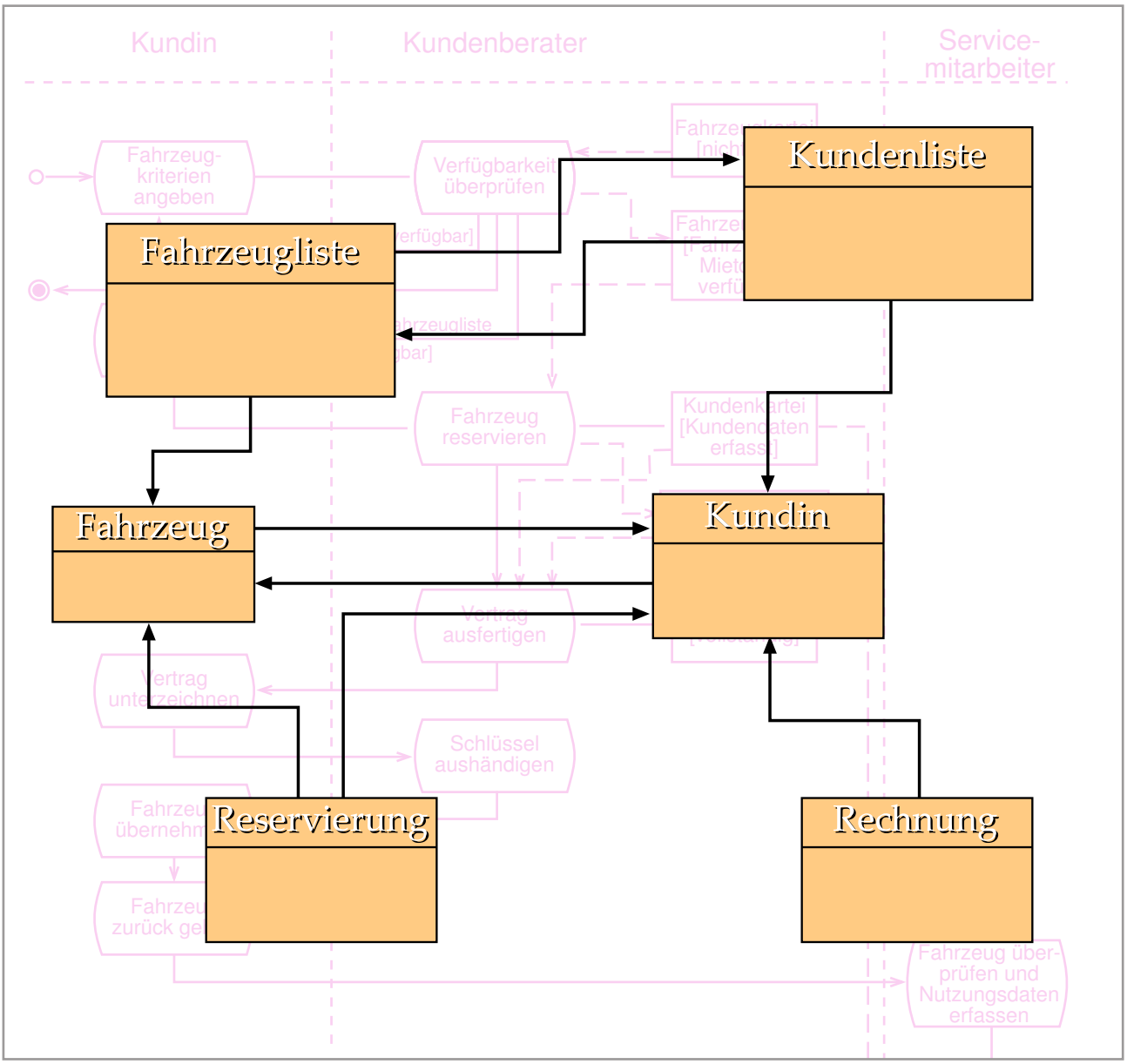
FernUniversität in Hagen

Objektorientiertes Programmieren

Lerneinheit 1: Von der Aufgabenstellung zum Programm

Autor: Bernd J. Krämer



Kursnummer: 20022

Vorwort

Willkommen zur zweiten, leicht erweiterten Auflage des Kurses **Objektorientierte Programmierung**. Der Kurs führt Sie in die Grundlagen der objektorientierten Programmierung ein. Grundlegende Strukturbegriffe wie →Objekt, →Klasse, →Vererbung, →Schnittstelle und alle wichtigen Programmierkonzepte im Kleinen werden an Hand einer Fallstudie motiviert, dann allgemein bestimmt und anschließend konkret in der →Java-→Syntax eingeführt.

Dieser Kurs soll Sie in die Lage versetzen, eine gegebene Aufgabe inhaltlich zu durchdringen, d. h. die vorgegebene oder erwünschte Funktionalität zu verstehen und so exakt und umfassend wie möglich zu dokumentieren, aber auch die bei der Programmentwicklung zu berücksichtigenden Randbedingungen zu erfassen.

Diese Vorgaben werden dann systematisch in ein lauffähiges Java-Programm umgesetzt. Dabei entwerfen und implementieren Sie einzelne Klassen, die zusammen mit den von uns vorgegebenen Klassen ein eigenständiges Programmsystem ergeben, das die Abläufe in einem Autoverleihbetrieb unterstützt.

Das Kursmaterial wird sowohl im HTML-Format für die Online-Nutzung als auch im PDF-Format für einen Ausdruck auf Papier angeboten. Wir empfehlen Ihnen dringend, die Online-Fassung wenigstens gelegentlich zu nutzen, weil das Modul zahlreiche interaktive und multimediale Lernelemente umfasst, die zum eignen Handeln anregen und damit den Lernprozess fördern sollen. Darüber hinaus stellen wir Ihnen didaktisch geeignete Modellierungs- und Programmierwerkzeuge zur Verfügung, die Sie zur Lösung Ihrer Aufgaben auf die eigenen PC laden können.

Kursziele

Dieser Kurs

- bietet eine *Einführung in die Programmierung* an Hand der Programmiersprache *Java*,
- erläutert die *Grundlagen des objektorientierten Programmentwurfs*,
- vermittelt *Kenntnisse elementarer Sprachkonzepte*,
- behandelt *Datenstrukturen* und *Algorithmen* und
- erklärt die Programmein- und -ausgabe mittels Dateien.

Weiterführende Programmierkonzepte wie die Programmierung nebenläufiger und verteilter Anwendungen werden Sie in im Folgekurs „Programmierkonzepte“ kennen lernen.

Vorgehensweise

Anstatt Ihnen eine Vielzahl unterschiedlicher Beispiele zur Erläuterung der wesentlichen Konzepte der Programmierung zu präsentieren, werden wir eine größere Anwendung im Stil eines Projekts Schritt für Schritt entwickeln und auf diesem Weg die Konzepte und Techniken darlegen, die wir bei jedem Schritt brauchen. Sie werden dabei lernen, wie man:

- eine gegebene Aufgabe genau versteht,
- sie gedanklich in ihre wesentlichen Aspekte zerlegt und

- systematisch in ein Programm umgesetzt.

Am Ende des Kurses werden Sie ein ablauffähiges Java-Programm konstruiert haben, das die Anforderungen der Anwendungsaufgabe möglichst vollständig und angemessen verwirklichen sollte.

Zur Rolle von Java

Die Lerneinheiten dieses Kurses sind nicht als Programmieranleitung für Java ausgelegt. Viele Einzelheiten der Sprache Java werden deshalb hier nicht behandelt, denn die Betonung des Kurses liegt auf der Vermittlung grundlegender Programmierbegriffe und Methoden der objektorientierten Programmierung (OOP). Java dient vor allem als konkrete Schreibweise für die Beispiele und Übungen des Kurses. Alle Sprachkonstrukte, die wir dafür benötigen, werden auch im Kurs erläutert.

Weitergehende Java-Kenntnisse können Sie aus einer ganzen Reihe ausgezeichneter Lehrbücher erwerben, die Sie in alle Details der Sprache einführen. Im Abschnitt Hintergrundmaterial finden Sie kommentierte Hinweise auf deutsche und englische Lehrbücher zu Java sowie auf Online-Quellen im Internet.

Bevor Sie sich mit den eigentlichen Lerneinheiten des Kurses befassen, sollten Sie sich den Abschnitt Informationen zum Kurs ansehen, in dem Sie Hinweise zum Veranstaltungsplan, zu erforderlichen Vorkenntnissen, zu technischen Anforderungen, zur Leistungsbewertung und zu den Konventionen finden, die wir in diesem Kurs verwenden.

Lernziele

Nach Abschluss dieses Kursteils sollen Sie:

- die Grundbegriffe und besonderen Merkmale der objektorientierten Programmierung erklären und anwenden können,
- für eine gegebene Aufgabe geeignete Datenstrukturen und Algorithmen entwickeln können, d. h. den Umgang mit Daten systematisch zu gestalten und programmierbare Lösungsmethoden zu entwerfen, die eine Aufgabe in endlichen Schritten bewältigen,
- Grundkenntnisse der Sprache Java beherrschen und in der Lage sein, die Datenstrukturen und Algorithmen Ihres Entwurfs in Java umzusetzen,
- ein gegebenes Java-Programm lesen und verändern können und
- das Prinzip der Rekursion in Programmen verstehen und anwenden können.

Voraussetzungen

Technische Voraussetzungen

Der Kurs „Objektorientiertes Programmieren“ integriert verschiedene Multimediaformate wie Farbgrafiken, Animationen, →Java-Applets und kurze Videoausschnitte und sollte deshalb nicht nur in der Papierversion studiert werden.

An technischen Voraussetzungen erwarten wir, dass Sie über folgende technische Ausstattung verfügen:

Technische Voraussetzungen

- einen leistungsstarken PC oder Arbeitsplatzrechner mit modernem Betriebssystem (Windows NT, Windows 95 oder höher, Mac OS 8 und höher, Linux oder Unix),
- eine Internet-Verbindung,
- aktuelle Browsersoftware,
- Tonunterstützung und →Browser-Plugins zur Mediendarstellung (eine aktuelle Liste der benötigten Plugins und Verweise auf entsprechende Softwarequellen finden Sie im Abschnitt Software).

Wir setzen auf den Kursseiten auch JavaScript ein. Ihr Browser muss JavaScript unterstützen, und JavaScript muss aktiviert sein, damit die Verweise auf Abbildungen, Tabellen und Glossarbegriffe korrekt verfolgt werden können.

Vorkenntnisse und Mitarbeit

Um diesen Kursteil studieren zu können, benötigen Sie Vorwissen zum Aufbau und zur Funktionsweise von Computersystemen und Kenntnisse über Dienste im Internet, die z. B. im Kurs „Rechnertechnik[1]“ gelehrt werden.

Die regelmäßige und selbständige Bearbeitung des Kursmaterials und der Einsendeaufgaben sind wichtige Voraussetzungen für den erfolgreichen Abschluss des Kurses.

Darüber hinaus erwarten wir, dass Sie sich weiterführend Literatur und Online-Informationen zu Details der Sprache Java selbständig erschließen. Im Abschnitt Hintergrundmaterial finden Sie kommentierte Quellenhinweise.

Sie sollten auch in der Lage sein, die auf unserem Server bereit gestellten Programmwerkzeuge oder eine selbst gewählte Programmierumgebung für Java auf Ihrem Rechner zu installieren und effektiv zu nutzen.

Studierhinweise

Informationen zum Kurs

Dieser Kurs führt in die Grundbegriffe und Techniken der objektorientierten Programmierung ein. Er beschreibt den Weg der Programmentwicklung von der Aufgabenanalyse über den objektorientierten Entwurf hin zur Konstruktion ausführbarer Programme. Zur Darstellung der Ergebnisse der Aufgabenanalyse und Programmentwürfe verwenden wir verschiedene Diagrammarten der standardisierten Beschreibungstechnik →UML (Unified Modeling Language).

UML

1 http://www.ice-bachelor.fernuni-hagen.de/lehre/k20046/

Programmbeispiele und Übungen werden in der Programmiersprache →Java angegeben. Eine Einführung in grundlegende Sprachkonstrukte und -konstruktionen von Java runden den Kurs ab.

Verfügbarkeit

Dieser Kurs ist über das Internet zugänglich und wird ständig auf dem Laufenden gehalten. Wir raten Ihnen daher dringend, sich gelegentlich über die letzten Änderungen[2] zu informieren und Ihre persönliche Kopie des Kursmaterials dem aktuellen Stand anzupassen.

Kursbearbeitung

Wir erwarten, dass Sie regelmäßig und selbständig das Kursmaterial und die Aufgaben bearbeiten. Anfängern raten wir, den Kurs sequenziell durchzuarbeiten, da die einzelnen Abschnitte inhaltlich aufeinander aufbauen.

Falls Sie mit dem Stoff schon vertraut sind, können Sie sich mit Hilfe des Inhaltsverzeichnisses frei durch den Lernstoff bewegen.

Um einige der Aufgaben bearbeiten zu können, ist es notwendig, die auf unserem Server bereitgestellten Programmwerkzeuge oder aber selbst gewählte Editier- und Programmierumgebungen für UML bzw. Java zu installieren und einzusetzen.

Weitere Information

Weitere Information über

- erwartete Vorkenntnisse und notwendige technische Voraussetzungen,
- Lernziele,
- Kursautor und Betreuer,
- Konventionen und
- das Glossar

finden Sie auf den folgenden Seiten.

Literatur zu Java

Da es eine Vielzahl hervorragender Lehrbücher und im Internet verfügbarer Informationen über Java gibt, halten wir die Ausführungen zum Aufbau der Sprache Java und ihren zahlreichen Sprachkonstrukten kurz. Sie sind deshalb gehalten, sich weitere Details zur Sprache selbst zu beschaffen.

Eine kommentierte Liste weiterführender Lehrmaterialien mag Sie bei der Auswahl leiten.

Programmierhinweise und -konventionen

Die Lerneinheiten dieses Kurses werden ergänzt um Hinweise und Merkregeln, Programmier- und Formatierhinweise sowie Hinweise auf übliche Kodierprobleme. Diese sollten Sie sich gut einprägen und bei der Programmierung immer vor Augen halten:

1. Verwenden Sie viel Aufmerksamkeit für die Anwendungserhebung und den Programmentwurf.

2 http://www.ice-bachelor.fernuni-hagen.de/lehre/k20022/#news

2. Vereinfachen Sie arithmetische und logische Ausdrücke, bevor Sie die entsprechenden Programmausdrücke entwickeln!
3. Kommentieren Sie Ihr Programm von Anbeginn an!
4. Fragen Sie sich, ob Sie das Programm verstünden, wenn sie nicht die Person wären, die es geschrieben hat!
5. Scheuen Sie sich nicht, von vorne zu beginnen!
6. Versichern Sie sich immer wieder, dass Kommentare und Programmkode miteinander übereinstimmen und der Programmspezifikation entsprechen!
7. Manche Softwarefachleute vertreten die Auffassung, dass ein Programmentwickler zumindest so viel Testkode schreiben muss, wie er Produktkode entwickelt hat. Wir werden nicht so rigoros sein, verlangen von Ihnen jedoch ab der fünften Lerneinheit, die Sie mit Testverfahren verraut machen wird, dass Sie Ihre Programme ausgiebig testen und die Testfälle sowie die Testergebnisse dokumentieren.
8. Testcode muss auf alle Fälle ebenso dauerhaft aufbewahrt werden, wie Analyse- und Entwurfsdokumente und die Anwendungsprogramme selbst.

Konventionen

Bei der Programmentwicklung sind Sie auch gehalten, eine Reihe von Konventionen einzuhalten, die sich in der Praxis bewährt haben und die Lesbarkeit von Programmen erhöhen.

Autoren- und Betreuervorstellung

Abb. 1: Prof. Dr.-Ing. Bernd J. Krämer
Leiter des Lehrgebiets DVT

Professor Krämer ist der Autor dieses Kurses. Er leitet das Lehrgebiet Datenverarbeitungstechnik[3] im Fachbereich Elektrotechnik und Informationstechnik[4] der FernUniversität in Hagen[5].

Abb. 2: Dipl.-Ing. Alexander Stuckenholz
wissenschaftlicher Mitarbeiter am Lehrgebiet DVT

Alexander Stuckenholz[6] ist wissenschaftlicher Mitarbeiter am Lehrgebiet Datenverarbeitungstechnik[7]. Er ist der Hauptbetreuer für diesen Kurs.

Danksagung

Ohne die Hilfe weiterer Mitarbeiterinnen und Mitarbeiter wäre der Kursteil nicht zustande gekommen. Besonders erwähnen möchte ich:

- Frau Poerwantoro[8], die zusammen mit Herrn Fakher viele der Programmbeispiele dieses Kurses entwickelte,
- Herrn Kötter[9], der Übungsaufgaben beitrug, und
- Herrn Winkler[10], der für die Animationen und andere Gestaltungselemente verantwortlich zeichnet.

Diese zweite Auflage des Kurses wurde mit dem neuen, auf der erweiterbaren Dokumentenauszeichnungssprache XML[11] aufbauenden Autorensystem FuXML der FernUniversität erzeugt. Ich hatte das Privileg, das Werkzeug noch während sei-

3 http://www.fernuni-hagen.de/DVT

4 http://www.et-online.fernuni-hagen.de/welcome.html

5 http://www.fernuni-hagen.de/

6 http://www.fernuni-hagen.de/DVT/Mitarbeiter/stuckenholz.html

7 http://www.fernuni-hagen.de/DVT/

8 http://www.fernuni-hagen.de/DVT/Mitarbeiter/poerwantoro.html

9 http://www.fernuni-hagen.de/DVT/Mitarbeiter/koetter.html

10 http://www.fernuni-hagen.de/DVT/Mitarbeiter/winkler.html

11 http://www.w3.org/XML/

ner Entwicklungszeit einsetzen zu können. Der Kurs konnte so gleichzeitig mit der Freigabe des Autorensystems erscheinen. Der Entwicklergruppe bin ich dafür sehr dankbar. Hervorheben möchte ich stellvertretend für die vielen Entwicklerinnen und Entwickler Herrn **G. Steinkamp**, der mich mit fortdauerndem Engagement und viel Geduld durch die Klippen der Pioniernutzung führte.

Hintergrundmaterial

Software

Wir stellen Ihnen verschiedene Programmwerkzeuge zur Verfügung, die Sie zur Bearbeitung der Entwurfs- und Programmierarbeiten verwenden können.

BlueJ

BlueJ[12] ist eine Programmierumgebung für Java. Mit BlueJ stellen wir Ihnen eine Lern- und Programmierumgebung für Java vor, mit deren Hilfe Sie das Erlernte durch Übungen vertiefen können. BlueJ bietet eine didaktisch elegante Möglichkeit[13], in die objektorientierte Programmierung mit Java einzusteigen[14]. Sie werden auch ohne Schwierigkeiten in der Lage sein, sich eigene Übungen auszudenken. Einen Bildschirmabzug zu BlueJ sehen Sie in Abb. 2.

12 http://www.bluej.org

13 http://www.ice-bachelor.fernuni-hagen.de/lehre/k20022/tools/BlueJ/Dokumente/2003-12-CSEd-bluej.pdf

14 http://java.sun.com/features/2002/07/bluej.html

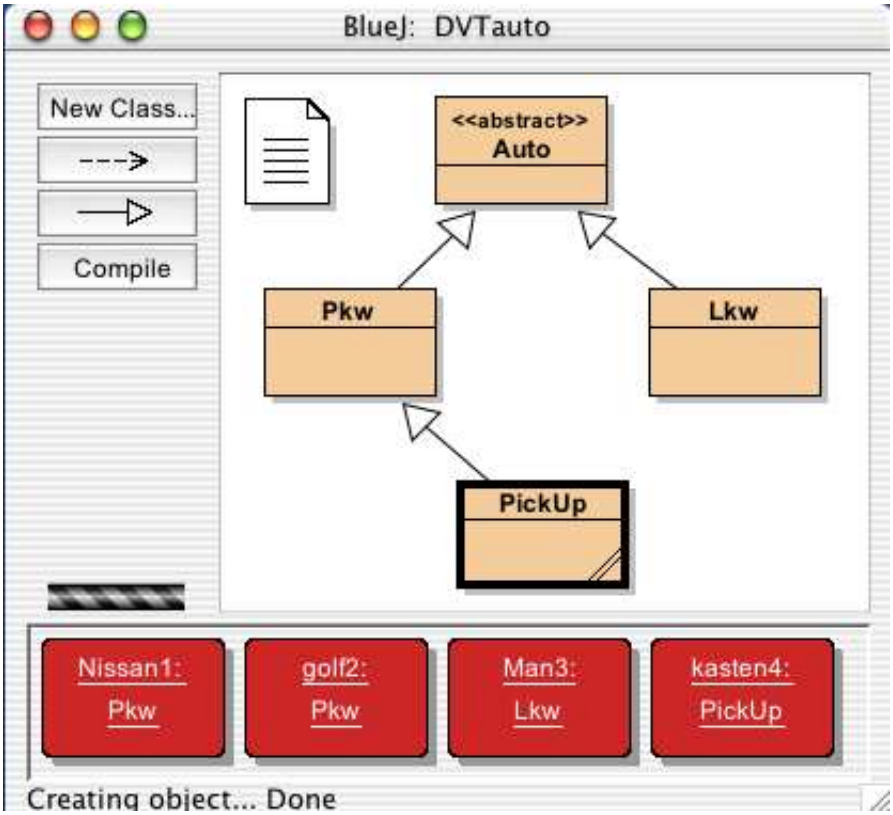


Abb. 2: Das Hauptfenster von BlueJ. Links: die Steuer- und Anzeigekommandos, Mitte: vier Klassen mit Vererbungsbeziehung, unten: zwei Exemplare der Klasse PKW und je eines der Klassen LKW und PickUp

BlueJ wurde an der „Faculty of Information Technology[15]“ der Monash University[16] in Melbourne, Australien, speziell im Hinblick auf den Einsatz in der Lehre entwickelt. Einer der Hauptautoren, Dr. Michael Kölling[17], ist jetzt Professor für Informatik an der Süddänischen Universität[18] und entwickelt BlueJ auch dort weiter. Michael Kölling ist auch Mitautor des Buchs „Objects First with Java[19]“.

BlueJ beinhaltet nützliche Werkzeuge zur Erstellung von Java-Programmen wie Editor, Compiler, virtuelle Maschine (engl. *virtual machine*) und ein Programm zur Eingrenzung von Fehlern (engl. *debugger*).

Eigenschaften von BlueJ

BlueJ ist nach objektorientierten Grundsätzen aufgebaut. BlueJ unterstützt eine graphische Klassendarstellung, die mit einem Klassen-Browser bearbeitet werden kann. Auch die interaktive Veränderung von Klassendefinitionen und Klassenstrukturen in der graphischen als auch in der Textdarstellung ist möglich. Objekte können per Mausklick erzeugt und die Methoden eines jeden Objekts können einzeln aufgerufen werden. Damit haben Sie die Möglichkeit, Ihre Objekte ohne Kenntnis

15 http://www.infotech.monash.edu.au/home.html

16 http://www.monash.edu.au/

17 http://www.mip.sdu.dk/~mik/

18 http://www.mip.sdu.dk/

19 http://www.mip.sdu.dk/~mik/homepage/publications.html

weiterer Sprachkonzepte und Umgebungsbedingungen wie etwa der Existenz einer `main`-Funktion, ausführen und in ihren Eigenschaften und ihrem Verhalten verändern zu können.

Weitere Vorteile von BlueJ sind die einfach gehaltene Bedienoberfläche und die Möglichkeit, die Programmierumgebung auf Ihren Rechner übertragen und kostenfrei nutzen zu können.

Download

BlueJ liegt in Form einer ausführbaren Jar-Datei auf den BlueJ-Seiten[20] der Monash University[21] und auf dem DVT-Server zum Herunterladen[22] bereit.

Eine englischsprachige Bedienungsanleitung[23] im PDF-Format ist auf den BlueJ-Seiten oder auf unseren Kursseiten[24] verfügbar.

Installation

Um BlueJ verwenden zu können, benötigt man lediglich eine Java 2 Plattform[25] (früher: Java Development Kit, JDK) der Firma Sun Microsystems[26], die ebenfalls unentgeltlich angeboten wird. Vorausgesetzt wird mindestens die Version 1.2.2, empfohlen wird die Version 1.3. Für Windows-, Linux-, Unix-Rechner und für Apple[27]-Rechner mit MacOS X[28]kann die Plattform von den BlueJ-Seiten[29] oder vom Kursserver heruntergeladen werden.

Bei Aufruf des Zeilenkommandos

```
java -jar bluej-112.jar
```

wird man aufgefordert, das Ziel- und das JDK-Verzeichnis anzugeben.

Nach erfolgreichem Abschluss der Installation befindet sich im Ziel-Verzeichnis eine Skriptdatei, welche die Anwendung durch Eingabe von

```
bluej
```

20 http://bluej.monash.edu.au/download/download.html

21 http://bluej.monash.edu.au/

22 http://bluej.monash.edu.au/download/download.html

23 http://www.bluej.org/doc/documentation.html

24 http://www.ice-bachelor.fernuni-hagen.de/lehre/k20022/tools/BlueJ/Dokumente/BlueJ-reference.pdf

25 http://java.sun.com/j2se/1.3/

26 http://java.sun.com/

27 http://www.apple.com/de/

28 http://www.apple.com/macosx/

29 http://www.bluej.org/download/download.html

in der Kommandozeile oder auch durch Doppelklick bei Windows startet.

Erste Schritte

Nach dem Start von BlueJ müssen Sie zunächst ein Projekt laden oder ein neues Projekt erstellen. BlueJ legt für jedes Projekt ein Verzeichnis an, in dem Quelltexte, Bytecode und weitere projektrelevante Daten abgelegt werden. Es besteht die Möglichkeit, die Quelltexte zu exportieren und mit anderen Werkzeugen weiter zu bearbeiten oder vorhandene Java-Quelltexte zu importieren.

Nach Erzeugen eines neuen Projektes kann der Anwender in einer Mischung aus grafischer und textueller Bearbeitung Klassen und Schnittstellen definieren.

Für die Übungen mit BlueJ erhalten Sie weitere Bedienungshinweise.

Diagramm-Editor

Ded ist ein Diagrammeditor zur grafischen Dokumentation und dient der Darstellung:

- der Ergebnisse der Rekonstruktion der Anwendungswelt,
- abstrakter Modelle des zu entwickelnden Programmsystems sowie
- konkreter Entwürfe eines entstehenden Programmsystems.

Der Diagrammeditor wurde von **Helge Dinse** am Lehrgebiet DVT entworfen und in der Sprache Java implementiert. Sie können das Werkzeug auf Ihren PC übertragen und lokal nutzen. Der Diagrammeditor ist so gestaltet, dass Sie zunächst nur eine Diagrammart bearbeiten können. Weitere Editierfunktionen können Sie dann zum gegebenen Zeitpunkt nachladen.

Shockwave- Plugin

Zum Abspielen der Animationen der ersten Kurseinheit benötigen Sie ein Shockwave-Plugin[30] für Ihren Browser. Aus lizenzrechtlichen Gründen können wir dieses Plugin nicht auf unserem Server bereithalten. Sie können es aber kostenfrei von der Webseite des Herstellers Macromedia[31] beziehen.

JDK

Zur Entwicklung von Java-Programmen bietet die Fa. Sun Microsystems[32] kostenlos das →Java Development Kit (JDK) an, mit dessen Hilfe Java-Programme entwickelt werden können. Daneben gibt es eine Reihe kommerzieller Java-Entwicklungsumgebungen, die meist, anders als JDK, eine integrierte Programmierumgebung einschließlich Fehlerbehebungswerkzeugen (engl. *debugger*) und grafischer Oberfläche anbieten.

Lehrbücher

Alle nachstehend aufgeführten Bücher können Sie aus der Universitätsbibliothek entleihen.

[Arnold99] Ken Arnold and James Gosling.
The Java Programming Language: Second Edition.

30 http://sdc.shockwave.com/shockwave/download/

31 http://www.macromedia.com/

32 http://www.sun.com/

Addison-Wesley Longman 1999

Gosling ist der Kopf hinter Java; umfangreiches Referenzwerk aus der Java-Serie der Fa. Sun, die die Java-Entwicklung betreibt und die Rechte an der Sprache besitzt.

[Barnes02] David J. Barnes and Michael Kölling.
Objects First with Java: A Practical Introduction using BlueJ[33].
Prentice Hall 2002

Die Einführung objektorientierter Programmiertechniken am Beispiel von Java folgt der Grundgedanken der BlueJ-Programmierumgebung und stellt den Umgang mit Objekten in den Vordergrund; eine empfehlenswerte Ergänzung zum Kurs.

[Doberkat99] Ernst-Erich Doberkat und Stefan Dißmann:
Einführung in die objektorientierte Programmierung mit Java.
Oldenbourg 1999

Entstand aus einer Lehrveranstaltung an der Universität Dortmund; stellt den Entwurf sequenzieller Algorithmen und wichtiger Datenstrukturen in den Vordergrund und erläutert die Vorgehensweise bei der objektorientierten Analyse und beim Entwurf.

[Echtle00] Klaus Echtle und Michael Goedicke:
Lehrbuch der Programmierung mit Java.
dpunkt Verlag 2000

Wird als Lehrbuch an der Universität Essen in der Programmierausbildung für Informatikerinnen und Nicht-Informatiker eingesetzt; beschreibt die Methoden der objektorientierten Programmierung und illustriert sie an Java-Beispielen; beginnt mit einer gut gelungenen Einführung in die Grundbegriffe der Informatik.

[Poetzsch-Heffter00] Arnd Poetzsch-Heffter:
Objektorientierte Programmierung: Mit einer Einführung in Java.
dpunkt Verlag 2000

Entstand aus einer Lehrveranstaltung am Fachbereich Informatik der FernUniversität; umfasst neben einer Einführung in die Grundkonzepte der objektorientierten Programmierung auch Beiträge zur Wiederverwendung von Programmen in der Form von Bibliotheksbausteinen und Programmgerüsten (engl. *framework*); geht auch auf die parallele und verteilte Programmierung ein.

33 http://www.mip.sdu.dk/~mik/homepage/publications.html

[Sedgewick02] Robert Sedgewick:
Algorithms in Java. Third Edition
Addisson Wesley 2002

Behandelt die Grundlagen von Datenstrukturen und untersucht zahlreiche Varianten von Such- und Sortieralgorithmen.

[Züllighoven98] Heinz Züllighoven:
Das objektorientierte Konstruktionshandbuch.
dpunkt.verlag 1998

Dieses Buch ist keine Einführung in die Programmierung und bezieht sich auch nicht auf Java. Es ist ein Handbuch zur Entwicklung großer objektorientierter Softwaresysteme, das die Begriffe Werkzeug und Material in den Vordergrund der Betrachtung stellt.

Quellen im Internet

Für die Gültigkeit der Verweise können wir über einen längeren Zeitraum hinweg leider keine Gewähr übernehmen.

