Dr. Klaus Ringhand, Ingo Patett

Entwickeln und Bereitstellen von Anwendungssystemen für IT-Berufe

Lernfeld 6

4. Auflage

Bestellnummer 225383

westermann

Die in diesem Produkt gemachten Angaben zu Unternehmen (Namen, Internet- und E-Mail-Adressen, Handelsregistereintragungen, Bankverbindungen, Steuer-, Telefon- und Faxnummern und alle weiteren Angaben) sind i. d. R. fiktiv, d. h., sie stehen in keinem Zusammenhang mit einem real existierenden Unternehmen in der dargestellten oder einer ähnlichen Form. Dies gilt auch für alle Kunden, Lieferanten und sonstigen Geschäftspartner der Unternehmen wie z. B. Kreditinstitute, Versicherungsunternehmen und andere Dienstleistungsunternehmen. Ausschließlich zum Zwecke der Authentizität werden die Namen real existierender Unternehmen und z. B. im Fall von Kreditinstituten auch deren IBANs und BICs verwendet.

Die in diesem Werk aufgeführten Internetadressen sind auf dem Stand zum Zeitpunkt der Drucklegung. Die ständige Aktualität der Adressen kann vonseiten des Verlages nicht gewährleistet werden. Darüber hinaus übernimmt der Verlag keine Verantwortung für die Inhalte dieser Seiten.

Druck: westermann druck GmbH, Braunschweig

service@westermann-berufsbildung.de
www.westermann-berufsbildung.de

Bildungshaus Schulbuchverlage Westermann Schroedel Diesterweg Schöningh Winklers GmbH, Postfach 33 20, 38023 Braunschweig

ISBN 978-3-14-**225383**-1

westermann GRUPPE

Vorwort

Das vorliegende Lehrbuch behandelt als dritter Band der IT-Buchreihe das Thema Softwareentwicklung für die Ausbildung in den IT-Berufen.

Alle Autoren der dreibändigen Lehrbuchreihe verwirklichen das Ziel, auf der Basis des Rahmenlehrplans für IT-Berufe handlungs- und geschäftsprozessorientierte Unterrichtsmedien und -hilfen zur Verfügung zu stellen. Die Auszubildenden werden damit für ihre vielfältigen Aufgaben im Berufsleben qualifiziert und erhalten das erforderliche Prüfungswissen.

Alle drei Bände gestalten ihre handlungsorientierten Inhalte auf der Basis von Strukturen und Geschäftsprozessen des Modellunternehmens ACI, einem typischen Systemhaus der IT-Branche, wodurch ein stärkerer Praxisbezug und eine größere Schülernähe angestrebt werden.

Die Einleitung durch Situationsbeschreibungen soll einerseits die Geschäftsprozessorientierung fördern und andererseits eine Verbindung zu den Handlungsschritten der Prüfungsaufgaben schaffen. Zu jedem Kapitel wird das Fachwissen in Verbindung mit Handlungen und kompakt in Wissenscontainern zur Verfügung gestellt, die gleichzeitig als Zusammenfassungen, Nachschlageregister oder für Wiederholungen geeignet sind. Mit den zahlreichen Aufgaben soll der Unterrichtserfolg gesichert werden.

Band 3 bezieht sich auf das Lernfeld 6 der Rahmenrichtlinien für die Ausbildung in den IT-Berufen und erfüllt ebenso die sich aus Lernfeld 5 (Englisch) ergebenden Forderungen an den Unterricht.

Die Anwendungsentwicklung wird von den Auszubildenden im Modellunternehmen am Beispiel eines Webshops demonstriert. Das Projekt „Webshop“ beinhaltet die typischen Arbeitsschritte zur Entwicklung einer Software für betriebswirtschaftliche Aufgaben und vermittelt das notwendige Fachwissen. Als Grundlage dient die im Band 1 „Wirtschafts- und Geschäftsprozesse für IT-Berufe“ vorgestellte Warenwirtschaft „ACI Teach Business Software“. Daraus entsteht mit den technischen Möglichkeiten von Java eine portable und von Laufzeitumgebungen unabhängige neue Anwendung, wobei die in Band 1 erarbeiteten Anforderungen verwendet und fortgeschrieben werden. Zusätzliche Schnittstellen schaffen die Voraussetzung, um daraus ein webbasiertes Warenwirtschaftssystem zu implementieren.

In der aktuellen Auflage wurden die Kapitel „Werkzeuge zur Softwareentwicklung“, „Programmierung in Java und C#“ und „Datenbankanwendungen“ stark überarbeitet. Alle Kapitel vermitteln einen direkten Bezug zum Rahmenlehrplan für die IT-Berufe und zum Stoffkatalog mit den Anforderungen zu den schriftlichen Abschlussprüfungen. Berücksichtigt werden aktuelle Trends der Softwareentwicklung, z. B. Entwurf und Design mithilfe von UML, Programmierung in Java und C# sowie die Anwendung von In-memory-Datenbanken. Die Anwendungsentwicklung berücksichtigt den Einsatz auf mobilen Endgeräten. Befehlsreferenzen für SQL und PHP erleichtern das Lernen und die praktische Anwendung.

Die Autoren danken Herrn Lutz Sattler für seine Korrekturhinweise.

Folgende Markierungen in den Kapiteln dienen der besseren Orientierung:

S	Situationsbeschreibung
G	Geschäftsprozessübersicht
W	Wissenscontainer
Aufgaben	Aufgaben zur Übung und Vertiefung
AH	Hinweis auf Aufgaben im Arbeitsheft

Die Verfasser

Inhaltsverzeichnis

4 Werkzeuge zur Softwareentwicklung

Klassische Werkzeuge zur Modellierung: PAP und Struktogramm → Objektorientierte Modellierung mit UML: Anwendungsfalldiagramm, Klassendiagramm und andere → Objektorientierte Analyse (OOA) und objektorientiertes Design (OOD) am Beispiel → Computersprachen: Historie und Überblick → Entwicklungsumgebung Eclipse → Übersetzer formaler Sprachen: Compiler und Interpreter, Browser

4.1 Überblick zu den Werkzeugen

Die Notwendigkeit einer systematischen Entwicklung von Software wurde bereits herausgearbeitet. In diesem Kapitel sollen die Werkzeuge zur systematischen Entwicklung von Software unter folgenden Schwerpunkten vorgestellt werden:

- **Werkzeuge zur Modellierung.** Es werden verschiedene Modelle zur Darstellung der Ergebnisse des Entwurfsprozesses besprochen, wobei UML den Schwerpunkt bildet.
- **Methoden zur Strukturierung.** Das objektorientierte Herangehen von der Analyse über den Entwurf bis hin zur Programmierung bildet eine wesentliche Grundlage erfolgreicher Softwareentwicklung.
- In der **Entwicklungsumgebung** werden die Anweisungen an den Computer als Quelltext erfasst, gestaltet, kontrolliert und im Rahmen der Tests auch ausgeführt.
- Die **Geschichte der Computersprachen** mit den Übergängen von den prozeduralen zu den objektorientierten Sprachen sowie dem Aufkommen deskriptiver Sprachen verdeutlicht die Fortschritte in der Technologie der Softwareentwicklung.
- Die **Übersetzer der Computersprachen** beenden den Überblick zu den Werkzeugen, denn schließlich bringen sie die entwickelte Software in Funktion.

Bei den Werkzeugen zur Entwicklung von Software fällt das Augenmerk wahrscheinlich zuerst auf die Programmiersprachen, wobei man besser den Begriff **Computersprachen** verwenden sollte. Es handelt sich um formale und damit maschinenverständlichen Sprachen, die der Übermittlung von Anweisungen an den Computer dienen.

Sprachen sind Werkzeuge zur Kommunikation. Der Mensch schreibt seine Anweisungen an den Computer als Programm. In der heutigen Zeit, die vom „Internet of Things“ gekennzeichnet wird, erfolgt die Kommunikation zunehmend direkt zwischen den Maschinen oder Robotern, aber auch hierfür bedarf es einer formalen Sprache.

Zu den Computersprachen gehören neben den Programmiersprachen die deskriptiven Sprachen. Deskriptive Sprachen sind z. B. die Datenbanksprache SQL und HTML5 als Sprache zur Beschreibung von Webseiten. Für die Kommunikation von Maschine zu Maschine verwendet man überwiegend deskriptive Sprachen, wie z. B. XML, also rein beschreibende Sprachen ohne Elemente zur Ablaufsteuerung (Zyklen und Alternativen).

Doch vor der Programmierung steht die Modellierung. Mit den Modellen werden die Erkenntnisse aus der Entwurfsphase festgehalten und als Vorgaben für die Programmierung dokumentiert. Auch die **Modelle** können als formale Sprachen angesehen werden. Dabei ist nicht die Verständlichkeit durch den Computer das vordergründige Ziel, sondern die Festschreibung der Ergebnisse des Entwurfs zur Dokumentation eines gemeinsamen Verständnisses zwischen Auftraggeber und Softwareentwickler.

Eine der wichtigsten Empfehlungen zur Modellierung bezeichnet sich selbst als Sprache (Language): **UML (Unified Modeling Language)**. Hier spricht man in Bildern, denn bekanntlich gilt: „Ein Bild sagt mehr als 1000 Worte!“

Das Schreiben eines Programmes geschieht allgemein mithilfe einer komfortablen Entwicklungsumgebung. Diese **integrierte Entwicklungsumgebung** (**IDE**, integrated development environment) enthält Texteditor, Compiler bzw. Interpreter, Linker, Debugger und Quelltextformatierungsfunktionen. IDEs werden

mehrheitlich als freie Software angeboten, wie z. B. die Produkte „Eclipse“ oder „Android Studio“. Es gibt aber auch proprietäre IDEs, also kostenpflichtige und firmenspezifischen Lösungen. Microsoft Visual Studio ist hierfür ein gutes Beispiel.

Die IDE „Eclipse“ wird im nächsten Kapitel als Umgebung für die C#- bzw. Java-Programmierung genutzt. Die IDE „Android Studio“, die vom Konzept her auf Eclipse aufbaut, wird für die Web-Entwicklung eingesetzt, wo XML und Java zusammenlaufen und so die Basis für eine einfache Smartphone-App bilden.

Zu jeder Computersprache benötigt man spezielle Software, die diese Sprache versteht und für den Computer übersetzt, also verständlich macht. Früher kamen **Compiler** zum Einsatz, womit ganze Programme übersetzt wurden. Heute werden mehrheitlich **Interpreter** genutzt, die aus der Sprache nacheinander Anweisungen für Anweisung analysieren und für den Computer übersetzen, also interpretieren. Auch die üblichen **Webbrowser**, wie Safari, Mozilla Firefox, Google Chrome oder Microsoft Edge, sind eigentlich Interpreter. Sie werten die in der Sprache HTML geschriebenen Webseiten Zeile für Zeile aus und erstellen so die gewünschten Ansichten der Webinhalte (siehe auch Kap. 7).

