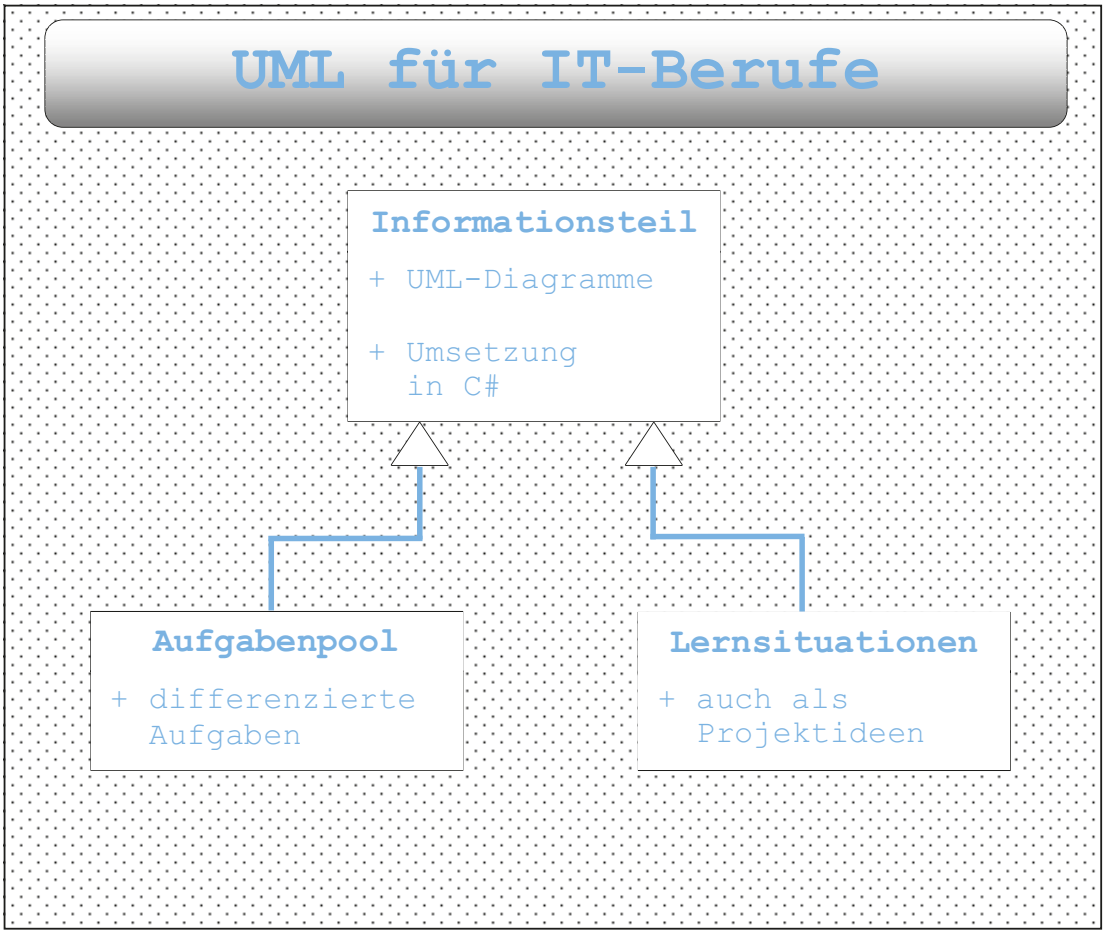
EUROPA-FACHBUCHREIHE
Programmierung für die
IT-Ausbildung



2. Auflage

VERLAG EUROPA-LEHRMITTEL · Nourney, Vollmer GmbH & Co. KG
Düsselberger Str. 23 · 42781 Haan-Gruiten

Europa-Nr.: 85580

Verfasser:
Dirk Hardy, 46049 Oberhausen

2. Auflage 2019

Druck 5 4 3 2

Alle Drucke derselben Auflage sind parallel einsetzbar, da sie bis auf die Korrektur von Druckfehlern identisch sind.

ISBN 978-3-8085-8588-7

www.europa-lehrmittel.de
Satz: Reemers Publishing Services GmbH, Krefeld
Umschlag: braunwerbeagentur, 42477 Radevormwald
Umschlagfotos: envfx-fotolia.com; Gina Sanders-fotolia.com; Gordon Bussiek-fotolia.com
Druck: Plump Druck & Medien GmbH, 53619 Rheinbreitbach

Vorbemerkung

Die moderne Softwareentwicklung geht weit über das Entwickeln von Algorithmen und die Umsetzung in eine Programmiersprache hinaus. Vielmehr geht es um eine umfassende Planung aller beteiligten Komponenten sowie eine solide Dokumentation der umzusetzenden Prozesse. Dieser Entwicklungsprozess kann durch die formale Sprache **UML** (**U**nified **M**odeling **L**anguage) hervorragend unterstützt werden. Dabei können die verschiedenen Diagramme der UML helfen, das geplante Softwaresystem in allen Phasen der Entwicklung zu beschreiben und damit auch die Grundlage für die Implementierung zu legen. Zusätzlich kann der Entwicklungsprozess verbessert werden, indem sogenannte CASE-Tools (**C**omputer-**A**ided-**S**oftware-**E**ngineering-Tools) eingesetzt werden. Damit ist nicht nur die Erstellung von UML-Diagrammen, sondern auch eine automatische Codeerzeugung sowie ein Reverse-Engineering möglich. Der Entwicklungsprozess wird dadurch professioneller und effizienter. Allerdings erfordert es eine hinreichende Einarbeitung in diese CASE-Tools. Für die Ausbildung im IT-Bereich ist die Auseinandersetzung gerade mit diesen Techniken ein sehr wichtiger Aspekt.

Aufbau des Buches

Das vorliegende Buch möchte die formale Sprache UML möglichst anschaulich, praxis- und unterrichtsnah vermitteln. Damit verfolgt dieses Buch einen **praktischen Ansatz**. Es ist die Ansicht des Autors, dass gerade in der schulischen Ausbildung der Zugang zu den komplexen Themen der Softwareentwicklung verstärkt und durch anschauliche und praktische Umsetzung vorbereitet werden muss. Neben dem Erlernen von wichtigen UML-Diagrammen wird ein besonderer Schwerpunkt auf die Umsetzung bestimmter Diagramme in die Programmiersprache C# gelegt. Damit soll die Lücke, die oftmals zwischen den eher theoretischen Konstrukten der UML und den eher praktischen Umsetzungen in den Programmiersprachen herrscht, geschlossen werden.

Das Buch ist in **drei Teile** gegliedert.

Der **erste Teil** des Buches dient als **Informationsteil** und bietet eine **systematische Einführung in die formale Sprache UML sowie die Umsetzungen in die Programmiersprache C#.**

Der **zweite Teil** des Buches ist eine **Sammlung von Übungsaufgaben**. Nach der Erarbeitung der entsprechenden Kenntnisse aus dem Informationsteil können die Aufgaben aus diesem Teil zur weiteren Auseinandersetzung mit den Themen dienen und durch verschiedene Schwierigkeitsgrade auch die Differenzierung im Unterricht ermöglichen.