Bücher

1. The Java Language Specification[341]
2. The Java Virtual Machine Specification[352]
3. The Java Tutorial[363] (Version zum Herunterladen[374])
4. Java 2 SDK, Standard Edition Documentation[385]
5. JDK 1.4 Documentation[396]

---

34 http://java.sun.com/docs/books/jls/second_edition/html/j.title.doc.html

1 http://java.sun.com/docs/books/jls/second_edition/html/j.title.doc.html (zuletzt besucht: 20.12.2003)

35 http://java.sun.com/docs/books/vmspec/index.html

2 http://java.sun.com/docs/books/vmspec/index.html (zuletzt besucht: 20.12.2003)

36 http://java.sun.com/docs/books/tutorial/

3 http://java.sun.com/docs/books/tutorial/ (zuletzt besucht: 20.12.2003)

37 http://java.sun.com/docs/books/tutorial/information/download.html

4 http://java.sun.com/docs/books/tutorial/information/download.html (zuletzt besucht: 20.12.2003)

38 http://java.sun.com/j2se/1.4.2/docs/index.html

5 http://java.sun.com/j2se/1.4.2/docs/index.html (zuletzt besucht: 20.12.2003)

39 http://java.sun.com/j2se/1.4/index.jsp

6 http://java.sun.com/j2se/1.4/index.jsp (zuletzt besucht: 20.12.2003)

6. JavaProgrammierhandbuch und Referenzfür die Java-2-Plattform, Standard Edition[407]

Weitere Informationsquellen

1. Java-Seiten der Fa. Sun[418]
2. Java-Lernzentrum der Fa. Sun[429]
3. Index häufig gestellter Fragen[4310]
4. Draft Java Coding Standard[4411] (von Doug Lea)
5. Java unter Yahoo[4512]
6. Java-Quellcode und Information bei Gamelan[4613]
7. Dokumentation der Java-Anwendungsschnittstellen (API)[4714]
8. foldoc : Ein frei zugängliches Wörterbuch zu Informatikbegriffen und Abkürzungen[4815]
9. UML-Tutorial der Univ. Magdeburg[4916]
10. Java-World Zeitschrift[5017]

---

40 http://www.dpunkt.de/java/

7 http://www.dpunkt.de/java/ (zuletzt besucht: 1.3.2004)

41 http://java.sun.com/

8 http://java.sun.com/ (zuletzt besucht: 20.12.2003)

42 http://java.sun.com/learning/new2java/index.html

9 http://java.sun.com/learning/new2java/index.html (zuletzt besucht: 20.12.2003)

43 http://java.sun.com/docs/faqindex.html

10 http://java.sun.com/docs/faqindex.html (zuletzt besucht: 20.12.2003)

44 http://gee.cs.oswego.edu/dl/html/javaCodingStd.html

11 http://gee.cs.oswego.edu/dl/html/javaCodingStd.html (zuletzt besucht: 20.12.2003)

45 http://dir.yahoo.com/Computers_and_Internet/Programming_and_Development/Languages/Java/

12 http://dir.yahoo.com/Computers_and_Internet/Programming_and_Development/Languages/Java/ (zuletzt besucht: 20.12.2003)

46 http://softwaredev.earthweb.com/java

13 http://softwaredev.earthweb.com/java (zuletzt besucht: 20.12.2003)

47 http://java.sun.com/j2se/1.3/docs/api/index.html

14 http://java.sun.com/j2se/1.3/docs/api/index.html (zuletzt besucht: 20.12.2003)

48 http://foldoc.doc.ic.ac.uk/foldoc/

15 http://foldoc.doc.ic.ac.uk/foldoc/ (zuletzt besucht: 20.12.2003)

49 http://ivs.cs.uni-magdeburg.de/~dumke/UML/index.htm

16 http://ivs.cs.uni-magdeburg.de/~dumke/UML/index.htm (zuletzt besucht: 20.12.2003)

50 http://www.javaworld.com/

17 http://www.javaworld.com/ (zuletzt besucht: 20.12.2003)

Glossar

Das Begriffsverzeichnis oder Glossar enthält kurze Begriffserklärungen für die wichtigsten Fachbegriffe, die in diesem Kurs verwendet werden.

abstract	**abstract** ist ein Schlüsselwort der Sprache →Java. Es wird in →Klassendefinitionen benutzt, um syntaktisch →abstrakte Klassen auszuzeichnen und somit auszudrücken, dass von diesen Klassen keine →Objekte erzeugt werden können. Ihre →Methoden und →Attribute können von →Unterklassen nur geerbt und in diesen Unterklassen implementiert werden.
Abstrakte Klasse	Eine **abstrakte Klasse** definiert die Gemeinsamkeiten einer Gruppe von →Unterklassen (engl. *subclass*). Sie kann eine oder mehrere →abstrakte Methoden enthalten. Andere Klassen können abstrakte Klassen erweitern und müssen abstrakte Methoden implementieren. Von einer abstrakten Klasse können keine →Objekte erzeugt werden.
Abstrakte Methode	Eine **abstrakte Methode** (engl. *abstract method*) wird in der Klasse, die sie einführt, nur mit Namen, Parameter→typen und Ergebnis→typ definiert aber nicht durch einen Methodenrumpf implementiert. Sie kann nur in →Unterklassen der einführenden Klasse implementiert werden. Enthält eine Klasse eine oder mehrere abstrakte Methoden, muss die Klasse selbst als →abstrakte Klasse vereinbart werden.
Abstrakte Operation	siehe →abstrakte Methode.

Abstraktion

Abstraktion bezeichnet

1. einen gedanklichen Vorgang, der darauf abzielt, wesentliche Merkmale und Eigenschaften eines Gegenstands oder Begriffs zu ermitteln und Unwesentliches zu unterdrücken (engl. *abstraction*);

2. die Festlegung der Funktionalität von Klassen, ohne eine Implementierung der Methoden und Attribute der Klassen anzugeben (engl. *interface*). Abstraktionen werden durch Klassen implementiert.

abstrakter Datentyp (ADT)

Ein **abstrakter Datentyp** (ADT, engl. *abstract data type*) besteht aus:

- einem Namen, der den ADT bezeichnet,
- Typen von Datenelementen,
- Operationen und
- einer Beschreibung der Eigenschaften dieser Operationen.

Der innere Aufbau (die Datenstruktur) des ADT ist nicht nach außen sichtbar. Eigenschaften des ADT können nur mit Hilfe der verfügbaren Operationen abgefragt und verändert werden. Die Eigenschaften können auf verschiedene Weise abstrakt, also unabhängig von einer konkreten Darstellung der eingekapselten Datentypen und unabhängig von einer konkreten Implementierung der Operationen festgelegt werden.
Die **algebraische Datenabstraktion** ist eine, bei die Eigenschaften der Operationen durch eine eine endliche Menge von Gesetzen in der Form von Gleichungen, bedingten Gleichungen oder Prädikaten angegeben werden. Diese Gesetze oder Axiome beschreiben die Wechselwirkungen der verschiedenen Operationen auf die Datenelemente des ADT. Man kann aber auch →Invariante sowie →Vorbedingungen und →Nachbedingungen für einzelne Operationen festlegen.

Ada

Die Programmiersprache **Ada** ist eine Mehrzwecksprache, die sich insbesondere auch für die Entwicklung von Programmen eignet, die in technische Anwendungen eingebettet sind. Ada umfasst eine Vielzahl von Programmierkonstrukten und ermöglicht auch die Programmierung →nebenläufiger und →verteilter Abläufe.
Die Sprache Ada, die im Jahr 1979 veröffentlicht wurde, wird von manchen auch als die letzte Elefantensprache bezeichnet. Entwicklung und Einsatz von Ada wurden vom US-amerikanischen Verteidigungsministerium voran getrieben.

Akteur

Ein **Akteur** (engl. *actor*) ist ein Benutzer oder eine Benutzerin eines zu implementierenden Systems aus der Sicht eines bestimmten →Anwendungsfalls. Ein Akteur ist nicht Teil des Systems, sondern interagiert mit ihm. Ein Akteur kann eine Person oder ein anderes System sein.

Der Begriff Akteur bezeichnet eine Rolle einer Systembenutzerin oder eines externen Programms, das direkt mit dem betrachteten System als Teil einer zusammenhängenden Arbeitseinheit zusammenwirkt. Ein realer Handlungsträger kann mehrere Rollen in den Anwendungsfällen eines Systems wahrnehmen und kann daher durch verschiedene Akteure dargestellt sein.

Aktivitätsdiagramm

Ein **Aktivitätsdiagramm** (engl. *activity diagram*) ermöglicht die übersichtliche Darstellung der verschiedenen Schritte in einem Arbeitsablauf. Aktivitätsdiagramme erlauben es, alternative und →nebenläufige Aktivitäten zu beschreiben.

Aktivitätsdiagramme werden vorwiegend benutzt, um Abläufe zu erfassen, die sich über verschiedene Anwendungsfälle und Akteure erstrecken. Sie fördern die Entdeckung nebenläufiger Handlungsstränge und verhindern so die unnötige Linearisierung von Abläufen.

Algol	Die Programmiersprache **Algol 60**, erschienen im Jahr 1960, ist nach →Fortran und mit →Cobol die zweitälteste höhere Programmiersprache. Algol 60 ist eine prozedurale Sprache, die mit dem Ziel entwickelt wurde, numerische wissenschaftliche Probleme leichter lösbar zu machen. Algol 60 führte die Blockstruktur sowie rekursive Prozeduren ein.
Algorithmus	Ein **Algorithmus** (engl. *algorithm*) ist eine Verfahrensvorschrift zur Lösung einer Aufgabe. Die Verfahrensvorschrift beschreibt in endlicher Form, welche einzelnen Verarbeitungsschritte bei der Bewältigung der gestellten Aufgabe ganz mechanisch und gesetzesmäßig auszuführen sind.
Analyse	Bei der **Analyse** (engl. *analysis*) werden die Anforderungen eines Auftraggebers an ein zu erstellendes Softwaresystem ermittelt. Das Ergebnis soll die Anforderungen vollständig, eindeutig, präzise und verständlich beschreiben.
Animation	Eine **Animation** ist eine Folge bewegter Bilder, die jedes einzeln berechnet werden und durch geringfügige Unterschiede zwischen aufeinanderfolgenden Bildern den Eindruck einer fließenden Bewegung erzeugen.
Anweisung	Eine **Anweisung** (engl. *statement*) beschreibt einen einzelnen Bearbeitungsschritt, der von einem Programm ausgeführt wird. In Java wird eine elementare Anweisung i. d. R. mit einem Semikolon (;) abgeschlossen.

Anwendungsfall

Ein **Anwendungsfall** (engl. *use case*) ist eine zusammenhängende Einheit von Aktionen, die ein System einem →Akteur außerhalb des Systems zur Verfügung stellt. Die Aktionen werden von Akteuren durch →Botschaftenaustausch angestoßen, um ein bestimmtes Ziel oder Ergebnis zu erreichen. Die Ergebnisse einer Systemaktion werden gleichfalls durch Botschaftenaustausch den Akteuren in der Umgebung des Systems übermittelt.

Die Summe aller Anwendungsfälle zeichnet überblicksartig ein Bild des Systems.

Anwendungsfalldiagramm

Ein **Anwendungsfalldiagramm** (engl. *use case diagram*) besteht aus einer Menge von →Akteuren, →Anwendungsfällen, →Kommunikationsbeziehungen zwischen →Akteuren und Anwendungsfällen sowie aus Verallgemeinerungsbeziehungen wie „erweitern“ (engl. *extend*) und „enthalten“ (engl. *include*), die zwischen Anwendungsfällen oder zwischen Akteuren bestehen können.

Argument

Ein **Argument** (engl. *argument*) oder ein aktueller Parameter (engl. *actual parameter*) bezeichnet einen konkreten Wert, der bei Aufruf und Ausführung einer Methode anstelle eines →Aufrufparameters eingesetzt wird.

Assembler

Assembler-Sprachen sind maschinenorientierte Programmiersprachen, die die hardwarenahe Programmierung erleichtern: Der Binärcode der Maschinensprachen wird durch →mnemonische Symbole dargestellt, sodass er für Menschen verständlich ist.

Für jeden Rechnertyp gibt es spezielle, auf den Befehlsvorrat des Prozessors zugeschnittene Assemblersprachen.

Assoziation	Eine **Assoziation** (engl. *association*) modelliert statische Beziehungen zwischen →Klassen von →Objekten.
Attribut	Ein **Attribut** (engl. *attribute*) ist ein benanntes und typisiertes Datenelement, das in einer →Klasse vereinbart wird und Werte eines bestimmten Typs und damit auch →Zustandsinformation nachhält. Wir unterscheiden →Exemplarattribute oder -variablen, die den Zustand eines einzelnen Attributs erfassen, und →Klassenattribute oder -variablen, die Zustandsinformation aufbewahren, die für alle Objekte einer Klasse sichtbar sind. Jedes aus einer Klasse erzeugte →Objekt besitzt eigene Kopien der in der Klasse vereinbarten Exemplarattribute, die im Allgemeinen bei verschiedenen Objekten einer Klasse mit unterschiedlicher Werten belegt sind. →Klassenattribute gehören zur Klasse und existieren genau einmal. Sie können von den Objekten der Klasse für die Verwaltung gemeinsamer Daten benutzt werden. Jedes Attribut besitzt einen bestimmten →Typ und einen Bezeichner, der innerhalb der Klasse eindeutig ist. Für ein Attribut kann in der Klasse ein Initialwert vereinbart werden, den das jeweilige Attribut bei der Erzeugung eines Objekts dieser Klasse annimmt.
Aufrufparameter	**Aufrufparameter** (engl. *invoking parameter*) ermöglichen es, Daten vom Sender einer →Botschaft zum Empfänger der Botschaft über die aufgerufene →Methode des Empfängers zu transportieren (s.a. →Argument).

Automatische Speicherverwaltung	**Automatische Speicherverwaltung** (engl. *automatic garbage collection*) bedeutet, dass zur Laufzeit eines Programms →Objekte ermittelt werden, die nicht mehr benötigt werden. Die für sie reservierten Speicherbereiche können deshalb frei gegeben werden. Eine Lösung, bei der - wie z. B. bei C++ - der von Objekten beanspruchte Speicherplatz durch den Programmierer verwaltet wird, ist dagegen sehr fehleranfällig. Nahezu die Hälfte aller Programmfehler in →C++ lassen sich auf Fehler in der Speicherverwaltung zurückführen.
AWT	Das **Abstract Window Toolkit** (AWT) des →Java-Systems besteht aus einer Sammlung von Komponenten zur Gestaltung →grafischer Benutzerschnittstellen (GUI, engl. *graphical user interface*). Diese Komponenten wurden mit Hilfe rechnerspezifischer Grundkomponenten implementiert. Die Komponenten fassen Funktionen zusammen, die gängigen Rechner- und Betriebssystemtypen (Sun Solaris, Linux, Windows, Mac OS) gemeinsam sind. Das neuere →Swing-Paket von Java ersetzt und erweitert die AWT-Komponenten.
Botschaft	In einem objektorientierten Programm wird eine →Methode durch die Übermittlung einer **Botschaft** (engl. *message*) an das →Objekt, das die Methode implementiert, aktiviert. Die Botschaft benennt die Methode, die ausgeführt werden soll, und die für die Ausführung der Methode notwendigen →Argumente.
Austausch von Botschaften	Der **Austausch** von **Botschaften** (engl. *message exchange*) bezeichnet einen Kommunikationsvorgang, bei dem eine Verarbeitungseinheit die Kopie eines Auszugs ihrer lokalen Daten als →Botschaft an eine empfangende Verarbeitungseinheit versendet. Der Kommunikationsvorgang erfolgt ggf. über ein Kommunikationsnetz, falls die Verarbeitungseinheiten verteilt sind.

Bündelung	**Bündelung** (engl. *bundling*) bezeichnet den Vorgang der Zusammenfassung einer Menge von →Methoden mit den Daten, auf denen die Methoden arbeiten.
Bytecode	**Bytecode** ist ein maschinenunabhängiger Programmcode, der von jedem Java-→Übersetzer erzeugt und von einem Java-→Interpreter, i. d. R. der →Java virtuellen Maschine, ausgeführt wird.
C	**C** ist eine Systemprogrammiersprache, die im Jahr 1972 erschien und u. a. im Kontext des Betriebssystems Unix eine weite Verbreitung erfuhr.
Cobol	Die Programmiersprache **Cobol**, die im Jahr 1960 erschien, ist nach →Fortran und mit →Algol 60 die zweitälteste höhere Programmiersprache und wurde v.a. für die Programmierung kaufmännischer DV-Anwendungen entwickelt (z. B. Gehaltsabrechnungsprogramme). Cobol 60 führte Dateien und Datenbeschreibungen ein und lehnte sich in seiner →Syntax stark an natürliche Sprachen an. Cobol war viele Jahre lang die am häufigsten gebrauchte Sprache und ist auch heute noch in vielen langlebigen Anwendungen zu finden.
C++	**C++** ist eine Programmiersprache, die 1984 erschien und objektorientierte Sprachkonzepte auf die Sprache →C aufsetzte.
Datenabstraktion	**Datenabstraktion** (engl. *data abstraction*) bedeutet, dass nur die auf ein →Objekt anwendbaren →Operationen nach außen sichtbar gemacht werden, die Implementierung dieser Operationen dagegen verborgen bleibt.
Datenstrom	Ein **Datenstrom** ist eine Verbindung zwischen einer Datenquelle und einer Datensenke.

Datenstruktur	Eine **Datenstruktur** (engl. *data structure*) beschreibt die Art und Weise, wie Daten aus anderen Daten zusammengesetzt sind. Bekannte Datenstrukturen beinhalten Liste, Stapel, Reihe, Baum oder Graph.
Deklaration	Eine Vereinbarung oder **Deklaration** (engl. *declaration*) ist eine Anweisung, die einen Namen und bestimmte Eigenschaften, wie etwa einen →Typ, mit diesem Namen verknüpft.
Eiffel	**Eiffel** ist eine objektorientierte Programmiersprache, die im Jahr 1988 von Bertrand Meyer veröffentlicht wurde und die darauf abzielte, den Programmierprozess zu systematisieren und entlang softwaretechnischer Prinzipien zu organisieren.
Einfachvererbung	Bei der **Einfachvererbung** (engl. *simple inheritance*) besitzt jede Klasse höchstens eine direkte →Oberklasse.
Einkapselung	**Einkapselung** (engl. *encapsulation*) bezeichnet ein wichtiges Prinzip beim Programmentwurf. Es unterstützt die Modularität von Programmen und vereinfacht deren Pflege und Weiterentwicklung. Einkapselung wird ermöglicht durch zwei untergeordnete Konzepte: →Bündelung und →Geheimnisprinzip. Letzteres sorgt dafür, dass der Anwender eines Objekts nichts weiter als die →Signatur und den Ergebnistyp der öffentlichen →Methoden einer →Klasse kennen muss, um ein →Objekt der Klasse benutzen zu können. Die Darstellung seiner Daten und die Implementierung seiner Methoden bleiben verborgen.
enthält-Beziehung	Die **enthält**-Beziehung (engl. *include*) zwischen zwei →Anwendungsfällen - im Sinne *A* schließt *B* ein, geschrieben *A*->*B* - bedeutet, dass der Anwendungsfall *A* den Anwendungsfall *B* umfasst.

erweitert-Beziehung	Eine **erweitert**-Beziehung (engl. *extend*) zwischen zwei →Anwendungsfällen, sagen wir *A* und *B*, bedeutet, dass ein Exemplar des Anwendungsfalls *B* das Verhalten, das für Anwendungsfall *A* angegeben ist, einschließen kann.
Exemplar	Ein **Exemplar** (engl. *instance*) ist eine Ausprägung einer →Klasse, also ein →Objekt. In Java-Programmen wird ein Exemplar der Klasse mit Hilfe des Operators →`new`, gefolgt von einem der →Konstruktoren der Klasse, erzeugt.
Exemplarattribut	**Exemplarattribute** (engl. *instance variable*) dienen dazu, die Eigenschaften und den Zustand der →Exemplare einer →Klasse nachzuhalten. Ein Exemplarattribut hat den in der →Klassendefinition vereinbarten Namen und Typ. Der Name ermöglicht den abfragenden oder verändernden Zugriff auf den Wert eines Exemplarattributs. Die Werte von Exemplarattributen können während der Lebenszeit eines →Objekts verändert werden.
Feld	Ein **Feld** (engl. *array*) ist eine Aneinanderreihung von gleichartigen Elementen, auf dessen Komponenten mittels eines Indexes zugegriffen werden kann. Felder sind →Objekte in →Java.
Fortran	Die Sprache **Fortran** ist eine imperative Programmiersprache für naturwissenschaftliche und Ingenieuranwendungen und wurde Ende der 50er Jahre entwickelt. Die Intention war, dass in dieser Sprache implementierte Programme möglichst schnell abgearbeitet werden sollten, was u. a. durch einen einfachen Aufbau und geringe Strukturierungsmöglichkeiten erreicht wurde. Fortran ist auf den meisten Computern so implementiert, dass die Programme sehr effizient ausgeführt werden können.

Geheimnisprinzip	Das **Geheimnisprinzip** (engl. *information hiding*) bedeutet, dass Eigenschaften und die Implementierung einer Betrachtungseinheit von außen nicht sichtbar sind. Zustandsveränderungen oder Abfragen über den Zustand der Betrachtungseinheit können nur mittels einer wohldefinierten Schnittstelle, die die Gesamtheit der sichtbaren Operationen und ggf. Datentypen angibt, erfolgen.
Generalisierung	**Generalisierung** (engl. *generalization*) bezeichnet den Vorgang der Festlegung einer Klasse *A*, in der gemeinsame Eigenschaften und Fähigkeiten anderer Klassen *B1, B2, ...* in *A* zusammengefasst werden. Die Klassen *B1, B2, ...* werden dabei zu Unterklassen von *A* gemacht .
Glossar	Ein **Glossar** (engl. *glossary*) bestimmt die Begriffe der jeweiligen Fachsprache.
Grammatik	Eine **Grammatik** (engl. *grammar*) ist die Festlegung der Syntax einer Sprache und besteht aus einer Menge von Regeln, die bestimmen, welche Folgen von Zeichen zu einer Sprache gehören und welche nicht.
GUI	Eine **grafische Benutzerschnittstelle** (engl. *graphical user interface*) bezeichnet grafische Darstellungstechniken sowie Eingabemedien wie Tastatur, Maus oder Tablett, die einfach zu bedienende Mensch-Maschineschnittstellen ermöglichen.
Höhere Programmiersprache	**Höhere Programmiersprachen** (engl. *high level programming language*) unterscheiden sich von maschinennahen Sprachen dadurch, dass sie von der Hardware unabhängig sind und anwendungsnahe Sprachmittel anbieten. Höhere Sprachen können mittels Interpretern ausgeführt werden und werden meistens durch Übersetzer in niedere Sprachen übersetzt.

Identität	Die **Identität** (engl. *identity*) oder der **Name** (engl. *name)* eines Objekts unterscheidet es eindeutig von allen anderen Objekten, die in der gleichen Objektwelt existieren.
Implementierung	Der Begriff **Implementierung** (engl. *implementation*) bezeichnet die Formulierung eines vorgegebenen →Algorithmus' zum Zweck der Ausführung auf einem Rechner.
Instantiierung	**Instantiierung** (engl. *instantiation*) beschreibt die Erzeugung eines →Exemplars (s.a. →Instanz) einer →Klasse.
Instanz	Der Begriff **Instanz** wird häufig als Synonym für →Objekt gebraucht und entstand wohl aus der falschen Übersetzung des englischen Worts *instance*, das soviel wie →Exemplar bedeutet.
Instanzvariable	siehe →Exemplarattribut.
Interaktives Programm	Ein **interaktives Programm** (engl. *interactive program*) zeichnet sich dadurch aus, dass es mit dem Benutzer interagiert, d. h. kontinuierlich auf Eingaben und Stimulationen durch den Benutzer reagiert.
Interpreter	Ein **Interpreter** (engl. *interpreter*) ist ein Programmsystem, das die Ausdrücke einer Programmiersprache in eine maschinennahe Sprache umsetzt und unmittelbar ausführt.
Invariante	Eine **Invariante** ist eine Aussage, die vor Eintritt in die Ausführung einer →Operation, einer →Methode oder eines →Algorithmus wahr ist und nach jedem Schritt der Operation (der Methode oder des Algorithmus) wieder wahr ist.

Java	**Java** ist eine objektorientierte Programmiersprache, die aufgrund der →virtuellen Java-Maschine (JVM) gegenüber anderen Programmiersprachen den Vorteil besitzt, dass jedes korrekt übersetzte Java-Programm auf jedem Computer, für den es eine →JVM gibt, ausgeführt werden kann.
Java-Anwendung	Eine **Java-Anwendung** (engl. *Java application*) ist ein →Java-→Programm, das wie andere Programme innerhalb des Betriebsystems Ihres Rechners abläuft.
Java-Applet	Ein **Java-Applet** ist ein kleines →Java-→Programm, das im Allgemeinen von einem entfernten Server geladen wird und im Web-Browser oder Applet-Viewer der Klientin abläuft. Jedes Applet unterliegt hohen Sicherheitsbeschränkungen, die verhindern, dass z. B. Dateien auf dem Gastrechner gelesen oder gar gelöscht werden.
JDK	Das **Java Development Kit** (JDK) ist ein Paket notwendiger Basisprogramme, um Java-Anwendungen programmieren zu können. Es enthält den Compiler, den Interpreter (die „virtuelle Maschine“), die Klassenbibliotheken (APIs) nebst kompletter Beschreibung in HTML, viele kleine Demos usw.
JVM	Die **virtuelle Maschine für Java** (JVM, engl. *Java virtual machine*) umfasst sowohl die Spezifikation als auch die Implementierung eines virtuellen, d. h. durch ein Programmsystem realisierten Rechners, der auf allen gängigen Rechnerplattformen in gleicher Weise den →Bytecode in Java-Klassendateien ausführt. Der von Übersetztern vieler anderer Programmiersprachen erzeugte Assembler- oder Maschinenkode kann i. d. R. nicht auf Rechnern anderer Art als der, auf dem das Programm übersetzt wurde, ausgeführt werden.

Klasse	Eine **Klasse** (engl. *class*) umfasst eine Gruppe oder Familie gleichartiger →Objekte und bezeichnet einen Begriff aus dem Anwendungsbereich.
Klassenattribut	Ein **Klassenattribut** (engl. *class attribute* oder *class variable*) gehört nicht einem einzelnen Objekt, sondern einer →Klasse und ist von der Existenz der Objekte unabhängig. Das Klassenattribut existiert nur einmal pro Klasse und wird bei der Ausführung der Klassenbeschreibung angelegt und mit einem passenden Standardwert belegt.
Klassenbeschreibung	Die **Klassenbeschreibung** definiert die gemeinsamen →Attribute, →Operationen und die gemeinsame →Semantik für eine Anzahl gleichartiger Objekte. Alle Objekte einer →Klasse entsprechen dieser Definition. In Java wird eine Klassenbeschreibung mit dem Schlüsselwort **class** und dem Namen der →Klasse eingeleitet. Darauf folgen, eingefasst in geschweifte Klammern, die →Methoden und →Attribute, die zu der Klasse gehören. Die Reihenfolge von Attribut- und Methodenvereinbarungen ist hierbei beliebig
Klassenbibliothek	Eine **Klassenbibliothek** (engl. *class library*) ist eine strukturierte Ansammlung von →Klassen.
Klassendefinition	siehe →Klassenbeschreibung.
Klassendiagramm	Ein **Klassendiagramm** (engl. *class diagram*) dient zur grafischen Darstellung von Klassen, ihren →Vererbungsbeziehungen und →Assoziationen mit anderen Klassen.

Klassenhierarchie	Von einer Klasse A können Klassen abgeleitet werden, die die Klasse A verfeinern und die Funktionalität und Attribute der Klasse A erben. Ausgehend von einer Ursprungsklasse ergibt sich somit eine **Klassenhierarchie** (engl. *class hierarchy*), die sich leicht durch eine Baumstruktur darstellen lässt.
Klassenoperation	**Klassenoperation** (engl. *class operation*) oder **Klassenmethoden** (engl. *class methods*) sind →Operationen bzw. →Methoden, die nicht auf den →Attributen eines Objekts, sondern auf den →Klassenattributen einer →Klasse operieren.
Klassenvariable	siehe →Klassenattribut
kommuniziert-Beziehung	Die **kommuniziert**-Beziehung (engl. *communicate*) zwischen einem →Akteur und einem →Anwendungsfall bezeichnet die Teilnahme eines Akteurs an einem Anwendungsfall. Es gibt keine anderen Beziehungen zwischen Akteuren und Anwendungsfällen.
Komplexes Objekt	Ein **komplexes Objekt** (engl. *complex object*) ist ein Objekt, dessen Attribute selbst wieder Objekte sind.
Komponente	Eine **Komponente** (engl. *component*) ist ein abgrenzbares Programmpaket mit definierten Schnittstellen und eigener Identität.
Konstruktor	**Konstruktoren** (engl. *constructor*) sind ausgezeichnete →Methoden zur Erzeugung und Erstbelegung von →Objekten mit →Attributwerten. Konstruktoren tragen immer den Namen der Klasse, der sie angehören.

Kontrollfluss	Der **Kontrollfluss** (engl. *control flow*) eines →Programms bestimmt die Reihenfolge der Ausführung der einzelnen →Anweisungen des Programms.
Maschinensprache	Die **Maschinensprache** (engl. *machine language*) ist die Programmiersprache eines Computers. Programme in dieser Sprache sind im Binärcode dargestellt, wodurch sie direkt vom Prozessor ausgeführt werden können, jedoch für Menschen kaum verständlich sind. Zur Behebung dieses Mankos werden diese hardwarenahen Programme nicht in der Maschinensprache, sondern in einer Assembler-Sprache geschrieben.
Mehrfachvererbung	Im Falle der **Mehrfachvererbung** (engl. *multiple inheritance*) kann eine Klasse mehrere direkte Oberklassen besitzen.
Methode	In manchen objektorientierten Programmiersprachen werden die Operationen als **Methoden** (engl. *methods*) bezeichnet. Methoden implementieren Operationen, die eine bestimmte Funktion ausführen oder einen Dienst erbringen.
mnemonisch	**Mnemonische Zeichen** sind leicht zu merken, legen Assoziationen mit bestimmten Begriffen nahe und erleichtern Rückschlüsse auf die Bedeutung des Bezeichneten.
Modell des Anwendungsbereichs	Das **Modell des Anwendungsbereichs** (engl. *model of the application domain*) orientiert sich an den Aufgaben und Verantwortlichkeiten des Anwendungsbereichs sowie den zur Erfüllung der Aufgaben relevanten Gegenständen und Verarbeitungsfunktionen.

Modula Die Sprache **Modula**, genauer MODULA-2, baut auf PASCAL auf und ist ebenfalls eine imperative Programmiersprache. Das besondere an dieser Programmiersprache ist das sog. *Modul-Konzept*: Jedes Modula-2-Programm ist ein Modul, welches wiederum aus anderen Modulen besteht und mit Hilfe dieser Module seine Funktionalität erbringt. Ein Modul ist eine Zusammenfassung von Konstanten, Datentypen, Variablen und Prozeduren zu einer Einheit. Soll ein Modul an einer anderen Stelle des Programms oder in einem anderen Modul benutzt werden, so muss angegeben werden, welche Teile dieses Moduls von aussen sichtbar sein sollen und welche nicht. Dabei bleibt aber die konkrete Realisierung der Datentypen und Prozedurrümpfe grundsätzlich vor allen anderen Modulen verborgen.