4.2 Entwurf und Modellierung

Bevor man mit der Programmierung beginnt, sollte ein Plan existieren. Dieser Plan muss Vorstellungen von der Funktionsweise und Bedienung der Software enthalten, aber auch den Ablauf der Entwicklung berücksichtigen. Die Planung des Ablaufs und der Arbeitsschritte, der Zeitplan sowie der Einsatz von Ressourcen (Personen, Technik, etc.) sind Gegenstand das Projektmanagements (vgl. Kap. 3).

Funktionsweise und Bedienung der zu erstellenden Software sollten sich nach den Anforderungen der Auftraggeber richten. Auch wenn man selbst eine Idee umsetzen möchte, also selbst der Auftraggeber ist, sollte man vorher genau überlegen, was man eigentlich realisieren möchte. Die Ermittlung der Anforderungen an das zu entwickelnde System kann durch Befragungen oder Analysen beim Auftraggeber (vgl. Anforderungsanalyse, Kap. 3) erfolgen. Es eignet sich auch die Analyse der Bedürfnisse des potenziellen Nutzers, also die Methode des „Design Thinking“. Wichtig ist auf jeden Fall die Dokumentation der Ergebnisse des Entwurfs.

Für die Dokumentation von Entwürfen haben sich die Modelle bewährt. Modelle verbinden Auftraggeber und Entwickler und dienen der Kommunikation zwischen beiden. Der Auftraggeber kann anhand des Modelles zeigen, was ihm gefällt oder nicht gefällt und was er sich noch wünscht. Der Entwickler kann anhand des Modells die Funktionsweise und Bedienung der Software veranschaulichen und auch auf weitere Leistungsmöglichkeiten verweisen. Es gibt viele Arten von Modellen, wie zum Beispiel:

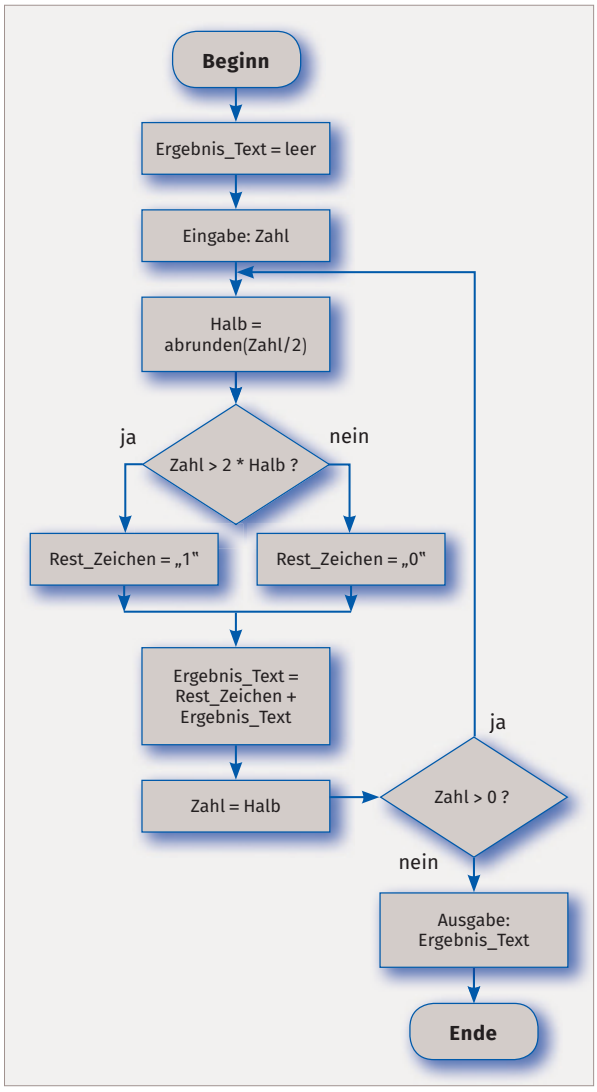
- **Haptische Modelle**, also Modelle zu Anfassen. Architekten oder Designer verwenden diese Modelle.
- **Funktionale Modelle**, also Prototypen der zukünftigen Produkte. Hier kann man ausgewählte Eigenschaften in Funktion erleben.
- **Abstrakte Modelle**, d. h. Gedankenspiele oder exakte mathematische Systeme zur Abbildung realer Zusammenhänge.
- **Grafische Modelle**, dargestellt mit einfachen Zeichnungen oder umfangreichen Skizzen.

Die grafischen Modelle finden in der Softwareentwicklung breite Verwendung. Meistens basieren sie auf wenigen eindeutigen Symbolen, sind mithilfe spezieller Editoren am Computer erstellbar und können häufig automatisch in Quelltext überführt werden (Modell Driven Development).

Bei der Softwareentwicklung dienen Modelle zur Beschreibung von Funktionsweise und Bedienung. Die Funktionsweise ergibt sich aus den implementierten Algorithmen, zu deren Modellierung traditionell Programmablaufpläne (Kap. 4.2.1) oder Struktogramme (Kap. 4.2.2) genutzt werden. Die Struktur und Arbeitsweise der Software lässt sich mit der vereinheitlichten Modellierungssprache UML (Kap. 4.3) beschreiben.

4.2.1 Programmablaufplan

Ein Programmablaufplan (PAP) wird auch als Ablaufdiagramm, Flussdiagramm oder Blockdiagramm bezeichnet. Genormt ist jedoch der Begriff **Programmablaufplan** nach DIN 66001 bzw. ISO 5807.



PAP zur Umrechnung in Dualzahlen

Als Beispiel dient die Umrechnung einer Dezimalzahl in eine Dualzahl durch die wiederholte Division durch den Wert 2 und das Notieren der Restwerte. Aus den Restwerten, gelesen von unten nach oben, ergibt sich die Dualzahl.

Gegeben: Zahl = 25
25 / 2 = 12 Rest 1
12 / 2 = 6 Rest 0
6 / 2 = 3 Rest 0
3 / 2 = 1 Rest 1
1 / 2 = 0 Rest 1

Ist der Quotient Null, bricht der Algorithmus ab.
Die Dualzahl ist von unten nach oben zu lesen:
$25_{dez} = 11001_{dual}$

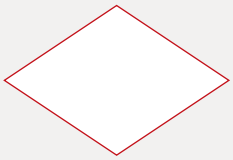
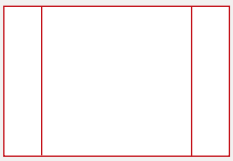
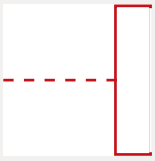
Im Algorithmus erreicht man dieses Lesen von unten nach oben durch das jeweilige Voranstellen des neuen „Rest_Zeichen“ vor den bisherigen „Ergebnis_Text“.

W **Programmablaufpläne** sind grafische Darstellungen mithilfe genormter Symbole. Sie sind weit verbreitete Hilfsmittel bei der Programmentwicklung. Die Programmstruktur lässt sich bildhaft als Ablauf darstellen.

Die Festlegung der Symbole basiert auf der Erkenntnis, wonach sich jeder Algorithmus durch die folgenden Elemente darstellen lässt:

- **Sequenz:** Folge von Anweisungen
- **Alternative:** Mögliche Auswahl zwischen weiteren Wegen der Abarbeitung
- **Zyklus:** Wiederholung von Anweisungsfolgen

Die einzelnen Symbole sind im Rahmen der angegebenen Standards mit ihrer Bedeutung festgeschrieben. Die Pfeile zeigen die Verarbeitungsrichtung an. Die wichtigsten Symbole werden in der folgenden Übersicht in ihrer Bedeutung erläutert.

Symbol	Bedeutung
	**Operation**, allgemeine Anweisung zur Darstellung der Sequenz
	**Verzweigung**, Symbol zur Darstellung der Alternative
	**Unterprogramm**, Symbol zur Zusammenfassung mehrerer komplexer Anweisungen
	**Eingabe und Ausgabe**, wobei die Art der Ein- oder Ausgabe aus der Beschriftung hervorgeht
	Ablauflinie möglichst mit Pfeilen, wobei die Vorzugsrichtungen von oben nach unten und von links nach rechts verlaufen
	**Zusammenführung von Abläufen**. Zwei sich kreuzende Ablauflinien bedeuten keine Zusammenführung.
	**Übergangsstelle**, kennzeichnet die mögliche Fortsetzung des Ablaufs auf einem anderen Blatt. Zusammenhängende Übergangsstellen tragen die gleiche Bezeichnung.
	**Grenzstelle**, allgemein genutzt zur Darstellung von Beginn und Ende
	**Bemerkungen**. Über dieses Symbol können zu allen anderen Symbolen Kommentare eingefügt werden.

Symbole des PAP laut Definition

Aufgaben

1. Beschreiben Sie anhand eines PAP den Ablauf, der das Betanken eines Fahrzeuges an einer Tankstelle darstellt.
2. Erstellen Sie einen PAP zur Umsatzberechnung aus einzelnen Kassenbuchungen.
3. Entwerfen Sie einen PAP, mit dessen Hilfe der Einbau einer Festplatte beschrieben wird.
4. Entwickeln Sie einen PAP, der beschreibt, wie nach der Eingabe einer Schulnote die Bewertung in Textform ausgegeben wird.

4.2.2 Struktogramm

S

Der Ausbildungsleiter Herr Köhler wünscht sich Struktogramme zur Dokumentation der Programmstruktur, weil sich daraus eine strukturierte Programmierung entwickeln lässt. Deshalb versuchen die beiden Azubis Anna und Stefan, den PAP in ein Struktogramm umzusetzen.

W 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.

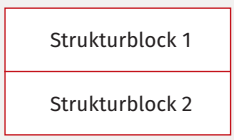
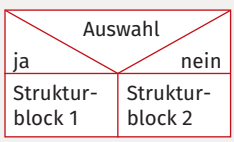
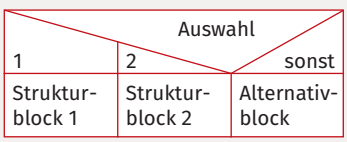
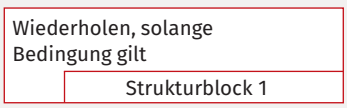
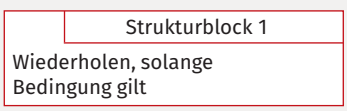
Struktogramme wurden im Jahre 1973 von I. Nassi und B. Shneiderman als Darstellungsmittel für einen Algorithmus beim strukturierten Programmentwurf entwickelt.

