

bfw
Unternehmen für Bildung.

Software zur Verwaltung von Daten anpassen

Modul: ITG / F

Version: 1.0

Juni 2022

Erstellt von: Torsten Schwarze

Modul: ITG / F - Software zur Verwaltung von Daten anpassen

Version, Datum: 1. Juni 2022

Erstellt von: Torsten Schwarze

Alle Rechte vorbehalten. Kein Teil des Werkes darf in irgendeiner Form (Druck, Fotokopie oder einem anderen Verfahren)
ohne Genehmigung des Herausgebers reproduziert oder unter Verwendung elektronischer Systeme verarbeitet, vervielfältigt oder verbreitet werden.

Die Modulunterlage und alle ihre Bestandteile wurden mit großer Sorgfalt erstellt und geprüft. Dennoch können Fehler nicht vollkommen ausgeschlossen werden. Der Herausgeber und die Autoren können für fehlerhafte Angaben und deren Folgen weder eine juristische Verantwortung noch irgendeine Haftung übernehmen.

Literaturhinweis

Titel: Programmentwicklung mit C#.NET und SQL Server

IT-Berufe Grundstufe

ISBN: 978-3-427-01166-8

978-3-14-220000-2

Verlag: Westermann Gruppe

1 Inhalt

1 Erste Schritte 9
1.1 Einleitung 9
1.2 Motivation 9
1.3 Technische Voraussetzungen 9
1.4 Erkundung 9
1.5 Einführung in .NET und C# 10
1.6 Einführung in Visual Studio 10
1.7 Erstellen einer Konsolenanwendung 10
2 Einfache Datenverarbeitung mit C# 15
2.1 Hauptauftrag 15
2.2 Variablen 15
2.2.1 Variablen deklarieren 15
2.2.2 Einfache Datentypen 16
2.2.3 Variablen initialisieren 16
2.2.4 Gleichzeitige Deklaration von Variablen 17
2.2.5 Variablen setzen 17
2.2.6 Typkonvertierung 17
2.2.7 Konstanten 18
2.2.8 Entwurf und Realisierung 18
2.3 Ein- und Ausgabe 18
2.3.1 Benutzereingabe 19
2.3.2 Konsolenausgabe 19
2.3.3 Entwurf und Realisierung 19
2.4 Arithmetische Operation 20
2.5 Weiterführende Themenbereiche 21
2.5.1 Weitere mathematische Funktionen 21
2.5.2 Literale 22
3 Kontrollstrukturen, Datentypen und Operatoren 23
3.1 Hauptauftrag 23
3.2 Kontrollstrukturen 23
3.3 Struktogramme 24
3.3.1 Entwurf und Realisierung 24
3.4 Vergleichsoperatoren 25
3.4.1 Entwurf und Realisierung 25
3.5 IF-Abfrage: Einseitige Auswahl 25
3.5.1 Entwurf und Realisierung 26
3.6 IF-Abfrage: Zweiseitige Auswahl 27
3.6.1 Entwurf und Realisierung 28

3.7 IF-Abfrage: Mehrfachauswahl ... 28
3.7.1 Entwurf und Realisierung ... 30
3.8 Switch ... 30
3.8.1 Entwurf und Programm ... 32
3.9 Eindimensionale Arrays ... 33
3.9.1 Arrays deklarieren ... 33
3.9.2 Arraywerte setzen ... 33
3.9.3 Arraywerte auslesen ... 34
3.9.4 Arrays mit Anfangswerten initialisieren ... 34
3.9.5 Entwurf und Realisierung ... 34
3.10 FOR-Schleife ... 35
3.10.1 Entwurf und Realisierung ... 36
3.11 Parameter der Main-Methode ... 37
3.11.1 Entwurf und Realisierung ... 37
3.12 WHILE-Schleifen ... 38
3.12.1 Entwurf und Programm ... 39
3.13 Exkurs: Klasse string ... 39
3.14 DO/While-Schleife ... 40
3.14.1 Break-Anweisung ... 40
3.14.2 Entwurf und Realisierung ... 41
3.15 MehrdimensionaIeArrays ... 41
3.15.1 Arrays deklarieren ... 42
3.15.2 Arrays setzen ... 42
3.15.3 Arrays mit Anfangswerten initialisieren ... 42
3.15.4 Zugriff mit verschachtelter FOR-Schleife ... 42
3.15.5 Entwurf und Realisierung ... 43
3.16 Logische Operatoren ... 44
3.16.1 UND-Verknüpfung ... 44
3.16.2 ODER-Verknüpfung ... 44
3.16.3 Entwurf und Realisierung ... 45
3.17 Strukturen ... 45
3.17.1 Entwurf und Realisierung ... 46
3.18 Enums ... 47
3.19 Weiterführende Themenbereiche ... 49
3.19.1 Else-if-Anweisung ... 49
3.20 PAP Programmablaufplan ... 51
3.20.1 Was sind Programmablaufpläne? ... 51
3.20.2 Veränderung der Pfeilführung ... 52
3.20.3 Wichtige Hinweise ... 52
4 Grundlagen der objektorientierten Programmierung ... 54