Nachbedingung Eine **Nachbedingung** bezeichnet Eigenschaften, die nach der Ausführung einer →Operation oder einer →Methode gelten, wenn diese terminiert und falls die zugehörige →Vorbedingung vor der Ausführung der Operation oder Methode galt.

Nachricht siehe →Botschaft

nebenläufig Zwei Aktivitäten oder Prozesse, die kausal voneinander unabhängig ausgeführt werden können, werden **nebenläufig** (engl. *concurrent*) genannt. Aufgrund der Unabhängigkeit kann sowohl zuerst der eine Prozess, dann der andere als auch umgekehrt ausgeführt werden, beide Prozesse können aber auch zeitlich überlappend ausgeführt werden.

new Der **`new`-Operator** in Java dient dazu, →Objekte oder →Felder zur Laufzeit eines Programms zu erzeugen. Der Ausdruck `new Robot ( )` bewirkte z. B. die Erzeugung eines Objekts der Klasse `Robot`, während der Ausdruck `new Schachbrett[ ][ ]` ein zweidimensionales Feld erzeugte.

Oberklasse	Jede Klasse, die Eigenschaften und Verhalten an eine andere Klasse vererbt, heißt **Oberklasse** (engl. *superclass*) dieser Klasse.
Objekt	Im täglichen Sprachgebrauch bezeichnen wir mit dem Wort **Objekt** einen Gegenstand des Interesses, der Beobachtung, der Veränderung, der Untersuchung oder der Messung. Im engeren, programmiersprachlichen Sinn besitzt ein Objekt einen →Zustand (repräsentiert durch Beziehungen und Attribute), es reagiert mit einem definierten Verhalten (durch die Ausführung von →Operationen) auf seine Umgebung, und es ist ein bestimmtes →Exemplar seiner →Klasse. Das Verhalten gilt für alle Objekte einer Klasse gleichermaßen, jedoch besitzt jedes Objekt eine eigene Identität, die es von anderen Objekten unterscheidet.
Objektdiagramm	Ein **Objektdiagramm** (engl. *object diagram*) ist eine grafische Darstellung der Beziehungen zwischen Objekten zu einem bestimmten Zeitpunkt.
Objektorientiertes Programm	Ein **objektorientiertes Programm** (engl. *object-oriented program*) besteht aus einer Menge kooperierender funktionaler Einheiten, die wir →Objekte nennen. Jedes Objekt spielt in diesem Verbund eine bestimmte Rolle. Es bietet Dienste an oder führt Aktionen aus, die von anderen Mitgliedern des Verbunds genutzt werden.
Operation	Eine **Operation** (engl. *operation*) ist eine Verarbeitungsmethode, die Zugriff auf die Attribute eines Objekts oder einer Klasse hat. (Synonym zu →Methode.)
Paket	Ein **Paket** (engl. *package*) in Java ist die Zusammenfassung mehrerer Klassen beliebigen Typs zu einer Gruppe, um die Systemstruktur auf einer höheren Abstraktionsebene darstellen zu können. Pakete können selbst wieder Pakete enthalten.

Paradigma	Ein **Paradigma** (engl. *paradigm*) erläutert die wesentlichen Bestandteile eines Computerprogramms, wie sie organisiert sind und welche Beziehungen zwischen ihnen bestehen.
Pascal	Die Programmiersprache **Pascal** wurde 1972 von Niklaus Wirth entwickelt und ist nach dem französischen Mathematiker Blaise Pascal (1623-1662) benannt. Pascal ist eine Weiterentwicklung von ALGOL 60 und zeichnet sich besonders durch ihr Datentypenkonzept aus. Sie kann leicht erlernt und aufgrund effizienter Übersetzer auf nahezu allen Rechnerplattformen in vielen Anwendungsbereichen eingesetzt werden.
Persistenz	**Persistenz** (engl. *persistence*) bezeichnet die Fähigkeit eines →Objekts, über die Laufzeit eines →Programms hinweg zu existieren. Ein persistentes Objekt steht nach dem Neustart des Programms wieder zur Verfügung.
Plugin	**Plugins** sind kleine →Programme oder Programmteile, die die Funktionalität einer Software vergrößern. Browser-Plugins werden häufig für multimediale Anwendungen im WWW benötigt.
Programm	Ein **Programm** (engl. *program*) ist eine konkrete Formulierung eines →Algorithmus' unter Verwendung der Ausdrücke einer Programmiersprache. Es ermöglicht die Ausführung eines Algorithmus' auf einem Computer.
Programmieren	Die Tätigkeit des **Programmierens** (engl. *programming*) besteht aus dem Entwerfen oder Finden eines →Algorithmus' zu einem gegebenen Problem und seiner anschließenden →Implementierung.

Prozedur

Prozeduren (engl. *procedure*) sind zentrale Konzepte imperativer Programmiersprachen. Hierdurch kann jede als Programm formulierte Vorschrift zu einer elementaren Anweisung in einem anderen Programm werden. Darüber hinaus dienen Prozeduren der Zerlegung und Strukturierung umfangreicher Programme, und sie erlauben die Verwendung rekursiver Techniken.

Eine Prozedur besteht aus einem Schlüsselwort gefolgt von einem Bezeichner (Name der Prozedur), aus einer Liste von formalen Parametern, aus einer Folge von Deklarationen und aus einer Folge von Anweisungen.

Jede Programmiersprache besitzt ein eigenes Schema für die Definition von Prozeduren.

Prozedurale Programmierung

Bei der **prozeduralen Programmierung** (engl. *procedural programming*) orientiert man sich beim Entwurf eines Programms an den →Prozeduren, die beschreiben, wie bestimmte Aufgaben ausgeführt werden. Diese Prozeduren tauschen untereinander →Datenstrukturen aus und verändern ihnen gemeinsame globale Datenstrukturen. Sobald auf Grund veränderter Anforderungen neue Datenstrukturen eingeführt werden, müssen neue Prozeduren implementiert werden, die auf diesen Datenstrukturen operieren. Dies schränkt die Erweiterbarkeit prozeduraler Programme wegen der Notwendigkeit, Änderungen über das gesamte Programm hinweg nachzuvollziehen, erheblich ein.

Schnittstelle

Eine **Schnittstelle** (engl. *interface*) in Java besteht aus den →Signaturen der →Methoden, die eine →Klasse mit diesen anbietet.

Semantik

Die **Semantik** (griech. *semantikos*= bezeichnend, engl. *semantics*) ist ein Teilgebiet der Linguistik, das sich mit der Bedeutung sprachlicher Ausdrücke befasst.

Sequenzdiagramm Ein **Sequenzdiagramm** (engl. *sequence diagram*) beschreibt die Interaktionen zwischen mehreren Objekten in einem bestimmten Kontext unter Verdeutlichung des zeitlichen Ablaufs. In einem einzelnen Sequenzdiagramm können alternative Abläufe nicht gut dagestellt werden.

Sichtbarkeit Die **Sichtbarkeit** (engl. *visibility*) von Namen bestimmt die Möglichkeit des Zugriffs auf Datentypen, Attribute und Methoden.

Signatur Die **Signatur** (engl. *signature*) einer →Methode besteht aus ihrem Namen sowie der Anzahl und den Typen ihrer Parameter.

Simula **Simula** wurde 1967 aus der Sprache →ALGOL 60 entwickelt. Simula bietet spezielle Möglichkeiten zur Ausführung von Simulationen (der Name leitet sich von „**Simu**lation **La**nguage“ ab) und besitzt bereits Datenstrukturen wie Referenzen und Klassen. Da Simula auch ein Modul- und Vererbungskonzept bietet, gilt diese Sprache als Vorläufer der objektorientierten Programmierung.

Smalltalk Die Sprache **Smalltalk** und der Nachfolger Smalltalk-80 gehören ebenfalls zur Gruppe der objektorientierten Programmiersprachen. Das besondere daran ist, das Smalltalk gleichzeitig eine vollständige Programmierumgebung ist, bei der viel Wert auf ein einheitliches und benutzungsfreundliches Erscheinungsbild gelegt wurde. Die Benutzerin braucht sich nicht in die Betriebsmittel wie Editoren, Übersetzer oder Betriebssysteme einzuarbeiten, da sich unter Smalltalk-80 das gesamte Computersystem in einheitlicher Form darstellt, durch das die Benutzerin mit Menüs geführt wird. Dem Benutzer ist dabei immer nur ein spezielles Objekt, der Bildschirm, sichtbar. Eingaben erfolgen mit der Maus oder über die Tastatur.

Subklasse	Siehe →Unterklasse
Superklasse	Siehe →Oberklasse
Swing	Die **Swing**-Komponenten umfassen eine Sammlung von Komponenten zur Implementierung →grafischer Benutzerschnittstellen. Die Komponenten stellen für alle gängigen Rechner- und Betriebssystemplattformen, welche die →Java virtuelle Maschine (→JVM) unterstützen, die gleiche Funktionalität in plattformspezifischer Gestaltung zur Verfügung. Die Komponenten sind selbst in Java implementiert und übersteigen damit in ihrer funktionellen Ausprägung die auf nativen Plattformfunktionen aufbauenden →AWT-Komponenten.
Syntax	**Syntax** (griech. *syntaxis*=Zusammenstellung) bezeichnet die korrekte Verknüpfung von Wörtern zu Wortgruppen und Sätzen einer Sprache. Die Syntax einer Sprache wird durch eine endliche Menge von Regeln, die den Aufbau wohlgeformter Ausdrücke aus Grundelementen dieser Sprache festlegen, angegeben.
Szenario	Ein **Szenario** (engl. *scenario*) beschreibt eine aktuelle Arbeitssituation im Hinblick auf die Aufgaben im Anwendungsbereich. Es beschreibt ferner die Art und Weise, wie die jeweilige Aufgabe in einzelnen Verarbeitungsschritten auszuführen ist und die Bedingungen, unter denen dies zu erfolgen hat.
this	Das Schlüsselwort **`this`** bezeichnet im Rumpf einer Methode das gerade aktive Objekt.

Typ

Jedes →Attribut und jeder Methodenparameter ist von einem bestimmten **Typ** (engl. *type*). Dies kann ein primitiver Datentyp (z. B. `char`), ein Aufzählungstyp, eine elementare Klasse oder eine Abstraktion sein. →Methoden, deren Ergebnis nicht `void` ist, liefern ebenfalls ein Ergebnis eines bestimmten Typs.

Überschreiben

Mit **Überschreiben** (engl. *overriding*) bezeichnet man die erneute Implementierung einer ererbten Methode. Name, Anzahl und Typen der Parameter sowie der Ergebnistyp der Methode müssen dabei beibehalten werden.

Übersetzer

Ein **Übersetzer** (engl. *compiler*) ist ein Programm, das Ausdrücke einer bestimmten Programmiersprache, der Quellsprache, in gleichbedeutende Ausdrücke einer Zielsprache umsetzt.

UML

Die **Unified Modeling Language** (**UML**) ist eine Sprache zur Beschreibung verschiedener Aspekte von Softwaresystemen. Der Grundgedanke bei der UML bestand darin, eine einheitliche Notation für viele Einsatzgebiete zu haben. Die UML erlaubt die Beschreibung von Datenbankanwendungen ebenso wie die Darstellung von Echtzeitsystemen und Workflow-Anwendungen.

Die UML bietet verschiedene Diagrammarten mit unterschiedlichen grafischen Elementen an, wobei jedes Element eine festgelegte Bedeutung besitzt.

In der UML sind mehrere Vorgängertechniken vereint: Die wichtigsten sind OOSE (Object-Oriented Software Engineering), OMT (Object Modeling Technique) und OOD (Object-Oriented Design). Die Entwickler dieser Methoden - Ivar Jacobson, James Rumbaugh bzw. Grady Booch - sind auch die Urheber der UML. Die UML kann durch so genannte „Stereotypen“ erweitert werden.

Unicode	**Unicode** ist ein 16-bit Zeichensatz, der im ISO-Standard 10646 festgelegt ist.
Unterklasse	Jede Klasse, die von einer anderen Klasse Eigenschaften und Verhalten erbt, wird **Unterklasse** (engl. *subclass*) dieser Klasse genannt.
Variable	Eine **Variable** (engl. *variable*) benennt einen Speicherort für einen Wert eines bestimmten →Typs. Der Inhalt des Speicherorts - und damit Wert der Variablen - kann sich während der Programmlaufzeit beliebig verändern. Über den Namen der Variablen kann auf den Wert zugegriffen und dieser verändert werden.
Vererbung	Mit dem Begriff **Vererbung** (engl. *inheritance*) bezeichnet man die Weitergabe der in einer Klasse angegebenen Eigenschaften und Fähigkeiten an die Unterklassen dieser Klasse. Diese ”spezialisierten Klassen” erweitern die Liste der Attribute, Methoden und Assoziationen der Oberklasse.
Verhalten	Das **Verhalten** (engl. *behavior*) eines Objekts umfasst die beobachtbaren Effekte aller Operationen, die auf ein Objekt angewendet werden können.
Verteilte Anwendung	Eine **verteilte Anwendung** (engl. *distributed application*) ist eine vollständige, in sich geschlossene Problemlösung für einen bestimmten Anwendungsbereich, wobei das Problem durch (räumlich) verteilte Prozesse, die auch auf unterschiedlichen Plattformen (Rechnertypen, Betriebssysteme) ablaufen können, bearbeitet wird.
Vorbedingung	Eine Vorbedingung bezeichnet Eigenschaften, die gelten müssen, wenn eine →Operation oder eine →Methode aufgerufen werden.

Zustand	Der **Zustand** (engl. *state*) eines Computers ergibt sich aus den Werten in den Registern, in Programmzählern, im Hauptspeicher und in Zwischenspeichern zu einem Zeitpunkt. Der Zustand eines →Programms wird bestimmt durch die Werte der Variablen und →Datenstrukturen zu einem Zeitpunkt. Der Zustands eines →Objekts ergibt sich aus der Wertebelegung der →Attribute eines Objekts zu einem bestimmten Zeitpunkt.
Zustandsdiagramm	Ein **Zustandsdiagramm** (engl. *state diagram*) ist eine grafische Darstellung der Zustände, die ein Objekt im Laufe seines Lebens einnehmen kann sowie der möglichen Zustandsübergänge und die Bedingungen, unter denen solche Übergänge stattfinden.

Inhaltsverzeichnis

Kurseinheit 1

1 **Von der Aufgabenstellung zum Programm** **41**
1.1 Motivation 41
1.2 Programmieren 42
1.3 Programme steuern Computer 46
1.4 Anwendungsgebiete für Programme 48

2 **Programmiersprachen** **51**
2.1 Entwicklungsgeschichte von Programmiersprachen 51
2.2 Eingabe, Verarbeitung, Ausgabe 55
2.3 Interaktive Programme 56
2.4 Zur Rolle von Java 57

3 **Aufgabenanalyse** **59**
3.1 Fallstudie 59
3.2 Analyse der Anwendungswelt 67
3.3 Anwendungsfälle undAkteure 68
3.4 Anwendungsfallbeschreibungen 71
3.5 Beziehungen zwischen Anwendungsfällen 74
3.6 Datenlexikon 77
3.7 Pflichtenheft 78

4 **Projektanforderungen** **80**
4.1 Soll-Analyse 80
4.2 Anwendungsfallbeschreibung: „Beratung und Information“ 82
4.3 Anwendungsfallbeschreibung: „Kunden erfassen“ 83
4.4 Anwendungsfallbeschreibung:„Fahrzeug mieten“ 84
4.5 Anwendungsfallbeschreibung:„Fahrzeug zurückgeben“ 85

5 **Konzepte objektorientierter Programmiersprachen** **87**
5.1 Analyse der Anwendungswelt: Gegenstände der Bearbeitung 87
5.2 Fachbegriffe der Anwendungswelt 88
5.3 Realität und Modell 90
5.4 Klassengeflecht 96
5.5 Botschaft und Auftrag 98
5.6 Verhalten: Umgang mit Objekten 99

6 **Zusammenfassung** **101**

Lösungshinweise **105**

Index **113**

Abbildungsverzeichnis

Abb. 1	Prof. Dr.-Ing. Bernd J. Krämer Leiter des Lehrgebiets DVT	vii
Abb. 2	Das Hauptfenster von BlueJ. Links: die Steuer- und Anzeigekommandos, Mitte: vier Klassen mit Vererbungsbeziehung, unten: zwei Exemplare der Klasse PKW und je eines der Klassen LKW und PickUp	x
Abb. 1.2-1	Programmierprozess	44
Abb. 1.3-1	von Neumann-Architektur	47
Abb. 1.3-2	Sprachübersetzung	48
Abb. 1.4-1	Fahrzeuginnenraumbelüftung	49
Abb. 1.4-2	Virtuelle Fertigungsanlage	50
Abb. 1.4-3	Mobiles Navigationsgerät	50
Abb. 2.1-1	Struktur prozeduraler Programme	52
Abb. 2.1-2	Struktur modularer Programme	53
Abb. 2.1-3	Aufbau eines Objekts	54
Abb. 2.1-4	Kooperierende Objekte	54
Abb. 2.2-1	EVA-Prinzip	55
Abb. 3.1-1	Ist-Situation	66
Abb. 3.3-1	Anwendungsfalldiagramm	70
Abb. 3.5-1	Beziehungen zwischen Anwendungsfällen	75
Abb. 3.5-2	erweitert-Beziehung	76
Abb. 4.1-1	Anwendungsfalldiagrammzur Sollanalyse	81
Abb. 5.3-1	Roboter -Wirklichkeitund Modell	91
Abb. 5.3-2	Robotereigenschaften im Modell	91
Abb. 5.3-3	Roboterverhalten im Modell	92
Abb. 5.3-4	Eigenschaften und Verhalten im Robotermodell	92
Abb. 5.4-1	Klassenhierarchie	96
Abb. 5.4-2	Klassengraph	97
Abb. 5.6-1	Fahrzeugdatei	99
Abb. 5.4-3	Klassengraph für geometrische Figuren	110

Tabellenverzeichnis

Tabelle 1.4-1 Anwendungsgebiete für Programme 48
Tabelle 3.4-1 Beschreibungsschema fürAnwendungsfälle 72
Tabelle 3.4-2 Anwendungsfall „Beratung“ 73
Tabelle 3.6-1 Schreibweise im Datenlexikon 77
Tabelle 4.2-1 Anwendungsfallbeschreibung: „Beratung und Information“ . 82
Tabelle 4.3-1 Anwendungsfallbeschreibung: „Kunden erfassen“ 83
Tabelle 4.4-1 Anwendungsfallbeschreibung: „Fahrzeug mieten“ 84
Tabelle 4.5-1 Anwendungsfallbeschreibung: „Fahrzeug zurückgeben“ 85
Tabelle 5.2-1 Fachbegriffe 89
Tabelle 3.4-3 Anwendungsfallbeschreibung: „Fahrzeug reservieren“ 106
Tabelle 4.5-2 Anwendungsfallbeschreibung: „Mietvertrag abrechnen“ 106
Tabelle 5.2-2 Einordnung der Fachbegriffe 107
Tabelle 5.3-1 Ordnen der Fachbegriffe nach objektorientierten Kernbegriffen 109
Tabelle 5.3-2 Attribute der Klasse Auto 109
Tabelle 5.6-1 Attribute und Methode der Klasse Kfz-Mietvertrag 110

Einführung

Ziele

Viele Lehrbücher beginnen ihre Einführung in die Java-Programmmierung mit der Kodierung einer scheinbar einfachen Anwendung:

```
public static void main(String[] args) {
    System.out.println("Hello World!");
}
```

Dieses kleine Programm bewirkt, dass die Zeichenfolge „`Hello World`“ auf dem Bildschirm erscheint.

Die wenigen Anweisungen in diesem Programm enthalten aber schon mehr als sechs verschiedene Sprachkonstrukte, deren Verständnis man Anfängerinnen nicht abverlangen kann.

Wir gehen anders vor, denn allzuoft vergessen die Programmierer vor der Kodierung ihres Programms, gründlich über die jeweilige Aufgabenstellung nachzudenken und den sozialen, organisatorischen und technischen Kontext, in dem das Anwendungsprogramm später verwendet werden soll, genau zu hinterfragen. Sie wiegen sich sicher in einer vermeintlichen Professionalität, sind aber oft nicht in der Lage, die Fachbegriffe der Anwendungswelt zu verstehen, geschweige denn sich mit ihrer Fachsprache den Auftraggeberinnen und Nutzern, die meist keine IT-Experten sind, verständlich zu machen.

Viele Probleme in der Praxis der Programmentwicklung rühren bekanntermaßen daher, dass Programmentwickler, Auftraggeberinnen und spätere Programmanwender kein ausreichend detailliertes gemeinsames Verständnis der Aufgabe entwickelten. Aus diesem Mangel entstehen unerwünschte Freiheitsgrade bei der Programmierung, die meist zu Enttäuschungen beim Kunden führen.

In dieser Lerneinheit versuchen wir zunächst, Ihnen anhand einer praxisnahen Fallstudie ein intuitives Verständnis der zentralen Begriffe der objektorientierten Programmierung wie →Klasse, →Objekt, →Methode oder →Nachricht zu vermitteln. Die Fallstudie dient auch dazu, deutlich machen, dass ein detailliertes Verständnis der gestellten Aufgabe sowie wie eine präzise Dokumentation notwendige Voraussetzungen für einen gelungenen Programmentwurf sind. Wir werden Schreibweisen und Konventionen einführen, mit deren Hilfe Sie eine größere Programmieraufgabe vernünftig und nach einem in der Praxis verbreiteten Standard dokumentieren können.

Fallbeispiel

Bei der Auswahl des Fallbeispiels mussten wir einen Kompromiss zwischen Einfachheit und Realitätstreue finden. Das Beispiel ist als Ganzes aber immer noch zu groß, um es im Rahmen dieses Kurses vollständig von Ihnen lösen zu lassen. Die Programmieraufgaben sind deshalb so gewählt, dass Sie abgrenzbare Teilaufgaben lösen, die sich mit den von uns vorprogrammierten Teilen zu einer Gesamtlösung zusammenfügen lassen. Bei Interesse können Sie sich später im Selbststudium im Einzelnen mit der Gesamtlösung auseinander setzen.

Bitte beachten Sie auch die allgemeinen Programmierhinweise zur dieser Lerneinheit.

Lernziele

- Die Fachbegriffe eines Anwendungsproblems, ihre Merkmale und typischen Verhaltensweisen verstehen und umfassung beschreiben können.
- Die Anforderungen einer Anwendung erheben und systematisch dokumentieren können.
- Anwendungsfälle, Gegenstände, Beziehungen, Verantwortlichkeiten, Abläufe und andere Phänomene und Erscheinungen von Belang in der Aufgabe erkennen und präzise in der Form eines Fachlexikons und verschiedener Diagrammarten darstellen können.
- Die behandelte Aufgabenstellung vervollständigen und auf ähnliche Aufgaben übertragen können.
- Die Begriffe Objekt und Klasse erläutern und gegeneinander abgrenzen sowie auf die Fachbegriffe einer gegebenen Problemstellung anwenden können.
- Verstehen, wie Objekte zusammenwirken und worin sich der Austausch von Botschaften und Prozeduraufrufe unterscheiden.
- Eine Vererbungshierarchie aufstellen und erklären können, wie sich ihre Klassen und das Verhalten dieser Klassen bestimmen lassen.
- Die Angemessenheit einer Klassenhierarchie beurteilen können.
- Die wesentlichen Merkmale der objektorientierten Programmierung wiedergeben können.
- Die Begriffe Klasse und Objekt und ihre Beziehung zueinander verstehen und anwenden können.
- Die Hauptbestandteile von Klassen und Objekten, nämlich Verhalten und Eigenschaften, erklären und konstruktiv einsetzen können.
- Die Vererbungsbeziehungen zwischen Klassen und ihre Auswirkungen auf den Programmentwurf begreifen.

Hinweise

- Bestimmen Sie das vorliegende Problem möglichst vollständig! Anforderungserhebung

  In der Analysephase ist es wichtig, möglichst alle Anwendungsfälle und Anforderungen für das zu entwickelnde System zu entdecken. In der Praxis wird man nicht alle finden. Sie sollten deshalb danach streben, zumindest die wichtigsten zu finden.

Programmierenmehr alskodieren

- Widerstehen Sie dem Drang, sofort mit dem Programmieren los zu legen!

  Beherzigen Sie das Motto:
  „Erst verstehen und denken, dann programmieren!“

Auswirkungenvon Programmen

- Stellen Sie sich während der Programmentwicklung immer wieder der Frage, ob und ggf. in welcher Weise
  - die Rechte von Personen, deren Daten verarbeitet werden, verletzt werden,
  - alle Anforderungen und Beschränkungen des Einsatzfelds beachtet werden,
  - die Arbeitsabläufe von Personen, die mit dem Programmsystem arbeiten oder es betreiben sollen, verändert werden

  und ob diese Veränderungen gewollt sind!

Ergonomie

- Beachten Sie anerkannte Richtlinien der Ergonomie und Benutzbarkeit von Programmsystemen!

Entwurf

- Denken Sie so früh wie möglich darüber nach, wie Sie Ihren Entwurf und die Spezifikation Ihres Programms möglichst allgemein gestalten können!

Klasseund Rolle

- Orientieren Sie sich bei der Entwicklung von Klassenmodellen nicht an den Daten, sondern an den Verantwortlichkeiten der Objekte einer Klasse!

Klasseoder Exemplar

- Beachten Sie den Unterschied zwischen Klassen und deren Exemplaren, den Objekten!

  Die Konzepte →Klasse und →Exemplar werden oft verwechselt. Dies drückt sich darin aus, dass →Attribute und →Methoden als statisch deklariert werden und →Java als prozedurale Sprache verwendet wird.

Literatur

[Parnas72] David L. Parnas:
A technique for software module specification with examples.
Communications of the ACM, 15(5):330-336, May 1972

Der Artikel führt das Geheimnisprinzip auf der Grundlage eines Modulbegriffs ein.

[Züllighoven98] HeinzZüllighoven:
Dasobjektorientierte Konstruktionshandbuch.
dpunkt.verlag 1998

Dieses Buch ist keine Einführung in die Programmierung und bezieht sichauch nicht auf Java. Es ist ein Handbuch zur Entwicklung großerobjektorientierter Softwaresysteme, das die Begriffe Werkzeug und Material inden Vordergrund der Betrachtung stellt.

1 Von der Aufgabenstellung zum Programm

Mit diesem Kurs wollen wir Ihnen fundierte Kenntnisse von Programmiertechniken, insbesondere der objektorientierten Programmierung vermitteln. Nach erfolgreichem Abschluss des Kurses sollten Sie in der Lage sein, einfache Programmsysteme in der Programmiersprache Java zu entwickeln und die Qualität überschaubarer Java-Programme zu bewerten.

Da Programme i. d. R. dazu dienen, menschliche Tätigkeiten zu unterstützen oder technische Systeme und Anlagen zu überwachen und zu steuern, ist es wichtig, die jeweilige Aufgabenstellung und die Umgebungsbedingungen für das geplante Programm genau zu verstehen.

1.1 Motivation

Um Ihnen den gesamten Programmierprozess von der Aufgabenstellung bis zur Ablieferung eines geprüften Programms anschaulich zu machen, führen wir informell aber dennochmethodisch Vorgehensweisen ein, die zeigen, wie man:

- eine Aufgabenstellunginhaltlich erfasst und in der Form von Geschäftsvorfällen oder technischen Abläufen, die es zuunterstützen oder zu automatisieren gilt, dokumentiert;
- Geschäftsvorfälle systematisch in einen Programmentwurf in derForm standardisierter grafischer Beschreibungstechniken überführt und
- derartige Entwurfsdokumente in ausführbare Java-Programme umsetzt.

Bei dieser Vorgehensweisewerden Sie intuitiv grundlegende Programmierkonstrukte wie Abfolge, Schleife,Verzweigung und Rekursion, aber auch wichtige Datenstrukturen wie Liste,Kellerstapel und Baum kennen lernen und benutzen.

Programmiersprachen

Heutegibt es eine Vielzahl unterschiedlicher Programmiersprachen, um Programme zu formulieren. Sie unterscheidensich jedoch erheblich von den Sprachen, die Menschen benutzen, wenn siemiteinander kommunizieren. Ein entscheidender Unterschied zwischen der Art, wie wir miteiner anderen Person reden und wie der Computer instruiert werden muss,besteht darin, dass Menschen ein großes Maß an Allgemeinwissen und gesundemMenschenverstand besitzen, die es ihnen ermöglichen, auch ungenaue undunvollständige Aussagen sinnvoll zu interpretieren. Solche Fähigkeiten besitzenComputer heute noch nicht. Ihre Anweisungen müssen daher sehr detailliert,Schritt für Schritt und in einer genau festgelegten Sprache, eben derProgrammiersprache, angegeben werden.

Allzu oft habenProgrammsysteme unvorhergesehene oder unerwünschte Auswirkungen auf dieUmgebung, in der sie eingesetzt werden:

- Piloten können bei der Landung denUmkehrschub nicht aktivieren, weil für das Steuerprogramm die Bedingungen einerBodenberührung nicht erfüllt sind – so geschehen bei der missglückten Landungeiner Maschine der Lufthansa in Warschau;
- Mitarbeiter in Organisationen undUnternehmen müssen sich nicht selten an ungewohnte Geschäfts- undFertigungsabläufe anpassen, weil das neue Programmsystem ohne ausreichendeKenntnis solcher Abläufe geplant wurde.
- Fehlbedienungen werden durchungewohnte oder unergonomisch gestaltete Bedienoberflächen von Programmsystemenprovoziert.

Wir betrachten der Programmierung als eine Ingenieurstätigkeit und legen an sie die gleichen Maßstäbe an wie an andere Technikdisziplinen: Bevor Sie beginnen zu konstruieren, denn Programmieren heißt Konstruieren, müssen Sie

- die Gesetzmäßigkeiten des Anwendungsfelds, die Fachkonzepte und Abläufe verstehen,
- professionelle Arbeitsweisen beherrschen,
- passende Standards kennen (hierzu gehören auch Normen, Richtlinien und Prüfverfahren zur Gestaltung von Benutzungsschnittstellen interaktiver Systeme[51], die allzu häufig ignoriert werden),
- Qualitäts-, Kosten- und Nutzendenken entwickeln sowie
- empirisches Wissen aufbauen, nutzen und wirksam vermitteln können.

In nachfolgenden Kursen werden Sie weitergehende Kompetenzen zur Softwaretechnik, zum Projekt- und Qualitätsmangement u. a.m. erwerben, die Sie befähigen sollen, nicht nur als Programmierer, sondern auch als Projektingenieur und Bereichsleiterin tätig werden zu können.

1.2 Programmieren

Programmieren

Programmieren bedeutet das Entwerfen, Formulieren, Dokumentieren und Überprüfen eines Programms. Der Vorgang der Programmierung umfasst demnach nicht nur das Aufschreiben von Programmcode, sondern schließt eine genaue

- Erhebung der fachlichen Begriffe, Abläufe, Rollen, Gegenstände und Phänomene des jeweiligen Anwendungsbereichs,
- Dokumentation der Anforderungen, die das Programm erfüllen muss,
- Modellierung der Gegenstände und Verfahrensweisen des Anwendungsbereichs und

51 http://www.procontext.com/de/richtlinien/

- umfassende Qualitätsprüfverfahren

mit ein.