Bei der Festlegung der Symbole für die Blöcke bezieht man sich wie beim Programmablaufplan auf die Erkenntnis, wonach jeder Algorithmus durch folgende Elemente dargestellt werden kann:

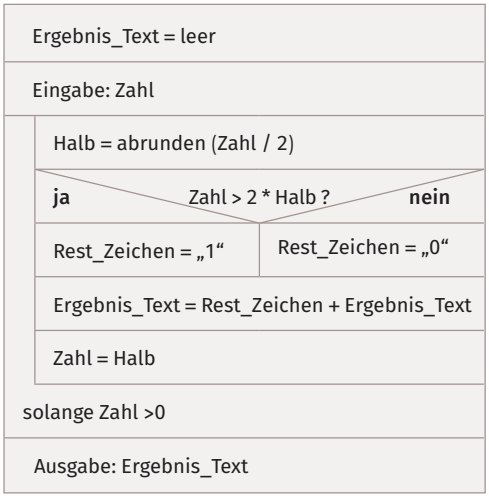
- **Sequenz:** Folge von Anweisungen
- **Alternative:** Mögliche Auswahl zwischen weiteren Wegen der Abarbeitung
- **Zyklus:** Wiederholung von Anweisungsfolgen

Ein Programmbaustein aus mehreren logisch zusammengehörenden Befehlen wird als **Strukturblock** bezeichnet. Ein einzelner Befehl heißt **Elementarblock**. Zur Darstellung von Programmabläufen in Struktogrammen werden Symbole verwendet, die in der Übersicht zusammengefasst sind.

Die Symbole stehen für die verschiedenen Operationsarten, genormt nach DIN 66261. Sie werden von oben nach unten betrachtet (Top-down-Betrachtung).

Verarbeitung (Prozess)		Mit dem Verarbeitungssymbol werden Struktur- und Elementarblöcke dargestellt, die Ein- und Ausgabebefehle, Berechnungen und Unterprogrammaufrufe enthalten.
Folge (Sequenz)		Abfolgen mit zwei oder mehreren Arbeitsschritten werden durch aneinandergereihte Strukturblöcke dargestellt.
Auswahl, Alternative (Verzweigung)		Mit dem Symbol „Alternative“ wird eine Bedingung im Programmablauf dargestellt. Ist die Bedingung erfüllt, wird der Strukturblock 1 ausgeführt, ansonsten der Strukturblock 2.
Fallauswahl (Mehrfachverzweigung)		Bei einer mehrfachen Bedingung im Programmablauf wird kontrolliert, welche Auswahl vorgenommen wurde, und dann wird in den entsprechenden Strukturblock verzweigt. Trifft keine der Bedingungen zu, wird der Alternativblock ausgeführt.
Zyklus/Wiederholung (Schleife)		Diese Wiederholungssymbole dienen zum Formulieren von Befehlen in Schleifen. **Der Strukturblock wird wiederholt, solange die Bedingung gilt.**
Zyklus/Wiederholung (solange Bedingung gilt)		Diese Wiederholungssymbole dienen zum Formulieren von Befehlen in Schleifen. **Der Strukturblock wird solange von Neuem ausgeführt, bis die angegebene Bedingung erfüllt ist.**

Symbole der Struktogramme laut Definition



Die Aufgabenstellung für das Struktogramm entspricht der Aufgabenstellung für die Entwicklung eines Programmablaufplanes (PAP) aus Kapitel 4.2.1. Als Beispiel dient wieder die Umrechnung einer Dezimalzahl in eine Dualzahl mithilfe der wiederholten Division durch den Wert 2 und das Notieren der Restwerte. Aus den Restwerten, gelesen von unten nach oben, ergibt sich dann die Dualzahl.

Beispiel für Struktogramm

Aufgaben

1. Beschreiben Sie mit einem Struktogramm den Ablauf, der das Betanken eines Fahrzeuges an einer Tankstelle darstellt.
2. Erstellen Sie ein Struktogramm zur Umsatzberechnung aus einzelnen Kassenbuchungen.
3. Entwerfen Sie ein Struktogramm, mit dessen Hilfe der Einbau einer Festplatte beschrieben wird.
4. Entwickeln Sie ein Struktogramm, das beschreibt, wie nach der Eingabe einer Schulnote die Bewertung in Textform ausgegeben wird.

4.2.3 Pseudocode

S Von ihrem IT-Lehrer erhalten die beiden Azubis Anna und Stefan den Tipp, statt der Struktogramme Pseudocodes zu verwenden und bei kniffligen Auswahlkriterien Entscheidungstabellen zu benutzen. Damit soll die Lesbarkeit des Programmablaufs erhöht werden.

W Der **Pseudocode** ist die umgangssprachliche Beschreibung eines Programmablaufs, der sich syntaktisch an eine Programmiersprache anlehnt. Der Pseudocode ist nicht genormt und kann deshalb relativ frei formuliert werden.

Der Aufbau von Pseudocodes sollte der Programmiersprache entsprechen, in der das Programm geschrieben wird. In Pseudocodes können bereits Variablen- und Konstantendeklarationen enthalten sein.

Die folgende Aufgabenstellung entspricht wieder der Aufgabenstellung für die Entwicklung eines Programmablaufplanes (PAP) aus Kapitel 4.2.1.

```
Beginne Modul Umrechnung_Dezimal_Dual
Definiere Dualfolge als leere Zeichenkette
Definiere Rest als Zeichen
Definiere Zwischenwert als ganze Zahl
Eingabe Dezimalzahl
Wiederhole bis Dezimalzahl gleich Null
      Zwischenwert = Dezimalzahl / 2
      Wenn glatt teilbar dann Rest="0"
      Ansonsten Rest="1"
      Dezimalzahl = Zwischenwert
      Dualfolge <= Rest vor Dualfolge
Ende der Schleife
Ausgabe der Dualfolge
Ende des Moduls
```

Als Beispiel dient die Umrechnung einer Dezimalzahl in eine Dualzahl mit wiederholter Division durch den Wert 2 und das Notieren der Restwerte. Im Pseudocode muss man nicht genau mahematisch definieren, wie der Restwert ermittelt wird. Wenn die Dezimalzahl glatt teilbar ist, also ohne Restwert, wird das „Rest“-Zeichen auf „0“ gesetzt, ansonsten auf „1“.

Aus den Restwerten baut sich dann die Dualzahl auf. Dazu wird der neue Rest immer vor die bestehende Zeichenkette gesetzt, was dem „Lesen von unten nach oben“ entspricht.

5 Programmierung in Java und C#

5.1 Auswahl einer Programmiersprache

S Kerstin und Stefan stehen vor dem Problem, mit welcher Programmiersprache sie ein Programm programmieren sollen. Um sich endgültig zu entscheiden, beschließen sie, im Internet zu recherchieren und sich über die möglichen Programmiersprachen zu informieren.

Unter einer **Programmiersprache** versteht man nach DIN 44300, Teil 4, eine formale Sprache zum Abfassen (Formulieren) von Verarbeitungsanweisungen für Rechnersysteme. Die durch eine Programmiersprache ausgedrückte, von einem Menschen lesbare Beschreibung heißt Quelltext (oder auch **Quellcode/Programmcode**).

Unter der **Syntax** versteht man eine formale Grammatik, die es ermöglicht, erlaubte Konstruktionen festzulegen und unerlaubte Konstruktionen auszuschließen. Der Quelltext besteht aus folgenden Elementen oder einer Kombination dieser Elemente:

- Wörter
- Symbole
- Regeln
- Trennzeichen

Die Bedeutung eines speziellen Symbols in einer Programmiersprache nennt man dessen **Semantik.** Die **Syntax** und die **Semantik** kann man der **Spezifikation,** teilweise auch der **Dokumentation** der Programmiersprache entnehmen. Die syntaktische Definition einer Programmiersprache wird meist in der formalen Notation **Backus-Naur-Form** angegeben.

Neben der Unterscheidung nach Generationen lassen sich Programmiersprachen nach der Art und Weise der Problemlösung bzw. -darstellung unterscheiden.

Als **prozedurale Programmiersprachen** werden die Sprachen der ersten bis dritten Generation bezeichnet. Bei diesen Programmiersprachen werden vor allem Programmabläufe beschrieben. Ein solcher Programmablauf gibt den Weg zur Problemlösung als Folge von Einzelschritten an. Die Entwicklung eines Algorithmus ist demnach als Vorstufe der Programmierung anzusehen.

Die prozedurale Entwurfsmethode konzentriert sich auf die Zerlegung von Aufgaben der einzelnen Verarbeitungsschritte, um sie danach in übersichtliche **Unterprogramme** zu gliedern. Diese Unterprogrammtechnik ist in den prozeduralen Programmiersprachen eine sinnvolle Möglichkeit der strukturierten Programmentwicklung, da jedes Unterprogramm einen Teilschritt bei der Problemlösung darstellt. Auch unter dem Aspekt der Wiederverwendbarkeit von Programmbausteinen können Unterprogramme effizient eingesetzt werden.

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.

Logische Programmiersprachen sind auf die Lösung von bestimmten Problemen abgestimmt (Datenbankabfragen, mathematische Beweise). Es wird kein Algorithmus zum Lösen eines Problems angegeben, sondern es werden lediglich die Bedingungen für eine korrekte Lösung bestimmt.

Die Weiterentwicklung der modularen Programmierung führte zur **objektorientierten Programmiersprache.** Das Problem der modularen Programmierung besteht darin,

- dass globale Variablen in jedem Teil des Programms aufgerufen und überschrieben werden können und
- Verbindungen zwischen den Daten eines Programms und den sie manipulierenden Funktionen fehlen.

Dies führt dazu, dass große Programme sehr leicht unübersichtlich werden und sich schwerer testen lassen.

Im Jahre 1970 erkannte David Parnas das Problem und hatte die Idee, in einem Modul jede einzelne Variable zu kapseln. Der direkte Zugriff auf die Variablen 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, indem sie die Variable über eine Schnittstelle für ein solches Modul aufrufen.

Statt ein Problem in Teilprobleme zu zerlegen und diese Teilprobleme durch Unterprogramme wie bei der prozeduralen Programmierung zu lösen, werden hier das Problem und seine Lösung durch eine Untergliederung der einzelnen Elemente reduziert. Es entstehen sogenannte **Klassenhierarchien.**

Typische Vertreter für diesen Lösungsansatz sind C++, Java oder C#. Dabei ist C++ die älteste dieser Programmiersprachen. Bereits im Jahre 1982 begann Bjarne Stroustrup eine Erweiterung der prozeduralen Programmiersprache C zu entwickeln. Im Jahre 1989 wurde die Basissprache definiert und 1996 der internationale Standard verabschiedet. C++ eignet sich auch zur Entwicklung von systemnahen Programmen.