4.1 Hauptauftrag ... 54
4.2 Was bedeutet objektorientiert? ... 54
4.3 Objektorientierte Analyse, Design und UML ... 56
4.4 OOP-Konzept Abstraktion ... 58
4.4.1 Klassen in Visual Studio anlegen ... 59
4.4.2 Klassenmember Feld ... 59
4.4.3 Klassenmember Funktion ... 60
4.4.4 Klassenmember Prozedur ... 60
4.4.5 Objekte instanzieren ... 61
4.4.6 Entwurf und Realisierung ... 62
4.5 Methoden mit Parametern ... 63
4.5.1 Parameter out und ref ... 64
4.5.2 Entwurf und Realisierung ... 64
4.6 Konstruktoren ... 65
4.6.1 Entwurf und Realisierung ... 66
4.7 OOP-Konzept Datenkapselung ... 66
4.7.1 Zugriffsmodifizierer private ... 66
4.7.2 Klassenmember Eigenschaften ... 67
4.7.3 WriteOnly-Property ... 68
4.7.4 ReadOnly-Property ... 68
4.7.5 Entwurf und Realisierung ... 69
4.8 OOP-Konzept Vererbung ... 69
4.8.1 UML-Diagramm ... 70
4.8.2 Klassen ableiten ... 70
4.8.3 Klassenmember der Basisklasse verwenden ... 71
4.8.4 Zugriffsmodifizierer protected ... 71
4.8.5 Konstruktoren in Unterklassen ... 71
4.8.6 Typumwandlung und späte Bindung ... 72
4.8.7 Entwurf und Realisierung ... 73
4.9 OOP-Konzept Polymorphie ... 74
4.9.1 Methoden Virtualisieren und überschreiben ... 74
4.9.2 Entwurf und Realisierung ... 75
4.10 Methoden überladen und ausblenden ... 76
4.10.1 Methoden überladen ... 76
4.10.2 Methoden ausblenden ... 76
4.11 Abstrakte Klassen ... 77
4.11.1 Entwurf und Realisierung ... 77
4.12 Statische Klassen und Klassenmember ... 77
4.12.1 Statische Eigenschaften und Felder ... 77
4.12.2 Statische Methoden und statische Klassen ... 78

4.13 Beziehungen zwischen Klassen ... 78
4.14 Weitere Themenbereiche ... 79
4.14.1 Klasse Objekt, Boxing und Unboxing ... 79
4.14.2 Werte- und Referenztypen ... 79
5 Fehlersuche und Fehlerbehandlung ... 81
5.1 Werkzeuge für die Fehlersuche ... 81
5.1.1 Fehlerarten ... 81
5.1.2 Breakpoint setzen ... 81
5.1.3 Variablen überwachen ... 82
5.1.4 Aufrufliste ... 84
5.2 Fehlerbehandlung (Exception Handling) ... 84
5.2.1 Unvorhergesehene Fehler abfangen ... 85
5.2.2 Mehrfaches catch ... 86
5.2.3 Fehler werfen ... 87
5.2.4 Finally ... 87
5.2.5 Eigene Exception definieren ... 88
6 Grundlegende .NET-Klassen ... 89
6.1 Texte verarbeiten mit Klasse string und StringBuIder ... 89
6.1.1 Textvariable deklarieren und initialisieren ... 89
6.1.2 Texte zusammensetzen ... 89
6.1.3 Texte teilen ... 90
6.1.4 Anzahl von Zeichen ermitteln und auf Einzelzeichen zugreifen ... 90
6.1.5 Texte vergleichen ... 91
6.1.6 Groß-/Kleinschreibung ... 92
6.1.7 Texte ersetzen ... 93
6.1.8 Weitere Themenbereiche: Klasse StringBuIder ... 94
6.2 Datum und Uhrzeit verarbeiten mit Klasse DateTime und TimeSpan ... 94
6.2.1 DateTime initialisieren ... 95
6.2.2 Vergleichsoperatoren ... 95
6.2.3 Auf Datumselemente zugreifen ... 96
6.2.4 Aktuelles Datum ... 97
6.2.5 Datumselemente verändern ... 97
6.2.6 Datum formatieren ... 97
6.2.7 Weitere Themenbereiche ... 98
6.3 Textdateien lesen mit Klasse StreamReader ... 99
6.3.1 Textdatei auf einmal auslesen ... 100
6.3.2 Textdatei zeilenweise auslesen ... 101
6.4 Textdateien schreiben mit Klasse StreamWriter ... 101
6.5 Dateien- und Verzeichnisoperationen mit Klassen Directory, Path und File ... 102
6.5.1 Prüfen, ob ein Verzeichnis existiert ... 102

6.5.2 Verzeichnis anlegen ... 102
6.5.3 Verzeichnisnamen zusammensetzen ... 102
6.5.4 Dateinamen extrahieren ... 103
6.5.5 Dateien suchen ... 103
6.5.6 Dateien verschieben ... 103
6.5.7 Weitere Themenbereiche ... 103
6.5.8 Entwurf und Realisierung ... 103
6.6 Auflistungen mit Klasse List ... 104
6.6.1 Collection deklarieren ... 104
6.6.2 Objekte hinzufügen ... 104
6.6.3 Auf Objekte zugreifen ... 104
6.6.4 Liste durchlaufen ... 105
6.6.5 Weitere Themenbereiche ... 105
6.6.6 Anwendung ... 105
6.7 Delegaten und Events ... 107
7 Die Verwaltung von Daten mithilfe von Datenbanken planen und umsetzen ... 112
7.1 Datenbanksysteme beschreiben ... 112
7.1.1 Bestandteile ... 112
7.1.2 ANSI/SPARC-Architektur für Datenbanksysteme ... 112
7.1.3 Unterscheidung nach dem Betriebskonzept ... 114
7.1.4 Unterscheidung nach dem Datenmodell ... 116
7.2 Daten und deren Beziehungen mithilfe eines ER-Modells beschreiben ... 118
7.2.1 Entität und Entitätstypen ... 118
7.2.2 Attribute ... 119
7.2.3 Beziehungstyp ... 119
7.2.4 Kardinalitäten ... 120
7.2.5 Beispiele für die Erstellung eines ER-Modells ... 120
7.3 Grundlagen des relationalen Datenmodells erläutern ... 123
7.3.1 Relationales Datenmodell ... 123
7.3.2 Tabellen ... 123
7.3.3 Datensatz ... 123
7.3.4 Schlüssel ... 124
7.4 ER-Modelle in relationale Datenmodelle überführen ... 131
7.5 Die Datenbanksprache SQL anwenden ... 133
7.5.2 Anlegen, Ändern, Löschen von Tabellen ... 135
7.5.3 Einfügen, Ändern, Löschen von Datensätzen ... 137
7.5.4 Abfragen von Daten ... 138
7.5.5 Abfrage mit Bedingungen ... 139
7.5.6 Rechnen in Abfragen ... 140
7.5.7 Sortieren der Ergebnismenge ... 140