Wir unterscheiden zwischen dem Programmieren-im-Kleinen und -im-Großen. Beim **Programmieren-im-Kleinen** konstruieren wir Folgen von Anweisungen, die auf einem gemeinsamen, durch Daten dargestellten Zustandsraum arbeiten und als einzelne oder im Verbund mit anderen Anweisungen einen mehr oder minder komplexen Zustandsübergang beschreiben.

Das **Programmieren-im-Großen** betrachtet ein Programm aus abstrakterer Warte als Kollektion größerer Programmeinheiten. Bei dieser abstrakten Betrachtung stehen nicht die Abläufe im Kleinen, sondern die Wirkungs- und Nutzungsbeziehungen zwischen den Programmeinheiten im Vordergrund.

Ein Modell des gesamten Programmierprozesses ist in Abb. 1.2-1 dargestellt. Es unterscheidet zwischen Aufgaben, die von Menschen ausgeführt werden müssen, weil sie Kreativität erfordern, und Aktivitäten, die vom Rechner mit Hilfe spezieller Programme wie Übersetzer (engl. *compiler*) und Lader (engl. *loader*) übernommen werden, weil sie mechanischen Charakter haben.

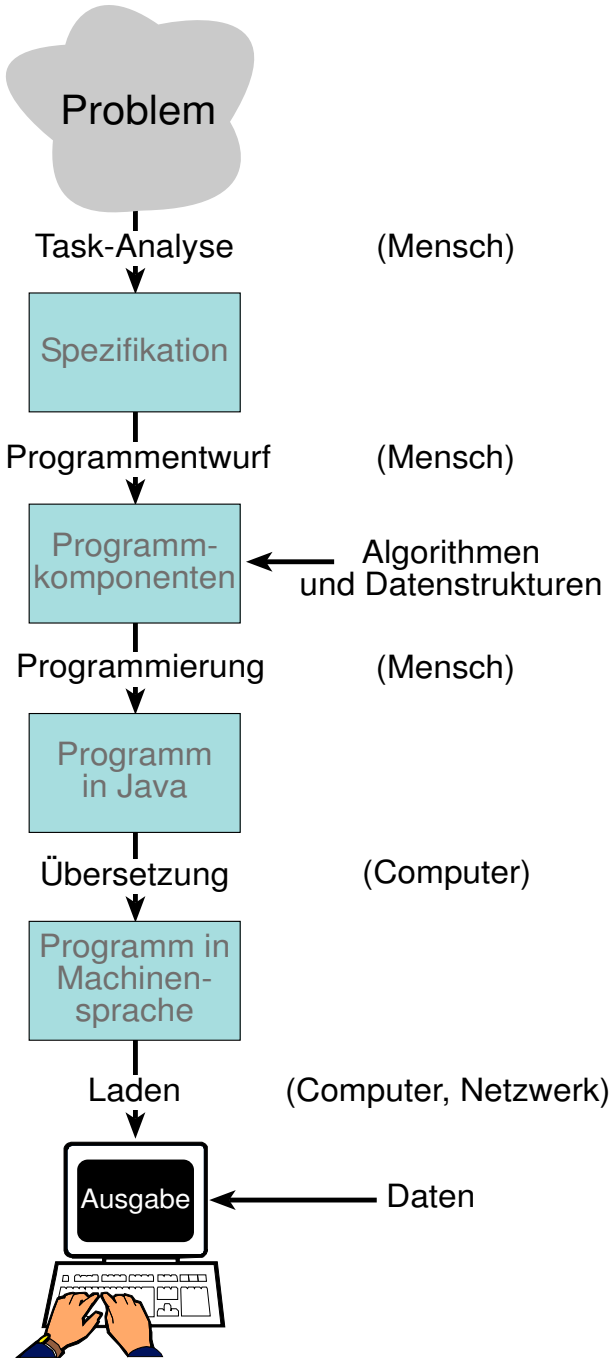


Abb. 1.2-1: Programmierprozess

Im Rest dieser Lerneinheit werden wir den Unterprozess diskutieren, der sich mit dem Verstehen der Aufgabe und der Dokumentation von Anforderungen und Randbedingungen befasst. In Lerneinheit 3 werden wir uns auf die Aufgabe des Algorithmenentwurfs konzentrieren, und in Lerneinheit 6 werden wir einige grundlegende Datenstrukturen einführen. Programmieren in Java ist Gegenstand des gesamten Kurses. Während der Programmierübungen in den folgenden Wochen werden Sie die Rolle des Java-Übersetzers und -Laders kennen lernen. Das Konzept der Sprachübersetzung wird im nächsten Abschnitt erklärt und veranschaulicht.

Definition 1.2-1: Programm
*Ein **Programm** beschreibt ein schematisches Lösungsverfahren für eine bestimm-*

te Aufgabenstellung mit den Ausdrucksmitteln einer Programmiersprache. Es ermöglicht die mechanische Ausführung eines Lösungs- oder Verarbeitungsverfahrens auf einem Rechner.

Ein Programm bestimmt die genauen Arbeitsschritte, die ein Rechner befolgen muss, um eine bestimmte Aufgabe zu erledigen.

□

Beispiel 1.2-1: Kaffee kochen
Wenn wir zum Beispiel einer Person erklären wollen, wie man Kaffee mit einer Kaffeemaschine zubereitet, könnten wir sagen:

- *Fülle den Wasservorratsbehälter der Maschine mit einer der gewünschten Tassenzahl entsprechenden Menge Wassers,*
- *lege eine Filtertüte in den Filteraufsatz ein und füge pro Tasse einen gehäuften Kaffeelöffel voll mittelfein gemahlenen Kaffees bei,*
- *schließe Filteraufsatz und Wasservorratsbehälter,*
- *setze die Kaffeekanne auf die Wärmeplatte der Maschine, und*
- *schalte die Maschine ein.*

□

Vergleichbare Anweisungen müssen wir angeben, wenn wir einen Rechner programmieren wollen.

Ein anderes, mehr rechnerorientiertes Beispiel ist im Folgenden angegeben:

Beispiel 1.2-2: Multiplikation
Rechner!
Frage mich nach zwei ganzen Zahlen, multipliziere sie und sage mir das Ergebnis.

□

Solche Anweisungen sind für einen der deutschen Sprache mächtigen Gesprächspartnerin ausreichend genau und ausführlich. Rechner verstehen jedoch kein Deutsch, und sie können diese oder andere natürliche Sprachen auch noch nicht zufriedenstellend verarbeiten. Sie können nur Maschinensprache ausführen. Unter Maschinensprache versteht man die Programmiersprache, die ein Prozessor direkt ausführen kann. Maschinensprache ist jedoch für Menschen kaum zu lesen.

Allein das Programmieren auf der Grundlage rechnernaher Sprachen hätte wegen der immensen begrifflichen Distanz zwischen Anwendungsbereich und Rechnerebene nicht die komplexen Anwendungen zustande gebracht, die wir heute sehen.

1.3 Programme steuern Computer

Im Kurs **Grundlagen der Informationstechnik** haben Sie gelernt, wie man ein Schaltnetz oder ein Schaltwerk entwirft, um eine gegebene Aufgabe mit Hilfe einer elektronischen Schaltung zu lösen. Jede Schaltung ist individuell aufgebaut und auf die jeweilige Aufgabe zugeschnitten. Jede neue Aufgabe erfordert die Entwicklung einer neuen, „hart verdrahteten“ Lösung.

Der entscheidenende Schritt vom Schaltwerk zum Computer besteht darin, dass ein Computer nicht nur Daten, sondern auch Programme speichern kann. Jedes Programm bestimmt die Aktionen des Computers in der Weise, dass die anstehende Aufgabe gelöst wird, ohne irgendwelche Verdrahtung verändern zu müssen. Der Rechner wird damit zu einer für viele Aufgaben einsetzbare universelle Problemlösungsmaschine.

Computer sind technische Geräte, die vorgegebene Anweisungen ausführen. Für Computer sind jedoch Anweisungen, wie wir sie aus Kochrezepten oder von Gebrauchsanweisungen her kennen, im Allgemeinen viel zu ungenau. Hinzu kommt, dass Computer natürliche Sprachen, die wir Menschen benutzen, heute und auf absehbare Zeit nicht ausreichend verstehen und Anweisungen in natürlicher Sprache nicht ausführen können.

Programmiersprache

Deshalb wurden schon in der Frühzeit der Computer spezielle **Programmiersprachen** entwickelt, die viel einfacher aufgebaut sind als natürliche Sprachen wie Deutsch, Englisch oder Chinesisch.

Programmiersprachen liefern uns den Vorrat an elementaren und in ihrer Bedeutung genau festgelegten Ausdrücken, den wir benötigen, um eine Verfahrensbeschreibung oder ein schematisches Lösungsverfahren für eine bestimmte Aufgabenstellung zu formulieren. Solche aus endlich vielen, ausführbaren elementaren Verarbeitungsschritten bestehenden Beschreibungen sind die Programme.

Programm

Wenn Sie einen Rechner kaufen, erwerben Sie in der Regel auch Programme, die andere Personen für Sie geschrieben haben. Diese Programme sind dafür zuständig, dass Ihr Rechner nach dem Einschalten in einen bestimmten Zustand „hochfährt“ und Ihnen eine einfach zu bedienende Benutzungsoberfläche anbietet.

Die Mehrzahl heutiger Rechner entspricht in ihrem Aufbau immer noch der von Neumann-Rechnerarchitektur (Abb. 1.3-1), die Sie ja im Kurs „Informationstechnische Grundlagen“ schon kennen gelernt haben.

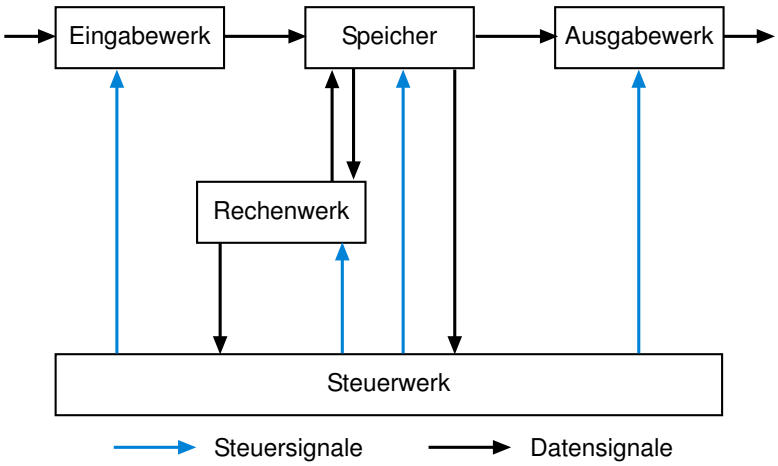


Abb. 1.3-1: von Neumann-Architektur

Gemäß dieser Architektur besteht ein Rechner aus einem Speicher, der Daten und Instruktionen verwaltet, einer Zentralrecheneinheit oder CPU (engl. *central processing unit*) und einer Ein-Ausgabeeinheit. Instruktionen werden eine nach der anderen aus dem Speicher in die CPU geladen und ausgeführt. Bei der Ausführung von Instruktionen werden auch Daten aus dem Speicher in die CPU eingelesen und durch logische oder arithmetische Operationen verändert. Die Ergebnisse dieser Verarbeitung werden in den Speicher zurück geschrieben.

Die Ausführung solcher Instruktionen bewirkt also eine Veränderung des →Zustands der Maschine, der im Wesentlichen durch den Speicherinhalt, Registerwerte und den Wert des Instruktionszählers dargestellt wird.

Zustandsveränderung

Auch auf der Webseite der Fa. Intel[52] können Sie sich anhand einer interaktiven Animation[53] noch einmal die Arbeitsweise eines Mikroprozessors vor Augen führen.

Die im Rechner verarbeiteten Instruktionen orientieren sich begrifflich an den Möglichkeiten der Maschine und werden daher als →Maschinensprachen bezeichnet. Im Kurs „Informationstechnische Grundlagen“ lernten Sie bereits maschinennahe Programmierkonstrukte kennen.

Jede Maschinensprache ist jedoch für menschliche Leser schwer zu interpretieren. Deshalb wurden relativ rasch so genannte →höhere Programmiersprachen entwickelt, um komplexe Anwendungen leichter programmieren zu können. Um Programme, die in einer höheren Programmiersprache geschrieben wurden, ausführen zu können, müssen die Programme in Maschinensprache übersetzt werden. Abb. 1.3-2 veranschaulicht diesen Prozess.

52 http://www.intel.com

53 http://www.intel.com/education/mpworks/

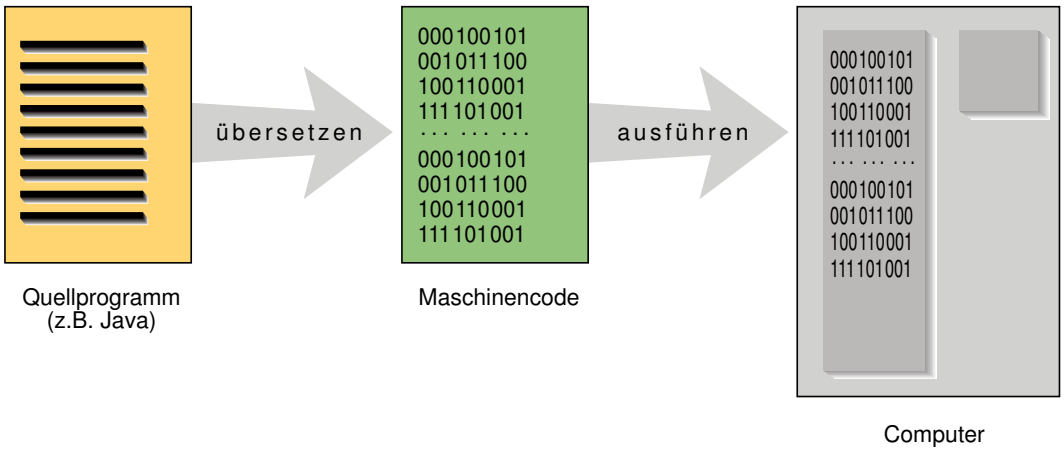


Abb. 1.3-2: Sprachübersetzung

1.4 Anwendungsgebiete für Programme

Programme werden entwickelt, um Menschen bei ihren Tätigkeiten zu unterstützen oder um Aufgaben, die bisher von mechanischen Maschinen oder Elektronik ausgeführt wurden, an Rechner zu delegieren. Programme können:

Tab. 1.4-1: *Anwendungsgebiete für Programme*

Gestaltungsaufgaben unterstützen, z. B. beim	• Karosserieentwurf oder bei der • Innenraumgestaltung von PKWs (s. Abb. 1.4-1);
Prozesse überwachen und steuern, z. B. bei der	• Fertigung von Chemikalien in Reaktoren, • Gepäckbeförderung in Flughäfen, • Energieverteilung in Hochspannungsnetzen;
Sicherheitsaufgaben gewährleisten, z. B. bei	• Signalsteuerungen von Stadt-, Regional- und Fernbahnen, • Antiblockiersystemen, • der automatischen Auslösung von Airbags;
komplexe Modellberechnungen durchführen und mathematische Modelle veranschaulichen, z. B. bei	• Wettervorhersagen oder • der Simulation und Optimierung von Betriebsabläufen (Abb. 1.4-2);

Fortsetzung auf der nächsten Seite

Fortsetzung von der vorhergehenden Seite	
Daten verarbeiten und aufbereiten, z. B. :	• die Quartalsabrechnung einer Arztpraxis erstellen, • die Lagerhaltung in Betrieben abwickeln, • Computertomografiedaten grafisch aufbereiten;
kontinuierlich Dienste anbieten, z. B. bei	• Geldautomaten, • mobilen Navigationsgeräten (Abb. 1.4-3).

Abb. 1.4-1: Fahrzeuginnenraumbelüftung

Abb. 1.4-2: Virtuelle Fertigungsanlage

Abb. 1.4-3: Mobiles Navigationsgerät

2 Programmiersprachen

Programmiersprachen sind Kunstsprachen zur Darstellung von Computerprogrammen. Sie sind in ihrer →Syntax und →Semantik wesentlich einfacher gestaltet als natürliche Sprachen. Statt der im Computer intern verarbeiteten Binärwerte werden in Programmiersprachen häufig verwendete Zahlen und Zeichen symbolisch angegeben. Ihrem Anwendungszweck entsprechend bieten sie Befehle, Steueranweisungen und andere Sprachkonstrukte an, die das Formulieren von Programmen für numerische Berechnungen, für die betriebliche Datenverarbeitung, für die Systemprogrammierung u. a.m. erleichtern. Zugleich abstrahieren sie von den elementaren Operationen des Rechners wie Zugriffe auf Speicherzellen oder logische und arithmetische Operationen.

2.1 Entwicklungsgeschichte von Programmiersprachen

Die ersten höheren Programmiersprachen wie →Fortran, →Algol oder →Pascal unterstützen einen sog. **imperativen** oder **prozeduralen Programmierstil**, der sich an den Abläufen (engl. *processes*) und Vorgehensweisen (engl. *procedure*) des Anwendungsgebiets (engl. *application domain*) orientiert. Programme werden bei der imperativen Programmierung entlang der Systemabläufe strukturiert.

Bei der prozeduralen Programmierung orientiert man sich beim Entwurf eines Programms an den Prozeduren, die beschreiben, wie bestimmte Aufgaben ausgeführt werden. Diese Prozeduren tauschen untereinander Daten aus und verändern ihnen gemeinsame globale Datenstrukturen. Der Aufbau der Datenstrukturen des Programms ist allen Prozeduren zugänglich. Die Strukturierung von Daten, die von den Prozeduren manipuliert werden, ist jedoch dem Entwurf von Prozeduren nachgeordnet.

Ein Nachteil dieses Programmieransatzes ergibt sich aus dieser ungleichen Behandlung von Prozeduren und Datenstrukturen. Sobald nämlich auf Grund veränderter Anforderungen vorhandene Datenstrukturen geändert werden müssen, ist im Extremfall eine Überarbeitung aller Prozeduren, die auf diesen Datenstrukturen aufbauen, fällig. Dies schränkt die Erweiterbarkeit prozeduraler Programme wegen der Notwendigkeit, Änderungen über das gesamte Programm hinweg nachzuvollziehen, erheblich ein.

Ein Computerprogramm, das z. B. in einer Bibliothek eingesetzt werden soll, besteht aus Prozeduren, die wesentlichen Anteile der Aufgaben Ausleihe, Rückgabe, Reservierung, Katalogisieren und Beschaffen von Büchern in den Rechner verlagern.

Abb. 2.1-1 zeigt schematisch, wie die Prozeduren gemeinsam die Daten der Anwendung bearbeiten.

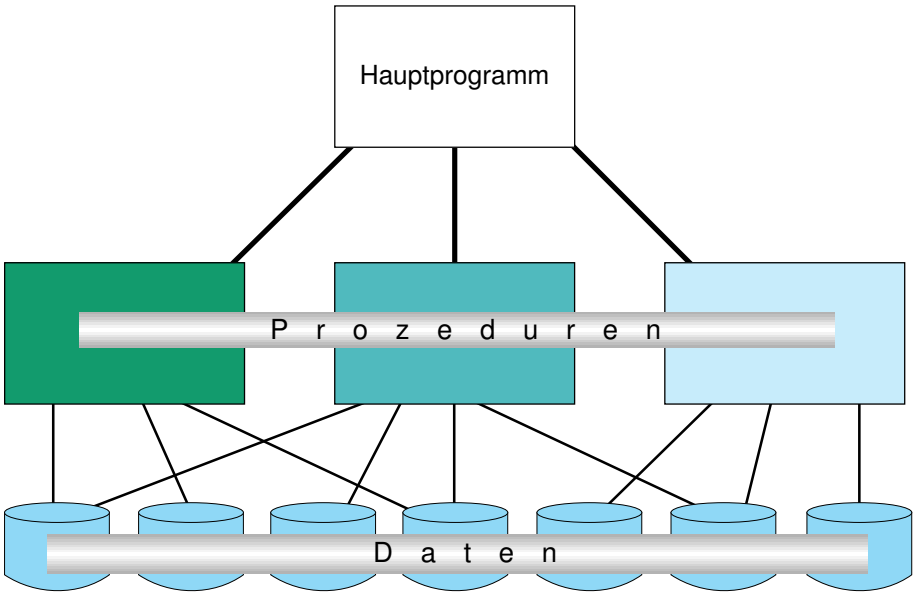


Abb. 2.1-1: Struktur prozeduraler Programme

Fortran

Die Entwicklung der Sprache Fortran war ein entscheidender Abstraktionsschritt, der von vielen Details der Maschinensprachen wie Registerbenutzung und Speicherverwaltung absah und die Aufmerksamkeit des Programmierers auf algebraische Operationen hin lenkte.

Algol, Pascal

Algol und Pascal gingen in der Abstraktion einen Schritt weiter, indem sie auch heute noch wesentliche Kontrollstrukturen einführten. Sie sind auch Bestandteil der Sprache Java und werden in späteren Lerneinheiten vorgestellt.

Etwas neuere Sprachen wie →Modula oder →Ada stellen die Datenabstraktion und die Zerlegung eines Programms in modulare Einheiten in den Vordergrund der Anwendungsprogrammierung (Abb. 2.1-2).

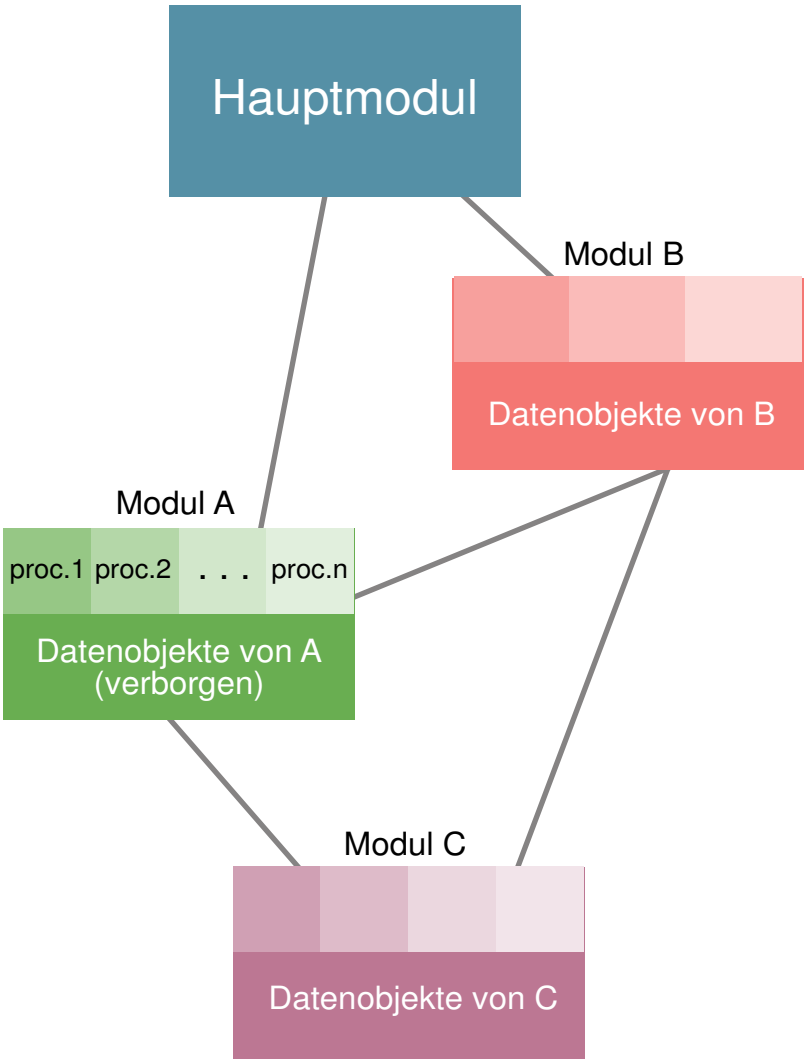


Abb. 2.1-2: Struktur modularer Programme

Hierbei werden die Datenstrukturen einer Anwendung in abgrenzbare Einheiten, sog. **Module** (im Fall von Modula) oder **Pakete** (bei Ada) so eingekapselt, dass Veränderungen an den Datenstrukturen ausschließlich über definierte Zugriffsoperationen erfolgen können. Die Module operieren nicht mehr auf einem gemeinsamen Datenbestand, sondern besitzen ihre eigenen Datenstrukturen, deren Aufbau nach außen verborgen bleibt. Erweiterungen werden so vereinfacht, und die Freiheitsgrade für Implementierungs- und Optimierungsentscheidungen werden größer. Dieser Konstruktionsansatz wurde von David Parnas im Jahr 1972 [Parnas72] unter der Bezeichnung **Geheimnisprinzip** eingeführt und liegt auch dem Prinzip der Datenabstraktion zu Grunde. Jedes einzelne Modul wird natürlich wieder mit den Mitteln des Programmierens-im-Kleinen realisiert.

Bei der **datenstrukturorientierten Programmierung** konstruierte man bei der Bibliotheksautomatisierung beispielsweise Module, die die verschiedenen Kataloge und Ablagen - Gesamtkatalog, Ausleihkatalog, verfügbarer Bestand, laufende Bestellungen, Rechnungen - im Rechner abbilden und entsprechende Verarbeitungsfunktionen zur Veränderung oder Abfrage dieser Datenstrukturen bereitstellen.

→Simula, eine für Simulationsanwendungen Ende der 60er Jahre entwickelte Programmiersprache, ermöglichte zum ersten Mal einen **objektorientierten Programmierstil**. Bei dieser Programmiertechnik werden Datenstrukturen und die sie manipulierende Verarbeitungsmethoden als Einheit betrachtet und **Objekte** genannt.

Die Ausführung einer Verarbeitungsmethode wird angeregt durch die Ankunft einer Botschaft, in der die zu aktivierende Methode zusammen mit weiterer Information von der Umgebung des Objekts übermittelt wird.

Abb. 2.1-3 veranschaulicht die Zusammenführung der Konzepte Methode, Datenstruktur und Botschaft zum neuen Begriff Objekt.

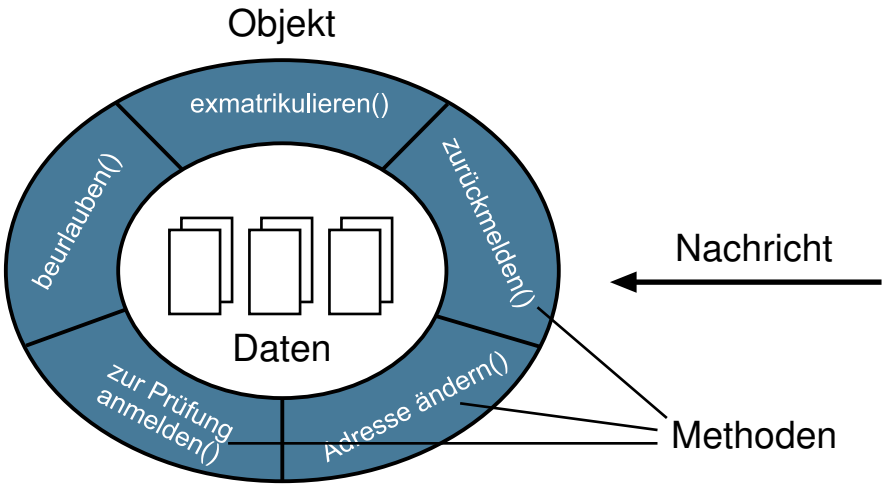


Abb. 2.1-3: Aufbau eines Objekts

Ein objektorientiertes Programm besteht aus einer Vielzahl kooperierender Objekte (Abb. 2.1-4), die untereinander Botschaften austauschen, um gemeinsam eine Aufgabe zu bewältigen.

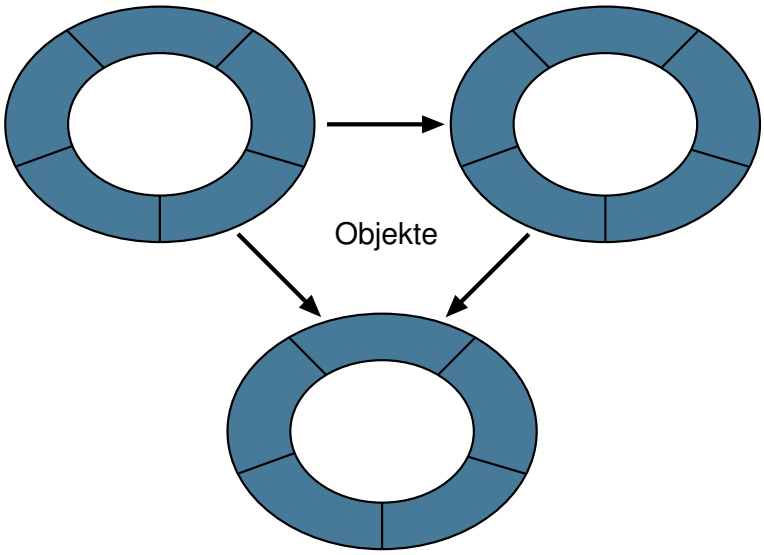


Abb. 2.1-4: Kooperierende Objekte

Bei der objektorientierten Programmierung werden Datenstrukturen und Operationen auf solchen Datenstrukturen zu Einheiten, so genannten →Objekten, zusammengefasst. Objekte repräsentieren gedankliche Gebilde, Gegenstände und Phänomene der Wirklichkeit.

Die objektorientierte Programmierung eröffnet eine neue Art, eine Aufgabenstellung zu verstehen und eine programmtechnische Lösung in Form eines Systems zusammenwirkender Objekte zu finden (Abb. 2.1-4). Objekte modellieren Dinge, Personen, Begriffe oder Gegenstände der Wirklichkeit. Sie sind eindeutig identifizierbare Einheiten, die Daten und Verarbeitung in sich vereinen. Die Daten repräsentieren wesentliche Merkmale eines Objekts und seinen Zustand. Die vom Objekt zur Verfügung gestellten Verarbeitungsmethoden bestimmen sein typisches Verhalten. Diese Methoden werden von anderen Objekten oder durch äußere Ereignisse, wie etwa Benutzereingaben, aktiviert.

Wir verwenden im Kurs die objektorientierte Programmiersprache Java zur →Implementierung programmtechnischer Lösungen, die auf Rechnern ausführbar sind.

2.2 Eingabe, Verarbeitung, Ausgabe

Bei den älteren Programmiersprachen scheint das Modell der sequenziellen, schrittweisen Anweisungsverarbeitung, wie es der von Neumann-Architektur innewohnt, sehr stark durch.

Die Vorstellung, Computer seien sequenziell arbeitende Problemlösungsmaschinen, und das Schreiben eines Computerprogramms sei vergleichbar mit der Beschreibung eines Kochrezeptes, wird auch heute noch von vielen Programmiersprachenbüchern vermittelt.

Ihnen liegt die Vorstellung zu Grunde, die Arbeitsweise eines Computers entspräche einem sequenziellen Ablauf, der von einem Ausgangszustand zu einem Ergebnis führt und dann endet. In dieser Vorstellungswelt besteht die Aufgabe des Programmierers darin, eine bestimmte Folge von Anweisungen zu finden, die bestimmte Eingabedaten in ein gewünschtes Ergebnis umformen. Dieser Programmieransatz wird gelegentlich auch das **EVA-Prinzip** genannt (Abb. 2.2-1).