Java wurde im Jahre 1991 von Ingenieuren der Firma SUN mit dem Ziel eines interaktiven Fernsehens unter dem Namen „OaK" entwickelt. Nach und nach änderte sich die Zielvorstellung und heute wird Java hauptsächlich für Internet- bzw. plattformübergreifende Anwendungen eingesetzt.

Im Jahr 2001 entwickelte Microsoft die Programmiersprache **C#** für ihre .Net-Plattform. Dabei wurde auf bewährte Konzepte der Programmiersprachen C, C++, Java, Delphi und Haskell zurück gegriffen. Wie Java ist auch C# eine reine objektorierentierte Sprache.

Aufgaben

1. Nennen Sie den Unterschied zwischen einer prozeduralen und einer objektorientierten Programmiersprache.
2. Diskutieren Sie den Unterschied zwischen den Programmiersprachen C++ , Java und C#.
3. Informieren Sie sich über weitere moderne Programmiersprachen wie Rust, Go, Swift und D. Erarbeiten Sie deren Gemeinsamkeiten und Unterschiede.

S

Anna und Stefan finden sowohl Java als auch C# sehr interessant. Deshalb beschließen sie, beide Sprachen besser kennenzulernen und in Projekten auszuprobieren. Anna möchte sich mit Java beschäftigen und Stefan mit C#. Anschließend informieren sich beide über Entwicklungsumgebungen (IDEs), mit deren Hilfe sie ein Programm in der entsprechenden Sprache schreiben und ausführen können.

5.2 Das erste Programm

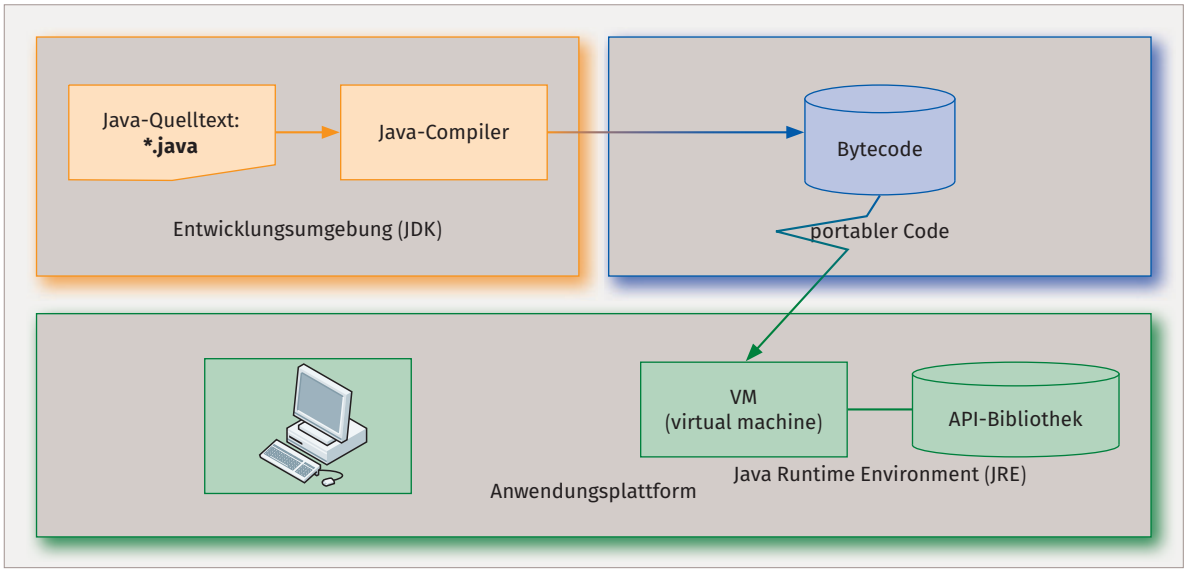
5.2.1 Grundlagen von Java

Java ist eine moderne, einfach zu erlernende und konsequent objektorientierte **Programmiersprache**. Gleichzeitig steht Java aber auch für eine spezielle **Technologie** der Bereitstellung von Software und hat sich quasi zu einer **Philosophie** der IT-Branche entwickelt.

Als Programmiersprache unterstützt Java den Programmierer dabei, kleine, zuverlässige und fehlerfreie Programmeinheiten zu erstellen. Wichtig ist in diesem Zusammenhang die konsequente Objektorientierung. Unter Java entstehen Objekte mit gesicherter Qualität, wobei die Performance etwas in den Hintergrund tritt.

Als Technologie umfasst Java zum einen das JDK (**J**ava **D**evelopment **K**it) als komfortable Entwicklungsumgebung und zum anderen die JRE (**J**ava **R**untime **E**nvironment) als Laufzeitumgebung. In der Entwicklungsumgebung werden die Programme erfasst, getestet und übersetzt. Die Laufzeitumgebung realisiert die Java-Plattform und muss auf den Computern installiert sein, wo Java-Anwendungen ausgeführt werden sollen. Die Laufzeitumgebung beinhaltet zum einen die virtuelle Maschine (VM) zur Interpretation und Ausführung des vorab übersetzten Java-Programms und zum anderen die Java-API (**A**pplication **P**rogramming **I**nterface) als Bibliothek mit vorgefertigten Softwarebausteinen.

Java zielt als Philosophie auf Unabhängigkeit und Qualität. Unabhängig will man von konkreten Hard- und Softwareplattformen sein, unabhängig aber auch von jeglichen kommerziellen Einflüssen. Java ist das Werkzeug der Open-Source-Community. Es gibt in diesem Bereich umfangreiche Unterstützung und eine Vielzahl von frei nutzbaren Tools, wie z. B. Eclipse, worauf bereits in Kapitel 4 verwiesen wurde.



Entstehung einer Java-Applikation

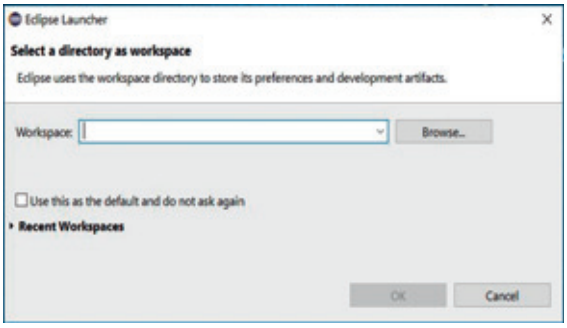
5.2.2 Programm in Java mithilfe von Eclipse erstellen

S Anna informiert sich über die Entwicklung von Java-Programmen und entsprechenden integrierten Entwicklungsumgebungen (IDEs). Dabei stellt sie fest, dass es möglich ist, Java-Programme auch ohne Hilfe einer IDE zu entwickeln. Allerdings lassen sich Programme mithilfe einer IDE komfortabler erstellen und Projekte besser verwalten. Nach einigen Recherchen und aufgrund von Empfehlungen entscheidet sich Anna für „Eclipse“ als Entwicklungsumgebung.

Eclipse ist ein sehr flexibles und leistungsfähiges Open-Source-Framework für eine Entwicklungsumgebung, das nach entsprechender Aufrüstung durch Plug-ins durchaus mit kommerziellen IDEs konkurrieren kann. Um ein Programm in „Eclipse“ zu erstellen, sind folgende Schritte notwendig.

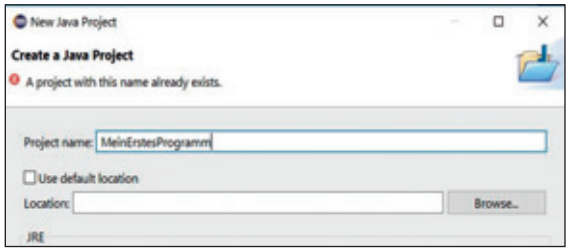
Eclipse starten

Gestartet wird Eclipse durch einen Doppelklick auf das Programmsymbol. In das folgende Fenster muss dann das Arbeitsverzeichnis eingegeben und mit „Ok“ bestätigt werden.



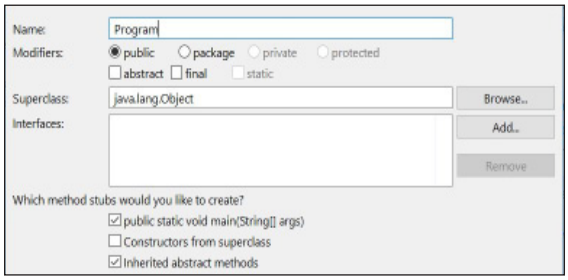
Projekt anlegen

Durch den Menübefehl **File** | **New** | **Java Project** wird ein neues Projekt erzeugt. Es ist dabei ein Projektname anzugeben, hier z. B. „MeinErstesProgramm“, und mit Betätigen des „Finish“-Buttons zu bestätigen.



Klasse erzeugen

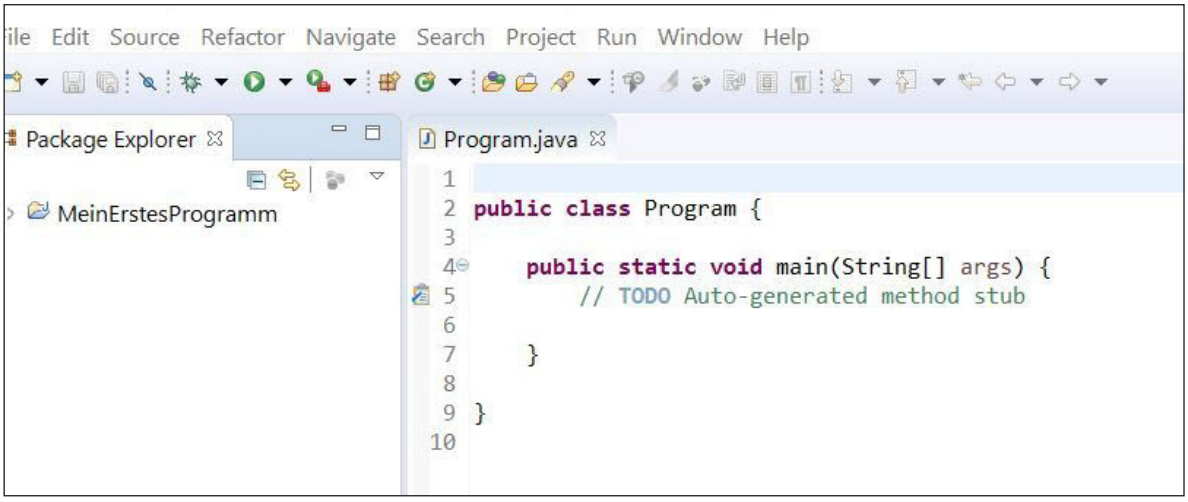
Durch den folgenden Menübefehl wird eine Klasse erzeugt: **File** | **New** | **Class**



Dabei ist ein Name für die Klasse anzugeben, hier z. B. der Name „Program“. Weiterhin ist der Haken bei dem Auswahlpunkt „public static void main (String[] args)“ zu setzen. Danach wird durch das Betätigen des Button „Finish“ die Klasse erzeugt.