7.5.8 Aggregatfunktionen und Gruppen ... 140
7.5.9 Abfragen über mehrere Tabellen ... 141
7.5.10 Datums- und Zeitfunktionen ... 142
7.6 Datenbankabfrage mit C# ... 144
8 Software testen und dokumentieren ... 145
8.1 Den Testprozess beschreiben ... 145
8.1.1 Prozessschritte ... 145
8.1.2 Testverfahren ... 146
8.1.3 Testarten ... 148
8.1.4 Teststufen ... 150
8.1.5 Testdokumentation ... 151
8.2 Testgetriebene Entwicklung erläutern ... 153
8.3 Softwaredokumentationen erstellen ... 154
8.3.1 Grundlegende Anforderungen ... 155
8.3.2 Ersteller ... 155
8.4 Verschiedene Dokumentationsunterlagen unterscheiden ... 156
8.4.1 Entwicklungsdokumentation ... 156
8.4.2 Benutzerdokumentation ... 157
9 Den Prozess der Softwareentwicklung evaluieren ... 159

1 Erste Schritte

1.1 Einleitung

Willkommen bei dem Modul „Software zur Verwaltung von Daten anpassen“.

Folgende Themen aus dem Rahmenlehrplan für alle IT-Berufe werden hier vermittelt.

§ 4 Absatz 2 Nummer 4	Entwickeln, Erstellen und Betreuen von IT-Lösungen	Programmiersprachen, insbesondere prozedurale und objektorientierte Programmiersprachen, unterscheiden
		Algorithmen formulieren und Anwendungen in einer Programmiersprache erstellen
		Datenbankmodelle unterscheiden, Daten organisieren und speichern sowie Abfragen erstellen
§ 4 Absatz 2 Nummer 10	Programmieren von Softwarelösungen	Programmspezifikationen festlegen, Datenmodelle und Strukturen aus fachlichen Anforderungen ableiten sowie Schnittstellen festlegen
		Programmiersprachen auswählen und unterschiedliche Programmiersprachen anwenden

1.2 Motivation

Kleine Tools, Datenbankabfragen erleichtern den Alltag eines jeden ITlers erheblich. Die Automatisierung von monotonen Arbeiten, ermöglicht Freiraum für die Weiterentwicklung des Systems. Dies führt zu einer stabileren und sicheren IT-Landschaft.

1.3 Technische Voraussetzungen

Sie benötigen für dieses Modul folgende Software in deutscher Sprache auf Ihrem Rechner installiert:

- *Visual Studio Community 2022*
- *Notepad++*
- *SQL Server 2017 Express*
- *SQL Management Studio (neueste Version)*

1.4 Erkundung

In diesem Modul und auch darüber hinaus werden Sie mit den unterschiedlichsten Programmiersprachen in Berührung kommen. Angefangen von einfachen Scriptsprachen, über prozedurale Sprachen, bis hin zu objektorientierten Sprachen.

Da jede Sprache ihre Vor- und Nachteile hat, ist es wichtig diese Sprachen vergleichen zu können, und sich passend zum Zweck für eine zu entscheiden.

Führen Sie für folgende Themen jeweils eine Recherche durch.

- *Suchen Sie nach unterschiedlichen Rankings von Programmiersprachen.*
- *Ermitteln Sie die Begriffe maschinennahe Sprachen und Hochsprachen, ordnen Sie die Sprachen aus den Rankings diesen Sprachen etwa zu.*
- *Erforschen Sie was ein Compiler, ein Interpreter und ein Just-in-time-Compiler ist, und ordnen Sie die Programmiersprachen diesen Begriffen zu.*
- *Das .NET war als plattformunabhängig geplant, wurde aber nur für Windows realisiert und sehr eingeschränkt für Linux. Mit dem neuen .NET Core wird jetzt dies wieder aufgegriffen. Eine bekannte Programmiersprache*

Java ist schon lange plattformunabhängig und daher sehr beliebt. Ermitteln Sie die Unterschiede zwischen Java SE, Java EE und Javascript.

1.5 Einführung in .NET und C#

.NET ist eine objektorientierte, sprach- und plattformunabhängige Softwareumgebung zur Entwicklung und Ausführung von Programmen auf Desktop-Computern, im Web und auf Mobiltelefonen.

Zum Entwickeln von Programmen existierten vor der .NET-Technologie verschiedene Sprachen, wie C, C++, Visual Basic 6 usw. Beispielsweise kann ein Programm zum Zeichnen von einfachen Grafiken (Balkendiagramm, Liniendiagramm) in C++ geschrieben sein. Ein anderes Programm, das Aktienkurse darstellen soll und in Visual Basic 6 geschrieben ist, will nun auf die Zeichnungsfunktionen des Grafikprogramms zugreifen, um beispielsweise das Rad nicht neu erfinden zu müssen oder um Kosten für die Eigenentwicklung zu sparen. Der Aufruf von Code innerhalb der gleichen Sprache ist meistens einfach, hingegen ist der Aufruf von Code zwischen verschiedenen Sprachen mit einigen technischen Hürden verbunden.

.NET stellt verschiedene Sprachen wie C# (ausgesprochen: C Sharp), Visual Basic, J# (ausgesprochen: J Sharp) und andere zur Verfügung. Die Sprachunabhängigkeit ermöglicht es, dass ein Programm, das in C#.NET geschrieben ist, sehr einfach Funktionalitäten aus einem Programm verwenden kann, das in Visual Basic.NET geschrieben wurde. Zusätzlich gibt es durch die sogenannte Interoperationalität die Möglichkeiten, Funktionalitäten von Nicht- .NET-Programmen, die in C++ oder Visual Basic geschrieben wurden, zu verwenden.