Der **dritte Teil** des Buches beinhaltet **Lernsituationen** basierend auf dem Lernfeld „Entwickeln und Bereitstellen von Anwendungssystemen“ aus dem Rahmenlehrplan für die IT-Berufe (speziell Fachinformatiker-Anwendungsentwicklung). Lernsituationen konkretisieren sich aus den Lernfeldern und sollen im Idealfall vollständige Handlungen darstellen (Planen, Durchführen, Kontrollieren). Aus diesem Grund werden die Lernsituationen so angelegt, dass neben einer Planungsphase nicht nur die Durchführung im Blickpunkt steht, sondern auch geeignete Testverfahren zur Kontrolle des Entwicklungsprozesses in die Betrachtung einbezogen werden. Die Lernsituationen können aber auch als **Projektideen** verstanden werden.

Das Buch ist für alle berufsbezogenen Ausbildungsgänge im IT-Bereich konzipiert. Durch die differenzierten Aufgabenstellungen kann es in allen IT-Berufen (speziell Fachinformatiker), aber auch von den informationstechnischen Assistenten genutzt werden.

In diesem Buch werden einige Diagramme der UML mit dem CASE-Tool **StarUML** erstellt. Das Tool wird kostenfrei zum Download (als Evaluierungs-Edition) angeboten. Für die Umsetzung in die Programmiersprache dient die Entwicklungsumgebung von Microsoft (Community-Edition für C#). Diese Entwicklungsumgebung ist ebenfalls kostenfrei als Download im Internet verfügbar.

Für Anregungen und Kritik zu diesem Buch sind wir Ihnen dankbar (gerne auch per E-Mail).

Dirk Hardy

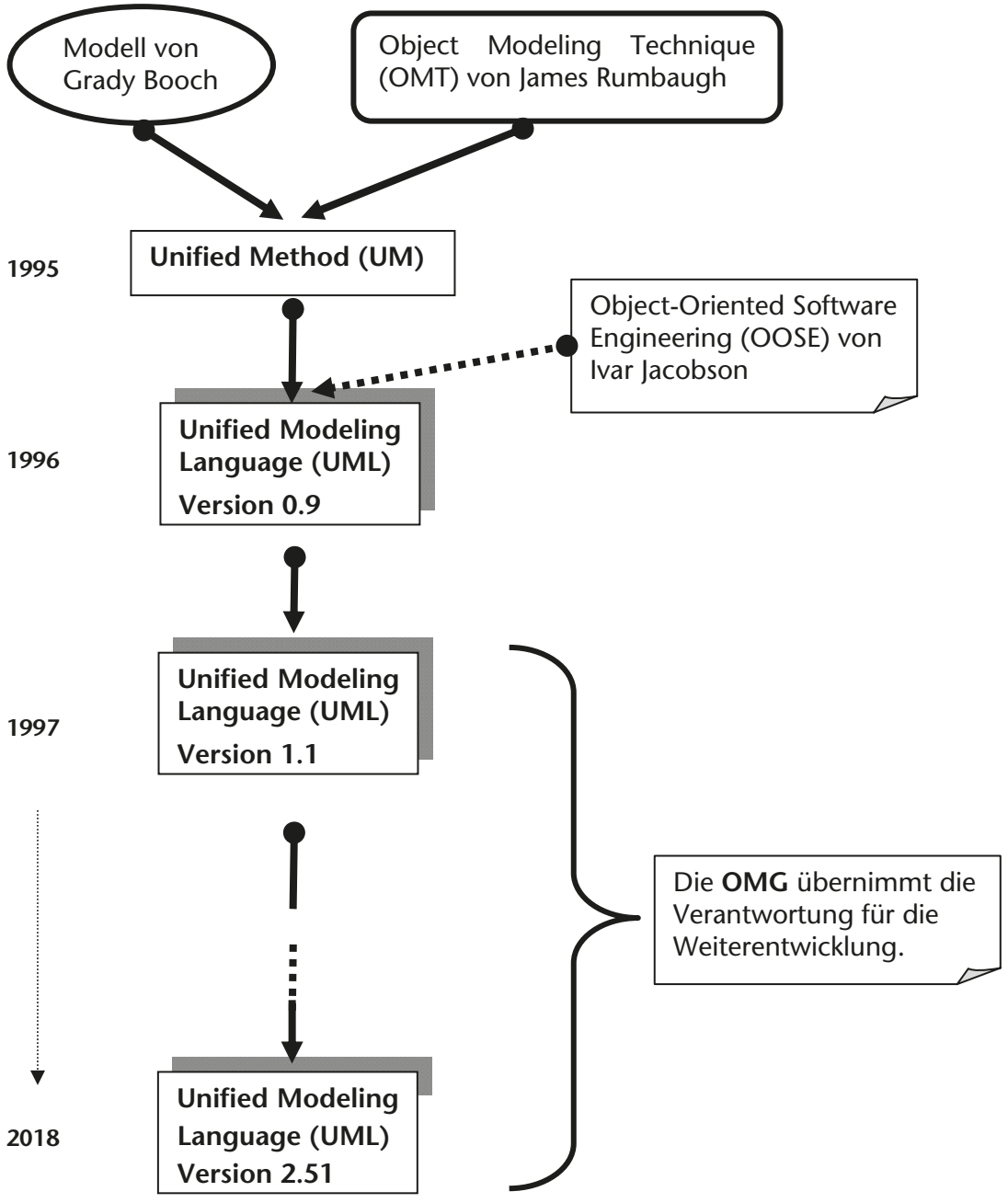
E-Mail: Hardy@DirkHardy.de

Im Frühjahr 2019

Verlag Europa-Lehrmittel

E-Mail: Info@Europa-Lehrmittel.de

Zeitliche Entwicklung der UML



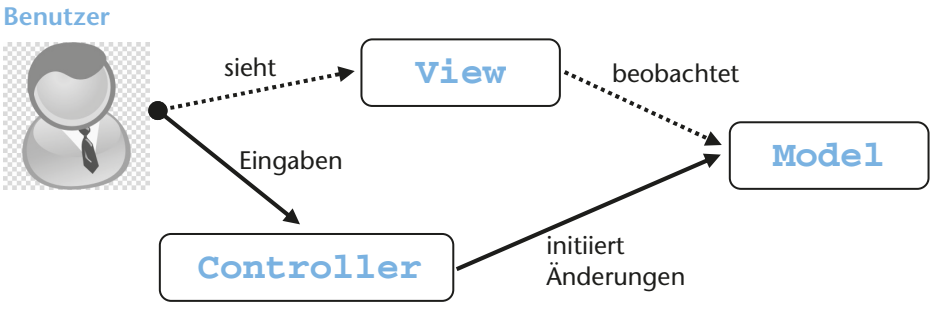
Die drei Gründer der UML wurden auch als die drei *Amigos* bezeichnet. Booch arbeitete seit den 80er-Jahren bei der Firma **Rational Software** in Kalifornien. Rational Software ist ein Unternehmen mit dem Schwerpunkt Systemanalyse und -design. Ein besonderer Schwerpunkt war die objektorientierte Analyse und das objektorientierte Design. Dafür entwickelte die Firma auch spezielle Tools wie das Produkt **Rational Rose** (ein mächtiges Softwaredesignwerkzeug, das auf der UML basiert). 1994 trat James Rumbaugh in die Firma ein und entwickelte mit Grady Booch zusammen die **Unified Method** (UM). Ein Jahr später kam wegen der Übernahme der Firma **Objectory AB** durch Rational Software auch der Besitzer dieser Firma, Ivar Jacobson, zu Rational Software.