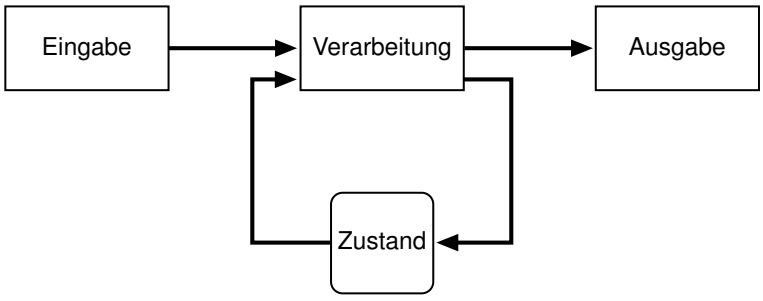


Abb. 2.2-1: EVA-Prinzip

EVA steht dabei für:

```
Eingabe, Verarbeitung, Ausgabe.
```

2.3 Interaktive Programme

Die Auffassung einer schrittweisen Abarbeitung von Anweisungen ist nicht falsch, liefert uns aber keinen brauchbaren Ansatz, um die Verhaltensweise vieler →interaktiver Anwendungen wie Platzreservierungssysteme, Videospiele, Tabellenkalkulatoren, Webserver, die Airbag-Steuerung in Pkws oder automatische Bankschalter zu erklären.

Interaktive Programme

Interaktive Anwendungsprogramme, denen wir heute vielfach begegnen, verhalten sich vielmehr wie eigenständige Agenten, die kontinuierlich auf Anforderungen oder Ereignisse in der Umgebung reagieren. Diese Anwendungen sind in ihre Umgebung eingebettet und interagieren mit dieser Umgebung, also mit Benutzerin, technischen Geräten wie Sensoren und Aktoren, Kommunikationseinrichtungen oder anderen Programmen.

Will man solche Anwendungen programmieren, genügt es nicht, sich zu fragen:

Welcher Schritt ist als Nächstes tun?

Die Aufgabe besteht eher darin, eine gedachte Welt miteinander kooperierender Agenten oder Einheiten zu entwerfen; sich also fragen:

- Welche Einheiten werden gebraucht?
- Welche Rolle spielen sie im Verbund mit anderen?
- Welchen Dienst bieten sie an, und welche Dienstleistung benötigen sie zur Erfüllung ihrer Aufgabe?
- Wie muss ihre innere Arbeitsweise gestaltet werden?
- Wie arbeiten sie zusammen?

Diese Denkweise wird von objektorientierten Programmiersprachen - so auch von Java - unterstützt.

Seit ihrer Veröffentlichung im Jahre 1995 erlebte die Programmiersprache Java einen unvergleichlichen Siegeszug - zunächst als noch Internetsprache. →Java-Applets boten eine neue Möglichkeit, aktive Webseiten zu gestalten. Inzwischen verdrängt Java aber selbst angestammte objektorientierte Sprachen wie →C++, →Smalltalk oder →Eiffel in der allgemeinen Anwendungsprogrammierung. Obwohl ähnlich zu C++ im Sprachaufbau, ist Java einfacher in seinen Sprachkonstrukten, und es vermeidet viele tückische Fehlerquellen von C++. Im Gegensatz zu Smalltalk weist Java ein strenges Typkonzept auf, und das Laufzeitsystem umfasst, anders als bei C++, eine →automatische Speicherbereinigung.

Selbsttestaufgabe 2.3-1:

Was ist die Grundidee, die hinter dem objektorientierten Entwurf steht?

Selbsttestaufgabe 2.3-2:

Warum ist das Geheimnissprinzip so wichtig für das Programmieren-im-Großen?

Selbsttestaufgabe 2.3-3:

Was bedeutet das Geheimnissprinzip?

2.4 Zur Rolle von Java

Wir werden im Verlauf des Kurses ein kleines Programmierprojekt bearbeiten und dabei nach Bedarf einige grundlegende Programmierkonzepte einführen. Zur konkreten Programmierung der jeweiligen Problemstellung werden wir die Programmiersprache →Java einsetzen.

Java

Abgesehen von den Vorteilen, die Java als objektorientierte Programmiersprache gegenüber prozeduralen Sprachen besitzt, gibt es noch andere Gründe, warum wir Java als Programmiersprache auswählten:

Warum Java?

- Java ist *plattformunabhängig*, d. h. es gibt für alle gängigen Rechnerplattformen Laufzeitumgebungen, unter denen ein und dasselbe Java-Programm ablaufen kann und dabei die gleichen Auswirkungen zeigt. Diese Eigenschaft hat den Vorteil, dass wir keine besonderen Voraussetzungen an Ihre Arbeitsumgebung stellen müssen. Es gibt sogar sehr klein gehaltene Laufzeitumgebungen, die speziell für den Einsatz in Mobiltelefonen, Taschenrechnern und eingebetteten Systemen gedach sind.
- Java erlaubt es, Programme, sog. →Java-Applets, zu entwickeln, die Webseiten interaktive Fähigkeiten mitgeben.
- Die zahlreichen Anleihen der Sprache Java bei C++ erleichtert später das Erlernen der Sprache C++, die in der Praxis weit verbreitet ist und Ihnen möglicherweise in Ihrem beruflichen Umfeld bereits begegnet ist, im Gegensatz zu Java aber für unerfahrene Programmierer zahlreiche Möglichkeiten für schwer zu findende Programmierfehler umfasst.

Zur Darstellung von **Modellen**, die abstrakte Lösungen gegebener Problemstellungen beschreiben, führen wir graphische Schreibweisen ein. Sie haben sich als Mittel zur Kommunikation mit den Auftraggebern und den späteren Benutzern des entstehenden Systems in der Praxis bewährt und sind unter der Bezeichnung →UML (engl. *Unified Modeling Language*) genormt.

Modell

Diese erste Kurseinheit führt Sie intuitiv in die objektorientierte Denkwelt ein, bevor wir in nachfolgenden Kurseinheiten auf Einzelheiten der Programmiersprache Java eingehen. Ausgangspunkt dieser intuitiven Einführung ist ein kleines Fallbeispiel, das das Geschehen in einer Autovermietung zum Gegenstand hat.

3 Aufgabenanalyse

Ausgangspunkt der Programmierung ist die Erschließung ausführlicher Informationen über Aufgaben, Arbeitspraktiken und Entwurfsoptionen aus Sicht der Benutzer eines Programmsystems. Erprobte **Vorgehensweisen** zur Bewältigung dieser Aufgabe benutzen Fragebögen, Interviewtechniken oder Aufgabenanalysemethoden, um einen möglichst umfassenden Satz an Anforderungen zu erhalten.

Vorgehensweise bei der Programmierung

Erst wenn die Anforderungen an das zu entwickelnde System und die Umgebungsbedingungen, unter denen es ablaufen soll, verstanden und dokumentiert sind, ist es sinnvoll, mit der Gestaltung des Programmaufbaus im Großen und der Implementierung konkreter Programmeinheiten zu beginnen.

3.1 Fallstudie

Weil es schwierig ist, abstrakte Theorie ohne konkrete Beispiele zu verstehen, wollen wir diese Vorgehensweise im Stil eines Projekts an einer einfachen Fallstudie vorführen. Im Laufe des Kurses werden wir immer wieder auf dieses Fallbeispiel zurückkommen, wenn es darum geht, Begriffe, Strukturen und Verhaltensweisen zu veranschaulichen.

Als Fallstudie wählen wir das Geschehen in der Autovermietung *Direkt-Vermietung und Transport* (kurz: DVT). Für die Fallstudie geben wir folgende **Aufgabenbeschreibung** vor:

Aufgabenbeschreibung

In der Autovermietung DVT sollen alle kundenbezogenen Geschäftsabläufe durch ein einheitliches Informationssystem unterstützt werden. Die bisherige papiergebundene Verwaltung der Kunden- und Fahrzeugstammdaten soll nur noch im Rechner abgewickelt werden. Die Kundenberaterinnen sollen mit Hilfe leicht verständlicher Menüs und Eingabemasken durch die verschiedenen Geschäftsprozesse geführt werden. Dazu gehören die

- Kundenberatung,
- Reservierung,
- Vermietung von Fahrzeugen und die
- Abrechnung mit den Kunden.

Weitere Arbeitsbereiche wie die interne Buchhaltung und Verwaltung, die Werbung, die Tarif- und Produktplanung sowie der Erwerb und Verkauf von Mietfahrzeugen sind nicht Gegenstand des Auftrags. Der Einfachheit halber nehmen wir an, dass sich der Fahrzeugbestand bei DVT nicht verändert.

Unser Ziel ist es, für dieses Fallbeispiel ein größeres Java-Programm zu entwickeln, das die Abläufe im Verleihbüro DV-technisch unterstützt.

Beispiel 3.1-1: Spielszenen in einem Autoverleih
Da wir weder eine Befragung der Akteure in der Autovermietung noch eine detaillierte Aufgabenanalyse vor Ort durchführen können, haben wir einige Szenen aus der Realität in den fünf animierten Spielszenen nachgestellt.

- *Die erste Szene „Beratung und Information“ zeigt einBeratungs- und Informationsgespräch zwischen einer Kundenbetreuerin und einemInteressenten.*

***Spielszene 3.1-1**: Spielszene: Beratung und Information*

Beratung und Information

Kunde: Guten Tag, ich möchte ein AUTO mieten.

Kundenberaterin: Welchen WAGEN hätten Sie denn gerne?Falls Sie einen PKW suchen, habe ich die folgenden Angebote für Sie:

- *Einen KLEINWAGEN für zwei Personen,*
- *einen Wagen der KOMPAKTKLASSE für vier bis fünf Personen mit etwas Gepäck oder*
- *einen VAN zur Beförderung einer kleinen Gruppe oder weniger Personen mit viel Gepäck.*

Sollten Sie Lasten befördern wollen, so gibt es einen KLEINTRANSPORTER

...oder einen LASTWAGEN, wenn Sie den entsprechenden FÜHRERSCHEIN haben.

Mit unserem PICKUP können Sie Personen und Lasten transportieren.

- *Die Szene „Fahrzeug mieten“ beschreibt die Situation,dass sich ein Kunde zum Autoverleih begibt, dort seine Wünsche vorträgt, einenMietvertrag abschließt und das Fahrzeug in Empfang nimmt.*

***Spielszene 3.1-2**: Spielszene: Fahrzeug anmieten*

Fahrzeug mieten

Kunde: Guten Tag, ich möchte ein AUTO mieten.

Kundenbetreuerin: Welchen WAGEN hätten Sie denn gerne?Falls Sie einen PKW suchen, habe ich die folgenden Angebote für Sie:

- *Einen KLEINWAGEN für zwei Personen,*
- *einen Wagen der KOMPAKTKLASSE für vier bis fünf Personen mit etwas Gepäck oder*
- *einen VAN zur Beförderung einer kleinen Gruppe oder weniger Personen mit viel Gepäck.*

Sollten Sie Lasten befördern wollen, so gibt es einen KLEINTRANSPORTER

...oder einen LASTWAGEN, wenn Sie den entsprechenden FÜHRERSCHEIN haben.

Mit unserem PICKUP können Sie Personen und Lasten transportieren.

Kunde: Ich möchte gerne einen Wagen der Kompaktklasse.Ich brauche ihn sofort und werde ihn am Sonntag abend in Berlin wieder abgeben.

Kundenbetreuerin: Einen Augenblick bitte ... Ja, das geht.Ich habe da einen Nissan Sentra.Er kostet 79 DM am Tag. 100 km sind frei, jeder weitere Kilometer kostet 49 Pf.Zusätzlich müssten Sie noch einen Aufschlag von 40 DM zahlen, weil Sie den Wagen an einem anderen Ort zurückgeben wollen.Wir nennen dies Einwegmiete.Soll ich einen Vertrag anfertigen?

Kunde: Ja, bitte.

Kundenbetreuerin: Gut, dann benötige ich Ihren Führerschein, Ihre Kreditkarte und einen Ausweis.

Kunde: Hier sind mein Führerschein und meine Kreditkarte. Die Adresse im Führerschein stimmt noch.

Kundenbetreuerin: Vielen Dank. Möchten Sie eine Vollkasko-Versicherung abschließen? Damit sind alle Schäden abgesichert.Die Selbstbeteiligung beträgt 450 DM und die Versicherung würde für den Zeitraum 38 DM betragen.

Kunde: Ja, gerne. Sicher ist sicher!

Kundenbetreuerin: Gut, ich belaste dann Ihre Karte mit 300 DM. Wenn Sie den Wagen vollgetankt zurückgegeben, wird der Überschuss auf Ihr Kartenkonto wieder gutgeschrieben.Ich möchte Sie nur noch bitten, den Vertrag zu unterzeichnen.

Kunde: Ja, natürlich ... So, fertig.

Kundenbetreuerin: Danke. Der Wagen steht für Sie auf unserem Parkdeck auf dem Platz A 56 bereit.Das Kennzeichen lautet HH-WS 531, der Kilometerstand ist 15 840 km.Den Schlüssel erhalten Sie von unserem Servicemitarbeiter.Und hier ist Ihre Kopie des Mietvertrages.

Kunde: Danke und auf Wiedersehen.

Kundenbetreuerin: Vielen Dank und gute Fahrt.

- *Die Szene „telefonische Reservierung“ illustriert denFall, dass ein Kunde telefonisch ein Fahrzeug, das er oder eine andere Personzu einem späteren Termin in Empfang nehmen und benutzen will, reservierenlässt.*

***Spielszene 3.1-3**: Spielszene: Telefonische Reservierung*

Telefonische Reservierung

Kundenberaterin: Guten Tag! DVT-Autoverleih, mein Name ist Gerda Busch. Wie kann ich Ihnen helfen?

Kunde: Hallo, hier Schneider. Ich möchte gerne einen Van reservieren.Wir kommen morgen um 18 Uhr am Flughafen Frankfurt an. Der Wagen soll dort angemietet werden.Wir benötigen ihn für zwei Wochen und würden ihn dann am Sonntag, den 14. August wieder in Frankfurt abgeben. Ist das möglich?

Frau Busch: Lassen Sie mich sehen - ja, das geht. Ich wiederhole: Sie holen den Wagen am Frankfurter Flughafen am 1.August um 18 Uhr ab.Die Rückgabe des Wagens wird am gleichen Ort erfolgen, am 14. August. Brauchen Sie Preisinformationen?

Herr Schneider: Nein, danke. Ich habe mich bereits gestern über die Wagenklassen und Preise informiert.

Frau Busch: Gut. Haben Sie schon einmal bei uns einen Wagen gemietet? Dann haben wir Sie in der Kundendatei.

Herr Schneider: Nein, dies ist meine erste Anmietung.

Frau Busch: Dann benötige ich die Nummer Ihrer Kreditkarte und das Gültigkeitsdatum.

Herr Schneider: Die Nummer lautet 2789-3456-5058 und ist gültig bis einschließlich September 2001.

Frau Busch: In Ordnung. Ihre Reservierungsnummer ist F-3580-108 SC.Ich werde sie Ihnen aber auch noch einmal - zusammen mit der Bestätigung Ihrer Reservierung - faxen.

Herr Schneider: Danke. Meine Faxnummer ist 987-1234.

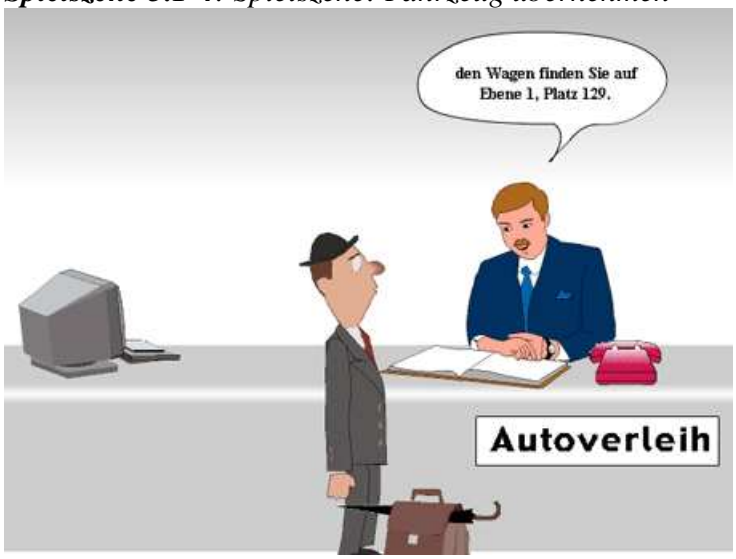
Frau Busch: Vielen Dank für Ihre Reservierung. Der Wagen wird zum gewünschten Termin in Frankfurt bereitstehen.

Herr Schneider: Auf Wiederhören.

Frau Busch: Danke und auf Wiederhören.

- *Die Szene „Fahrzeug übernehmen“ ergänzt das vorherigeSzenario genau um den Fall, dass ein Kunde einen zuvor reservierten Wagen inEmpfang nehmen möchte.*

***Spielszene 3.1-4**: Spielszene: Fahrzeug übernehmen*



Reserviertes Fahrzeug abholen

Kunde: Guten Tag, Petermann ist mein Name. Ich hatte telefonisch ein Auto reserviert.

Kundenberater: Guten Tag, Herr Petermann. Haben Sie auch die Reservierungsnummer parat?

Petermann: Hm, ja, irgendwo in meiner Aktentasche.

Kundenberater: Aach, lassen Sie ruhig, ich habe Ihre Reservierung schon gefunden.Sie hatten einen Golf vorbestellt. Leider haben wir im Moment keinen da;sind Sie auch mit einem Mercedes A-Klasse einverstanden? Natürlich zu den gleichen Bedingungen wie bei dem Golf.

Petermann: Ja, gerne. Hier ist meine Kundenkarte.

Kundenberater: Danke. Wir buchen dann die Kosten von Ihrem Firmenkonto.Hier ist Ihr Vertrag, entsprechend der Daten Ihrer Kundenkarte.

Petermann: Ja, genau, mit Vollkasko-Versicherung und Rabatt. Ich bringe den Wagen dann wie immer ungetankt zurück.

Kundenberater: Das geht in Ordnung. Hier ist Ihr Schlüssel, den Wagen finden Sie auf Ebene 1, Platz 129.Das Kennzeichen lautet HH-DV 123. Der derzeitige Kilometerstand ist 12.345.

Petermann: Vielen Dank und Tschüß.

- *Die letzte Szene „Fahrzeugrückgabe und Bezahlung“veranschaulicht den Abschluss des Mietverhältnisses mit Fahrzeugrückgabe,Abrechnung und Bezahlung des Mietpreises.*

***Spielszene 3.1-5**: Spielszene: Fahrzeugrückgabe und Bezahlung*

Wagenrückgabe und Abrechnung des Vertrags

Kunde: Guten Tag, ich möchte meinen Leihwagen zurückgeben und die Rechnung bezahlen.Hier sind der Vertrag und der Kontrollschein Ihres Mitarbeiters vom Parkdeck.

Kundenberaterin: Danke, einen Augenblick bitte ... Das macht dann 428 DM.Da wir Ihnen 400 DM im voraus von Ihrer Karte abgebucht hatten, bleibt ein Restbetrag von 28 DM.Soll der Betrag von Ihrer Kreditkarte abgebucht werden?

Kunde: Ja, bitte.

Kundenberaterin: Bitte unterschreiben Sie hier ...Danke.Hier ist Ihre Quittung. Auf Wiedersehen!

Kunde: Auf Wiedersehen, ich werde Sie weiterempfehlen.

□

Was Sie aus den Spielszenen nicht entnehmen konnten, ist die Tatsache, dass die Autoverleihfirma DVT ihr Geschäft noch mit herkömmlichen Mitteln wie Karteikästen, Aktenordnern und Schreibmaschinen ausführt.

Die **Ist-Situation**, nach der bei DVT Geschäftsvorfälle derzeit noch abgewickelt werden, ist in Abb. 3.1-1 bildlich dargestellt.

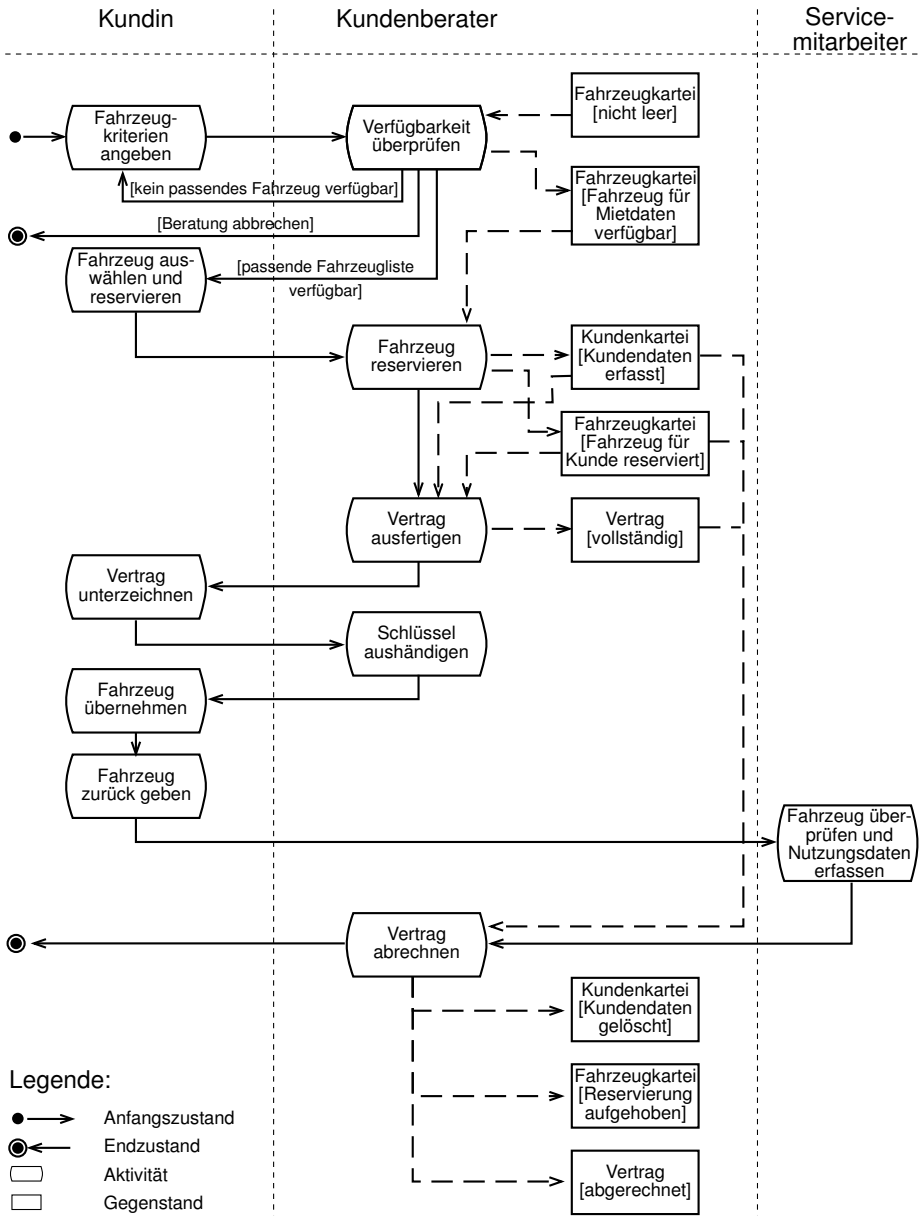


Abb. 3.1-1: Ist-Situation

Das Aktivitätsdiagramm in Abb. 3.1-1 illustriert den Ist-Zustand unseres Autoverleihs am Beispiel einer abgeschlossenen Anmietung. Der Einfachheit halber wurden bestimmte Ausnahmesituationen weggelassen. Ein Aktivitätsdiagramm ist eine spezielle Form eines Zustandsdiagramms, das im Wesentlichen über Aktivitäten redet. Eine Aktivität beschreibt einen einzelnen Schritt in einem Geschäftsvorfall. Jedes Aktivitätsdiagramm besitzt einen eindeutigen Anfangszustand, und es kann mehrere Endzustände haben.

Das Diagramm ist wie folgt zu lesen: Eine eingehende Transition (grafisch durch eine gerichtete Kanten dargestellt) löst eine Aktivität aus. Abgehende Transitionen werden durch den Abschuss einer Aktivität ausgelöst. Alternative ausgehende Transitionen können mit Bedingungen beschriftet sein, die sich gegenseitig logisch ausschließen.

Die Ausführung einer Aktivität kann auch vom Zustand eines (konkreten oder abstrakten) Gegenstands abhängen oder diesen verändern. Gegenstände werden grafisch durch Rechtecke dargestellt, die mit dem Namen des Gegenstands und dem in eckige Klammern angegebenen Zustand beschriftet sind. Eine gestrichelt dargestellte Transition zwischen einem Gegenstand und einer Aktivität bedeutet, dass die Aktivität diesen Zustand des Gegenstands voraussetzt. Eine Transition zwischen einer Aktivität und einem Gegenstand besagt, dass sich der genannte Zustand nach Abschluss der Aktivität ergibt.

3.2 Analyse der Anwendungswelt

Fachbegriffe verstehen

Jede Beschreibung der Realität hängt von der Wahl der Fachbegriffe sowie von einer vorsichtigen Erläuterung der Phänomene ab, die jeder Begriff bezeichnet. Wichtig ist, dass wir uns vor dem Entwurf eines Programmsystems ein genaues Bild der Wirklichkeit verschaffen und dieses auch mit viel Disziplin umfassend und möglichst eindeutig beschreiben. Unsere Wahrnehmung der Wirklichkeit müssen wir nachvollziehbar mit den schematischen und formalen Beschreibungen, die ein Programm ausmachen, abgleichen. Nur so können wir sicher sein, dass die Auswirkungen des Programms die gewünschten Resultate liefern.

In jeder Anwendungswelt werden Begriffe benutzt, die von vorne herein nur selten von Systemanalytikern voll verstanden werden. Sehen wir uns jetzt einzelne Szenen der Autovermietung noch einmal genauer an, um:

- die Handlungsträger,
- ihr Zusammenwirken und
- ihre für die Geschäftsabläufe wesentlichenHandlungen

zu erkennen,

- die Informationen, die sie austauschen,

nachzuvollziehen,

- die Gegenstände, mit denen sie umgehensowie
- die Beziehungen, die zwischen diesen Gegenständenbestehen,

zu beschreiben und

- erste Ideen zu diskutieren, wie die Geschäftsvorfälledurch Computer unterstützt werden können.

Wir werden die Analyse der durch die Fallstudie gegebenen Anwendungswelt in drei Schritten angehen:

1. Bestimmung der wesentlichen Systemfunktionen und Anwenderrollen,
2. Festlegung der Gegenstände der Kommunikation und Bearbeitung,
3. Detaillierte Beschreibung der Programmieraufgabe.

3.3 Anwendungsfälle undAkteure

Zunächst wollen wir das Verhalten des Systems DVT abstrakt dokumentieren, ohne auf die inneren Strukturen und Abläufe einzugehen. Die Arbeitsabläufe im Anwendungsfeld werden wir in so genannten Anwendungsfällen beschreiben. Anwendungsfälle erfassen die Funktionalität, die von Nutzern gefordert wird.

Definition 3.3-1: Anwendungsfall
*Ein **Anwendungsfall** (*engl. *use case) bezeichnet eine typische Interaktion zwischen einem Anwender und einem System aus der Sicht des Anwenders. Ein Anwendungsfall umfasst Folgen von Aktionen, die reguläre, aber auch vom normalen Verhalten abweichende Abläufe beschreiben.*

Anwendungsfälle erfassen genau die Systemaktionen, die für das Zusammenwirken zwischen dem System und seinen Benutzerinnen erforderlich sind und dazu beitragen, bestimmte Ziele der Benutzer zu erreichen.

Anwendungsfälle werden durch Anwender oder durch Zeitereignisse angestoßen.

□

Anwendungsfälle stellen einen Ausgangspunkt für spätere *Funktionstests* des fertigen Systems dar.

Systemaktionen, die zum Anwender hin nicht sichtbar sind, werden nicht in Anwendungsfällen erfasst.

Beispiel 3.3-1: Anwendungsfall
Die Spielszene 3.1-1 aus unserer Fallstudie DVT können wir als Anwendungsfall verstehen. Der Anwendungsfall „Beratung und Information“ bezeichnet eine wichtige Dienstleistung, die ein Interessent in Anspruch nehmen kann, um sich über Mietpreise, das Fahrzeugangebot und Sonderkonditionen zu informieren.

Die Dienstleistung des Systems DVT wird angestoßen durch ein äußeres Ereignis, nämlich durch die fernmündliche Anfrage oder persönliche Vorsprache im Verleihbüro.

Vier weitere Anwendungsfälle ergeben sich unmittelbar aus den Spielszenen:

- *Fahrzeug reservieren (Spielszene 3.1-3),*
- *Fahrzeug mieten (Spielszene 3.1-2),*
- *Fahrzeug zurückgeben (Spielszene 3.1-5),*
- *Mietvertrag abrechnen (Spielszene 3.1-5).*

□

In Anwendungsfällen treten Anwender und andere Systeme, die mit dem betrachteten System interagieren, immer in bestimmten Rollen auf. In diesen Rollen sind sie mit bestimmten Verantwortlichkeiten und Rechten ausgestattet.

Definition 3.3-2: Akteur
Anwender und externe Systeme, die an Anwendungsfällen teilnehmen aber selbst nicht Teil des zu realisierenden Systems sind, werden als ***Akteure*** *(engl. actor) bezeichnet.*

□

Es ist hier wichtig, zwischen *Rollen* und konkreten Personen wie Frau Busch zu unterscheiden. Ein und dieselbe Person kann z. B. die Rolle der Kundendienstleiterin und gelegentlich die Rolle der Kundenberaterin einnehmen.

Umgekehrt können verschiedene Personen zu verschiedenen Zeiten oder auch zur gleichen Zeit als Kundenberater aktiv sein. Diese Individuen sollen in Anwendungsfällen nicht weiter unterschieden werden, da sie in Bezug auf die betrachteten Geschäftsvorgänge alle die gleiche Rolle spielen.

Akteure nehmen also immer Bezug auf Rollen und nicht auf konkrete Personen.

Beispiel 3.3-2: Akteure
Im Fall der Autovermietung erkennen wir die folgenden Akteure:

- *Interessenten, die sich über dasFahrzeugangebot der Autovermietung informieren lassen,*
- *Kundinnen, die ein bestimmtes Fahrzeugreservieren lassen und zu diesem Zweck persönliche Daten hinterlegen müssen,*
- *Kundenbetreuer, die Kundentelefonisch oder am Kundenschalter behilflich sind, und*
- *Servicemitarbeiter, die bei derRückgabe eines Fahrzeugs den Kilometerstand, die Füllstandsanzeige des Tanksund den Gesamtzustand des Fahrzeugs überprüfen und einen Kurzbericht für die Kundenbetreuer im Büro anfertigen.*

□

Hinweis

Auf diese vier Akteure wollen wir uns im Projekt beschränken. Andere Rollen wie die der Geschäftsführerin oder des Fahrers werden wir nicht berücksichtigen. Hinzu kommen wird später jedoch noch der Akteur „Kreditkartengesellschaft“, die für die Abrechnung der Verträge verantwortlich ist.