Programme, die mit Visual C++ oder Visual Basic 6 geschrieben werden, laufen nur unter Windows. Die Portierung auf andere Betriebssysteme, wie Linux, ist sehr aufwendig, da viele Teile neu geschrieben werden müssen. Mit .NET ist es prinzipiell möglich, Programme plattformunabhängig umzusetzen. Praktisch existiert derzeit nur das Monoprojekt für Linux, allerdings nicht in vollem Umfang. Beispielsweise wird die Entwicklung von Webanwendungen nicht unterstützt.
Ein Computer kann nur Anweisungen ausführen, die als Maschinensprache im Speicher liegen. Höhere Sprachen, wie Visual C++ oder Visual Basic 6, sind für den Menschen besser lesbar und verständlich, aber nicht für den Computer. Daher muss ein Compiler den Programmcode in Maschinensprache übersetzen

In .NET übersetzt der Compiler den Programmcode nicht direkt in Maschinencode, sondern in einen Intermediate Language Code. Das ist ein Zwischencode, der nicht direkt am Computer ausgeführt werden kann. Erst dieser Zwischencode ermöglicht eine sprach- und plattformunabhängige Programmierumgebung. Erst zur Laufzeit wird der Zwischencode in Maschinencode übersetzt.

1.6 Einführung in Visual Studio

Wenn Sie ein Programm entwickeln möchten, müssen Sie den Programmcode (also die Befehle) in einem Editor schreiben und diesen vom Compiler in einen Zwischencode übersetzen lassen. Den Programmcode könnten Sie theoretisch auch in einem gewöhnlichen Texteditor schreiben und den Compiler über die Kommandozeile aufrufen.

Für die Programmentwicklung ist es jedoch bequemer, eine integrierte Entwicklungsumgebung (IDE: engl. Integrated Development Environment) zu verwenden. Die Entwicklungsumgebung von Microsoft für .NET heißt Visual Studio. Diese enthält bereits einen Editor für den Programmcode (Code-Editor), Compiler und andere nützliche Tools für ein angenehmes Programmierumfeld.

Microsoft bietet die kostenlose Version „Visual Studio Community Edition" an, die im Internet zum Download angeboten wird. Die Datenbank SQL Server Express Edition kann ebenfalls kostenfrei heruntergeladen werden.

1.7 Erstellen einer Konsolenanwendung

Arbeitsauftrag

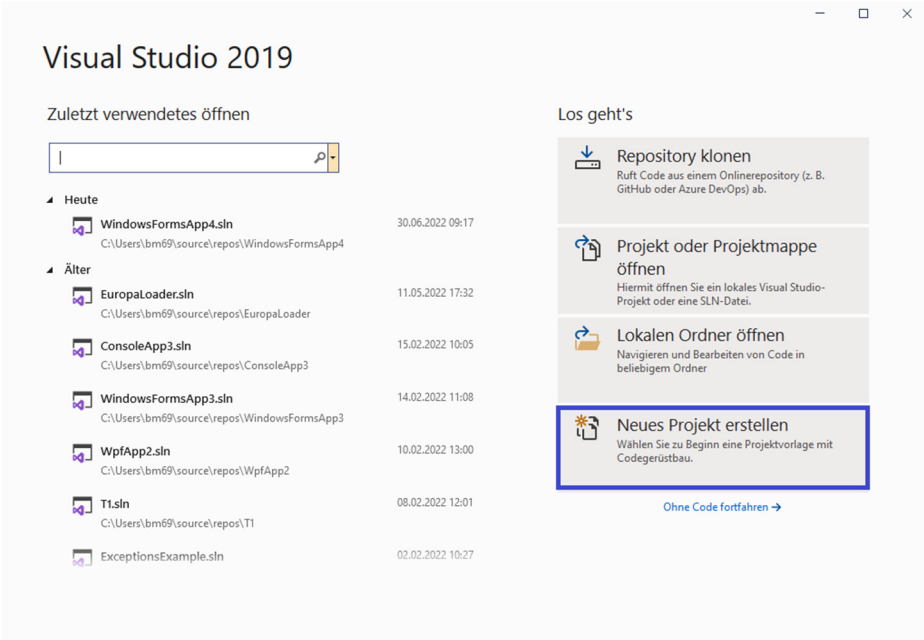
Erstellen Sie eine Konsolenanwendung.

Der PING- oder DIR-Befehl, den man über die Eingabeaufforderung absetzen kann, sind typische Beispiele von Konsolenanwendungen. Konsolenanwendungen (Dateiendung .exe) haben eine textbasierte Bedienungsoberfläche. Sie werden in der Praxis für Programme verwendet, die ohne Benutzerbedienung auskommen sollen. Beispielsweise können damit Programme, wie automatische Datenimporte, auf einem Server von einem Schedule Task zu einer bestimmten Uhrzeit aufgerufen werden, ohne dass ein Mensch die Anwendung starten muss.

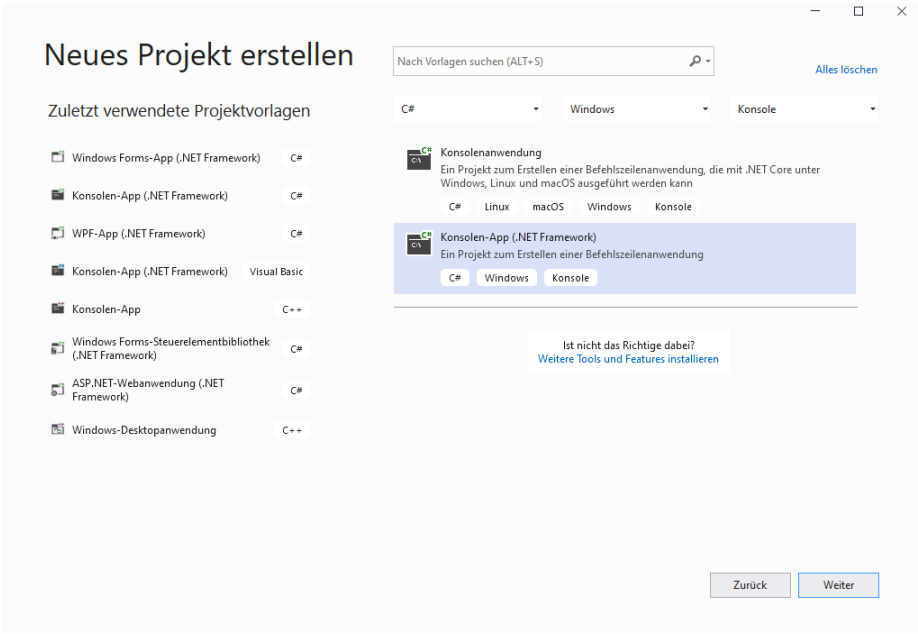
Auch wenn Konsolenanwendungen auf den ersten Blick nicht so spannend erscheinen mögen, sind sie ein guter Einstieg, um die grundlegenden Sprachelemente einer Sprache wie C# zu erlernen. Für einen professionellen Programmierer ist die Beherrschung der Grundlagen sehr wichtig, um später optimale und kreative Lösungen für einen Kunden ausarbeiten zu können.