Programm schreiben

Jetzt sollte folgende Ansicht zu sehen sein und es kann mit dem Schreiben des Programmes begonnen werden.



Als erstes Programm soll das klassische „Hello World“ ausprobiert werden. Nachfolgend ist der Quelltext für dieses Programm angegeben.

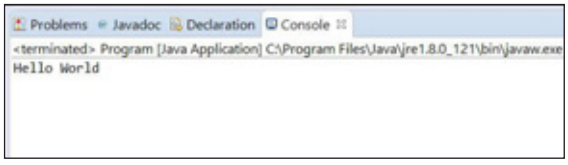
```
public class Program
{
    public static void main( String[] args )
    {
      System.out.println(“Hello World”);
    }
}
```

Dieser Quelltext kann in die IDE „Eclipse“ zum Ausprobieren übertragen werden. Dabei ist zu beachten, dass jene Teile, die automatisch generiert wurden und im Quelltext schon vorhanden sind, nicht erneut eingegeben werden müssen. In unserem Fall reicht es also, die Anweisung „System.out.println(“Hello World”);“ zu übertragen. Danach kann das Programm ausgeführt und getestet werden.

Programm starten

Durch den folgenden Menübefehl wird das Programm gestartet: **Run** | **Run**

Alternativ kann auch der in der Menüleiste abgebildete Button zum Ausführen betätigt werden. Das Programm wird ausgeführt und das Ergebnis erscheint im Ausgabefenster von Eclipse.



5.2.3 Grundlagen der Programmiersprache C#

C# ist eine einfache, sichere, moderne und leistungsfähige sowie rein objektorientierte Sprache. Sie wurde unter anderem von Andres Hejlsberg entwickelt, der früher z. B. auch für die Entwicklung des Produktes Borland Delphi zuständig war. In die Entwicklung von C# sind die Erfahrungen aus unterschiedlichen Programmiersprachen der letzten 30 Jahre eingeflossen, wie z. B. von C++ und Java. Die Grundlage für das Programmieren mit C# ist das **.Net-Framework**. Das .Net-Framework wurde von Microsoft entwickelt. Dabei handelt es sich nicht nur um eine besondere Laufzeitumgebung, sondern das .Net-Framework stellt gleichzeitig eine umfangreiche Klassenbibliothek für die Programmierung unter Windows zur Verfügung. Microsoft hat sich bei der Entwicklung von .Net von vielen schon vorhandenen Technologien inspirieren lassen. Das Ergebnis war eine Technologie, die der von Java sehr ähnlich ist. Auch hier wird aus dem Quellcode zunächst ein Zwischencode erzeugt, der erst zur Laufzeit in nativen Code übersetzt wird. Dieser Zwischencode wird **Intermediate Language (IL)** genannt und zum Ausführungszeitpunkt durch einen Just-In-Time-Compiler übersetzt und ausgeführt.

5.2.4 Programm in C# mithilfe von Visual Studio erstellen

S Während sich Anna mit Eclipse als Entwicklungsumgebung für Java beschäftigt, lernt Stefan Visual Studio als IDE für C#-Programme kennen.

Visual Studio ist eine von Microsoft entwickelte integrierte Entwicklungsumgebung (IDE), womit standardmäßig verschiedene Sprachen unterstützt werden. Neben C# sind das unter anderem C++ und Visual Basic. NET. Mithilfe von Plug-ins können weitere Sprachen in Visual Studio genutzt werden. Außerdem existiert mit Visual Studio Community eine Version, die aus dem kostenpflichtigen Visual Studio hervorgegangen ist und von allen Anwendern gratis verwendet werden darf.

Um ein Programm in „Visual Studio“ zu erstellen, sind folgende Schritte notwendig:

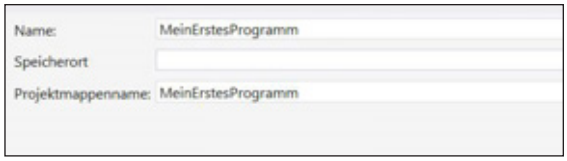
Visual Studio starten

Gestartet wird Visual Studio durch einen Doppelklick auf das Programmsymbol.

Projekt anlegen

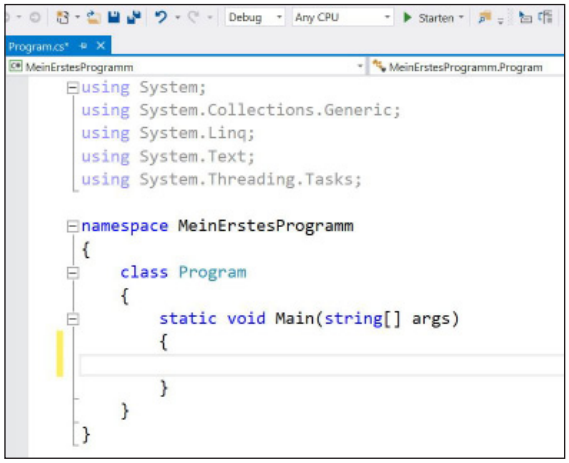
Durch den folgenden Menübefehl wird ein neues Projekt erzeugt: **Datei | Neu | Projekt**

Es wird ein Fenster angezeigt, wo die Programmiersprache und die Art der Anwendung ausgewählt werden muss. Für die ersten Übungen wird der Menüpunkt „Konsolenanwendung“ ausgewählt, ein Projektname eingegeben und ein Speicherort ausgewählt. Anschließend sind die Eingaben mit „Ok“ zu bestätigen.



Programm schreiben

Nachdem die folgende Ansicht erscheint, kann mit dem Schreiben des Programmes begonnen werden.



Das erste Programm, das ausprobiert werden soll, ist auch hier das klassische „Hello World“. Der Quelltext für das Programm erscheint in der folgendem Box.

Zur besseren Unterscheidung sind die Quelltexte für C#-Programme im gesamten Kapitel grün eingerahmt.

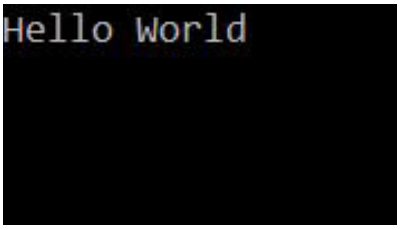
```
using System;

namespace MeinErstesProgramm
{
    class Program
    {
        static void Main(string[] args)
        {
             Console.WriteLine("Hello World");
             Console.ReadKey();
        }
    }
}
```

Dieser Quelltext kann nun nach Visual Studio zum Ausprobieren übertragen werden. Dabei ist zu beachten, dass bereits automatisch generierte Teile, die schon im Quelltext vorhanden sind, nicht noch einmal eingegeben werden müssen. In unserem Fall reicht es, die Anweisungen „Console.WriteLine("Hello World");" und „Console.ReadKey();" einzugeben. Danach kann das Programm ausgeführt und getestet werden.

Programm starten

Durch den folgenden Menübefehl wird das Programm dann gestartet: **Debuggen** | **Debugging starten**

Alternativ kann auch der in der Menüleiste abgebildete Startbutton betätigt werden. Das Programm wird ausgeführt und das Ergebnis erscheint in einem sich öffnenden, separaten Fenster.



5.3 Grundlegende Sprachelemente

5.3.1 Grundgerüst eines Programmes

Die nachstehenden Beispiele zeigen jeweils das **Grundgerüst für Java- und C#-Programme**. Die Quelltexte, die für Java-Programme, C#-Programme und für beide Programme gelten, sind farblich verschieden unterlegt.

```
public class Program
{
     public static void main( String[] args )
     {
     }
}
```

```
class Program
{
     static void Main(string[] args)
     {
     }
 }
```

Wie man in den Beispielen sehen kann, sind Java und C# in den grundlegenden Sprachelementen syntaktisch sehr ähnlich. Folgende syntaktische Eigenschaften gelten deswegen auch für beide Sprachen:

1. Es wird zwischen Groß- und Kleinschreibung unterschieden. Beide Sprachen sind „case sensitive".
2. Die Programme bestehen aus Klassen. Die Klassendefinition wird mit dem Schlüsselwort **class** eingeleitet.
3. Damit aus der Klasse eine Anwendung (Application) wird, muss eine Funktion mit dem Namen **main** vorhanden sein. Dies ist die Hauptfunktion, die beim Aufruf der Anwendung gestartet wird. In C# muss **Main** großgeschrieben werden.
4. Die Schlüsselwörter der Funktion **main** verkörpern folgende Sachverhalte:
   - **void** - Diese Funktion hat keinen Rückgabewert. Der Datentyp der Funktion ist damit unbestimmt.
   - **static** - Diese Funktion gehört fest zu der Klasse und kann aufgerufen werden, ohne das ein Objekt von der Klasse erzeugt wurde.
5. Geschweifte Klammern { } markieren die Anweisungenblöcke. Das erste Klammerpaar fasst alles zusammen, was zur Klasse gehört. Das innere Klammerpaar schließt die Anweisungen der Funktion **main** ein.
6. Anweisungen werden immer mit einem Semikolon abgeschlossen. Das Komma ist nur ein Trennzeichen für Aufzählungen, wie z. B. bei der Aufzählung der Variablen, die mit dem Datentyp **int** deklariert werden.

Java- und C#-Quelltexte verwenden kein vorgegebenes Format und es gibt keine vorgeschriebene Zeilenstruktur. Das Leerzeichen, das Zeilenendezeichen, der Zeilenvorschub, der Tabulator und Kommentare werden außerhalb von Zeichenkonstanten und Zeichenketten (Strings) als Trennzeichen behandelt, wodurch auch

alle Schlüsselwörter und Anweisungen voneinander getrennt werden. Eine Folge von Trennzeichen wirkt sich dabei wie ein Trennzeichen aus. Im Prinzip kann der Quelltext in einer Datei in einer einzigen Zeile geschrieben werden. Allerdings ist es für den besseren Überblick zweckmäßig, eine Strukturierung des Quelltextes vorzunehmen. Für die Schreibweise von Namen für Variablen, Klassen oder Funktionen gibt es Namenskonventionen, die sinnvoll, aber nicht notwendig sind.