Damit war das Trio komplett und die drei *Amigos* entwickelten die erste Version der UML.

Die Firma Rational Software wurde 2003 von **IBM** übernommen, die Produkte laufen aber weiter unter den bekannten Namen.

Model View Controller (MVC)

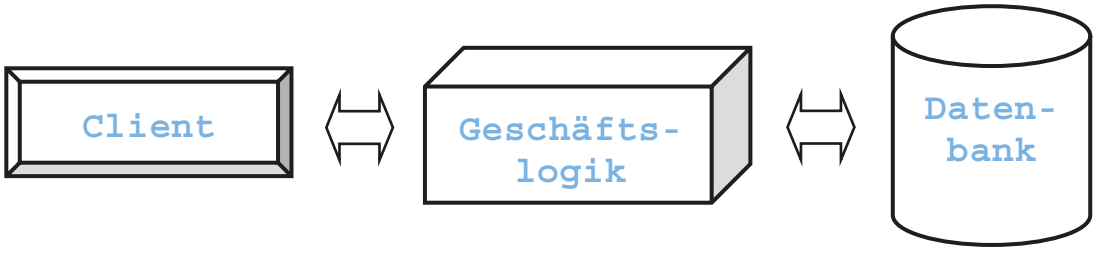
Dieses Konzept trennt ein System in eine Modell-Klasse, eine Ansichts-Klasse (oder mehrere Ansichts-Klassen) und eine Steuerungs-Klasse. Die Modell-Klasse repräsentiert die Daten des Systems, die mithilfe der Ansichts-Klassen in verschiedenen Sichtweisen dargestellt werden können. Angenommen, es handelt sich bei den Daten um Messwerte, so könnten diese Werte beispielsweise in einer Tabellenansicht, aber auch in einer Grafik in einem Koordinatensystem angezeigt werden. Der Vorteil dieser Trennung ist die Unabhängigkeit von Datenhaltung und Sicht. Das Hinzufügen weiterer Ansichten ist unabhängig von der Modell-Klasse. Die Steuerungs-Klasse ist das Bindeglied zwischen Modell und Ansicht. Sie registriert Benutzereingaben, die über die Ansicht kommen, und kommuniziert dann mit dem Modell, um beispielsweise weitere Daten zu erhalten oder Änderungen der Daten zu initiieren.



Das Schichtenmodell

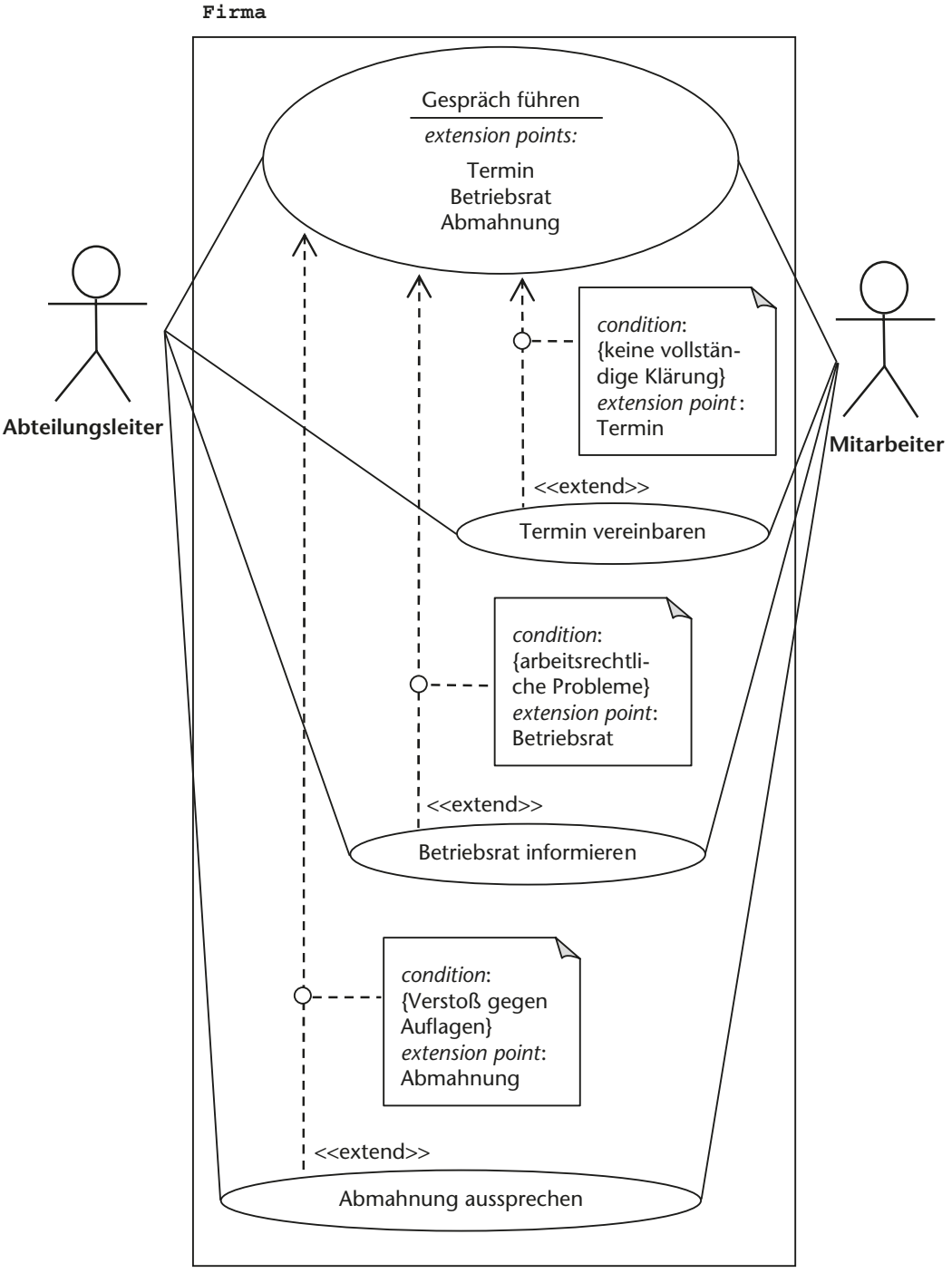
Das MVC-Konzept liefert einen Entwurf für ein Softwaresystem, welches auch ein einzelnes Programm sein kann. Das Schichtenmodell geht noch einen Schritt weiter und liefert einen Entwurf für ein Softwaresystem, das über mehrere Komponenten (Schichten, engl. *tier*) verfügt, die auch physikalisch getrennt sein können und über ein Netzwerkprotokoll miteinander kommunizieren. Häufig genutzt werden das Zwei-Schichten-Modell (engl. *two tier*) und das Drei-Schichten-Modell (engl. *three tier*). Das Zwei-Schichten-Modell ist oftmals das klassische Client-Server-Modell. In der Regel ist der Client ein Programm, das eine Benutzeroberfläche und die zugehörige Geschäftslogik anbietet. Die benötigten Daten fragt der Client dann vom Server ab. Das Drei-Schichten-Modell koppelt hingegen die Geschäftlogik von der Benutzoberfläche ab – damit wird eine weitere Schicht angelegt. In der Web-Programmierung wird ein Drei-Schichten-Modell sehr oft durch einen Client in Form eines Browsers und durch die Anzeige mit HTML, XHTML oder XML umgesetzt. Die Geschäftslogik liegt dann auf einem Webserver und wird mit einer Skriptsprache wie PHP oder Perl implementiert. Die Daten werden von einem zusätzlichen Server mithilfe geeigneter Schnittstellen wie ODBC ausgelesen. Dieses System hat den großen Vorteil, dass Änderungen an einer Schicht (Ansicht, Logik oder Datenhaltung) die anderen Schichten nicht tangieren. Wartbarkeit und Erweiterbarkeit eines solchen Systems sind damit sehr gut.