Definition 3.3-3: Beziehung zwischen Akteur und Anwendungsfall
Die Teilnahme eines Akteurs an einem Anwendungsfall wird durch eine ***Kommunikationsbeziehung*** *zwischen beiden ausgedrückt.*

Andere Beziehungen zwischen Akteuren und Anwendungsfällen gibt es nicht.

□

Die Beziehungen zwischen Anwendungsfällen und Akteuren lässt sich übersichtlich mit Hilfe von Anwendungsfalldiagrammen darstellen. Anwendungsfall-

diagramme sind dafür gedacht, die ermittelten Anforderungen zu dokumentieren, ohne dabei auf das Verhalten des künftigen Systems einzugehen.

Definition 3.3-4: Anwendungsfalldiagramm
*Ein **Anwendungsfalldiagramm** beschreibt grafisch die Zuammenhänge zwischen einer Menge von Akteuren und Anwendungsfällen. Ein solches Diagramm besteht aus:*

- ***Akteuren**, die wir durchStrichmännchen darstellen, wobei die Rolle des Akteurs unter das grafische Symbol geschrieben wird;*
- ***Anwendungsfällen**, die als Ellipsenerscheinen, in denen die Bezeichnung des Anwendungsfalls aufgeführt wird,*
- ***Kommunikationsbeziehungen**, die als ungerichtete Kanten zwischen Akteurenund Anwendungsfällen dargestellt werden, und*
- ***Beziehungen zwischen Anwendungsfällen**, die als gerichtete Kanten zwischen Anwendungsfällen erscheinen.*

□

Beispiel 3.3-3: Anwendungsfalldiagramm
Abb. 3.3-1 zeigt alle bisher erwähnten Anwendungsfälle und Akteure in einem Anwendungsfalldiagramm.

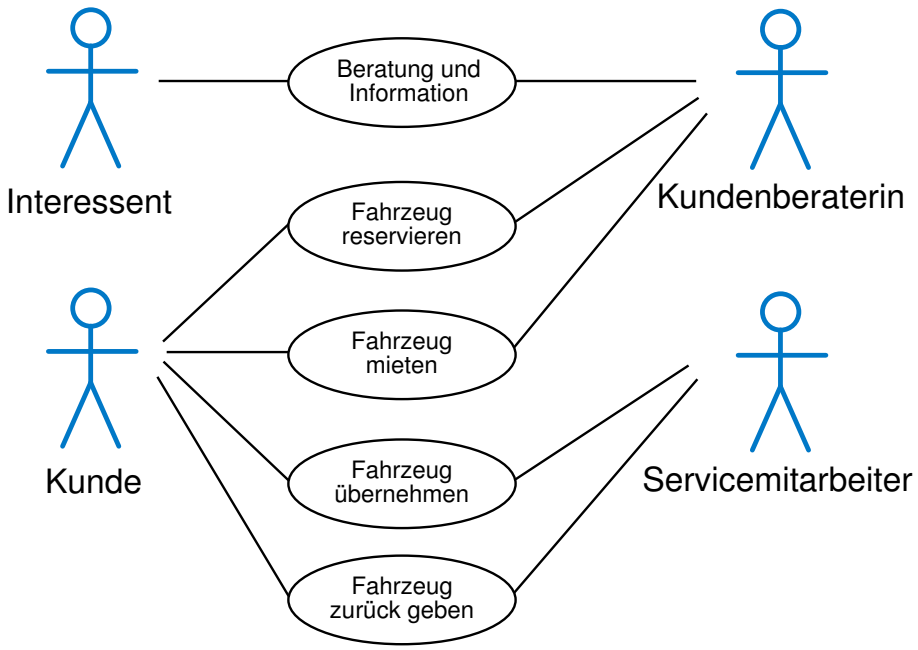


***Abb. 3.3-1**: Anwendungsfalldiagramm*

Die Kommunikationsbeziehungen geben an, welche Akteure an welchen Anwendungsfällen beteiligt sind. Sie machen aber keine Angaben über den Inhalt der Kommunikation.

□

Der Informationsgehalt der Anwendungsfalldiagramme ist offensichtlich zu gering, um die wesentlichen Aspekte von Anwendungsfällen zu dokumentieren. Im nächsten Schritt führen wir deshalb eine ergänzende Beschreibungstechnik ein.

3.4 Anwendungsfallbeschreibungen

Die grafische Darstellung von Anwendungsfällen allein bietet keine ausreichende Informationsbasis, um danach einen gezielten Programmentwurf anzufertigen. Wir müssen auch die Abläufe, von denen jeder Anwendungsfall abstrahiert, mögliche Abweichungen von Standardabläufen und die Bedingungen, unter denen ein normaler oder abweichender Ablauf angestoßen wird, festlegen.

Definition 3.4-1: Szenario
Wir nennen eine einzelne Abfolge von Aktionen, d. h. einen Ablaufpfad in einem Anwendungsfall, ein ***Szenario****.*

*Szenarien beschreiben aktuelle Arbeitssituationen. Ein Szenario, das die Sicht der Anwender wiedergibt, fasst das zu entwickelnde Systems als schwarzen Kasten (*engl. *black box) auf. Die innere Arbeitsweise des System bleibt den Anwendern verborgen, von Interesse ist allein das externe Systemverhalten, wie es sich aus der Sicht eines Anwenders darstellt. Das Augenmerk liegt bei dieser Perspektive auf typischen Interaktionen zwischen Benutzern und dem Anwendungssystem.*

□

Szenarien spielen in der objektorientierten Programmierung eine wichtige Rolle. Sie tauchen in verschiedenen Entwicklungssituationen auf, etwa bei der Aufstellung der Anforderungen oder bei der detaillierten Bestimmung der Verhaltensspezifikation eines neuen oder eines zu verändernden Anwendungssystems.

Auf einer detaillierteren Betrachtungsebene, zum Beispiel während des Entwurfs, benutzt man Szenarien dazu, die Interaktion zwischen Softwarekomponenten darzustellen. Hier liegt das Augenmerk auf der Beschreibung der Art und Weise, wie die jeweilige Aufgabe unter Verwendung von Arbeitsmitteln und Arbeitsgegenständen erledigt werden kann [Züllighoven98].

Eine tabellarische Darstellung der Szenarien, die alle möglichen Interaktionen zwischen einem Anwendungsfall und den mit ihm kommunizierenden Akteuren beschreiben, ergänzt deshalb die grafische Darstellung eines Anwendungsfalls.

Definition 3.4-2: Beschreibungsschema für Anwendungsfälle
Für die Beschreibung der in Anwendungsfällen auftretenden Interaktionen werden wir das in Tabelle 3.4-1 angegebene Schema verwenden.

Tab. 3.4-1: *Beschreibungsschema fürAnwendungsfälle*

Im folgendenSchema sind nur an den nicht fett angegebenen Stellen entsprechende Eingabenvorzunehmen			
Anwendungsfall		*Bezeichnung*	
Kurzbeschreibung			
Beteiligte Akteure		*Kürzel* *...*	*Bezeichnung* *...*
Auslöser,Vorbedingungen		*Aktion oder Ereignis, das den Anwendungsfallauslöst*	
Ergebnis, Nachbedingungen		*Ergebnis der Interaktion*	
		Ablauf, Interaktionen	
Nr.	*Akteur / System*	*Kurzbeschreibung dereinzelnen Schritte eines erfolgreichen Ablaufs, der entweder mit der auslösendenAktion oder dem Eintreten der Vorbedingung beginnt und mit dem gewünschten Ergebnis oder dem Eintritt der erwartetenNachbedingung endet; dabei sollte immer ein Tätigkeitswort verwendetwerden*	
...	*...*	*...*	
		Erweiterungen, alterativeInteraktionen	
Nr.	*Akteur / System*	*Bedingung, die zurErweiterung des Hauptablaufs führt, sowie Angabe der entsprechenden Aktion oder derBezeichnung eines untergeordneten Anwendungsfalls*	
...	*...*	*...*	

Beschreibungen nach diesem Schema enthalten:

- *eine* ***Bezeichnung desAnwendungsfalls****;*
- *eine****Kurzbeschreibung****;*
- *die****beteiligten Akteure****;*
- *die Angabe eines* ***auslösendenEreignisses****;*
- ***Ergebnisse*** *oder* ***Bedingungen****, diesich nach Abschluss des Anwendungsfalls ergeben;*
- *eine* ***Ablaufbeschreibung*** *ineinzelnen Schritten, wobei jeder Schritt durch eine der beiden folgendenEreignisse bestimmt wird:*
  *- eine****Aktion*** *und den an diesem Anwendungsfall beteiligten Akteur(oder das System) oder*
  *- ein****Zeitereignis****, das den Anwendungsfall auslöst;*
- ***Ausnahmen*** *und* ***alternativeAbläufe*** *zusammen mit den* ***Bedingungen****, unter denen siebearbeitet werden;*
- *offene Punkte und Fragen, die mit der Auftraggeberin oder dem Nutzer noch nicht abschließend geklärt werden konnten;*

- *Bezugsdokumente.*

□

Der schematische Aufriss in Tabelle 3.4-1 wird uns helfen, keine wesentlichen Anteile einer Anwendungsfallbeschreibung zu vergessen und die Beschreibungen verschiedener Entwicklerinnen zu vereinheitlichen.

Beispiel: Anwendungsfall Beratung
Eine konkrete Anwendungsfallbeschreibung für den Anwendungsfall „Beratung und Information“ ist in Tabelle 3.4-2 angegeben.

Tab. 3.4-2: *Anwendungsfall „Beratung“*

Anwendungsfall		*Beratung und Information*
Kurzbeschreibung		*Ein Interessent wird über das Fahrzeugangebot, dieVerfügbarkeit und den Mietpreis für ein gewähltes Mietdatuminformiert*
Beteiligte Akteure		*Kb* — *Kundenberaterin* *In* — *Interessent*
Auslöser,Vorbedingungen		*Telefonische oder persönliche Anfrage einesInteressenten*
Ergebnis, Nachbedingungen		*Dem Interessenten wird die Preisinformation fürdie gewählte Fahrzeugkategorie und Mietdauerübermittelt*
		Ablauf, Interaktionen
1		***Erfassung derInteressentenwünsche***
1.1	*In*	*Interessent nennt die gewünschte Fahrzeugkategorie, den Abholort, das Datum der Abholung und die Mietdauer*
1.2	*Kb*	*Die Verfügbarkeit eines den angegebenen Kriterien entsprechenden Fahrzeugs wird mit Hilfe der Fahrzeugkartei ermittelt und im positiven Fall dem Interessenten mitgeteilt.*
1.3	*Kb*	***Ausnahme:****Kein Fahrzeug am gewählten Abholort zum gewünschten Zeitpunkt verfügbar*
1.4	*In*	*Gibt neue Kriterien an oder bricht die Beratung ab*
		AbweichendeInteraktionen
2		***Interessent hat untere Altersgrenze für Anmietung einesLKW nicht erreicht***
3.1	*Kb*	*Interessent ist für Anmietung dergewünschten Fahrzeugkategorie zu jung (unter 21 Jahren generell nicht möglich,unter 25 Jahren ist die Anmietung eines größeren LKW nicht zulässig)*

Fortsetzung auf der nächsten Seite

Fortsetzung von der vorhergehenden Seite		
3.2	*In*	*Gibt andere Fahrzeugkategorie an oder bricht Beratung ab*

Die Anwendungsfallbeschreibung umfasst einen Hauptablauf, eine Ausnahmesituationen und einen abweichenden Ablauf.

□

Hinweis Folgende Fragen sollte man sich stellen, wenn man die Abläufe für einen Anwendungsfall erhebt:

- Warum benutzt ein Akteur das System?
- Welche Art der Antwort erwartet der Akteur von einerAktion?
- Was muss der Akteur tun, um das System benutzen zukönnen?
- Welche Information muss der Akteur dem Systemübermitteln?
- Welche Information erwartet der Akteur als Antwortvom System?

Selbsttestaufgabe 3.4-1:

Erstellen Sie eine Anwendungsfallbeschreibung unter Verwendung des Schemas in Tabelle 3.4-1 für den Anwendungsfall „Fahrzeug reservieren“! Nehmen Sie gegebenenfalls Spielszene 3.1-3 zu Hilfe.

3.5 Beziehungen zwischen Anwendungsfällen

Sehen wir uns die Anwendungsfälle aus der Autovermietung etwas genauer an, so erkennen wir, dass einzelne Anwendungsfälle andere Anwendungsfälle enthalten.

Der Anwendungsfall „Fahrzeug mieten“ umfasst z. B. die gleichen Tätigkeiten, wie sie auch notwendig sind, um ein Fahrzeug zu reservieren. Weiterhin gilt, dass ein Kunde, der ein Fahrzeug reservieren möchte und sich über den Typ des gewünschten Fahrzeugs noch nicht im Klaren ist, zunächst Informationen über das Fahrzeugangebot einholen wird.

Beispiel 3.5-1: Beziehungen zwischen Anwendungsfällen
In Abb. 3.5-1 sehen Sie unser erstes Anwendungsfalldiagramm ergänzt um den Anwendungsfall „beraten und informieren“ und die Beziehungen „enthält“ zwischen den Anwendungsfällen:

- *„beraten und informieren“ und „Fahrzeug reservieren“sowie*
- *„Fahrzeug reservieren“ und „Fahrzeugmieten“.*

*Die **enthält-Beziehung** erlaubt es uns, auf anschauliche Weise bestimmte Abläufe, die in verschiedenen Anwendungsfällen vorkommen, herauszuziehen, um Wiederholungen zu vermeiden.*

Das Prinzip der Wiederverwendung und Vermeidung von Redundanz wird uns im Laufe des Kurses noch häufiger begegnen.

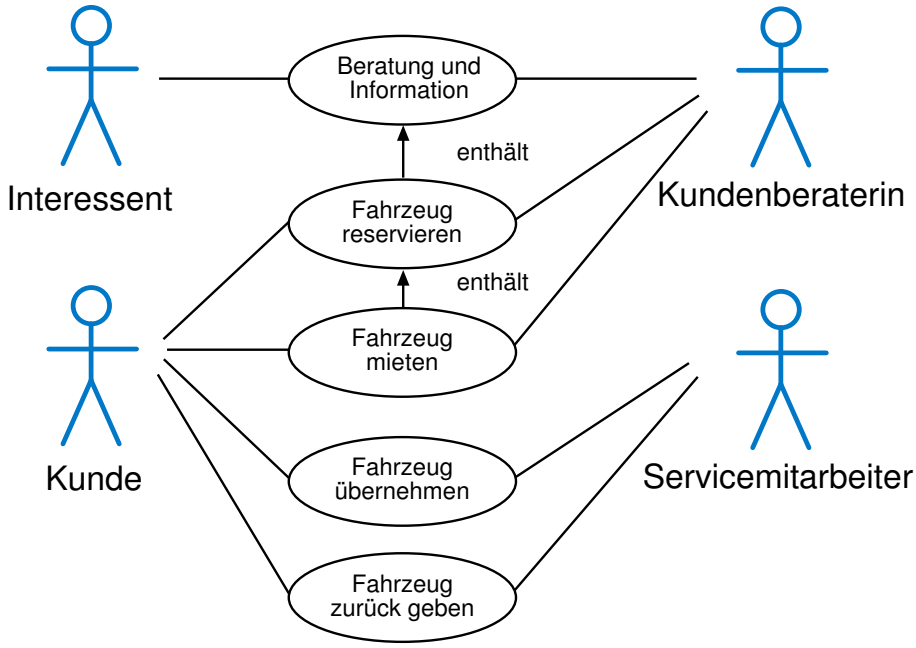


***Abb. 3.5-1**: Beziehungen zwischen Anwendungsfällen*

□

Unser Anwendungsfall „Beratung und Information“ deutet bereits an, dass DVT zwischen Privat- und Firmenkunden unterscheidet. Eine Nachfrage beim Auftraggeber ergibt, dass Firmenkundenverträge monatlich über ein vereinbartes Firmenkonto abgewickelt werden, während Privatkundenverträge in jedem Einzelfall nur über eine Kreditkartengesellschaft abgerechnet werden. Fallstudie

Definition 3.5-1: erweitert-Beziehung
Die erweitert-Beziehung drückt aus, dass ein Anwendungsfall an einer bestimmten Stelle unter bestimmten Bedingungen durch einen anderen erweitert wird. Die Beziehung wird häufig dafür benutzt, optionales Verhalten vom Standardverhalten zu unterscheiden.

□

Wir ergänzen unsere Sammlung von Anwendungsfällen um den Spezialfall „*Firmenkundendaten erfassen*“ und betrachten ihn als Erweiterung des Anwendungsfalls „*Kundendaten erfassen*“. Die Bedingung dafür ist, dass ein Kunde sich als Firmenkunde ausweist.

Wir sagen: „*Firmenkundendaten erfassen*“ **erweitert** „*Kundendaten erfassen*“.

In Anwendungsfalldiagrammen stellen wir die erweitert-Beziehung durch eine gerichtete Kante vom erweiternden zum erweiterten Anwendungsfall dar.

Später beim Programmentwurf werden wir darauf eingehen, an welcher Stelle im Ablauf des erweiterten Anwendungsfalls die Erweiterung den Standardablauf verändert.

Abb. 3.5-2 veranschaulicht diese Ergänzung in unserem Anwendungsfalldiagramm. Die Differenzierung in Privat- und Firmenkunden, die der Unterscheidung der beiden Anwendungsfälle zu Grunde liegt, geht einher mit einer Spezialisierung des Akteurs Kunde in Privat- und Firmenkunde.

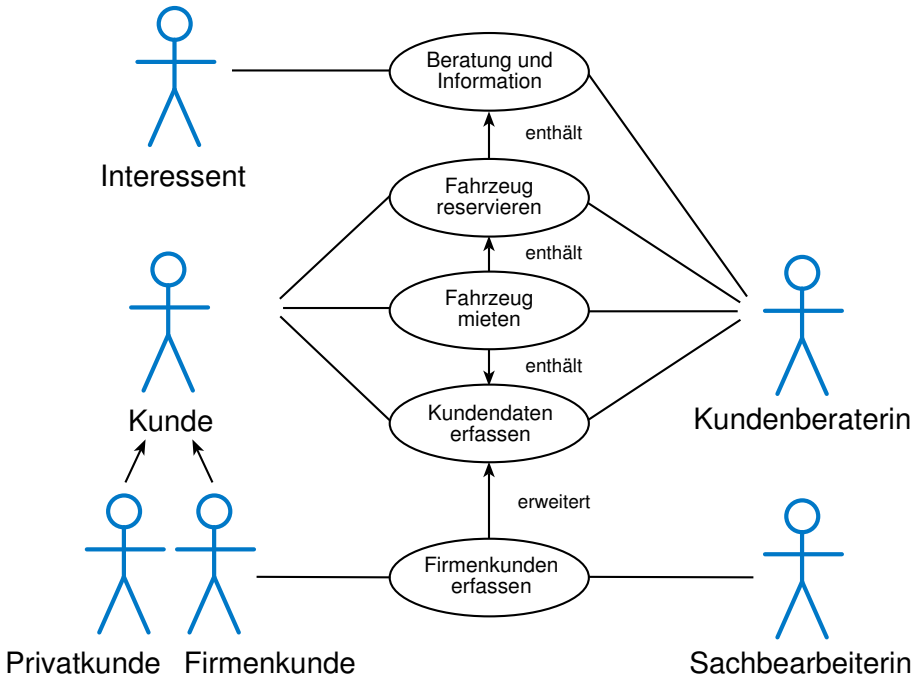


Abb. 3.5-2: erweitert-Beziehung

Aus Gründen der Übersichtlichkeit wurden die Anwendungsfälle „Fahrzeug übernehmen“ und „Fahrzeug zurück geben“ sowie der Akteur Servicemitarbeiter weggelassen.

Für den Anwendungsfall „Kreditwürdigkeit überprüfen“ werden Information von einer externen Einrichtung, nämlich der Kreditkartenzentrale, benötigt. Solche Fälle werden durch sekundäre Akteure beschrieben.

Sekundäre Akteure

Die bisher betrachteten Anwendungsfälle dokumentieren das Ergebnis der **Ist-Analyse**, d. h. das Ergebnis der Untersuchung der existierenden Abläufe und Zuständigkeiten.

Ist-Analyse

Bei der Beschreibung der Anwendungsfälle im Hinblick auf das zu entwickelnde System - wir wollen dies als **Soll-Analyse** bezeichnen - müssen wir unsere Sichtweise leicht verschieben. Wir wollen nämlich genau die Schnittstelle zwischen den Angestellten des Verleihbüros und dem Programmsystem erfassen, um erste Vorgaben für die Programmierung zu erhalten.

Soll-Analyse

Betritt z. B. eine Kundin das Verleihbüro und äußert Ihren Beratungswunsch, so kann dieser Vorgang zwar von einem Kundenbetreuer, nicht jedoch vom System

wahrgenommen werden. Der Kundenbetreuer muss erst ein entsprechendes Ereignis an der Schnittstelle zum System vornehmen, das auch dem System bekannt ist - z. B. das Bearbeitungsmenü „Beratung und Information“ am Rechnerbildschirm aktivieren.

In den folgenden Anwendungsfallbeschreibungen werden wir uns daher auf Ereignisse und Phänomene beschränken, die - sowohl für das zu entwickelnde Programmsystem als auch für die Umgebung des Systems, also Kundenbetreuer, Servicemitarbeiter und Sachbearbeiter im Hintergrundbüro - sichtbar sind.

Die Kunden treten in diesen Anwendungsfallbeschreibungen demnach nicht mehr direkt in Erscheinung. Die Akteure der Anwendungsfälle sind die Angestellten im Verleihbüro auf der einen und das System auf der anderen Seite.

3.6 Datenlexikon

In den verschiedenen Diagrammarten der voranstehenden Abschnitte verwendeten wir zahlreiche Fachbegriffe aus dem Anwendungsfeld, die nur vage bestimmt sind. Wir haben z. B. noch nicht daüber geredet, wie eine Kunden- oder Fahrzeugkartei aufgebaut sein soll, welche Daten ein Mietvertrag enthalten soll oder wie eine Reservierung abgelegt wird.

Ein Datenlexikon ist eine nach festen Regeln aufgebaute präzise Bestimmung aller Datenelemente, die für das Anwendungsgebiet wichtig sind. Das Datenlexikon, trägt dazu bei, Missverständnisse sowie unterschiedliche Interpretationen von Auftraggeberinnen und Entwicklern zu zu vermeiden. Tabelle 3.6-1 gibt die in einem Datenlexikon üblichen Symbole und Schreibweisen an. Kommentare können dafür benutzt werden, umgangssprachlich Definitionsbereiche oder Maßeinheiten zu anzugeben.

Tab. 3.6-1: *Schreibweise im Datenlexikon*

Schreibweise	**Bedeutung**
=	setzt sich zusammen aus
+	Folge
|	Auswahl
{ }	beliebige Wiederholung
( )	optionale Angabe
**	Kommentar

Einige der zuvor benutzten Fachbegriffe aus dem Autoverleih werden im folgenden Datenlexikon präzise (wenn auch aus Vereinfachungsgründen nicht ganz realitätsgetreu) bestimmt:

```
Kunde                  = Name + Anschrift + Führerscheinnr +
                         Kreditkarte + Alter
   Name                = Buchstabe + {(Buchstabe | - | _)}
                         ** _ symbolisiert das Leerzeichen
   Anschrift           = Straße + Hausnummer + PLZ
   Führerscheinnr      = 100 | ... | 199
   Kreditkarte         = Kreditkaternnr + Gültigkeitsdatum
   Alter               = 18 | ...| 80
      Buchstabe        = a | b | ... | z |  ä | ö | ü| ß
      Straße           = Name
      Hausnummer       = 1 | ... | 99
      PLZ              = 0001 | .. | 9999
      Kreditkaternnr   = 500 | ... | 599
      Gültigkeitsdatum = 01 | ... | 12 + / + 04 | ... | 07

Buchungscode           = P | U | K | + K | M | O + Ziffer +
                         Ziffer + Ziffer
                         ** der erste Buchstabe kodiert die Art des
                         ** Fahrzeugs (PKW, PickUp, Kleinlaster),
                         ** der zweite die Wagenklasse
                         ** (Kompakt-, Mittel-, Oberklasse)
   Ziffer              = 0 | ... | 9

Bericht                = km-Stand + Füllstand + Datum + Zeit
   km-Stand            = 1 | ... | 40000
   Füllstand           = 0 | 1/3 | 1/2 | 2/3 | 1/1
   Datum               = 01 | ... | 31 + . + 01 | ... | 12 + . +
                         04 | ... | 05
   Zeit                = 0 | ... | 24 + : 00 | ... | 59
...
```

Auf diese Festlegungen werden wir im Zuge der Implementierung des DVT-Verleihsystems zurück kommen.

3.7 Pflichtenheft

In einem professionellen Entwicklungsprojekt wird üblicherweise zwischen Auftraggeberinnen und Softwareentwicklern ein sog. Pflichtenheft aufgestellt. Im Pflichtenheft sind alle wichtigen organisatorischen undtechnischen Vorgabe zur Erstellung der Software zusammengefasst. Das Pflichtenheft bildet die Grundlage für die technische Realisierung der Software. Das Pflichtenheft dient damit zwei Hauptzwecken:

- Es muss so abgefasst sein, dass es vom Auftraggeber und den Spezialisten in den Fachabteilungen, die die Software später einsetzen sollen, verstanden wird
- Die erfasste Information muss ebenso für die Softwareentwickler hilfreich sein.

Der angestrebte Sollzustand ist der Hauptbestandteil des Pflichtenheftes. Der Istzustand soll nur dann in das Pflichtenheft mit aufgenommen werden, wenn er zur Verdeutlichung des Sollzustandes beiträgt. Ein Beispiel für ein Pflichtenheft finden Sie auf der Seite für Zusatzkomponenten zum Kurs[54]. Das Beispiel stellt ein stark vereinfachtes Pflichtenheft für unser Fallbeispiel dar.

54 http://www.ice-bachelor.fernuni-hagen.de/lehre/k20022/zusatzkomponenten.html

4 Projektanforderungen

Nach der erfolgreichen Bestandsaufnahme, der wir uns im vorigen Abschnitt widmeten, wollen wir in diesem Abschnitt die Anforderungen an das zu implementierende Programmsystem erheben.

4.1 Soll-Analyse

Unsere Entwicklungsaufgabe im Laufe des Kurses wird nun darin bestehen, einen Großteil der manuellen Arbeit im Autoverleihbüro programmtechnisch zu lösen und damit in den Rechner zu verlagern. Einige dieser Programmteile werden Sie selbst programmieren, andere stellen wir für Sie zur Verfügung.

Statt Karteien und Karteikarten wollen wir die Daten zu Kunden und Fahrzeugen im Rechner verwalten.

Die Kundenberaterinnen sollen bei der Betreuung der Kunden

- beim Suchen geeigneter Fahrzeuge,
- bei der Fahrzeugreservierung,
- bei der Ausstellung und Abrechnung von Mietverträgen sowie
- bei der Bezahlung

durch Programmfunktionen unterstützt werden.

Die Mitarbeiter im Fahrzeugservice geben die bei der Fahrzeugannahme festgestellten Nutzungsdaten wie

- Kilometerstand,
- Füllstand der Tankanzeige und
- Fahrzeugschäden

in den Rechner ein, so dass sie bei der anschließenden Abrechnung im Büro zur Verfügung stehen.

Vereinfachungen

Um diese Aufgabe überschaubar zu halten, nehmen wir eine Reihe von Vereinfachungen vor:

1. Der Fahrzeugbestand der Autovermietung wird nicht verändert. Dadurch ersparen wir uns den Entwurf und die Implementierung entsprechender Systemfunktionen.
2. Zur Vereinfachung unterscheiden wir im Weiteren nicht mehr zwischen Privat- und Firmenkunden. Die Kundendaten werden nur für die Dauer der laufenden Mietverhältnisse eines Kunden gespeichert. Die Speicherung der Kundendaten beginnt mit der Reservierung eines Fahrzeugs und endet mit der Bezahlung der Mietkosten, sofern nicht weitere Reservierungen bestehen. Das bedeutet, dass ein Kunde mehr als eine Reservierung für Zeitpunkt in der Zukunft vornehmen kann.

3. Kunden können nur per Kreditkarte bezahlen. Abgerechnet wird immer über die Kreditkartenzentrale.
4. Den Anwendungsfall „Kreditwürdigkeit überprüfen“ wollen wir nicht weiter betrachten.
5. Wir unterscheiden die Fahrzeugklassen Kompakt-, Mittel- und Oberklasse in der Kategorie PKW, eine Klasse in der Kategorie Pick-Up und zwei Klassen „bis 3 Tonnen“ und „über 3 Tonnen“ in der Kategorie Kleinlaster (Lastwagen bis 7,5 Tonnen).

Dieser Aufgabenstellung entsprechend wollen wir - wie bei der Ist-Analyse - Anwendungsfälle entwickeln, die diese Anforderungen in der Form einer Soll-Analyse etwas präziser beschreiben (Abb. 4.1-1).

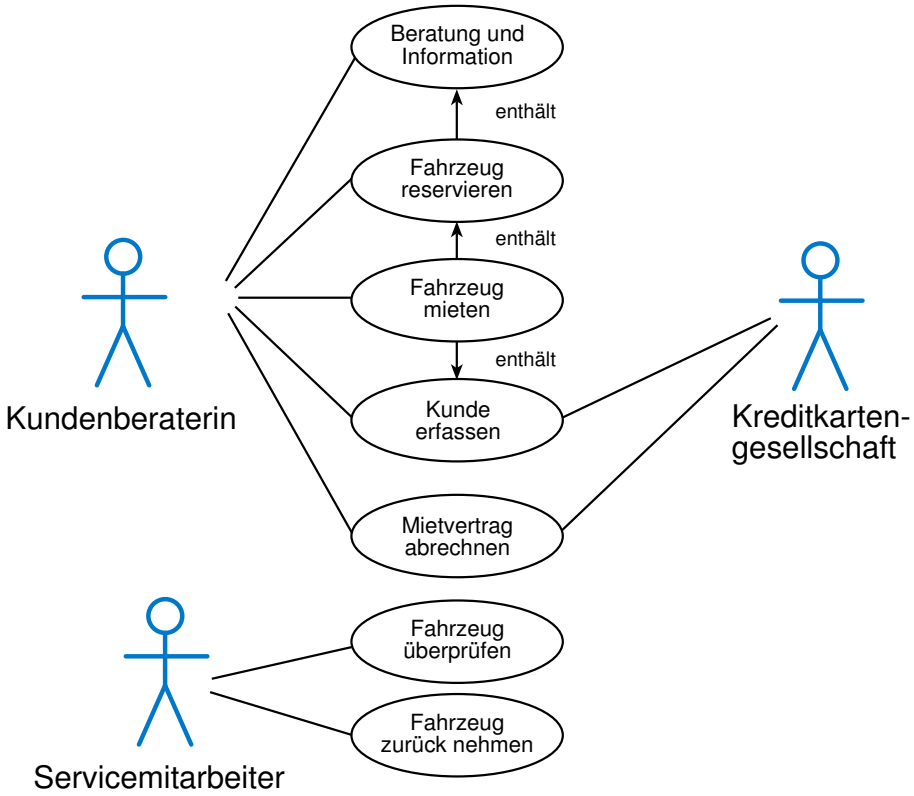


Abb. 4.1-1: Anwendungsfalldiagrammzur Sollanalyse

Wir unterscheiden die folgenden Anwendungsfälle:

- Beratung und Information,
- Fahrzeug reservieren,
- Kunde erfassen,
- Fahrzeug mieten,
- Fahrzeug übergeben,
- Fahrzeug zurücknehmen,
- Mietvertrag abrechnen,

Die zugehörigen Anwendungsfallbeschreibungen sind auf den folgenden Seiten mit Ausnahme von „Mietvertrag und Firmenvertrag abrechnen“ angegeben. Diese Anwendungsfallbeschreibung sowie die Erstellung des Anwendungsfalldiagramms überlassen wir Ihnen als Einsendeaufgabe.