Wir beginnen mit der Erstellung einer neuen Konsolenanwendung:

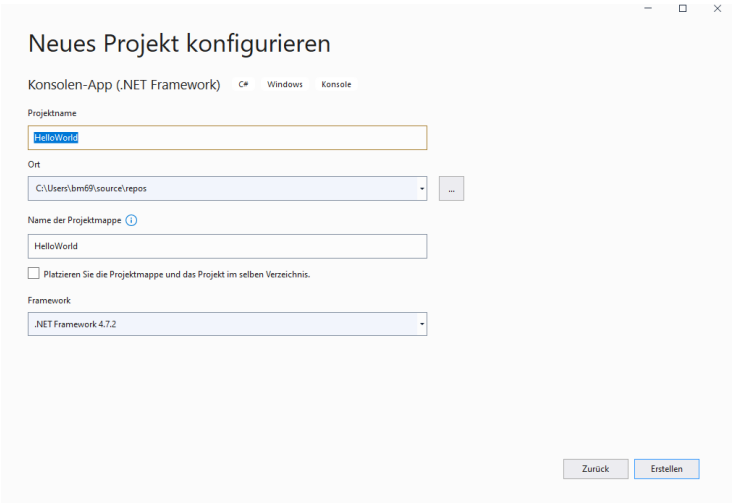
- *Starten Sie die Visual Studio Community Edition.*
- *Visual Studio Community präsentiert sich nach dem Start mit einer Startseite.*
- *Klicken Sie auf dieser Startseite den Link „Neues Projekt erstellen". Es werden alle installierten Vorlagen (Programmarten) aufgelistet.*



- *Wählen Sie den Projekttyp „Konsolenanwendung" („Konsolen App") aus der rechten Liste aus.*
  *Verwenden Sie hier im Modul immer **.NET Framework**, nicht .NET Core.*



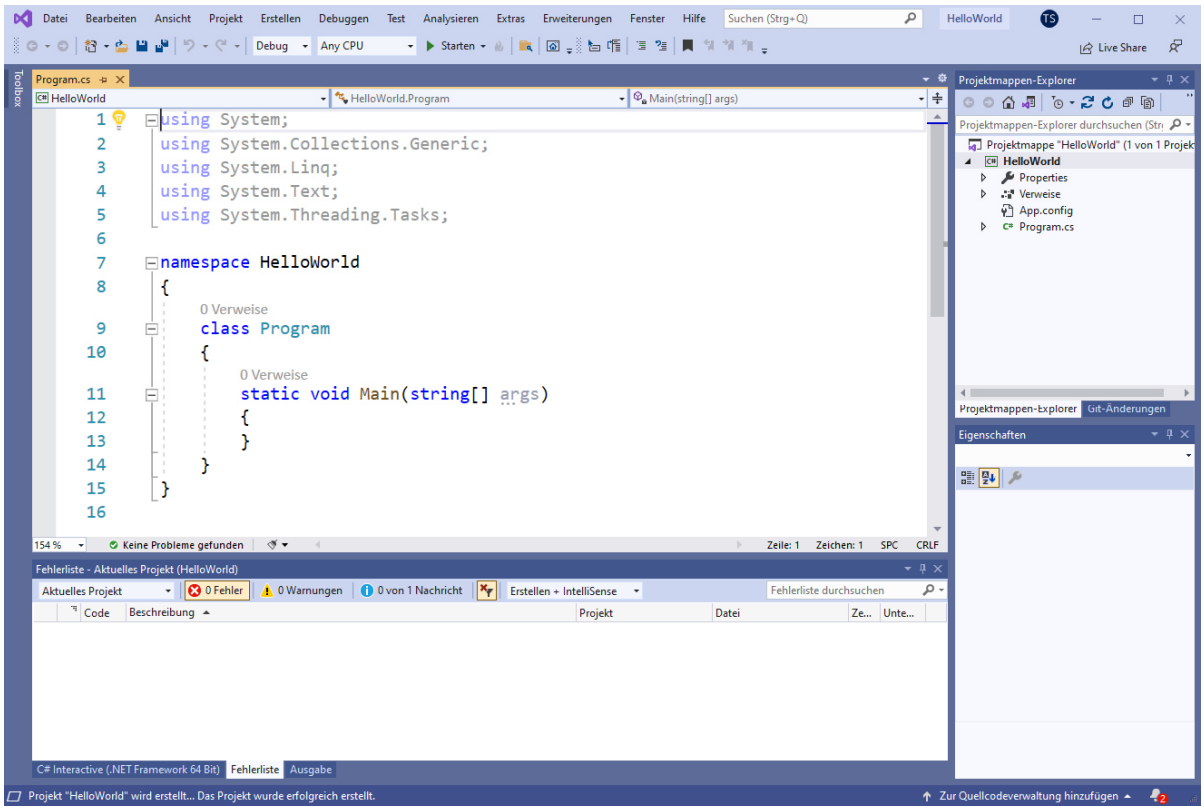
- *Vergeben Sie einen Namen für das Projekt, beispielsweise, HalloWelt".*



- *Bestätigen Sie Ihre Eingaben mit „Erstellen".*

Visual Studio legt nun ein Projekt für Konsolenanwendung an und präsentiert die folgende Entwicklungsumgebung:

Visual Studio ist ein sehr mächtiges Werkzeug, lassen Sie sich nicht von der Vielzahl an Fenstern verwirren. Die wichtigsten Fenster sind:



- *Der Code-Editor auf der linken Seite (Reiter mit der Beschriftung der jeweiligen Datei, hier: Program.cs). Im Code-Editor werden die Anweisungen für das Programm geschrieben.*
- *Der Projektmappen-Explorer (Solution Explorer) auf der rechten Seite enthält alle Elemente, die für das Projekt notwendig sind. Eine Projektmappe kann ein oder mehrere Projekte enthalten. Beispielsweise kann eine Projektmappe eine Konsolenanwendung und eine Klassenbibliothek enthalten. In den ersten Kapiteln werden wir immer nur mit einem Projekt pro Projektmappe arbeiten. In diesem Beispiel hat die Projektmappe den gleichen Namen wie das Projekt selbst: „Projektmappe HaIIoWelt".*

- *Im Eigenschaftsfenster (Properties) in der rechten unteren Ecke werden die Eigenschaften des ausgewählten Projektelements angezeigt. Die Anzahl und Art der Eigenschaften hängt von der Art des Projektelements ab.*
  Auch der beste Programmierer macht Fehler. Um die Fehler leichter zu finden, werden diese in der Fehlerliste (Error List) im Fenster links unten angezeigt (erreichbar über das Menü „View\Error List").

Im Projektmappen-Explorer ist unter anderem das Projektmappenelement Program.cs zu finden. ProjektmappeneIemente mit der Dateiendung .cs enthalten Anweisungen (Code) in der Sprache C#. Program.cs wird standardmäßig von Visual Studio angelegt und auch im Code-Editor geöffnet. Sollte der Code im Code-Editor nicht sichtbar sein, genügt ein Doppelklick auf das Projektmappenelement Program.cs.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace HelloWorld
{
    class Programm
    {
        static void Main(string[] args)
        {
        }
    }
}
```

Der Code in Program.cs besteht aus den folgenden Teilen:

- *Optional: ein oder mehrere Using Direktiven*
- *Beispiel: using System;*
- *Ein Namespace*
- *Beispiel: Namespace HaIIoWelt*
- *Die Definition der Klasse Programm*
- *Beispiel: class Programm*
- *Die Definition der Methode (Prozedur) Main*
- *Beispiel: static void Main(string[] args)*

Für eine erste Betrachtung genügt die Erläuterung der Methode Main. Damit .NET die Anweisungen einer Konsolenanwendung ausführen kann, muss die .NET Common Language Runtime (Laufzeitumgebung) die erste Anweisung finden können. Daher muss jede Konsolenanwendung eine Main-Methode besitzen, in der die erste Anweisung steht. Diese Methode ist also der Einstiegspunkt, an dem das Programm beginnt und endet.

Dies wird kurz an einem einfachen Beispiel demonstriert. Fügen Sie in Program.cs folgenden Code hinzu. Die Anweisung Console.WriteLine schreibt einen beliebigen Text in die Konsole:

```
static void Main(string[] args)
{
    Console.WriteLine("Hallo Welt");
}
```

Bevor Sie den Programmcode ausführen können, müssen Sie das Projekt wie folgt speichern:

- *Rufen Sie das Menü „Datei\Speichern Alle" („File\Save All") auf.*
- *Geben Sie als Projektnamen „HalloWelt" ein.*

Nun muss diese erste C#-Anwendung in Zwischencode wie folgt kompiliert werden, damit die .NET-Laufzeitumgebung sie ausführen kann:

Rufen Sie das Menü „Erstellen\Projektmappe erstellen" („Build\Build Solution") auf oder drücken Sie die Funktionstaste F6.

- *Bei erfolgreicher Kompilierung steht in der Statusleiste (ganz unten) die Meldung „Erstellen erfolgreich"*
- *War das Kompilieren erfolgreich, kann die Anwendung wie folgt gestartet werden:*

- *Rufen Sie das Menü „Debuggen\Debuggen starten" („Debug\Start Debugging“) auf oder drücken Sie die Funktionstaste F5.*

Wenn Sie die Anwendung gestartet haben, werden Sie sich vermutlich wundern, warum das Konsolenfenster nur ganz kurz zu sehen ist. Die Anweisung Console.WriteLine sagt dem Computer nur, dass er den Text „Hallo Welt" ausgeben soll. Nachdem diese Anweisung ausgeführt ist, wird die Anwendung wieder beendet. Grund dafür ist, dass der Computer nicht die Anweisung hatte, solange zu warten, bis der Benutzer beispielsweise eine Taste gedrückt hat.

Soll die Ausgabe „Hallo Welt" sichtbar bleiben, muss der Befehl Console.ReadKey() hinzugefügt werden, damit die Anwendung auf den Tastendruck des Benutzers wartet.