Drei-Schichten-Modell:



Beispiel:

Der Anwendungsfall *Gespräch führen* kann nicht nur durch den Anwendungsfall *Termin vereinbaren*, sondern auch durch die Anwendungsfälle *Betriebsrat informieren* und *Abmahnung aussprechen* erweitert werden.

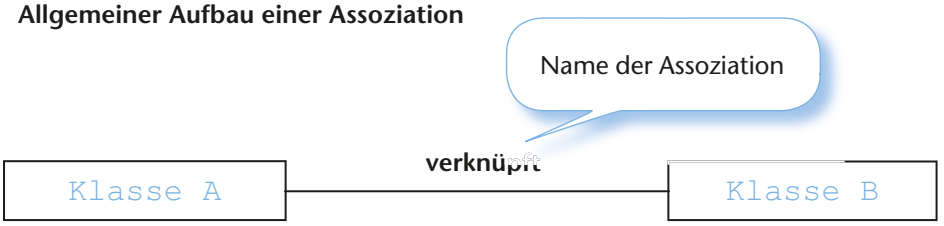


2.4 Beziehungen zwischen Akteuren

2.4.1 Generalisierung und Spezialisierung

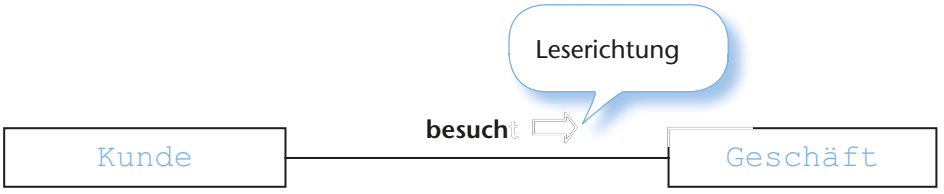
Die Generalisierung bzw. Spezialisierung bei Akteuren erfolgt analog zu den Anwendungsfällen. Ein Akteur spezialisiert sich zu einem weiteren Akteur. Dabei ist ganz wichtig, dass der spezialisierte Akteur automatisch an allen Anwendungsfällen des generalisierten Akteurs beteiligt ist. Das bedeutet mit anderen Worten, dass der spezialisierte Akteur die Beteiligungen an den Anwendungsfällen einfach *erbt*. Die Beziehung wird wie gewohnt durch einen Pfeil in Richtung des generalisierten Akteurs beschrieben.

3.3.1 Allgemeiner Aufbau einer Assoziation



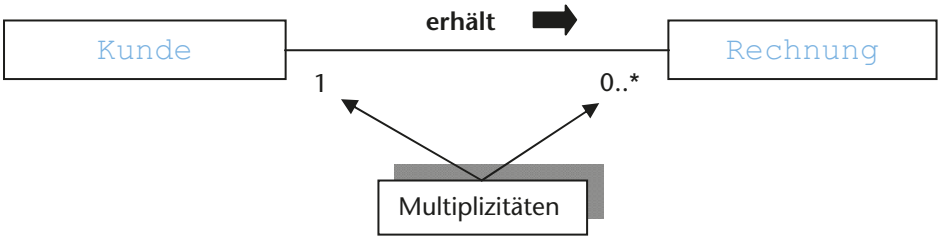
Leserichtung einer Assoziation

Um eine Assoziation näher zu spezifizieren, kann eine Leserichtung (durch einen Richtungspfeil) hinzugefügt werden.



Multiplizitäten einer Assoziation

Die Multiplizitäten einer Assoziation geben an, wie viele Objekte der einen Klasse mit wie vielen der anderen Klasse in Verbindung stehen.



Hinweis:

Die Multiplizitäten sind so zu lesen:

- Ein Kunde erhält keine oder beliebig viele Rechnungen.
- Eine Rechnung gehört genau zu einem Kunden.

Mögliche Multiplizitäten:

Multiplizität	Beschreibung
`0`	keins
`1`	genau eins
`*`	beliebig viele
`0..*`	keins oder beliebig viele (wie *)
`1..*`	eins oder beliebig viele
`1..3`	eins, zwei oder drei
`4..20`	4 bis 20
`1,5,7`	eins, fünf oder sieben

```
            return false;
        }
        public int GetNummer()
        {
            return nummer;
        }
    }
    class CKunde
    {
        private string name;
        private ArrayList konten = new ArrayList();

        public CKunde()
        {
            NeuesKonto();
        }
        public void SetName(string n)
        {
            name = n;
        }
        public string GetName()
        {
            return name;
        }
        public void NeuesKonto()
        {
            if (konten.Count==5) return;
            CKonto k = new CKonto();
            bool ok;
            int nr;
            do
            {
              Console.Write("Bitte die Nummer fuer Konto " +
                     (konten.Count + 1) + " angeben: ");
              nr = Convert.ToInt32(Console.ReadLine());
              ok = k.SetNummer(nr);
              if (!ok)
                Console.WriteLine("Fehlerhafte Kontonummer!");
            }
            while (!ok);
            konten.Add(k);
        }
        public void Ausgabe()
        {
            Console.WriteLine("Kunde " + name
                              + " hat folgende Konten:");
```

Die `ArrayList konten` verwaltet die fünf Konten.