5.3.2 Reservierte Wörter

W **Reservierte Wörter** (Schlüsselwörter) sind feste, vorgegebene Wörter, die der entsprechenden Sprache entstammen. Da jedes dieser Wörter eine feste, definierte Bedeutung besitzt, dürfen diese Wörter nicht als Bezeichner in Namen verwendet werden.

Die reservierten Wörter für Java und C# sind in den folgenden Tabellen zusammengefasst.

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

Schlüsselwörter in C# (Auszug)				
abstract	continue	finally	new	string
as	decimal	float	null	struct
base	default	for	private	switch
bool	do	foreach	protected	this
break	double	goto	public	true
case	else	if	readonly	try
catch	enum	int	return	using
char	event	interface	sbyte	void
class	extern	long	short	volatile
const	false	namespace	static	while

5.3.3 Kommentare

W **Kommentare** dienen dazu, Notizen oder Bemerkungen direkt in den Quelltext des Programms aufzunehmen.

Der Programmierer gestaltet mithilfe von Kommentaren sein Programm für die Leser einfacher und verständlicher. Folgende Kennzeichnungen für Kommentare sind möglich und können in Java und C# mit gleicher Syntax genutzt werden.

6 Datenbankanwendungen

Externe Speicherung → File-System → ANSI-SPARC-Architektur → Datenbankmanagementsysteme (DBMS) → Datenmodell → ER-Modell → Tabellen → Schlüssel → Referenzen → Normalformen → SQL → Abfragen SELECT → Einfügen, Ändern, Löschen → Zugriffsrechte → Transaktionen → ACID → Arbeit mit MySQL → Vom Modell zur Tabelle → Alternative DBMS → Hadoop → NoSQL → In-memory-database

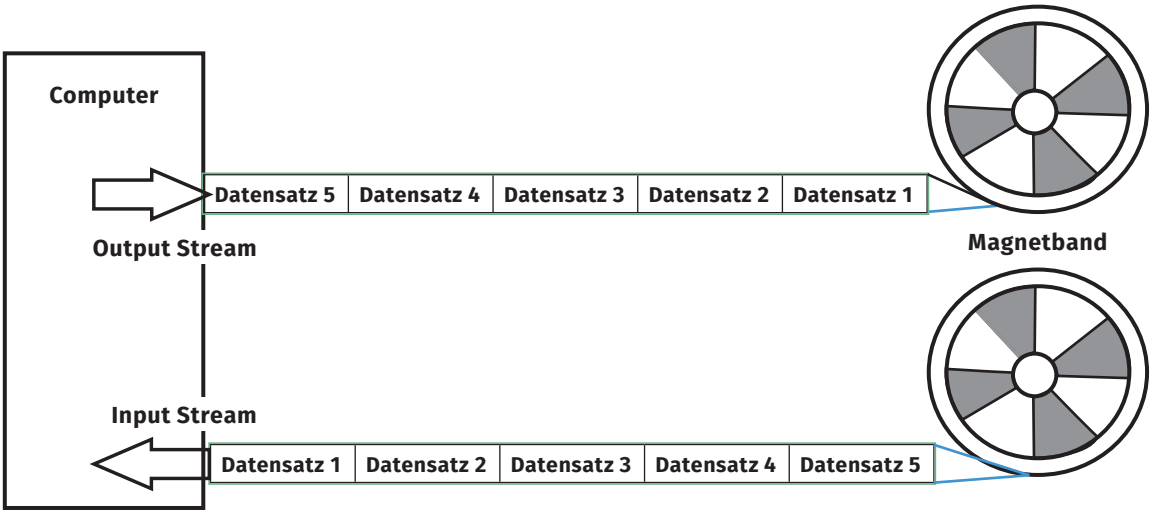
6.1 Von der Datei zur Datenbank

6.1.1 Dauerhafte externe Speicherung von Daten

Die Daten computergestützter Anwendungssysteme müssen für eine längere Aufbewahrung auf externen Datenträgern gespeichert werden. Der Arbeitsspeicher des Computers benötigt eine ständige Energiezufuhr, während er mit der Unterbrechung oder Abschaltung der Energiezufuhr seinen Inhalt verliert. Der Arbeitsspeicher eignet sich so nicht zur dauerhaften Speicherung von Daten.

So muss zum Beispiel jedes Smartphone einen niedrigen Ladezustand seiner Batterie erkennen und rechtzeitig die aktuellen Daten vor dem Ausfall der Energiezufuhr auf eine eingebaute Speicherkarte „retten". Das braucht Zeit, denn es müssen normalerweise viele Daten abgespeichert werden. Umgekehrt kostet es nach dem erneuten Einschalten ebenfalls Zeit, die Daten von der Speicherkarte wieder in den Arbeitsspeicher zu laden. Auch die Programme bzw. Apps liegen als Daten auf der Speicherkarte und müssen zuerst gelesen und dann gestartet werden. Wird ein Smartphone abgeschaltet, so befindet es sich eigentlich nur in einem Ruhemodus. Die Daten verbleiben im Arbeitsspeicher, die Programme arbeiten weiter, warten z. B. auf Anrufe oder andere Nachrichten. Im diesem Ruhemodus verbrauchen die Programme weniger Prozessorleistung und damit weniger Energie, denn das Smartphone ist eigentlich weiter in einem eingeschalteten Zustand.

Die externe Speicherung erfordert eine Technologie, die die Daten ohne ständige Energiezufuhr erhält. Zuerst dienten zur dauerhaften Speicherung Lochstreifen, dann **Magnetbänder** oder **Festplatten** mit magnetisierbaren Oberflächen. Jetzt werden **optische Speicher** oder **Feststoffspeicher** mit veränderbarer Kristallstruktur genutzt.



Strom der Daten (Stream) zwischen Computer und Magnetband

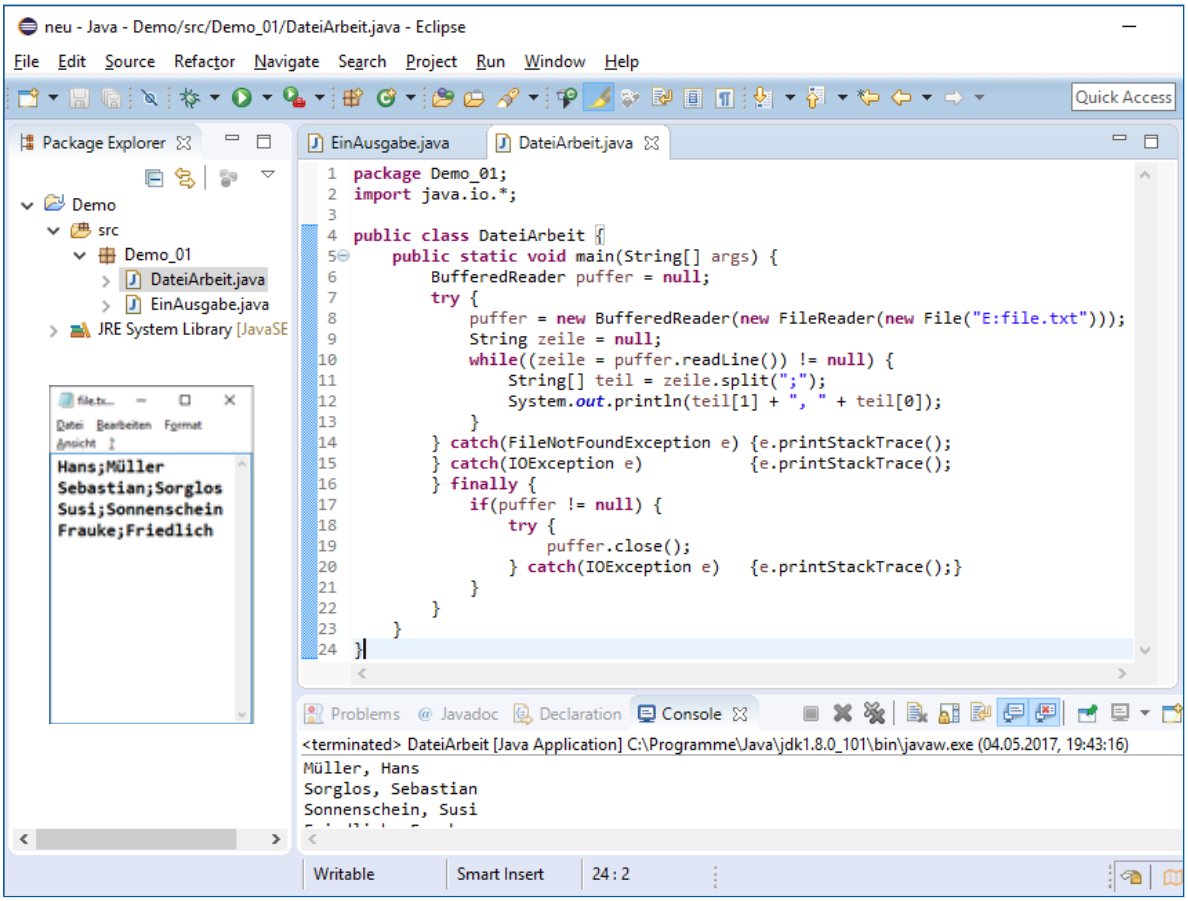
Aus der Anfangszeit der Magnetbänder stammt das Verständnis des Schreibens und Lesens der Daten als Datenstrom. Beim Schreiben „fließen“ die Daten „hinaus“ auf den Datenträger und beim Lesen „fließen“ sie wieder „hinein“ in den Arbeitsspeicher des Computers. Jedes Betriebssystem unterstützt diese Form der Datenströme. Im Konzept der UNIX-Betriebssysteme ist der Stream ein elementares Konstrukt. „Everything should be a stream!“ heißt es dort.

6.1.2 Dateiorganisation

Der Datenstrom fließt in eine Datei bzw. in einen File, wenn der englische Begriff verwendet wird. Jeder File ist eine sequenzielle Struktur von Daten, wo diese Daten ordentlich hintereinander aufgereiht sind. Jeder File ist damit eine eindimensionale Datenstruktur, also nur auf eine Dimension beschränkt, aber in der Länge unbegrenzt. Der Strom der Daten besteht, solange Daten vorhanden sind. Einzig der Datenträger zum Speichern bestimmt die maximale Länge eines Files.

Das Lesen und Schreiben von Files wird von allen Betriebssystemen hocheffizient unterstützt. Diese Systeme schreiben die Daten in einen Puffer und veranlassen dann das Kopieren des Puffers auf den Datenträger. Umgekehrt wird ein Block in den Puffer kopiert und dann dort nach Bedarf ausgelesen.

