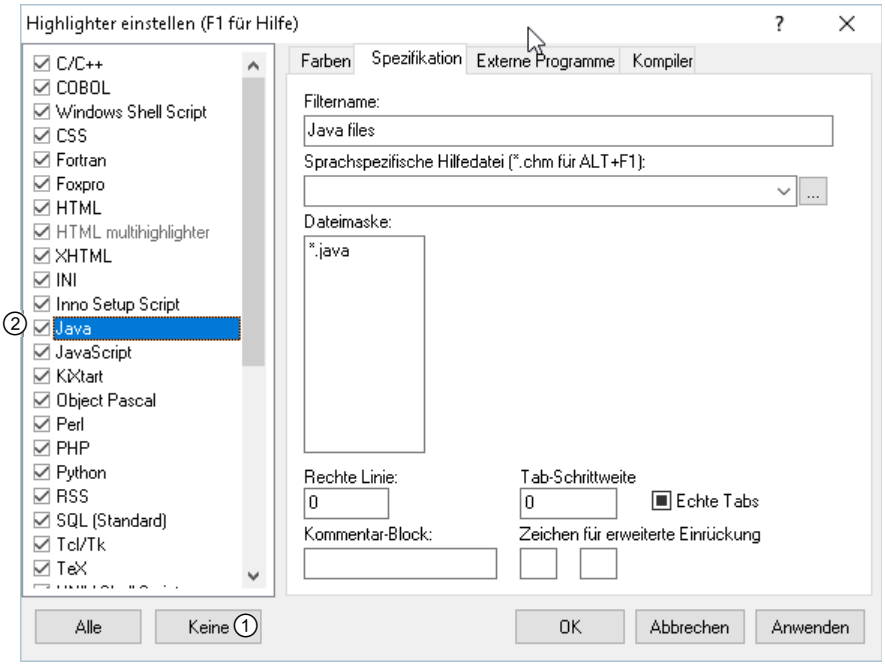
Download und Installation von PSPad

- Laden Sie das Programm PSPad von der Webseite *http://www.pspad.com/de/*.
- Um PSPad zu installieren, folgen Sie den Schritten des Installationsassistenten.
- Starten Sie nach erfolgreicher Installation das Programm.

Editor-Einstellungen verändern

Über das Menü *Einstellungen - Highlighter einstellen* können Sie die Einstellung für das Einrücken der Programmzeilen vornehmen. Damit lässt sich die Struktur besser erkennen.

- Deaktivieren Sie mit der Schaltfläche *Keine* ① alle Kontrollfelder im linken Bereich und aktivieren Sie anschließend das Kontrollfeld *Java* ②.
- Aktivieren Sie das Register *Spezifikation*.
- Wählen Sie als Tab-Schrittweite 2.
- Bestätigen Sie die Eingaben mit *OK*.



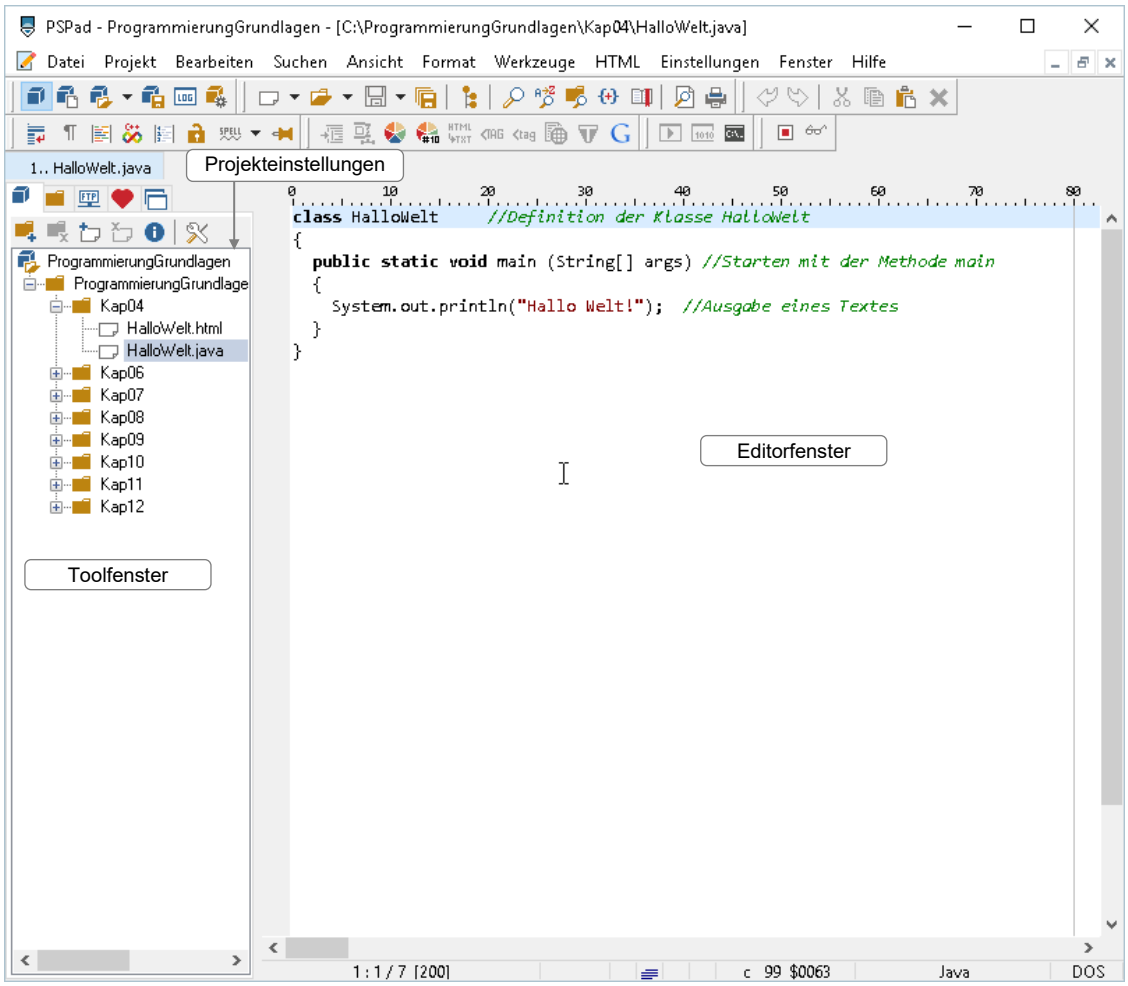
Einstellung für Java im Register „Spezifikation“ in PSPad