Umsetzung der *1..5*-Beziehung. Ein Konto wird immer angelegt.

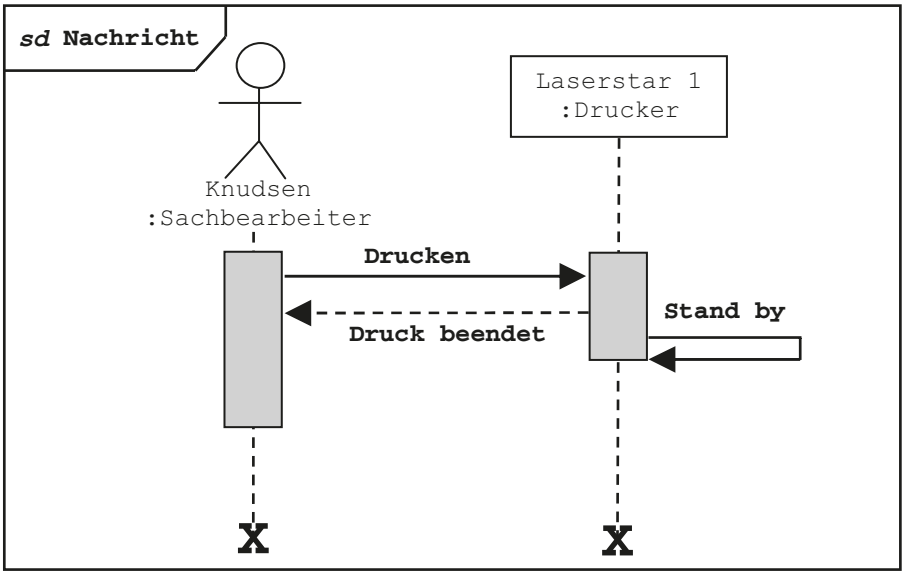
Maximal fünf Konten sind erlaubt.

Hinzufügen eines neuen Kontos

Synchrone Nachrichten

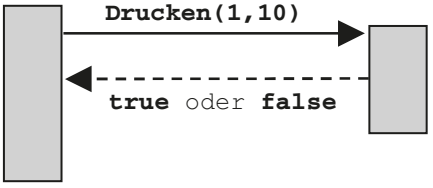
Die synchrone Nachricht wird durch eine Linie mit einer geschlossenen Pfeilspitze symbolisiert.

In dem folgenden Sequenzdiagramm sendet der Sachbearbeiter die Nachricht „Drucken“ an den Laserdrucker. Der Drucker antwortet mit der Meldung „Druck beendet“. Bis zu dieser Rückmeldung muss der Sachbearbeiter warten und kann keine weiteren Aktionen durchführen. Die Rückmeldung wird durch eine gestrichelte Linie mit einer geschlossenen Pfeilspitze symbolisiert. Im Anschluss sendet der Drucker sich selbst die Nachricht, in den „Stand by“-Modus zu wechseln.



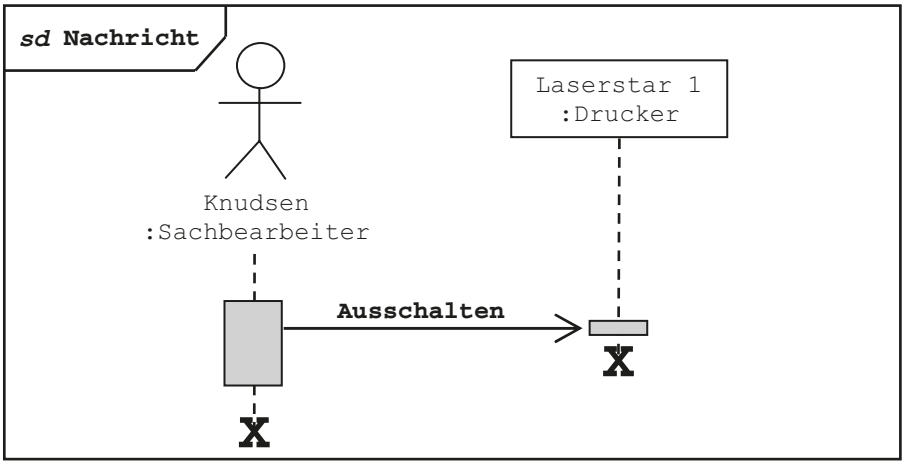
Hinweis:

Nachrichten können weiter spezifiziert werden, indem Übergabeparameter angegeben werden. Auch die Antworten können Rückgabewerte enthalten. Beispielsweise sollen die Seiten 1..10 gedruckt werden. Bei erfolgreichem Druck wird „`true`“ zurückgegeben, ansonsten „`false`“.



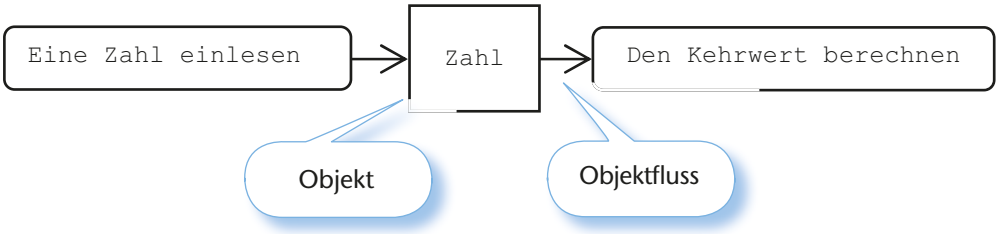
Asynchrone Nachrichten

Die asynchrone Nachricht wird durch eine Linie mit einer offenen Pfeilspitze symbolisiert. Der Sender der Nachricht wartet nicht auf eine Antwort, sondern führt seine Aktionen weiter durch.