Der Programmierer kennt die Verwendung der Puffer bei der Dateiarbeit. Zuerst wird die Datei bzw. der File geöffnet, d. h., es wird ein Puffer angelegt und der Puffer wird mit dem externen File verbunden. Man muss sich hier entscheiden, ob man den File lesen oder schreiben möchte. Das Lesen von Daten aus dem File erfolgt immer sequenziell, also immer hintereinander. Auch beim Schreiben kann man Daten nur am Ende des Files, d. h. am Ende des Datenstroms anfügen. Wird der File nicht mehr gebraucht, sollte man ihn schließen. Praktisch gibt man den Speicherplatz für den Puffer frei. Beim Schließen wird nach dem Schreiben der Pufferinhalt auf den Datenträger kopiert. Da dies häufig vom Programmierer vergessen wird, sorgen die Compiler am Ende jeder Prozedur automatisch für das Schließen der Dateien und die Freigabe der Puffer.



Puffer beim Input und Output in Java

Im Programmbeispiel stehen in der Datei „file.txt“ auf dem Laufwerk E: die Daten zeilenweise (**puffer.readLine()**) organisiert. In einer Zeile stehen Vorname und Familienname, getrennt durch ein Semikolon (**zeile.split(“;”)**), die dann in umgekehrter Reihenfolge (erst **teil[1]**, dann **teil[0]** ausgegeben werden. Ansonsten sind noch einige Vorkehrungen zur Fehlerbehandlung im Quelltext zu finden. Es kann passieren, dass die Datei nicht aufzufinden ist (**FileNotFoundException**) oder dass bei der Eingabe oder Ausgabe Fehler (**IOException**) auftreten. Zum Schluss wird der Puffer geschlossen (**puffer.close()**).

Es gilt: „Jedes Programm muss die Struktur seiner Files kennen.“ Diese Erkenntnis macht den entscheidenden Nachteil der File-Organisation deutlich. Nur nach Hinweisen des Programmautors kann auf „fremde“ Files zugegriffen werden. Jedes Programm kann seine individuelle Dateistruktur definieren.

Für Unbeteiligte ist es damit fast unmöglich, unbekannte, fremde Dateien zu lesen. Für unsere heutige Welt der digitalen Kommunikation wäre das ein fatales Hindernis gewesen. So musste man schnell nach Lösungen suchen, wobei sich folgende Lösungsvorschläge entwickelten:

- Die **Datei erhält im Namen eine Information zum verursachenden Programm.** Auf diese Weise entstand der Namensteil „Typenbezeichnung“ im Dateinamen nach dem Punkt, womit ggf. auch das Betriebssystem erkannt wird, z. B. dass eine Datei „neu.docx“ wahrscheinlich vom Programm Microsoft Word gelesen und interpretiert werden kann. Man könnte die Datei umbenennen und die Typenbezeichnung ändern. Dann würde Microsoft Word nicht mehr automatisch zugeordnet werden können, das Programm könnte aber weiterhin die Datei verarbeiten. Für ein eigenes Programm kann man eine eigene Typenbezeichnung definieren. Man muss nur aufpassen, dass diese Bezeichnung nicht bereits vergeben ist.
- Es wird **eine einheitliche, einfache Dateistruktur** festgelegt. So ergeben sich einfache Textdateien, in denen die kleinste Einheit ein Zeichen ist und die größte Einheit eine Zeile darstellt. Zur Abgrenzung von Worten kann man beliebige Trennzeichen definieren, üblicherweise Komma, Semikolon oder Leerzeichen. Dieses austauschbare Dateiformat wird als CSV-Format (Comma Separated Values) bezeichnet.
- Es wird **eine Sprache zur Beschreibung der Dateistruktur** definiert und verwendet. Diese Lösung ist komplizierter, aber wesentlich flexibler. XML oder HTML sind derartige Sprachen, auf die im Kapitel 7 näher eingegangen wird.
- Die **Entkopplung der Datenverwaltung von den Programmen** stellt die universellste Lösung zur Bereitstellung von Daten für unterschiedliche Programme dar. Die Datenverwaltung wird unabhängigen Datenbanksystemen übertragen, mit denen die Programme über definierte Schnittstellen zusammenarbeiten können.

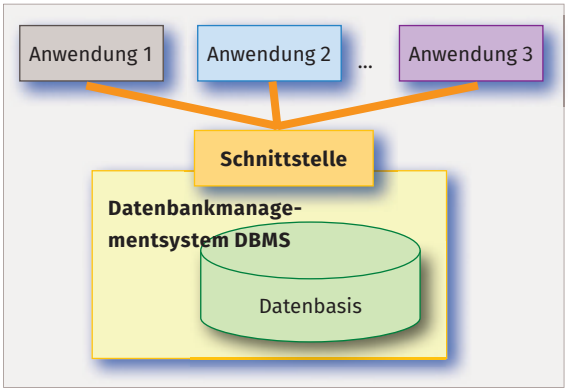
6.1.3 Datenbanken

Die Datenbanken „kapseln“ die Datenverwaltung, d. h., sie überlagern die Daten mit einem Datenbankmanagementsystem (DBMS), das den Zugriff auf die Daten organisiert. Über eine definierte Schnittstelle erhalten die Anwendungsprogramme Zugriff auf die Daten. Die Datenverwaltung geschieht unabhängig von den Anwendungsprogrammen.

W

Ein **Datenbanksystem** besteht aus den folgenden Bestandteilen:

- **Datenbestand** als einer strukturierten Sammlung von Daten. Der Datenbestand wird grafisch oft noch als Zylinder dargestellt, was an den rotierenden Stapel von magnetisierbaren Oberflächen aus einem Festplattenlaufwerk erinnern soll.
- **Datenbankmanagementsystem** (DBMS) zur Verwaltung der Zugriffe auf den Datenbestand, womit der Datenbestand umschlossen (gekapselt) wird. Nur über das Datenbankmanagementsystem kann man auf den Datenbestand zugreifen;
- **Schnittstelle**, die dem Datenaustausch zwischen den Anwendungen und dem Datenbankmanagementsystem dient.



Datenbanksystem

Typische Vertreter von Datenmanagementsystemen sind Microsoft Access, MySQL, SAP HANA, IBM DB2 oder Oracle DB. Alle diese Datenbankmanagementsysteme haben einen standardisierten Aufbau, der dem ANSI-SPARC-Architekturmodell folgt.

6.1.4 ANSI-SPARC-Architektur für Datenbanksysteme

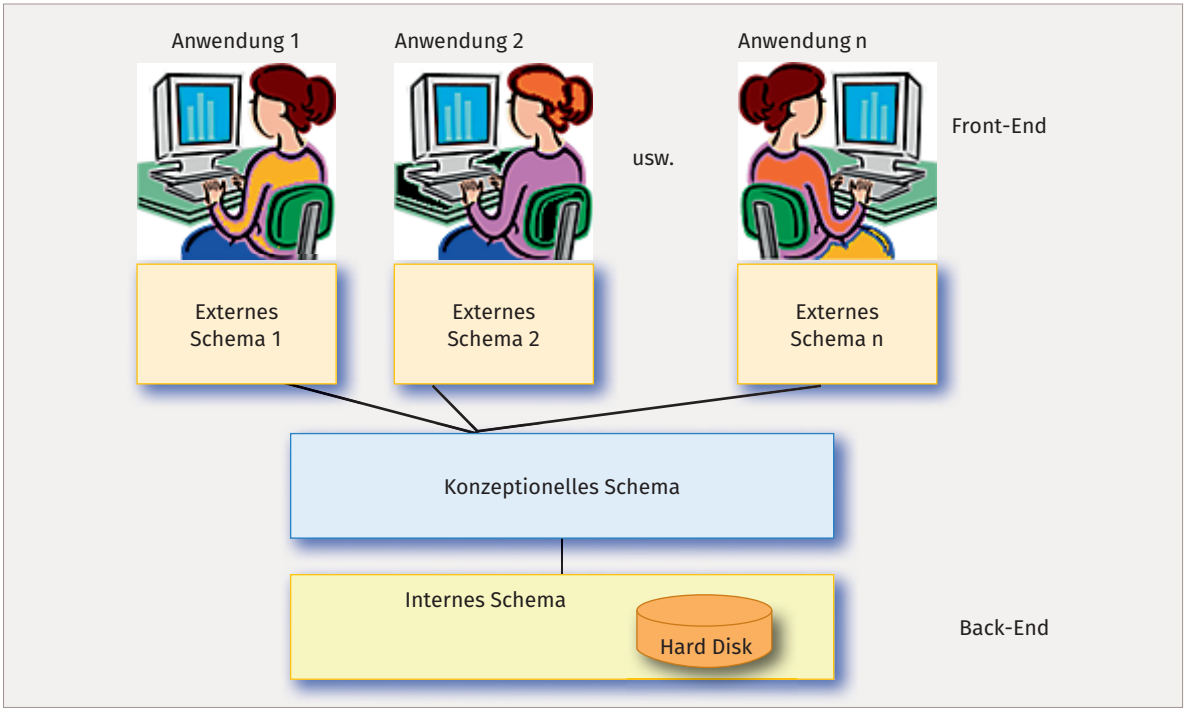
Die ANSI-SPARC-Architektur wurde im Jahre 1975 vom „Standards Planning and Requirements Committee“ (SPARC) des „American National Standards Institute“ (ANSI) entwickelt. Das American National Standards Institute ist der nationale Normenausschuss der USA, der dem Deutschen Institut für Normung (DIN) in Deutschland entspricht. Mit dem Architekturmodell unterbreitete ANSI einen Vorschlag für die prinzipielle Architektur von Datenbanksystemen, an dem sich sowohl die Hersteller von Datenbanksoftware als auch die Betreiber von Datenbanken orientieren sollten. Ziel war die Entkopplung von Anwendung und Datenspeicherung, um

- den Benutzer einer Datenbank vor nachteiligen Auswirkungen von Änderungen in der Datenbankstruktur zu schützen und
- den Entwicklern von Datenbankmanagementsystemen den Raum für kreative Lösungen zu geben.

Im Zentrum der ANSI-SPARC-Architektur befindet sich die konzeptionelle Ebene bzw. das **konzeptionelle Schema**. Über dem konzeptionellen Schema liegt das **externe Schema** und unter dem konzeptionellen Schema befindet sich das **interne Schema**. Im internen Schema wird die Speicherung der Daten mit ihren Zugriffspfaden auf den physischen Speicher festgelegt. Das externe Schema regelt, welche Daten bestimmte Benutzer bzw. Programme sehen und bearbeiten können.