Mit Projekten arbeiten

Um beispielsweise auf alle Beispieldateien des Buchs schnell Zugriff zu haben, können Sie im Editor ein neues Projekt für diese Dateien öffnen.

- Rufen Sie im Menü *Projekt* den Menüpunkt *Projekt aus Verzeichnis erstellen* auf, und öffnen Sie z. B. den Ordner *C:\ProgrammierungGrundlagen.*

Im Toolfenster werden die Ordner und Dateien des Projektes angezeigt. Mit einem Doppelklick auf eine Java-Datei im Toolfenster können Sie den Programmcode der Datei im Editorfenster öffnen.



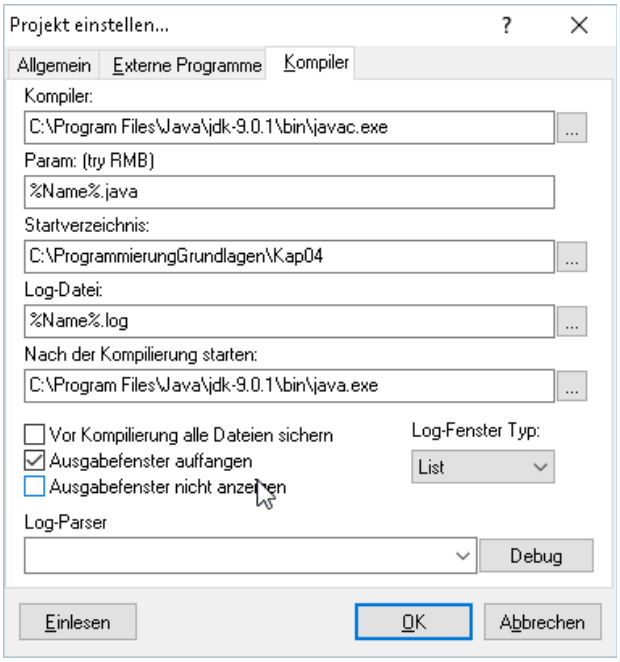
Projekt mit geöffneter Java-Datei "HalloWelt.java" in PSPad

Damit Sie Programmzeilen schneller finden, können Sie im Menü *Ansicht* mit dem Befehl *Zeilennummerierung* die Anzeige der Zeilennummern einschalten.

Kompiler-Einstellungen

- Klicken Sie auf den Menübefehl *Projekt* und dann auf *Projekt einstellen…*, um Einstellungen für das Projekt vornehmen zu können.
- Aktivieren Sie das Register *Kompiler* und nehmen Sie die Einstellungen entsprechend der nebenstehenden Abbildung vor. Den Platzhalter *%Name%* können Sie jeweils über das Kontextmenü einfügen.

Starten Sie für eine Datei mit dem Menüpunkt *Datei - Kompilieren* die Übersetzung einer Java-Datei, wird automatisch im Anschluss die Ausführung gestartet unter der Voraussetzung, dass die Kompilierung fehlerfrei beendet werden konnte.



Einstellungen für Kompilierung und Ausführung

B.2 Quellangaben im Internet

Android Studio	*https://developer.android.com/sdk/index.html*
Anleitung zur Erstellung von App unter Android	*http://www.app-entwickler-verzeichnis.de/apps-programmierung/24-android/297-android-programmierung-tutorial-der-grosse-android-newbie-guide*
Cordova	*https://cordova.apache.org/*
Dia Diagram Editor	*http://dia-installer.de/*
Diagram Designer	*http://meesoft.logicnet.dk/*
Eclipse	*http://www.eclipse.org*
Hamster-Modell	*http://www.java-hamster-modell.de/*
JDK	*http://www.oracle.com/technetwork/java/javase/downloads/index.html*
PapDesigner	*http://friedrich-folkmann.de/papdesigner/*
PsPad	*http://www.pspad.com/de/*
Scratch	*https://scratch.mit.edu/*
W3C	*http://www.w3.org*

*

<

4

A

B

C

D

N

O

P

Q

R

S

T

U

V

W

Z

Impressum

Matchcode: PG

Autor: Ralph Steyer

Produziert im HERDT-Digitaldruck

4. Ausgabe, Dezember 2017

HERDT-Verlag für Bildungsmedien GmbH
Am Kümmerling 21-25
55294 Bodenheim
Internet: www.herdt.com
E-Mail: info@herdt.com