```
static void Main(string[] args)
{
    Console.WriteLine("Hallo Welt");
    Console.ReadKey();
}
```

Kompilieren Sie wieder die Anwendung mit F6 und starten Sie die Anwendung mit F5. Nun sollte am Bildschirm die gewünschte Ausgabe stehen. Erst wenn Sie eine beliebige Taste drücken, wird die Anwendung beendet.

Gratulation: Damit haben Sie Ihre erste Anwendung in C# geschrieben!

2 Einfache Datenverarbeitung mit C#

Dieses Kapitel demonstriert das Grundprinzip der Datenverarbeitung: das EVA-Prinzip.

Hinweis: Folgende Anweisungen im Source Code des Arbeitsauftrags werden ausgelassen:

- *using Direktive*
- *namespace Direktive*
- *Klassendefiniton*

Diese Anweisungen wurden einerseits aus Platzgründen ausgelassen, andererseits fügt Visual Studio diese Befehle automatisch ein und der Programmierer kann den Programmcode aus den Beispielen direkt in die Main-Methode übernehmen.

2.1 Hauptauftrag

Der Projektleiter beauftragt Sie, ein Programm zu entwickeln, das ein Angebot für Hotel-Übernachtungen erstellen soll.

Die Analyse des Hauptauftrages ergibt folgende notwendige Verarbeitungsschritte:

- *Eingabe von:*
  - *Anzahl der Personen*
  - *Aufenthaltsdauer*
- *Der Gesamtpreis wird wie folgt berechnet, wobei der Zimmerpreis 70 € pro Person und Nacht beträgt:*
- *Nettopreis = Zimmerpreis × Aufenthaltsdauer × Anzahl Personen*
- *Gesamtpreis = Nettopreis + Nettopreis × MwSt./100*
- *Am Bildschirm wird Folgendes ausgegeben:*
  - *Gesamtpreis*

Um das Programm zu entwerfen und zu implementieren, sind folgende Sprachelemente notwendig:

- *Variablen*
- *Eingabe und Ausgabe*
- *arithmetische Operationen*

2.2 Variablen

Arbeitsauftrag

Definieren Sie die notwendige Datenstruktur (Variablen) für die Berechnung.

Wie in der Einleitung beschrieben, besteht zwischen Daten und Funktionen ein Zusammenhang: Um einen Gesamtpreis berechnen zu können, werden als Eingabe(daten) der Zimmerpreis, die Aufenthaltsdauer und die Anzahl der Personen benötigt. Diese Daten speichert ein Computer im Arbeitsspeicher. Im Programm werden die Daten über Variablen angesprochen.

2.2.1 Variablen deklarieren

Dazu müssen zunächst ein oder mehrere Variablen deklariert werden:

Syntax

```
Datentyp Variablenname;
```

Ein Variablenname wie Zimmerpreis wird beispielsweise wie folgt definiert:

```
double Zimmerpreis;
int Aufenthaltsdauer;
```

Der Variablenname Zimmerpreis ist vermutlich intuitiv verständlich, darin soll der Zimmerpreis von beispielsweise 50 (€) abgespeichert werden. Wahrscheinlich noch unbekannt dürfte hingegen der Datentyp sein. Der Datentyp ist

notwendig, damit die .NET-Laufzeitumgebung genügend Speicher reserviert, d.h., erst wenn das Programm gestartet wird und .NET auf die obigen Anweisungen stößt, wird der Speicher reserviert und kann in anderen Anweisungen verwendet werden.

Was ist mit genügend Speicher gemeint? Untersuchen wir dazu die benötigten Daten näher. Auch wenn die benötigten Daten auf den ersten Blick nur einfache Zahlen sind, können bei näherer Betrachtung Unterschiede festgestellt werden. So ist die Aufenthaltsdauer eine ganze Zahl, da der Gast nur ganze Tage übernachten kann. Der Zimmerpreis benötigt jedoch einen anderen Datentyp, da dieser Nachkommastellen enthalten kann (z. B. 50,90 €).

Die ganze Zahl benötigt einen anderen Speicherbedarf als eine Zahl mit Nachkommastellen, da diese Zahlen im Speicher unterschiedlich abgebildet werden. Den Speicherbedarf ermittelt die .NET-Laufzeit über den Datentyp. Daher muss eine Variablendeklaration einen Datentyp enthalten. In diesem Zusammenhang wird bei C#.NET auch von einer typsicheren Sprache gesprochen, da es eine Variable ohne Typ per Definition nicht geben darf. (Hinweis: Es gibt auch nicht typsichere Programmiersprachen wie beispielsweise VB6. Hier können einige Variablen jeden beliebigen Typ annehmen, was oft zu Problemen und Bugs führen kann, daher gilt diese Technik als unsauberer Programmierstil.)

2.2.2 Einfache Datentypen

Zunächst werden die **einfachen Datentypen** aufgelistet, hingegen sind die Datentypen wie Datum schon **komplexe Datentypen** und werden später noch detaillierter erklärt. C#.NET unterscheidet bei den einfachen Datentypen zwischen

Zahlen (numerisch)		
Datentyp	Wertebereich	Variablengröße
int	-2.147.483.648 bis 2.147.483.647	Ganze 32-Bit-Zahl mit Vorzeichen
long	-9.223.372.036.854.770.000 bis 9.223.372.036.854.775.807	Ganze 64-Bit-Zahl mit Vorzeichen
byte	0 bis 255	Ganze 8-Bit-Zahl ohne Vorzeichen
sbyte	-128 bis 127	Ganze 8-Bit-Zahl mit Vorzeichen
short	-32.768 bis 32.767	Ganze 16-Bit-Zahl mit Vorzeichen
ushort	0 bis 65.535	Ganze 16-Bit-Zahl ohne Vorzeichen
uint	0 bis 4.294.967.295	Vorzeichenlose 32-Bit-Ganzzahl
ulong	0 bis 18.446.744.073.709.551.615	Vorzeichenlose 64-Bit-Ganzzahl
float	±1,5e-45 bis ±3,4e38	7 Stellen
double	±5,0e-324 bis ±l,7e308	15-16 Stellen
decimal	(-7.9 x 10^28 bis 7.9 × 10^28)	29 Stellen
Text (alphanumerisch)		
char	Genau ein Unicodezeichen	10-Bit-Unicode-Zeichen
string	null oder kein oder beliebige Unicodezeichen	
Sonstige		
bool	wahr oder falsch (true/false)	

2.2.3 Variablen initialisieren

Eine Variable kann optional bei der Deklaration auf einen Anfangswert gesetzt werden. Dadurch versetzt man die Variable in einen definierten Ausgangszustand. Ein definierter Ausgangszustand ist wichtig, damit ein Programm richtige Ergebnisse liefert. C# setzt zwar verschiedene Datentypen, wie Zahlen, automatisch auf einen Anfangswert, jedoch wird empfohlen, Variablen zu initialisieren. Einerseits ist nicht garantiert, dass C# diese Vorgehensweise ändert (und in der nächsten Version nicht mehr die Werte auf einen Anfangswert setzt), andererseits erleichtert eine Initialisierung die Fehlersuche und Lesbarkeit von Code.

Syntax