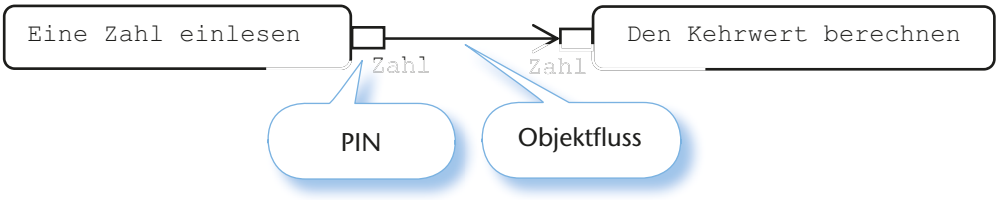


Beispiel:

Das Objekt `Zahl` wird in den Steuerungsfluss eingebunden:



Alternativ kann diese Darstellung auch mit der sogenannten **PIN-Notation** erfolgen:

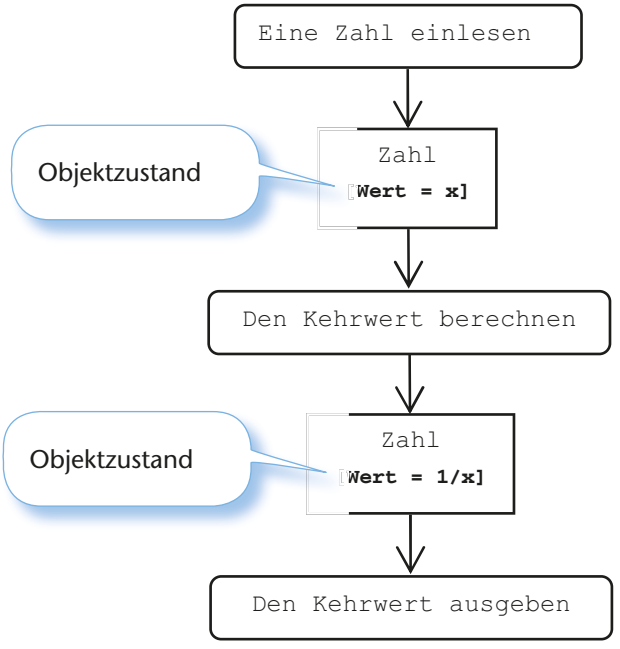


Hinweis:

Unter Objekten sind natürlich Instanzen von Klassen zu verstehen. Unter Objektfluss kann man sich beispielsweise die Übergabe eines Objektes an eine Methode (in Form eines Parameters) vorstellen. In dem obigen Beispiel könnte die Aktion „Eine Zahl einlesen" beispielsweise ein Objekt `Zahl` erzeugen und an eine Methode `Kehrwert` übergeben, die die Aktion „Den Kehrwert berechnen" umsetzt.

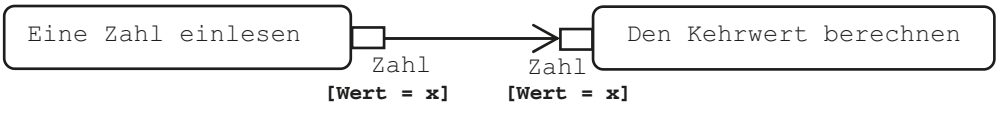
Zustände von Objekten

Die Objekte können im Laufe des Objektflusses ihren Zustand ändern (die Attributwerte ändern). Solche Änderungen können im Diagramm durch Zustandsangaben gekennzeichnet werden:

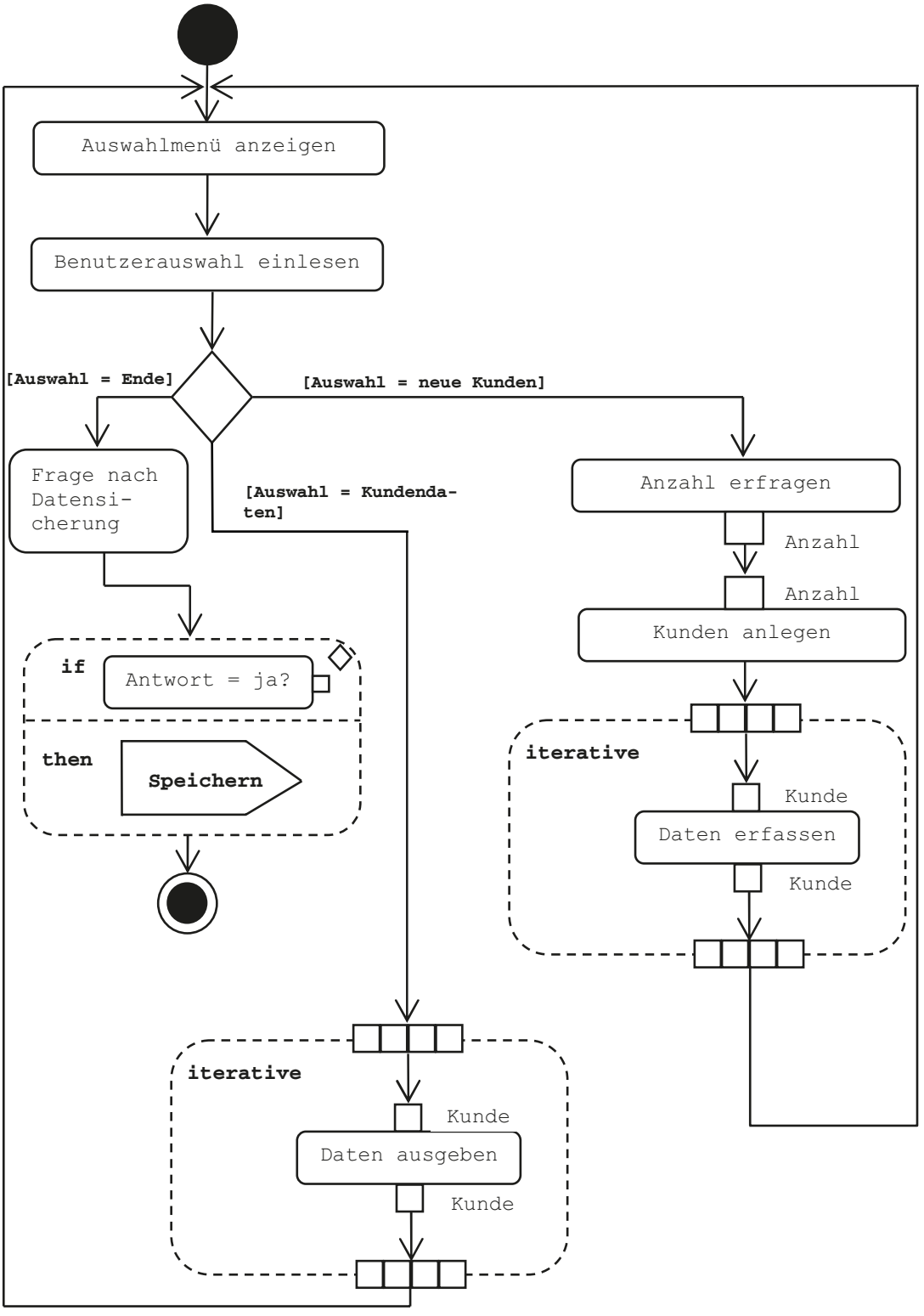


Hinweis:

In der PIN-Notation wird der Zustand unter dem Objektnamen angegeben:



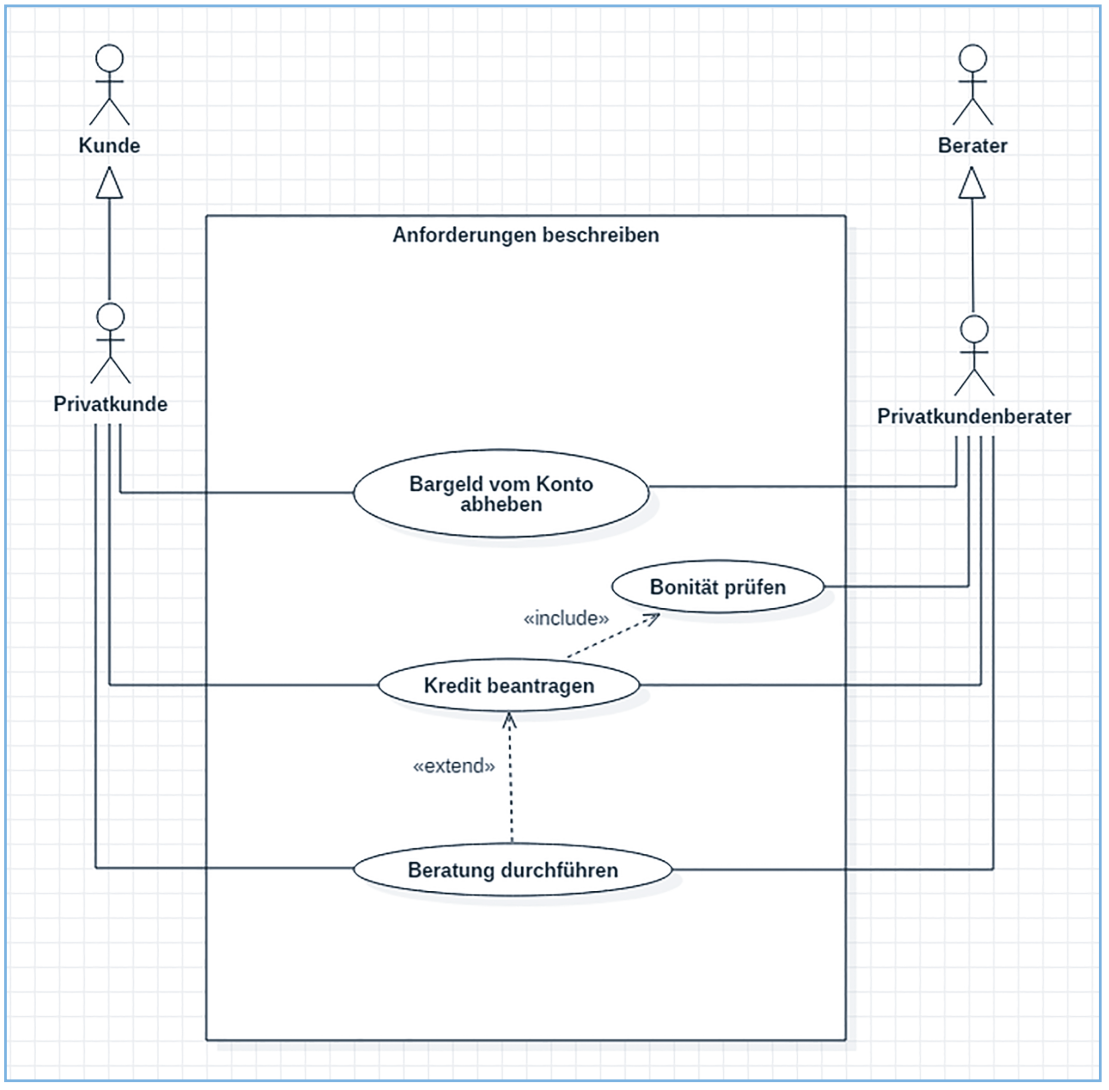
und Ausgeben der Kunden wird mit Expansionsbereichen umgesetzt, die Sicherung wird durch ein Signal in Gang gesetzt. Ein Unterbrechungsbereich soll die Datensicherung kontrolliert beenden, wenn ein Fehler auftritt. Zusätzlich werden noch eine Verzweigung und eine einseitige Selektion verwendet, um das Auswahlmenü für den Benutzer zu gestalten und eine Abfrage vor dem Beenden des Programms durchzuführen.



Die Anwendungsfälle „Bargeld vom Konto abheben“, „Kredit beantragen“ und „Beratung durchführen“ sind angelegt worden. Die beiden Akteure „Privatkunde“ und „Privatkundenberater“ sind mit den Anwendungsfällen assoziert.

7.2.2 Objektorientierte Analyse (OOA)

Das Ziel der objektorientierten Analyse ist das möglichst vollständige Erfassen der Anforderungen an das System. Dazu wird beispielsweise das Anwendungsfalldiagramm weiter spezifiziert. Die Beziehungen zwischen den Akteuren und Anwendungsfällen sowie mögliche Generalisierungen werden analysiert und ergänzt.



- Kunde und Berater können als Generalisierungen von Privatkunde bzw. Privatkundenberater identifiziert werden. Das ist sinnvoll, denn das System könnte zu einem späteren Zeitpunkt auch um Geschäftskunden erweitert werden.
- Der Anwendungsfall „Bonität prüfen“ wurde als weiterer Anwendungsfall angelegt und hat eine `<<include>>`-Beziehung zum Anwendungsfall „Kredit beantragen“. Bei jedem Kreditantrag muss die Bonität geprüft werden.
- Der Anwendungsfall „Kredit beantragen“ hat eine `<<extend>>`-Beziehung zum Anwendungsfall „Beratung durchführen“. Es ist nicht zwingend eine Beratung anzusetzen, nachdem ein Kunde einen Kreditantrag gestellt hat, aber durchaus möglich.

Identifizierung der Klassen

Mithilfe des Anwendungsfalldiagrammes sollen nun die beteiligten Klassen ermittelt. Damit soll ein erstes einfaches Klassendiagramm erstellt werden. Dieser Prozess ist relativ schwierig und hängt stark von der Erfahrung und dem Weitblick der Entwickler ab. Das bedeutet, dass es keine allgemein gültige und immer anwendbare Methode gibt, um die beteiligten Klassen zu ermitteln. An dieser Stelle soll deshalb ein einfaches Verfahren dargestellt werden.

```
   private void buttonAnlegen_Click(object sender, EventArgs e)
   {

      CPrivatkunde k = null;
      CPrivatkundenberater b = null;
      vKredit = null;

      string kname = comboBoxKunde.SelectedItem.ToString();
      string bname = comboBoxBerater.SelectedItem.ToString();

      foreach (CPrivatkunde pk in kListe)
      {
         if (kname == pk.NAME) k = pk;
      }
      foreach (CPrivatkundenberater pb in bListe)
      {
         if (bname == pb.NAME) b = pb;
      }

      if (DBABfrage(k.KONTO) == true)
      {
         if (k.BONITAET == true)
         {
            vKredit = new CKredit();
            vKredit.KUNDEN_ID = k.ID;
            vKredit.BETRAG = Convert.ToDouble(textBoxKreditHoehe.Text);
            vKredit.LAUFZEIT = Convert.ToInt32(textBoxLaufzeit.Text);
            vKredit.ZINSSATZ = Convert.ToDouble(textBoxZinssatz.Text);

            b.AddKredit(vKredit);
         }
         else MessageBox.Show("Kein Kredit möglich - fehlende Bonität");
      }
      else MessageBox.Show("Kein Kredit möglich - Konto überzogen");
      this.Close();
   }
}
public partial class GUIPrivatkunden : Form
{
   private CPrivatkunde vPrivatkunde;
   public GUIPrivatkunden()
   {
      InitializeComponent();
   }
```