Außerdem definiert das ANSI-Architekturmodell die Zuordnung von personellen Instanzen, die als Anwendungs-, Unternehmens- und Datenbankadministrator bezeichnet werden und die für das externe, konzeptionelle oder interne Schema zuständig sind.

Das **ANSI-SPARC-Architekturmodell** für Datenbanksysteme zeigt die verschiedenen Teile eines Datenbankkonzeptes, aber nicht die Schritte der Datenbankentwicklung. W



ANSI-SPARC-Architekturmodell für Datenbanksysteme

6.1.4.1 Konzeptionelles Schema

W Das **konzeptionelle Schema** benennt und beschreibt alle logischen Dateneinheiten (entities) sowie die Beziehungen (relationships) zwischen den Dateneinheiten. Das konzeptionelle Schema enthält keine Datenwerte (values), sondern beschreibt lediglich deren Struktur in anwendungsübergreifender Form.

Im konzeptionellen Schema werden alle Daten und ihre Beziehungen zueinander modelliert. Hier liegen nicht die Daten sondern die Informationen über die Struktur der Daten. Der Inhalt dieses Schemas ist unabhängig von der eingesetzten Hardware zur Datenspeicherung und unabhängig von den Anforderungen einzelner Benutzer. Dadurch wird die Entkopplung der Anwendungsprogramme von der physischen Datenspeicherung erreicht.

Die Form der Beschreibung und die Beschreibungsmöglichkeiten werden durch das Datenmodell festgelegt, das zur Erstellung des konzeptionellen Schemas herangezogen wird. So kann ein hierarchisches, ein netzwerkartiges oder ein relationales Datenmodell gewählt werden. Keinesfalls enthält ein konzeptionelles Schema Angaben zur physischen Organisation der Speicherung, die nur dem internen Schema zugeordnet sind.

Als anwendungsunabhängiges Instrument zur Beschreibung der Struktur der Datenwelt eines Unternehmens zeichnet sich das konzeptionelle Schema durch folgende Vorteile aus:

- Beschreibung der Datenbasis für alle aktuellen und künftigen Anwendungen eines Unternehmens
- Dokumentation der Informationszusammenhänge eines Unternehmens in einer einheitlichen Form
- Änderung im Vergleich zu den einzelnen Anwendungen erfolgt nur langsam.

6.1.4.2 Internes Schema

W Als **internes Schema** bezeichnet man die Organisation der physischen Datenspeicherung bezüglich der im konzeptionellen Schema definierten logischen Datenstrukturen.

Das interne Schema legt die physikalische Realisierung auf den Speichermedien fest und enthält die Algorithmen zur Speicherung, Indizierung, Sortierung und Selektion der Daten aus dem Datenbestand. Diese Informationen sind das eigentliche Know-how des Entwicklers des Datenbankmanagementsystems, die dem Anwender im Allgemeinen verborgen bleiben. So sieht ein Microsoft-Office-Anwender eine Microsoft-Access-Datenbank immer nur als eine einzige Datei, die außerdem noch eine unverständliche Größe besitzt. Das interne Schema muss der Anwender aber nicht kennen, denn in der Hauptsache muss es funktionieren.

Eine besondere Anforderung im internen Schema ist die Transaktionssicherheit, d. h., es muss die Sicherheit einer Transaktion (siehe hierzu auch Kap. 6.4.8) gewährleistet sein.

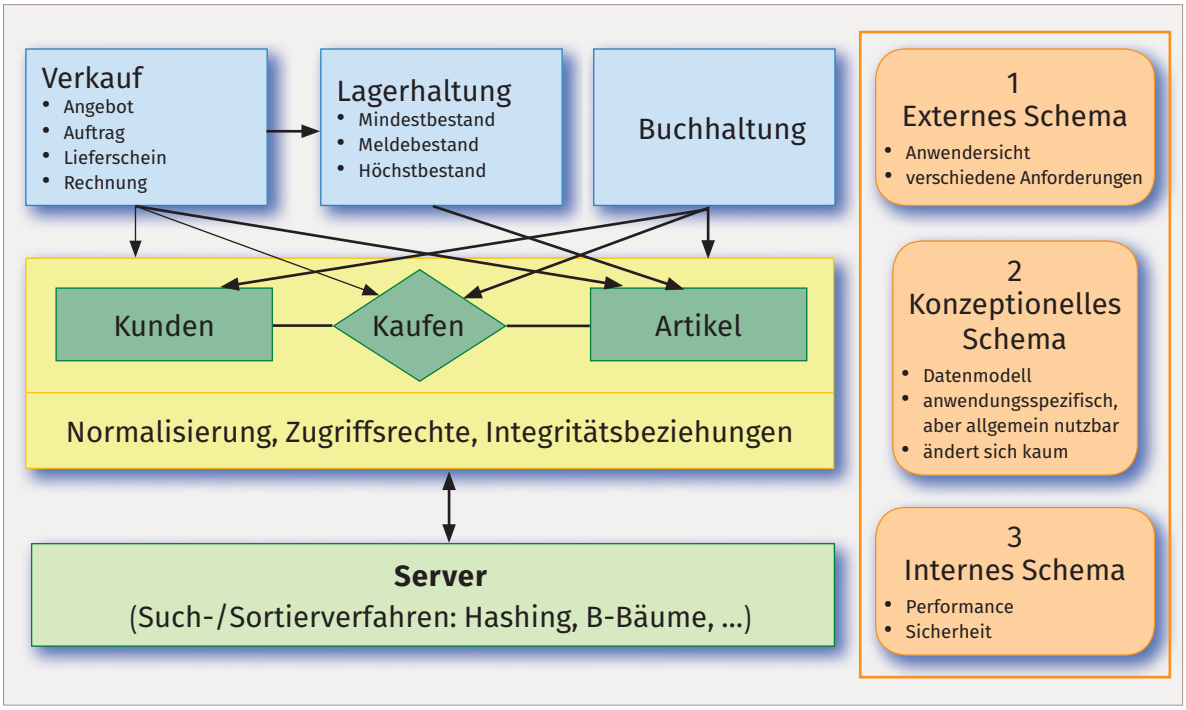
6.1.4.3 Externes Schema

Das **externe Schema** beschreibt einen Ausschnitt aus dem konzeptionellen Schema eines Unternehmens, der auf die spezielle Datensicht einer bestimmten Benutzergruppe zugeschnitten ist. W

Das externe Schema verdeckt gegenüber der betroffenen Benutzergruppe die logische Gesamtsicht, d. h., es zeigt nur den Teil der logischen Gesamtsicht, der für die Anwendungen der betroffenen Benutzergruppe von Interesse ist.

Da ein externes Schema nur einen Teil der konzeptionellen Gesamtsicht wiedergibt, bezeichnet man es auch als Subschema. Das auf die Bedürfnisse einer Benutzergruppe abgestimmte externe Schema soll nicht die Dateneinheiten und Beziehungen enthalten, die diese Benutzer nicht sehen wollen oder nicht sehen sollen. In einem Unternehmen existieren in der Regel mehrere Benutzergruppen. Es sind daher mehrere, unterschiedliche Subschemata zu entwickeln – je ein Schema pro Benutzergruppe.

Unterschiedliche externe Subschemata ermöglichen den **Multi-User-Betrieb**, d. h., mehrere Anwender (User) können aus mehreren (unterschiedlichen) Anwendungen gleichzeitig auf die Daten im Datenbanksystem zugreifen. Das stellt einen gewaltigen Fortschritt gegenüber der Arbeit mit einzelnen Files dar. Der Multi-User-Betrieb verlangt aber zusätzlich die Einrichtung eines Berechtigungskonzepts, welches sicherstellt, dass nur Anwender auf die Daten zugreifen, die hierzu gemäß ihrer Tätigkeit auch berechtigt sind.



Beispiel für Inhalte der Schemata

6.2 Entwurf von Datenbanken

Der Entwurf einer Datenbank kann auf bekannten Daten basieren, wobei oft eine vorhandene, gewachsene Tabelle als Ausgangsbasis dient. Wenn eine Datenbank komplett neu konzipiert wird, empfiehlt es sich davon auszugehen, in welchen Beziehungen die Daten untereinander stehen. Aus diesem Entwurf werden die Tabellen direkt abgeleitet, die dann häufig bereits normiert sind. Es ist aber unerlässlich, die entstandenen Tabellen trotzdem auf die Einhaltung der Normalitätsregeln zu überprüfen (siehe Kap. 6.3.3)

In diesem Abschnitt wird die Analyse der Daten und die Darstellung der Ergebnisse mithilfe eines Entity-Relationship-Modells (ER-Modell) erläutert.

6.2.1 Datenanalyse

Bei der Datenanalyse für eine neue Datenbank werden zuerst alle projektrelevanten Informationen ermittelt. Das können zum Beispiel Informationen über Arbeitsprozesse, Personen oder verschiedene Objekte sein, die später in der Datenbank erfasst werden sollen. Die Erhebung der Daten kann im Rahmen einer Bedarfsanalyse, einer Fragebogenaktion oder ganz einfach in einem oder mehreren Gesprächen mit dem Auftraggeber erfolgen. Am Ende dieser Phase steht eine Anforderungsliste, die mit dem Auftraggeber abgestimmt ist und als Vorlage für den Datenbankentwurf dient. Aus den erhobenen Daten werden dann Entitäten mit ihren Attributen und deren Beziehungen untereinander abgeleitet und dargestellt.

6.2.2 Entity-Relationship-Modell (ER-Modell)

Das Entity-Relationship-Modell (oder auch ER-Modell) wird dazu benutzt, um Informationen darzustellen, die bei der Datenanalyse erhoben wurden. Dieses Modell kommt u. a. in der Planungsphase zum Einsatz und dient zur Verständigung zwischen dem Anwender und dem Entwickler der Datenbank.

Das ER-Modell wurde im Jahre 1976 das erste Mal von Peter Chen vorgestellt und ist inzwischen zum

Bildquellenverzeichnis

ACI EDV-Systemhaus GmbH & Co KG, Lüneburg: 10, 93, 95, 104, 131, 133, 137, 139, 149
adpic Bildagentur, Köln: 65, 65, 67, 67, 67, 67, 67, 67, 67
Informationstechnikzentrum Bund - ITZBund, Bonn: 104
iStockphoto.com, Calgary: bagotaj Titel; Gaul, Pawel 9
Lithos, Wolfenbüttel: 9, 11, 12, 12, 12, 12, 30, 38, 65, 66, 75, 81, 81, 81, 81, 81, 81, 314, 379
stock.adobe.com, Dublin: Pakmor 106; Rudie 98; trodler1 66
The Apache Software Foundation / Apache Open Office PMC: 283
Valentinelli, Mario, Rostock: 28, 46, 66, 66, 66, 66, 114