4.2 Anwendungsfallbeschreibung: „Beratung und Information“

Tab. 4.2-1: *Anwendungsfallbeschreibung: „Beratung und Information“*

	Beratung und Information	
Anwendungsfall Kurzbeschreibung	Ein Interessent wird über das Fahrzeugangebot, die Verfügbarkeit und den Mietpreis für ein gewähltes Mietdatum informiert	
Beteiligte Akteure	Kb	Kundenberaterin
Auslöser, Vorbedingungen	Telefonische oder persönliche Anfrage eines Interessenten	
Ergebnis, Nachbedingungen	Dem Interessenten wird die Preisinformation für die gewählte Fahrzeugkategorie und Mietdauer übermittelt	
	Ablauf, Interaktionen	
1	***Erfassung der Interessentenwünsche***	
1.1 Kb	Die gewünschte Fahrzeugkategorie und der Abholort werden per Auswahlmenü festgelegt, das Alter des Kunden und die Mietdaten werden in der Form TT.MM.JJ für das Datum der Abholung und das Datum der Rückgabe eingegeben	
1.2 System	System prüft die Plausibilität der Daten; wenn Daten in Ordnung, listet das System den Mietpreis sowie die Aufschläge für Verlängerungstage und -stunden für eine passende Auswahl an Fahrzeugen auf	
1.3 System	***Ausnahme:*** Datum der Rückgabe ist später als Datum der Abholung	
1.4 Kb	Gibt neue Mietdaten ein	
1.5 System	wie 1.2	
1.6 System	***Ausnahme:*** Kein Fahrzeug am gewählten Abholort zum gewünschten Zeitpunkt verfügbar	
1.7 Kb	Gibt neue Daten ein oder bricht Beratung auf Wunsch des Kunden ab	

Fortsetzung auf der nächsten Seite

Fortsetzung von der vorhergehenden Seite		
		Abweichende Interaktionen
2		***Interessent hat untere Altersgrenze für Anmietung eines LKW nicht erreicht***
2.1	System	Das System meldet, dass Fahrer für Anmietung der gewünschten Fahrzeugkategorie zu jung ist (unter 21 generell nicht möglich, unter 25 Anmietung eines größeren LKW nicht zulässig)
2.2	Kb	Gibt andere Fahrzeugkategorie ein oder bricht den Vorgang auf Wunsch des Interessenten ab

4.3 Anwendungsfallbeschreibung: „Kunden erfassen“

Tab. 4.3-1: *Anwendungsfallbeschreibung: „Kunden erfassen“*

Anwendungsfall	Kunden erfassen	
Kurzbeschreibung	Die Kundendaten wie Anschrift, Alter, Führerschein- und Ausweisnummer werden aufgenommen	
Beteiligte Akteure	Kb	Kundenberaterin
Auslöser, Vorbedingungen	Besuch eines Kunden im Verleihbüro	
Ergebnis, Nachbedingungen	Der Kunde bestätigt seine Daten, die am Bildschirm angezeigt werden	
	Ablauf, Interaktionen	
1	***Erfassung der Kundendaten***	
1.1 Kb	Kundenberaterin gibt Namen, Anschrift, Führerscheinnummer und Kreditkartendaten eines Kunden ein	
	Ausnahme: Kunde kann sich nicht ausweisen oder keinen gültigen Führerschein oder keine Kreditkarte vorlegen ***Geschäftsvorfall wird abgebrochen!***	
1.2 System	Es wird überprüft, ob der Kunde dem System bereits bekannt ist (es existiert ein laufender Mietvertrag des selben Kunden); die eingegebenen Daten werden angezeigt	
1.3 Kb	Kundenberaterin bestätigt die Systemausgabe oder gibt neue Mietdaten ein	

4.4 Anwendungsfallbeschreibung:„Fahrzeug mieten“

Tab. 4.4-1: *Anwendungsfallbeschreibung: „Fahrzeug mieten“*

Anwendungsfall	Fahrzeug mieten	
Kurzbeschreibung	Ein Kunde reserviert einen Mietwagen	
Beteiligte Akteure	Kb	Kundenberaterin
Auslöser, Vorbedingungen	Buchungscode für reserviertes Fahrzeug liegt vor, oder Kunde ohne Reservierung möchte Vertrag abschließen	
Ergebnis, Nachbedingungen	Der Mietvertrag ist vom Kunden unterzeichnet, der Kunde erhält eine Kopie des Vertrags sowie den Fahrzeugschlüssel	
	Ablauf, Interaktionen	
1	***Mietvertrag ausfertigen***	
1.1 Kb	Kundenberaterin gibt den Buchungscode ein	
1.2 System	System zeigt die Nummer und den Standort des reservierten Fahrzeugs an und druckt den Mietvertrag aus	
1.3 Kb	Kundenberaterin bestätigt die Vorlage eines gültigen Führerscheins, die Aushändigung und Beurkundung des Mietvertrags sowie die Übergabe der Fahrzeugschlüssel	
1.4 Kb	***Ausnahme***: Der Kunde kann keinen gültigen Führerschein vorgelegen, ***Abbruch des Geschäftsvorfalls!***	
	Abweichende Interaktionen	
2	***Kunde hat nicht reserviert***	
2.1 Kb	[benutzt „Fahrzeug reservieren“]	

Bemerkung Die bei der Reservierung erfassten Kundendaten müssen aus dem System gelöscht werden.

4.5 Anwendungsfallbeschreibung:„Fahrzeug zurückgeben“

Tab. 4.5-1: *Anwendungsfallbeschreibung: „Fahrzeug zurückgeben“*

Anwendungsfall	Fahrzeug zurückgeben	
Kurzbeschreibung	Die Nutzungsdaten des Fahrzeugs werden aufgenommen	
Beteiligte Akteure	Sm	Servicemitarbeiter
Auslöser, Vorbedingungen	Kunde bringt Leihwagen zurück	
Ergebnis, Nachbedingungen	Fahrzeug übernommen	
	Ablauf, Interaktionen	
1	***Erfassung der Nutzungsdaten***	
1.1 Sm	Servicemitarbeiter gibt die Fahrzeugnummer, den Kilometerstand, den Füllstand des Tanks sowie Rückgabezeit und -datum ein	

Selbsttestaufgabe 4.5-1:

Entwickeln Sie eine Anwendungsfallbeschreibung für den Anwendungsfall „Mietvertrag abrechnen“ gemäß folgender Skizze:

Auslöser:
Die Kundenberaterin trägt eine Vertragsnummer ein, um den Vertrag für ein abgegebenes Fahrzeug auszufertigen.

Hauptablauf:

1. *Die Kundenberaterin gibt die Nummer des Mietvertrags ein.*
2. *System zeigt Vermietungsdaten und den Mietpreis an.*
3. *Die Kundenberaterin dokumentiert die Bestätigung des Kunden.*
4. *System gibt Beleg aus und bucht Betrag von Kreditkartenkonto des Kunden ab. (Bemerkung: Kundendaten werden gelöscht, falls kein weiteres Mietverhältnis besteht; Fahrzeug ist für Ausleihe wieder verfügbar).*

Mit der Aufstellung der Anwendungsfälle zum neuen Verleihsystem sind die ersten Schritte in Richtung Programmentwicklung getan.

Wir wollen damit zunächst die Dokumentation der Systemfunktionen beschließen und uns den Fachbegriffen und Gegenständen des Anwendungsfelds zuwenden.

5 Konzepte objektorientierter Programmiersprachen

ObjektorientierteEntwurfstechniken und Programmiersprachen werden von vielen Fachleuten aus Hochschule und Praxisgegenüber vielen anderen Programmiersprachen, die wir im Abschnitt „GeschichtlicheEntwicklung der Programmiersprachen“ kurz ansprachen, deshalb bevorzugt, weil siefachliche und DV-technische Begriffe vereinheitlichen. Die Begriffe derAnwendung dienen hierbei als Grundlage zur Modellierung einerprogrammtechnischen Lösung. Das Anwendungsmodell kann ohne gedanklichen Bruchin ein objektorientiertes Programm überführt werden.

5.1 Analyse der Anwendungswelt: Gegenstände der Bearbeitung

Fachbegriffeverstehen

Jede Beschreibung der Realität hängt von der Wahl der Fachbegriffe und einer vorsichtigen Erläuterung der Phänomene ab, die jeder Begriff bezeichnet. Wichtig ist, dass wir uns vor dem Entwurf eines Programmsystems ein genaues Bild der Wirklichkeit verschaffen und dieses auch mit viel Disziplin umfassend und möglichst eindeutig beschreiben.

Unsere Wahrnehmung der Wirklichkeit müssen wir nachvollziehbar mit den schematischen und formalen Beschreibungen, die ein Programm ausmachen, in Beziehung setzen. Nur so können wir sicher sein, dass die Auswirkungen des Programms die gewünschten Resultate liefern.

Beispiel 5.1-1: UngenügendeModellanalyse
Der Unfall der Lufthansa-Maschine beim Landeanflug in Warschau vor mehreren Jahren ist eindeutig auf eine unvollständige Modellierung der Phänomene der Wirklichkeit zurückzuführen. Ein Steuerprogramm sollte verhindern, dass der Umkehrschub, der ein Flugzeug nach der Landung stark abbremst, eingeschaltet werden kann, bevor die Maschine tatsächlich gelandet ist. Diese Steuerung sollte nämlich genau den Fall verhindern, der eine Maschine der Lauda-Air über Thailand zum Absturz gebracht hatte. Die Piloten hatten in der Luft versehentlich den Umkehrschub eingeschaltet und die Maschine damit in einen unkontrollierbaren Sturzflug gebracht.

In Warschau herrschten zum Zeitpunkt der Landung der Lufthansa-Maschine extrem widrige Landebedingungen. Die Landebahn war auf Grund starken Regens extrem glatt, und strenge Scherwinde zwangen die Piloten dazu, das Flugzeug mit starker Seitenneigung auf die Landebahn zu bringen. Infolgedessen setzte die Maschine mit einem Fahrwerk auf der Landebahn auf. Die Ingenieure hatten aber den Zustand des Gelandetseins so festgelegt, dass beide Fahrwerke mit einem vorgegebenen Anpressdruck auf der Fahrbahn aufliegen müssen. Da dies bei der Seitenneigung des Flugzeugs für geraume Zeit nicht der Fall war, konnte der Umkehrschub

nicht rechtzeitig aktiviert werden, und die Maschine rollte mit zu hoher Geschwindigkeit über die Landebahn hinaus.

□

In jeder Anwendungswelt werden Begriffe benutzt, die von vorne herein nur selten von Systemanalytikern voll verstanden werden.

Bisher untersuchten und dokumentierten wir die:

- Handlungsträgerinnen und ihre Rollen,
- ihr Zusammenwirken,
- ihre für die Geschäftsabläufe wesentlichen Handlungen und
- die Informationen, die sie austauschen.

Im nächsten Schritt wollen wir einzelne Spielszenen der Autovermietung noch einmal genauer betrachten, um:

- die Gegenstände, mit denen die Akteure umgehen sowie
- die Beziehungen, die zwischen diesen Gegenständen bestehen,

zu verstehen und zu beschreiben.

1. Spielszene 3.1-1,
2. Spielszene 3.1-3,
3. Spielszene 3.1-2.

5.2 Fachbegriffe der Anwendungswelt

Zur Bestimmung der wesentlichen Fachbegriffe der Autovermietung müssen wir uns fragen:

Gegenstände
- Welche dinglichen und abstrakten Gegenstände werden bearbeitet, verändert oder tauchen in Kommunikationssituationen auf?
- Welche Eigenschaften zeichnen diese Gegenstände aus?

Beziehungen
- Welche Beziehungen bestehen zwischen ihnen?
- Wie wird mit ihnen umgegangen?

Rollen
- Welche Rollen treten auf, und für welche Handlungen sind sie verantwortlich?

Tabelle 5.2-1 gibt eine ganze Reihe von Fachbegriffen wieder, die in unseren Spielszenen benutzt wurden.

Tab. 5.2-1: *Fachbegriffe*

Adresse	Kleinwagen	Preis
Ausweis	Kompaktklasse	PKW
Auto	Kontonummer	Rabatt
15.03.01	Kreditkarte	Rechnung
Fahrzeug	Kundenkartei	Reservierung
Servicemitarbeiter	Lastwagen	Reservierungsnummer
Firmenkonto	Mercedes	Herr Schneider
Frau Busch	Mietbeginn	Van
Führerschein	Mietdauer	Vertrag
HH-DV 123	Mietpreis	VW Golf
Kfz-Kennzeichen	Nissan Sentra	VWGolfMitKzHHWS531
Kilometerstand	Petermann	2745 5678 3498
Kleintransporter	Pick-Up	

Die Substantive bezeichnen

- **Rollen**,
- **Individuen**,
- **Gegenstände**,
- **Merkmale** und
- **Werte**.

Die Verben, die in der Tabelle nicht angegeben sind, bezeichnen **Aktivitäten**.

Beispiel 5.2-1: Begriffseinordnung
Der Begriff Servicemitarbeiter bezeichnet die Rolle eines Akteurs im Autoverleih.

Frau Busch und Herr Schneider benennen individuelle Personen, die in den Rollen Kundenberaterin bzw. Kunde auftreten.

Die Substantive Auto, PKW, Kleinwagen, Van, Golf, Lastwagen, Führerschein, Vertrag und Firmenkonto bezeichnen Gegenstände.

Ein Kennzeichen besitzen alle Personen- und Lastkraftfahrzeuge; wir wollen es deshalb als Attribut oder Merkmal solcher Fahrzeuge ansehen.

HH-WS 531 ist offensichtlich der konkrete Wert eines Fahrzeugkennzeichens.

□

Selbsttestaufgabe 5.2-1:

Sortieren Sie die in der Übung Fachbegriffe aufgelisteten Fachbegriffe in die vier angegebenen Kategorien Individuum, Gegenstand, Merkmal und Wert ein.

***Animation 5.2-1**: Übung: Fachbegriffe einordnen*
interaktive Übung: Fachbegriffe einordnen

5.3 Realität und Modell

Nachdem wir die Fachbegriffe der Anwendung erhoben und verstanden haben, müssen wir uns überlegen, wie wir diese Begriffe in ein Modell, also in ein Abbild der Realität übertragen. Die Begriffe der Anwendung bilden dabei die Grundlage der objektorientierten Modellierung. **Modellieren** heißt,

- →Abstrahieren, d. h. Wesentliches von Unwesentlichem zu trennen und nur das für die Anwendung Wesentliche zu erfassen,
- den Umgang mit Gegenständen zu beschreiben,
- Begriffe zu bilden und zu rekonstruieren,
- ein einheitliches Verständnis zu erarbeiten und
- Ähnlichkeiten und Beziehungen zu erkennen.

Beispiel 5.3-1: Roboter
Abb. 5.3-1 veranschaulicht den Prozess der Modellbildung am Beispiel eines Roboters.

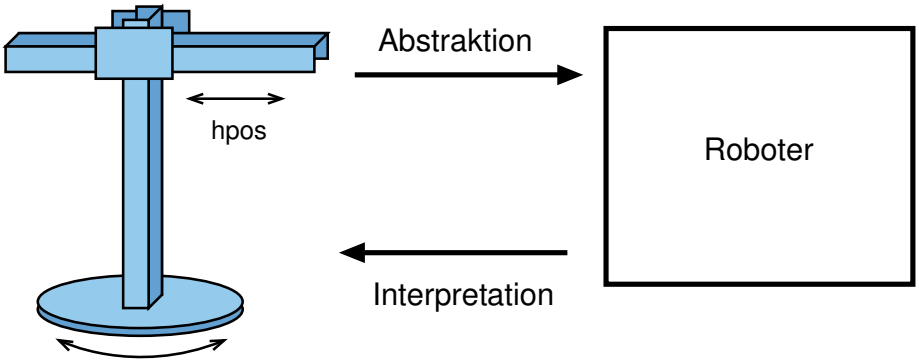


***Abb. 5.3-1**: Roboter -Wirklichkeitund Modell*

In Abb. 5.3-1 erkennen wir ein abstrahiertes Abbild des Roboters mit seiner Steuerung, die uns zunächst als undurchsichtiger oder schwarz (umrandet)er Kasten (engl. black box) erscheint.

In Abb. 5.3-2 sind wesentliche Eigenschaften des Roboters, wie sein ausfahrbarer Arm, sein drehbarer Fuß und sein Bewegungszustand (aktiv, inaktiv) in den Merkmalen Extension, Winkel und Aktivierung zusammengefasst. Diesen Merkmalen sind Datentypen zugeordnet, die etwas über ihre zulässigen Werte aussagen. (Das Merkmal Aktivierung kann offensichtlich nur zwei Werte annehmen, da der Datentyp `boolean`*, den Sie bereits aus früheren Kursen kennen, nur die Werte wahr und falsch umfasst. Dazu später mehr.)*

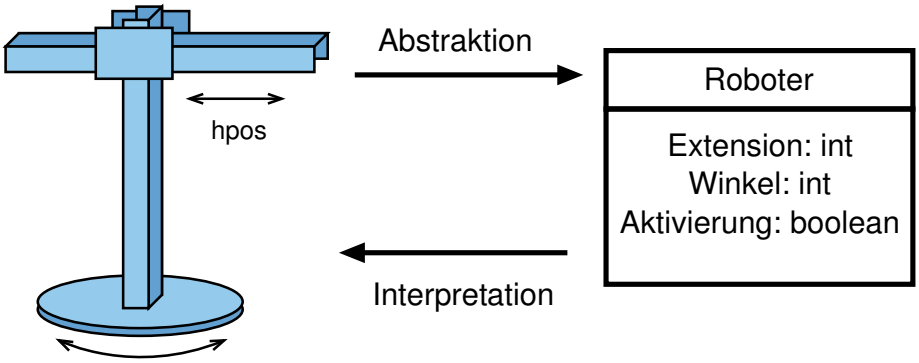


***Abb. 5.3-2**: Robotereigenschaften im Modell*

In Abb. 5.3-3 sind die Fähigkeiten des Roboters durch Operationen, die etwas über den möglichen Umgang mit dem Gegenstand „Roboter“ mitteilen, dargestellt.

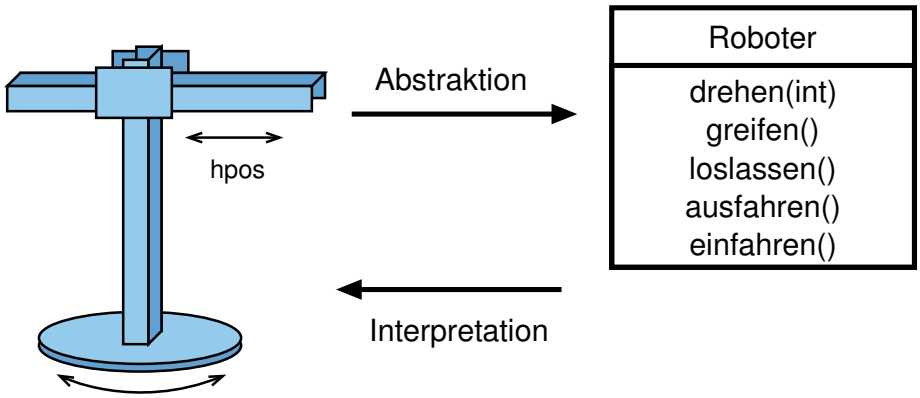


***Abb. 5.3-3**: Roboterverhalten im Modell*

Abb. 5.3-4 zeigt alle Aspekte zusammen.

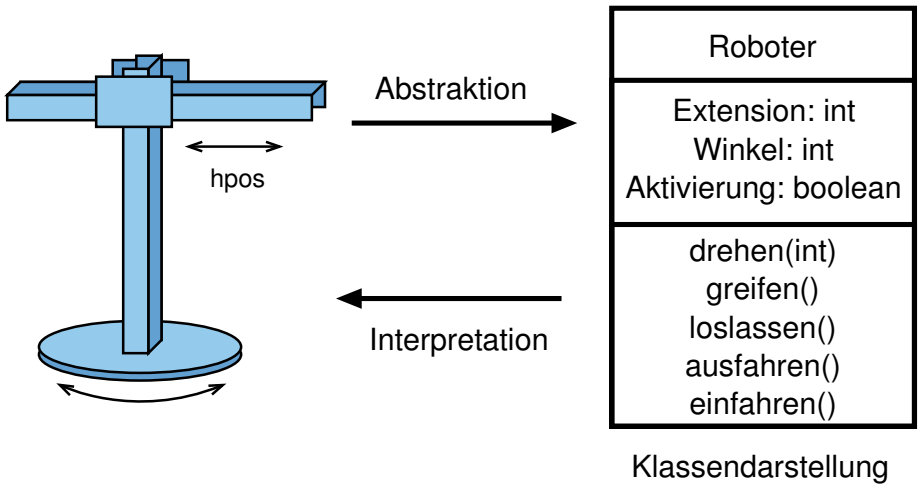


***Abb. 5.3-4**: Eigenschaften und Verhalten im Robotermodell*

□

Definition 5.3-1: Objekt
*Die objektorientierte Programmierung bezeichnet die Abbilder konkreter Gegenstände und Träger individueller Rollen in Entwurfsdokumenten und Programmen als **Objekte**.*

□

Beispiel 5.3-2: Objekt
Ein VW Golf mit seiner eindeutigen Fahrgestellnummer oder einem bestimmten Kennzeichen ist ebenso ein Objekt wie ein Mietvertrag, der konkrete Personen- oder Fahrzeugdaten umfasst, ein Ausweisdokument, eine bestimmte Person in der Rolle einer Sachbearbeiterin oder ein ganz bestimmter, mit einer individuellen Kennung versehener Roboter.

□

Bei der Modellbildung vereinfachen wir die Komplexität realer Objekte durch die Konzentration auf wesentliche Eigenschaften und Fähigkeiten. Mit Fähigkeiten meinen wir mögliche Formen des Umgangs mit den Objekten.

Im Zusammenhang mit dem Autoverleihgeschäft interessieren uns nur die Merkmale eines Mietvertrags, die für die Berechnung und Begleichung der Kosten und die Verfolgung missbräuchlicher Nutzung entscheidend sind. Dazu gehören z. B. die Identität und Kreditkartennummer des Kunden, das Kennzeichen des Fahrzeugs, der Kilometerstand, Datum und Uhrzeit sowie Füllstand des Tanks bei Zu- und Abgang. Auf welcher Art von Trägermaterial der Vertrag abgedruckt ist, welche Farbe und Größe der Vertrag aufweist, dass man den Vertrag falten oder zerknüllen kann, sind Eigenschaften und Formen des Umgangs, die nicht von Belang sind.

Nun wäre es sehr mühsam, wollte man alle Objekte eines Anwendungsbereichs in ihren Eigenschaften und Fähigkeiten einzeln angeben und programmieren. Es sind einfach zu viele, und zahlreiche Objekte weisen auch Ähnlichkeiten auf, die wir bei der Abstraktion nutzen können.

Reflektieren wir noch einmal die verschiedenen Begriffe, die in den Spielszenen verwendet werden, so stellen wir drei Dinge fest:

1. Einige Gegenstände, wie etwa der Nissan Sentra und der Mercedes-Benz A-Klasse-Wagen, können in Bezug auf bestimmte Merkmale wie Länge, Innenraumgröße, Verbrauch und Preis als gleichartig aufgefasst werden, während das Fahrzeug, das Herr Schneider ausleiht und der zugehörige Mietvertrag so gut wie nichts gemein haben.
2. In vielen Gesprächen wird gar nicht über konkrete Gegenstände, sondern über Kategorien oder Klassen von Gegenständen geredet. Der Begriff „Kompaktfahrzeug“ bezeichnet nämlich eine Preiskategorie im Verleihgeschäft, in die der Nissan und der A-Klasse-Wagen fallen. Mittelklassewagen und Van bezeichnen dagegen andere Preiskategorien.
3. Die Kategorien „Kompaktfahrzeug“, „Mittelklassewagen“ und „Van“ weisen selbst wieder wenigstens ein gemeinsames Merkmal auf, das sie von der Kategorie „Lastkraftwagen“ unterscheidet: Sie dienen der Personenbeförderung, während Lastkraftwagen zur Lastenbeförderung eingesetzt werden.

Definition 5.3-2: Klasse
*Der Begriff **Klasse** wird in der objektorientierten Programmierung genau dazu benutzt, einen Bauplan für viele ähnliche, aber individuell unterscheidbare Objekte anzulegen. Eine Klasse umfasst eine Gruppe oder Familie gleichartiger Objekte, und bezeichnet einen Begriff aus dem Anwendungsbereich.*

Klassen werden durch Substantive bezeichnet.

□

Beispiel 5.3-3: Klasse
Beispiele für Klassen in der Anwendung „Autoverleih“ sind:

Kartei, Wagen, PKW, Lastwagen, Van, Pick-Up, Mietvertrag, Rechnung, Kunde.

□

Definition 5.3-3: Exemplar
*Objekte sind Ausprägungen oder **Exemplare** einer bestimmten Klasse.*

□

Beispiel 5.3-4: Exemplar
Herr Schneider ist ein Exemplar der Klasse Kunde, der rote Golf mit dem Kennzeichen HH-WS 531 ist ein Exemplar der Klasse Kompaktfahrzeug.

□

Definition 5.3-4: Attribut
*Die Klassenbeschreibung bestimmt die Eigenschaften und Fähigkeiten aller Exemplare der Klasse. Die Eigenschaften werden durch Merkmale, auch **Attribute** genannt, dargestellt.*

Attribute werden durch Substantive bezeichnet.

□

Beispiel 5.3-5: Attribut
Kennzeichen, Farbe, Anzahl Türen sind Attribute der Klasse PKW.

□

Definition 5.3-5: Methode
*Das gemeinsame Verhalten aller Objekte einer Klasse wird durch die Methoden der Klasse bestimmt. Jede **Methode** beschreibt eine Fähigkeit oder mögliche Form des Umgangs mit einem Objekt der Klasse, in der Eigenschaften und Fähigkeiten festgelegt sind.*

Methoden werden durch Verben bezeichnet und durch ein nachstehendes Klammerpaar „(“ und „)“ abgeschlossen.

□

Beispiel 5.3-6: Methode
Mögliche Methoden der Klasse Kartei sind:

sortieren( ), einfügen( ), suchen( ).

□

Selbsttestaufgabe 5.3-1:

Ordnen Sie die in Übung Begriffe ordnen aufgesammelten Fachbegriffe in die Kategorien Klasse, Objekt und Attribut ein.

***Animation 5.3-1**: Übung: Begriffe in objektorientierte Kategorien einordnen*
interaktive Übung: Begriffe nach Kategorien ordnen

Wir haben gelernt, dass wir Gegenstände, die wir als gleichartig sehen, zu Klassen und somit zu einem Begriff zusammenfassen können. Die Klassenbildung ist meist nicht eindeutig. Sie hängt vom Verständnis der beteiligten Personen ab.

Nachdem wir die Klassen Fahrzeug, PKW, Kleinwagen, Van, Pick-Up und andere bildeten, liegt die Frage nahe, ob es Zusammenhänge zwischen Klassen gibt, die derart ähnliche oder gemeinsame Attribute haben, und wie diese Zusammenhänge gefunden und modelliert werden.

Selbsttestaufgabe 5.3-2:

*Aus Sicht einer Autovermietung - nennen wir sie einfach **Direkt-Vermietung von Transport- und Personenfahrzeugen** oder kurz **DVT** - ist es wichtig, über jedes Fahrzeug einen Datenbogen zu führen, um sowohl eine reibungslose Vermietung als auch eine kontinuierliche Instandhaltung zu ermöglichen.*

***Aufgabe:** Bestimmen Sie für die Klasse Auto Attribute, die für die Datenerfassung eines Autos aus Vermietersicht wichtig sind.*

5.4 Klassengeflecht

Definition 5.4-1: Klassenhierarchie, Spezialisierung, Attribut und Verhalten

Klassenhierarchie

*Ähnlich wie in der Biologie können die Klassen objektorientierter Modelle und Programme auf der Basis einer Verwandtschaftsrelation zu hierarchischen Strukturen verknüpft werden. Sie werden **Klassenhierarchien** genannt.*

Attribut
Verhalten
Spezialisierung

*In einer Klassenhierarchie weisen nachgegliederte Klassen alle Eigenschaften (**Attribute**) und Fähigkeiten (**Verhalten**) der Klassen auf, von denen sie abgeleitet wurden. Sie sind **Spezialisierungen** der übergeordneten Klassen. Sie können die geerbten Eigenschaften und Fähigkeiten neu bestimmen und durch eigene Attribute und Fähigkeiten erweitern.*

□

Beispiel 5.4-1: Klassenhierarchie

Abb. 5.4-1 zeigt eine Klassenhierarchie für verschiedene Arten motorgetriebener Fahrzeuge:

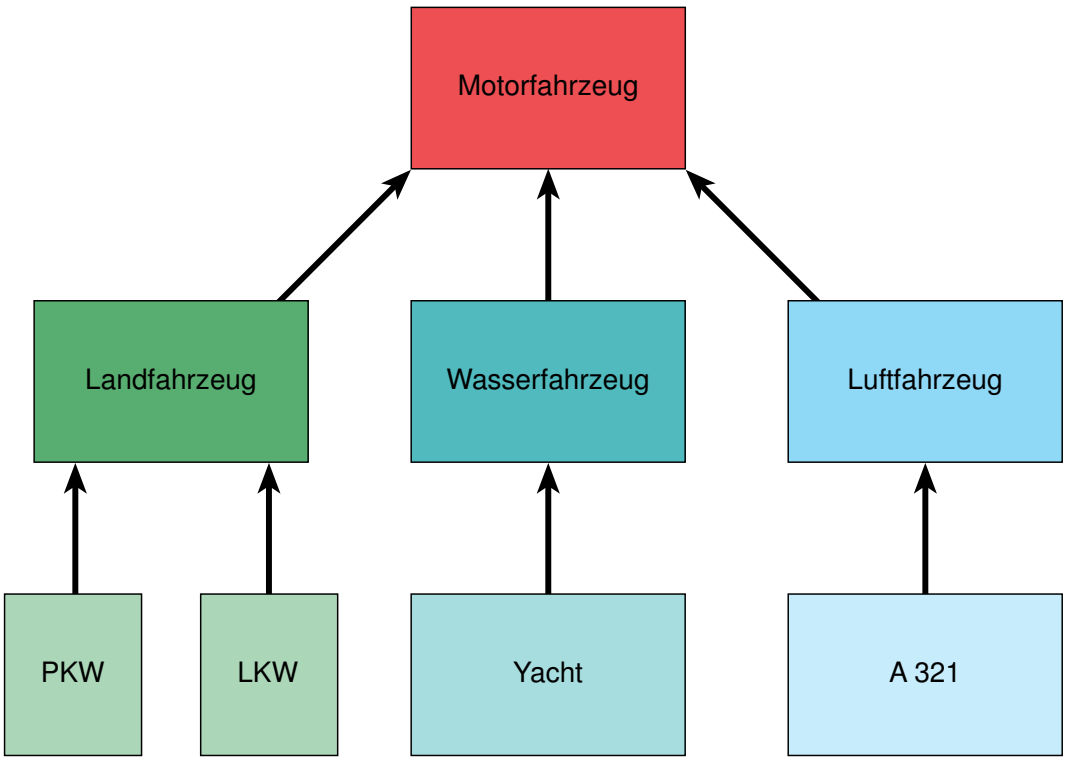


***Abb. 5.4-1**: Klassenhierarchie*

□

Definition 5.4-2: Unterklasse, Oberklasse, Einfach- und Mehrfachvererbung

Unterklasse
Oberklasse

*Eine Klasse A, die ihre Eigenschaften und Fähigkeiten von einer anderen Klasse B erbt, wird **Unterklasse** von B genannt. B ist die **Oberklasse** von A.*

Eine Klasse besitzt in der Regel mehr als eine Unterklasse.