Kredit auf `null` setzen – noch kein Kredit

Namen aus den Kombinationsfeldern merken

Kunden zu dem Namen finden

Berater zu dem Namen finden

Kreditwürdigkeit prüfen – nach Vorgabe im Aktivitätsdiagramm

Neuen Kredit anlegen

Kreditdaten zuweisen

Umsetzung der Aggregation

Der „Kunden anlegen“-Dialog

Verweis für das neue Kundenobjekt

Teil 2
Aufgabenpool

Aufgabe 6.2

Entwickeln Sie ein Aktivitätsdiagramm zu der folgenden Ausgangssituation:

Die Kunden einer Kfz-Versicherungsgesellschaft können ihre Schäden online über eine entsprechende Maske eintragen. Die Daten werden anschließend von einem Sachbearbeiter geprüft. In einem ersten Schritt muss die Schuldfrage geklärt werden. Hat der Versicherte Schuld (oder Teilschuld) an dem Schadensfall, so fordert der Sachbearbeiter ein Gutachten an, in dem die Schadenssumme genau berechnet wird. Liegt diese Schadenssumme unter 12.500 Euro, so muss der Sachbearbeiter die Vertragsdaten des Versicherten anpassen (Anpassung des Schadensfreiheitrabatts). Danach kann eine Reparaturfreigabe erfolgen, allerdings unter einschränkenden Bedingungen, die dem Versicherten mitgeteilt werden.

Liegt die Schadenssumme über 12.500 Euro, so muss der Sachbearbeiter neben der Anpassung des Vertrags auch den Abteilungsleiter informieren. Erst danach kann dann die Reparaturfreigabe unter Einschränkung erfolgen.

Falls der Versicherte keine (Teil-)Schuld trägt, so kann eine uneingeschränkte Reparaturfreigabe erfolgen.

7 Aufgaben zur Softwareentwicklung

Hinweis:

Die Aufgaben zur Softwareentwicklung sind in der Regel sehr komplex, da sie mehrere Diagrammtypen und die entsprechenden Implementierungen enthalten. Deshalb werden an dieser Stelle keine einzelnen Aufgaben gestellt, sondern auf die Lernsituationen verwiesen. Die Lernsituationen bieten den Rahmen für solche komplexen Aufgaben.

8 Aufgaben zu den weiteren Diagrammen

Aufgabe 8.1 Kompositionsstrukturdiagramm

Stellen Sie die folgende Problematik mithilfe eines Kompositionsstrukturdiagramms dar:

Eine Schule hat Lehrer, die wiederum Klassen (maximal sieben) unterrichten. Die Lehrer haben einen Zugang zu der Vertretungsplansoftware, über welche sie die aktuellen Vertretungspläne abrufen können.

Weiterhin kommunizieren Lehrer und Schüler, wenn es um die Beurteilung von Leistungen geht. Dabei kann die Kommunikation zwischen Sportlehrern und deren Schülern als ein Spezialfall betrachtet werden, denn die Mitarbeitsnote wird anders ermittelt als im Unterricht, der in einem Klassenraum stattfindet.

Aufgabe 8.2 Komponentendiagramm

Ausgangssituation:

Ein Online-Auktionshaus möchte seine Software komplett neu gestalten. In einem ersten Schritt soll eine Übersicht der zugrunde liegenden Komponenten entwickelt werden. Das neue System soll ein Drei-Schichten-System sein (*three tier*). Auf einem Server soll ein Webserver installiert werden, der über einen Perl-Interpreter verfügt. In der Sprache Perl sollen dann auch die nötigen Klassen für die Abwicklung der Auktionen geschrieben werden (Kunden-Klasse, Auktions-Klasse etc.). Die Daten sollen in einer Datenbank gesichert werden, die auch auf einem anderen separaten Server installiert werden kann. Der Zugriff der Kunden erfolgt über einen beliebigen Browser, der allerdings für JavaScript geeignet sein muss, da einige JavaScripts lokal auf dem Browser laufen müssen.

Entwickeln Sie ein Komponentendiagramm zu der obigen Ausgangssituation.

Aufgabe 8.3 Verteilungsdiagramm

Ergänzen Sie das Komponentendiagramm zu der geplanten Online-Auktions-Software (aus der Aufgabe 8.2) durch ein Verteilungsdiagramm. Dabei werden die folgenden Daten zur benötigten Hardware und Software zugrunde gelegt:

Hinweis:

Das obige Modell ist eine einfache Variante der MVC-Architektur, da nur ein Modell, eine Ansicht und eine Steuerung instanziert werden sollen. In der Regel gibt es mehrere Ansichten oder auch Steuerungen, die auf das Modell zugreifen.

Eine Bildschirmausgabe der Konsolenanwendung könnte so aussehen:

```
Auswahl:
<1> Neue Temperatur in Celsius eingeben
<2> Neue Temperatur in Fahrenheit eingeben
<3> ENDE
1
Bitte die Temperatur in Celsius: 10
```

Controller

```
T E M P E R A T U R - U M R E C H N U N G <Version 1.0>
Temperatur in Celsius:    10
Temperatur in Fahrenheit: 50
```

View

```
Auswahl:
<1> Neue Temperatur in Celsius eingeben
<2> Neue Temperatur in Fahrenheit eingeben
<3> ENDE
1
Bitte die Temperatur in Celsius: 20
```

Controller

```
T E M P E R A T U R - U M R E C H N U N G <Version 1.0>
Temperatur in Celsius:    20
Temperatur in Fahrenheit: 68
```

View

Durchführung:

Implementieren Sie die drei Klassen entsprechend des Klassendiagramms. Die Eingabelogik (Menü und Benutzereingabe) soll dabei als Methode der Steuerungs-Klasse umgesetzt werden.

Die Ausgabe auf dem Bildschirm wird als Methode der Ansichts-Klasse umgesetzt, die von der Steuerungs-Klasse aufgerufen wird, sobald der Benutzer eine neue Wahl getroffen hat. Die Modell-Klasse wird sowohl von der Steuerungs-Klasse (zur Speicherung neuer Werte) als auch von der Ansichts-Klasse genutzt, um die Daten für die Bildschirmausgabe zu erhalten.

Das Hauptprogramm in C# ist vorgegeben:

```
static void Main(string[] args)
{
   CModel modell = new CModel(10);
   CView ansicht = new CView(modell);
}
```