In manchen objektorientierten Programmiersprachen kann jede Klasse höchstens eine Oberklasse haben. Man spricht in diesen Fällen von der ***Einfachvererbung****.*

Einfachvererbung

Ist ***Mehrfachvererbung*** *in einem Modell oder einer Sprache zugelassen, kann eine Klasse auch mehr als eine Oberklasse haben.*

Mehrfachvererbung

□

Die Einfachvererbung erzeugt eine Baumstruktur für die Klassen einer Anwendung, während die Mehrfachvererbung die Klassen in einem azyklischen, gerichteten **Klassengraphen** (engl. *acyclic directed graph*, kurz ***DAG***) aufspannt.

Beispiel 5.4-2: Klassengraph
Wenn wir die Sammlung von Klassen aus Abb. 5.4-1 um die Klassen

- *Amphibienfahrzeug,*
- *Wasserflugzeug und*
- *Pick-Up*

ergänzen, erhalten wir einen DAG (Abb. 5.4-2), weil ein Amphibienfahrzeug sowohl Merkmale eines Land- als auch eines Wasserfahrzeugs in sich vereint, ein Wasserflugzeug sowohl fliegen als auch schwimmen kann und ein Pick-Up oder Kleinlaster sowohl Personen als auch Lasten befördern kann.

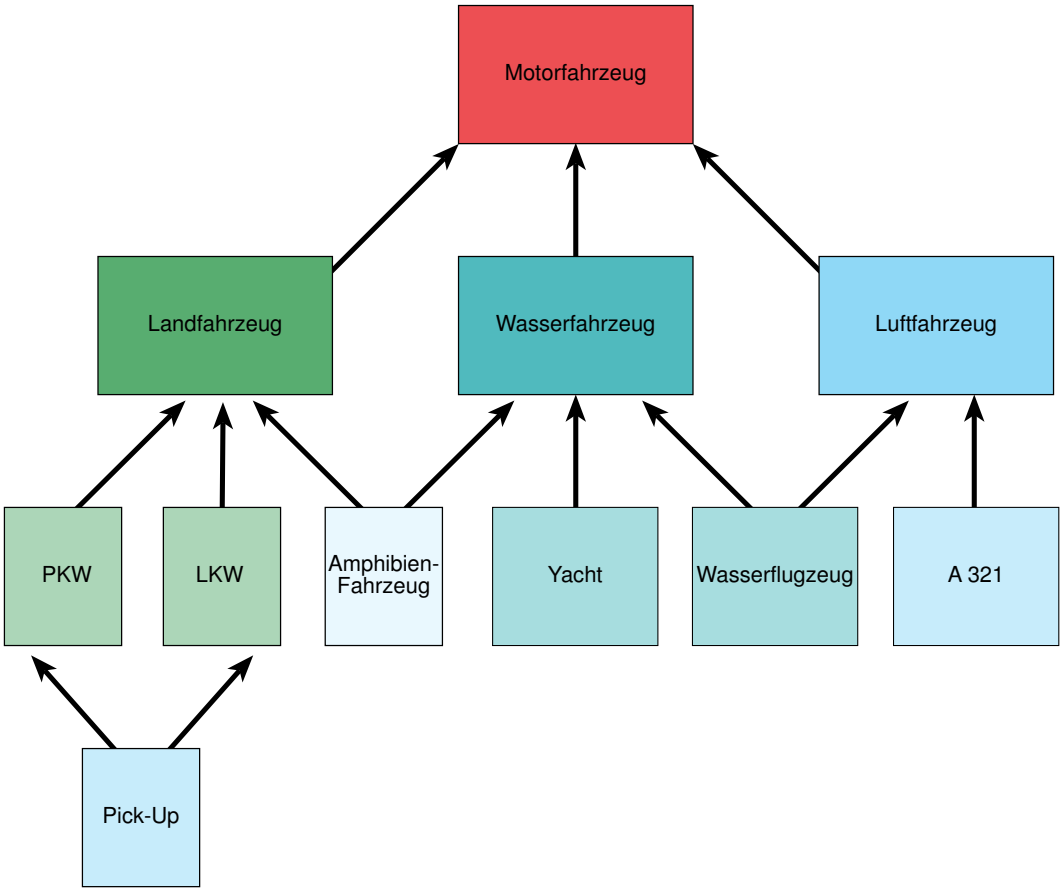


Abb. 5.4-2*: Klassengraph*

□

Selbsttestaufgabe 5.4-1:

Bestimmen Sie mindestens drei Unterklassen der Klasse „geometrische Figur“ und geben Sie für jede Klasse mindestens ein unterscheidendes Attribut an.

5.5 Botschaft und Auftrag

Spielszene 3.1-3, die wir hier noch einmal in Erinnerung rufen wollen, umfasst zwei Handlungsträger: eine Kundenbetreuerin und einen Kunden.

Kommunikationmittels-Botschaften

Die beiden Personen tauschen offensichtlich Informationen über das Mietwagenangebot der Firma aus. Der Kunde beauftragt schließlich die Kundenbetreuerin, ihm einen Mietwagen zu reservieren. Umgekehrt bittet die Kundenbetreuerin den Kunden, ihr seine für die Vermietung nötigen persönlichen Daten mitzuteilen.

→Botschaft

Übertragen in die Welt der objektorientierten Programmierung würden wir sagen: der Kunde übermittelt der Kundenbetreuerin die **Botschaften**

```
„informiereÜberMietwagen“ und
„reserviereFahrzeug“.
```

Diese Botschaften sind mit Informationen zur gewünschten Wagenkategorie und zum Zeitpunkt und zur Dauer der Miete versehen.

Auftrag

Die Kundenbetreuerin führt diese **Aufträge** auf eine bestimmte Art und Weise, die der Kunde nicht kennen muss, aus und liefert gegebenenfalls ein Ergebnis an ihn zurück.

Den Vorgang der Erhebung persönlicher Daten für die Reservierung kann man umgekehrt als einen Auftrag der Kundenbetreuerin an den Kunden ansehen, der durch die Botschaft

```
„bittePersönlicheDatenMitteilen“
```

angeregt wird.

5.6 Verhalten: Umgang mit Objekten

Den Benutzern eines Programmsystems kann es gleichgültig sein, wie erteilte Aufträge im Einzelnen ausgeführt werden.

Für den Programmentwurf und die Implementierung der Objektklassen ist es jedoch notwendig, die Art und Weise, wie mit Gegenständen der Anwendung umgegangen wird, zu verstehen und zu dokumentieren.

Den Gegenständen der Anwendungswelt sind i. d. R. typische Arbeitsweisen zugeordnet.

Beispiel 5.6-1: Fahrzeugkartei
Abb. 5.6-1 zeigt als Beispiel eine Kartei, in der Karteikarten zu Mietfahrzeugen verwaltet werden.

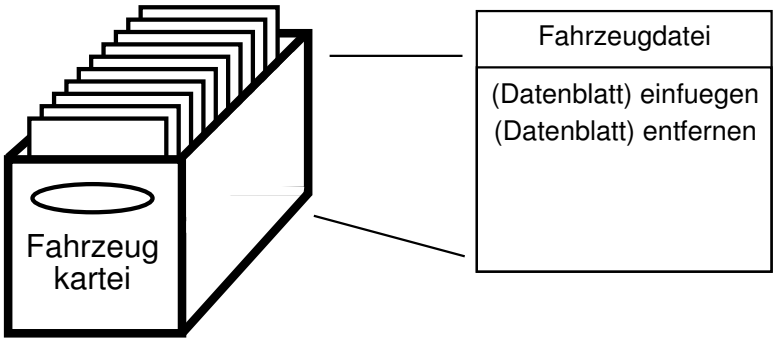


***Abb. 5.6-1**: Fahrzeugdatei*

Typische Formen des Umgangs mit Karteien sind:

- *(Karteikarte) in (Kartei) einfügen*
- *(Karteikarte) aus (Kartei) entnehmen*
- *(Kartei) sortieren*

□

Einige der Bearbeitungsschritte, die die Kundenbetreuerin ausführen muss, sind in den Szenen nicht im Einzelnen dargestellt; sie sind für unsere weiteren Überlegungen aber bedeutsam.

Beispiel 5.6-2: Umgang mit Objekten
So muss die Kundenbetreuerin zum Beispiel eine neue Personenkarte anlegen, die persönlichen Daten des Kunden in die Karte übertragen, anhand der Fahrzeugkartei die Verfügbarkeit des gewünschten Wagens überprüfen, die Karteikarte für den ausgesuchten Wagen aus der Fahrzeugkartei entnehmen und zusammen mit der Personenkarte in der Kartei für entliehene Wagen ablegen.

□

Wir fassen zusammen:

Akteur
Gegenstand

Die Systeme unserer Betrachtung bestehen aus **Akteuren** und **Gegenständen**. Die Akteure haben verschiedene Aufgaben und Verantwortlichkeiten.

Handlung

Sie werden durch die Dienstleistung, die Akteure anderen gegenüber erbringen, oder durch die **Handlungen**, die den Umgang mit Gegenständen wie Verträgen oder Karteikarten ausführen, bestimmt.

Botschaft

Das Erbringen einer Dienstleistung wird offensichtlich durch die Übermittlung einer **Botschaft**, etwa die Bitte, den Vertrag zu unterschreiben, ausgelöst.

Bevor wir jedoch darangehen, ein Programm zu entwerfen, das die Vorgänge der Anwendungswelt durch Rechnerfunktionen unterstützt, müssen wir uns fragen, was die verwendeten Fachbegriffe in ihrem Kontext genau bedeuten und wie sie zusammenhängen.

Ein objektorientiertes Begriffsmodell des Anwendungsbereichs hilft uns, diese Fragen zu beantworten und legt zugleich die Grundlagen für ein Entwurfsmodell, aus dem wir die spätere Implementierung entwickeln werden.

Selbsttestaufgabe 5.6-1:

Geben Sie Attribute und wesentliche Methoden an, mit denen Sie die Eigenschaften und Fähigkeiten einer Klasse Mietvertrag beschreiben.

6 Zusammenfassung

Mit den vorangegangenen Überlegungen haben wir versucht, Ihnen das Denken in Objekten näher zu bringen.

Nachdem wir das „Denken in Objekten“ verinnerlicht haben, werden wir daraus in den folgenden Lerneinheiten eine Programmiertechnik entwickeln, die wir konkret am Beispiel der objektorientierten Programmiersprache Java anwenden werden.

Was „Denken in Objekten“ intuitiv bedeutet, lässt sich am besten anhand einer Analogie verdeutlichen.

Beispiel 6-1: Computer und Bestandteile
Nehmen wir an, Sie beträten - ausgestattet mit den im Kurs Computer-Hardware erworbenen Kenntnissen - einen Computerladen und stellten sich aus verschiedenen Einheiten wie Mutterplatine, CPU, Videokarte, Haupt- und Hintergrundspeicher, Tastatur, Bildschirm usw. Ihren eigenen PC zusammen.

Sie erhalten damit ein komplexes System, dessen Einzelteile zusammenarbeiten, um bestimmte Anwendungsaufgaben wie Textverarbeitung, Grafikgestaltung oder Finanzverwaltung bewältigen zu können.

Jede einzelne Einheit kann als Objekt aufgefasst werden, das bestimmte Funktionen über eine wohl definierte Schnittstelle zur Verfügung stellt.

Diese Einheiten können von verschiedenen Herstellern produziert worden sein, und Sie müssen die innere Arbeitsweise jeder Komponente nicht verstehen, um ihre Leistung nutzen zu können. Die Passgenauigkeit kooperierender Einheiten hängt allein von der Einhaltung vereinbarter Schnittstellen ab.

Sobald Sie das Zusammenspiel der einzelnen Einheiten verstanden haben, ist die Konstruktion eines größeren Systems ein Leichtes.

□

Wir fassen also noch einmal zusammen:

In der Welt der objektorientierten Programmierung werden konkrete und abstrakte Gegenstände durch **Objekte** modelliert. — Objekt

Der Umgang mit solchen Gegenständen wird durch **Methoden** realisiert. — Methode

Alle Objekte sind **Exemplare** einer **Klasse**. Die Klasse definiert die Eigenschaften und Fähigkeiten jedes Objekts dieser Klasse. — Exemplar, Klasse

Die Eigenschaften werden durch eine Menge von **Attributen** in der Klassendefinition bestimmt. Die jeweiligen Werte der Attribute eines Objekts bestimmen seinen inneren **Zustand**. — Attribut, Zustand

Die Fähigkeiten oder das **Verhalten** aller Objekte der Klasse werden durch die in der Klassendefinition angegebenen Methoden bestimmt. — Verhalten

Erzeugen von Objekten

Das Erzeugen von Gegenständen, wie das Anlegen einer neuen Karteikarte, wird in einem objektorientierten Programm durch das **Erzeugen eines Objekts** der entsprechenden Klasse, z. B. der Klasse Karteikarte, realisiert.

Die Programmierung von Klassen mit ihren Attributen und Methoden werden wir im Einzelnen in den folgenden Lerneinheiten erörtern. Dazu ist es notwendig, eine ganze Reihe programmiersprachlicher Begriffe zu erlernen und einzuüben.

In der nächsten Lerneinheit werden wir uns zunächst einigen Details der Programmieraufgabe, wie der

- Berechnung von Kosten,
- der Speicherung von Daten,
- der Verknüpfung von Bedingungen und
- der Verarbeitung von Zeichen

widmen und dabei einige elementare Programmierkonzepte kennen lernen, bevor wir auf die Begriffe Klasse und Objekt in ihrer programmiersprachlichen Bedeutung weiter eingehen.

Wir haben in dieser ersten Lerneinheit gesehen, wie die objektorientierte Programmierung fachliche und DV-technische Begriffe vereinheitlicht. Die Begriffe der Anwendung dienen als Grundlage zur Modellierung einer programmtechnischen Lösung.

Das Anwendungsmodell kann ohne gedanklichen Bruch in ein objektorientiertes Programm überführt werden. Die Änderung, die Erweiterung und die Wiederverwendung aller mit der programmtechnischen Lösung zusammenhängenden Dokumente werden damit unterstützt.

Literatur

[Parnas72] David L. Parnas:
A technique for software module specification with examples.
Communications of the ACM, 15(5):330-336, May 1972

Der Artikel führt das Geheimnisprinzip auf der Grundlage eines Modulbegriffs ein.

[Züllighoven98] HeinzZüllighoven:
Dasobjektorientierte Konstruktionshandbuch.
dpunkt.verlag 1998

Dieses Buch ist keine Einführung in die Programmierung und bezieht sichauch nicht auf Java. Es ist ein Handbuch zur Entwicklung großerobjektorientierter Softwaresysteme, das die Begriffe Werkzeug und Material inden Vordergrund der Betrachtung stellt.

Lösungen zu Selbsttestaufgaben der Kurseinheit

Lösung zu Selbsttestaufgabe 2.3-1:

Das Grundgedanke der Objektorientierung besteht darin, dass Software-Objekte als Abstraktionen der Wirklichkeit erscheinen. Objekte sind eindeutig identifizierbar. Sie haben Eigenschaften, die durch Attribute modelliert werden, und sie weisen ein bestimmtes Verhalten auf, das durch Methoden beschrieben wird. Eigenschaften und Methoden werden in einem logischen Verbund gekapselt. Neben der Kapselung und der Abstraktion sind weitere grundlegende Prinzipien der Objektorientierung die Modularisierung und die Hierarchie. Ein Programmsystem kann aus Objekten zusammengebaut werden, die nebeneinander existieren und untereinander Botschaften austauschen, um fremde Methoden zu benutzen. Objekte werden als Exemplare von Klassen erzeugt. Klassen beschreiben die ihren Exemplaren gemeinsamen Attribute und Methoden. Die Klassen eines objektorientierten Entwurfs sind hierarchisch organisiert, so dass die Objekte einer Klasse die Rolle von Objekten einer Klasse, die an höherer Stelle der Klassenhierarchie rangiert, einnehmen können.

Lösung zu Selbsttestaufgabe 2.3-2:

Das Geheimnissprinzip ermöglicht die arbeitsteilige Entwicklung großer Programmsysteme, weil das Zusammenspiel der Programmmodule unabhängig von deren Implementierung ist. Die Entwickler einzelner Module müssen nur die öffentlichen Schnittstellen der benutzten Module kennen, um deren öffentliche Operationen verwenden zu können. Die Auswirkungen von Programmänderungen werden beherrschbar. Die Auswirkungen von Fehlern werden verringert, weil der Entwickler eines Moduls keine Möglichkeit hat, über die dokumentierten Schnittstellen anderer Module hinweg deren Implementierungsdetails in die Implementierung des eigenen Moduls einzubeziehen.

Lösung zu Selbsttestaufgabe 2.3-3:

Das Geheimnisprinzip besagt, dass nur die Funktionalität eines Programmmoduls oder einer Klasse, d. h. die von diesem Modul oder der →Klasse angebotenen →Operationen bekannt gemacht wird. Die interne Realisierung der Operationen und auch der Aufbau der Datenstrukturen, auf denen sie die Operationen arbeiten, bleiben nach außen verborgen. Das Geheimnisprinzips stellt sicher, dass die Implementierungsdetails eines Moduls oder einer Klasse beliebig oft geändert werden, ohne dass andere Programmteile betroffen sind, solange die Schnittstellen des geänderten Moduls nicht betroffen sind.

Lösung zu Selbsttestaufgabe 3.4-1:

Tab. 3.4-3: *Anwendungsfallbeschreibung: „Fahrzeug reservieren“*

Anwendungsfall	Fahrzeug reservieren	
Kurzbeschreibung	Ein Kunde reserviert einen Mietwagen	
Beteiligte Akteure	Kd Kb	Kunde Kundenberaterin
Auslöser, Vorbedingungen	Telefonische oder persönliche Anfrage eines Kunden	
Ergebnis, Nachbedingungen	Der Kunde erhält den Buchungscode für das reservierte Fahrzeug	
	Ablauf, Interaktionen	
1	***Erfassung der Kundenwünsche***	
1.1 Kd	Gibt die gewünschte Fahrzeugkategorie und -klasse, Abholort und -termin sowie die Mietdauer an	
1.2 Kb	Nennt, sofern die Eingabedaten plausibel sind, alle freien Kapazitäten für das gewünschte Angebot	
1.3 Kb	***Ausnahme:*** Kein Fahrzeug am gewählten Abholort zum gewünschten Zeitpunkt verfügbar	
1.4 Kd	Gibt andere Kriterien an	
1.5 Kb	Kundenberaterin benennt passendes Angebot	
1.6	["Kundendaten erfassen" wird benutzt]	
1.7 Kd	Legt Kreditkarte vor	
1.8 Kb	...	

Das Anwendungsfalldiagramm in Abb. 3.3-1

Lösung zu Selbsttestaufgabe 4.5-1:

Tab. 4.5-2: *Anwendungsfallbeschreibung: „Mietvertrag abrechnen“*

Anwendungsfall		Mietvertrag abrechnen	
Kurzbeschreibung		Die Kosten der Fahrzeugnutzung werden über die Kreditkarte des Kunden abgerechnet	
Beteiligte Akteure		Kb	Kundenberaterin
Auslöser, Vorbedingungen		Kunde hat Leihwagen an Servicemitarbeiter übergeben und legt seinen Vertrag zwecks Abrechnung vor	
Ergebnis, Nachbedingungen		Vertrag ist abgerechnet	
		Ablauf, Interaktionen	
1		***Berechnung der Nutzungskosten***	
1.1	Kb	Kundenberaterin gibt die Vertragsnummer ein	
1.2	System	System druckt Vertrag ergänzt um Nutzungskosten aus	
1.3	Kb	Kundenberaterin lässt Vertrag unterzeichnen und bestätigt Abschluss der Transaktion	
1.4	System	System bucht Kosten von der Kreditkarte des Kunden ab	

Auf Ausnahmesituationen wie fehlerhafte Eingabe wollen wir hier nicht weiter eingehen.

Lösung zu Selbsttestaufgabe 5.2-1:

Die einzuordnenden Begriffe sind in der nachstehenden Tabelle mit entsprechenden Kürzeln versehen (M = Mertkmal, I = Individuum, G = Gegenstand, W = Wert).

Tab. 5.2-2: *Einordnung der Fachbegriffe*

	Adresse	*G*	*Kleinwagen*		*Preis*
M				*M*	
	Ausweis	*G*	*Kompaktklasse*		*PKW*
G				*G*	
	Auto	*M*	*Kontonummer*		*Rabatt*
G				*M*	
	15.03.01	*G*	*Kreditkarte*		*Rechnung*
W				*G*	
	Fahrzeug	*G*	*Kundenkartei*		*Reservierung*
G				*G*	
	Servicemitarbeiter	*G*	*Lastwagen*		*Reservierungsnummer*
G				*M*	
	Firmenkonto	*G*	*Mercedes*		*Herr Schneider*
G				*I*	
	Frau Busch	*M*	*Mietbeginn*		*Van*
I				*G*	
	Führerschein	*M*	*Mietdauer*		*Vertrag*
G				*G*	
	HH-DV 123	*M*	*Mietpreis*		*VW Golf*
W				*G*	
	Kennzeichen	*G*	*Nissan*		*VWGolfMitKzHHWS53*
M			*Sentra*	*I*	
	Kilometerstand	*I*			*2745 5678 3498*
M	*Kleintransporter*		*Petermann*	*W*	
		G			
G			*Pick-Up*		

Lösung zu Selbsttestaufgabe 5.3-1:

Die einzuordnenden Begriffe sind in der nachstehenden Tabelle mit entsprechenden Kürzeln K (für Klasse), O (für Objekt) und A (für Attribut) versehen. (Begriffe, die wir nicht einsortieren können, haben wir mit dem Kürzel na (nicht anwendbar) markiert.)

Tab. 5.3-1: *Ordnen der Fachbegriffe nach objektorientierten Kernbegriffen*

	Adresse		*Kleinwagen*		*Preis*
A	*Ausweis*	*K*	*Kompaktklasse*	*A*	*PKW*
K	*Auto*	*K*	*Kontonummer*	*K*	*Rabatt*
K	*15.03.01*	*A*	*Kreditkarte*	*A*	*Rechnung*
na	*Fahrzeug*	*K*	*Kundenkartei*	*K*	*Reservierung*
K	*Servicemitarbeiter*	*K*	*Lastwagen*	*K*	*Reservierungsnummer*
K	*Firmenkonto*	*K*	*Mercedes*	*A*	*Herr Schneider*
K	*Frau Busch*	*K*	*Mietbeginn*	*O*	*Van*
O	*Führerschein*	*A*	*Mietdauer*	*K*	*Vertrag*
K	*HH-DV 123*	*A*	*Mietpreis*	*K*	*VW Golf*
na	*Kennzeichen*	*A*	*Nissan Sentra*	*K*	*VWGolfMitKzHHWS531*
A	*Kilometerstand*	*K*	*Petermann*	*O*	*2745 5678 3498*
A	*Kleintransporter*	*O*	*Pick-Up*	*na*	
K		*K*			

Lösung zu Selbsttestaufgabe 5.3-2:

Die folgenden Attribute dienen dazu, für jedes Fahrzeug den aktuellen Zustand im Verleihgeschäft nachzuhalten:

Tab. 5.3-2: *Attribute der Klasse Auto*

Attributname	***Verwendungszweck***
Fahrzeugnummer	*zur eindeutigen Identifizierung aller Leihfahrzeuge*
Standort	*Nummer des aktuellen Abstellplatzes*
entliehenAn	*Verweis auf Kunde, der Fahrzeug derzeit benutzt*
Reservierungen	*Liste Reservierungen für Fahrzeug*
kmStand	*aktueller Wert des Kilometerzählers*
Vorschäden	*Angaben zu Schäden am Fahrzeug*
nächsteWartung	*Datum der nächsten Wartung*
Füllstand	*Stand der Tankanzeige*

Lösung zu Selbsttestaufgabe 5.4-1:

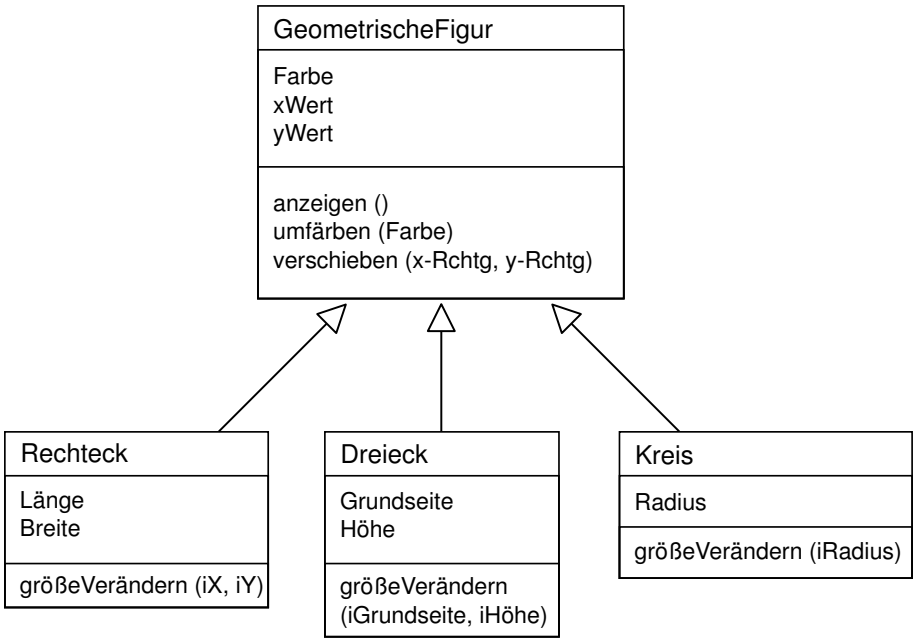


Abb. 5.4-3: Klassengraph für geometrische Figuren

Lösung zu Selbsttestaufgabe 5.6-1:

Tab. 5.6-1: *Attribute und Methode der Klasse Kfz-Mietvertrag*

Kfz-Mietvertrag
Attribute
Name
Vorname
Rückgabedatum
Ausweisnummer
Führerscheinnummer
Marke
Typ
Kennzeichen
kmBeginn
kmEnde
DatumBeginn
täglicheRate
euroProKilometer
extraTag
Versicherungen
Methoden
PreisBerechnen
VertragAusdrucken

Die Methoden zum Setzen der einzelnen Einträge eines Vertrags bei Vertragsabschluss wurden weggelassen. In Java gibt es dafür Standardmethoden, die sich aus des Zeichenfolge „set“ und den angehängten Namen des jeweiligen Attributs ergeben. Ein Beispiel wäre: `setkmEnde()`.

Index

Symbole

Anwendungsfall 68

Programm
    interaktives 56

A

Ada xviii, **52**
Akteur xviii, **69**
Aktivität 66
Aktivitätsdiagramm xviii, 66
Algol xix, xxiii, 51, **52**
Analyse xix
    Ist- 76
    Soll- 76, 80
Anwendungsfall xx
    -beschreibung **71**
    -diagramm xx, **70**
Assembler xx
Attribut xxi, 94
Aufgabenbeschreibung 59

B

Beziehung
    enthält- xxiv
    erweitert- xxv, 74, 75
    kommuniziert- xxx, 69
BlueJ ix, x
Botschaft xxii, 98
    Austausch von xxii
Bytecode xxiii

C

C++ xxiii, 56
Cobol xxiii

D

Datenlexikon **77**
Datenstruktur xxiv, 51, 55

E

Eiffel xxiv, 56
Eigenschaft 96
Einfachvererbung xxiv, 96
enthält-Beziehung xxiv, 74
erweitert-Beziehung xxv, **75**
EVA-Prinzip **55**
Exemplar xxv, 94

F

Fähigkeit 96
Fallstudie 59
Fortran xxiii, xxv, 51, 52

G

Geheimnisprinzip xxvi, 53, 105

H

höhere Programmiersprache xxvi, 47

I

Implementierung xxvii, 55
interaktives Programm xxvii, **56**
Interpreter xxvii
Ist-
    Analyse 76
    Situation 65

J

Java xxviii, **56**, 57

K

Klasse xxix, 93
    Ober- xxxiii, 96
    Unter- xxxix, 96
Klassen-
    diagramm xxix
    graph 97
    hierarchie xxx, 96
kommuniziert-Beziehung xxx, **69**

M

Maschinensprache xxxi, 47
Mehrfachvererbung xxxi, 96
Methode xxxi, 94
Modul 53
Modula xxxii, **52**

O

Oberklasse xxxiii, 96
Objekt xxxiii, 55, 92
Operation xxxiii, 55

P

Paket 53
Pascal xxxiv, 51, **52**
Pflichtenheft **78**
Programm xxxiv, **44**
    interaktives xxvii
Programmieren xxxiv, 42
    -im-Großen **43**
    -im-Kleinen **43**
    imperatives **51**
    prozedurales **51**
Programmieren
    datenstrukturorientiertes **53**
    objektorientiertes **54**
Programmierprozess **43**
Programmiersprache **46**
    höhere xxvi, 47
Programmierung
    prozedurale xxxv
Prozedur xxxv
prozedurale Programmierung xxxv

S

Semantik xxxv, 51
Simula xxxvi, 54
Smalltalk xxxvi, 56
Soll-Analyse 76, 80
Spezialisierung 96
Syntax xxxvii, 51
Szenario xxxvii, **71**

U

Übersetzer xxxviii
UML v, vi, xxxviii, 57
Unterklasse xxxix, 96

V

Vererbung xxxix
    Einfach- xxiv, 96
    Mehrfach- xxxi, 96
Verhalten xxxix
von Neumann-Architektur 55
von Neumann-Architektur **47**

Z

Zustand 47

Zustand xl