```
Datentyp Variablenname = Anfangswert;
int MwSt. = 19;
```

2.2.4 Gleichzeitige Deklaration von Variablen

Bis jetzt wurde pro Codezeile eine Variable deklariert (int Aufenthaltsdauer). In einer Zeile können auch mehre Codezeilen deklariert werden, wenn diese vom gleichen Datentyp sind.

Syntax

```
Datentyp Variable 1, Variable 2, Variable n
double Zimmerpreis, Gesamtpreis;
```

Diese Variablen können auch initialisiert werden:

Syntax

```
Datentyp Variable 1 = Wert 1, Variable 2 = Wert 2, Variable n = Wert n
double Zimmerpreis = 0, Gesamtpreis = 0;
```

Ab einer gewissen Anzahl an Variablen wird diese Variante jedoch unübersichtlich. Aus Platzgründen wird in diesem Modul in den Programmbeispielen diese Variante bevorzugt.

Für eine bessere Lesbarkeit und Wartbarkeit wird folgende Variablendeklaration stets bevorzugt.

```
Datentyp Variable 1 = Wert 1
Datentyp Variable 2 = Wert 2
Datentyp Variable n = Wert n

double Zimmerpreis = 0
double Gesamtpreis = 0;
```

2.2.5 Variablen setzen

Durch die Variablendeklaration reserviert die .NET-Laufzeitumgebung auf Basis des Datentyps genügend Speicher im Computer. Erst dadurch kann das Programm Daten (Werte) im lokalen Speicher abspeichern. Damit die Werte in den Speicher kommen, wird einerseits die Variableninitialisierung verwendet, anderseits muss ein Wert auch während des Programmablaufs verändert werden können. Der Wert einer Variable wird in C# mit folgender Syntax verändert (gesetzt).

Syntax

```
Variablenname = Wert;
```

Dieses Code-Snippet setzt die Aufenthaltsdauer auf 2.

```
int Aufenthaltsdauer;
Aufenthaltsdauer = 2;
```

2.2.6 Typkonvertierung

Mit einer Variablendeklaration - wie int Aufenthaltsdauer - legen Sie beispielsweise den Datentyp für die Aufenthaltsdauer auf eine Ganzzahl fest. Der Nettopreis wird durch Multiplikation von Aufenthaltsdauer, Zimmerpreis und Anzahl der Personen berechnet. Der Nettopreis ist jedoch als Gleitkommazahl (double) definiert, da der Zimmerpreis zwei Nachkommastellen haben kann. Wenn wir auf Papier oder im Kopf die Rechnung Aufenthaltsdauer × Zimmerpreis × Anzahlpersonen durchführen würden, wäre es für uns leicht, das Endergebnis auszurechnen. Hingegen ist das für einen Computer nicht ohne Weiteres möglich, da es sich um zwei unterschiedliche Datentypen handelt. Daher muss das Programm eine Konvertierung zwischen den Datentypen int und double durchführen. Das Programm muss dann aus der Formel Nettopreis = Aufenthaltsdauer × Zimmerpreis x Anzahlpersonen implizit oder explizit folgende Formel anwenden:

Nettopreis = konvertiere Aufenthaltsdauer auf double × Zimmerpreis × konvertiere Anzahlpersonen auf double.

Diese Konvertierung zwischen den Datentypen passiert im Hintergrund durch den Compiler implizit (d.h., der Compiler führt in bestimmten Situationen die Konvertierung selbstständig durch), oder der Programmierer implementiert eine Konvertierung (cast genannt) explizit. (Der Vollständigkeit halber sei noch erwähnt, dass es auch

eine benutzerdefinierte Typkonvertierung und eine Typkonvertierung mit Hilfsklassen wie Convert etc. gibt. Dazu später mehr.)

Syntax

```
Variable2 = (Datentyp)Variable1
Variable2 = Variable1
```

```
double Zimmerpreis = 20.5;
int Anzahlpersonen =0, Aufenthaltsdauer = 0;
double Nettopreis;
Anzahlpersonen = 2;
Aufenthaltsdauer = 7;
Nettopreis = Zimmerpreis * (double)Anzahlpersonen * (double)Aufenthaltsdauer;
```

2.2.7 Konstanten

Werte, die im Programm nicht geändert werden, können als Konstanten definiert werden.

Syntax

```
const Datentyp Variablenname = Wert;
const double Pi = 3.24;
```

Das Code-Snippet definiert die (mathematische) Konstante Pi.

2.2.8 Entwurf und Realisierung

Lösung zum Arbeitsauftrag

Laut Hauptauftrag werden folgende Werte benötigt:

- *Berechnung: Nettopreis, Zimmerpreis (nicht veränderbar), Aufenthaltsdauer, Mehrwertsteuer(MwSt.)*
- *Ausgabe: Gesamtpreis*

Daher ergibt sich für die Variablen folgender Programmcode:

```
static void Main(string[] args)
{
    const double Zimmerpreis = 70;
    double Gesamtpreis, Nettopreis;
    int Anzahlpersonen, Aufenthaltsdauer, MwSt.;
}
```

Aufgaben

1. Erstellen Sie eine neue Konsolenanwendung namens „Zinsenberechnung". Definieren Sie die Variablen, um die Zinsen für ein Jahr auszurechnen. Berechnung: Zinsen = Startkapital × (Zinssatz/100)
2. Erstellen Sie eine neue Konsolenanwendung namens „Energieverbrauch".

Definieren Sie die Variablen, um den Energieverbrauch eines elektrischen Geräts für ein Jahr zu berechnen. Berechnung: Verbrauch = Elektrische Leistung × Stunden je Tag × Tage pro Jahr

2.3 Ein- und Ausgabe

Arbeitsauftrag

Schreiben Sie ein Programm, das den Benutzer mit einer Meldung begrüßt („Berechnen eines Angebots"). Danach wird der Benutzer aufgefordert eine Aufenthaltsdauer einzugeben. Nach der Eingabe wird der eingegebene Wert bestätigt.

In der Praxis werden Konsolenanwendungen unter anderem so implementiert, dass sie ohne Benutzerinteraktion auskommen. Beispielsweise wird auf einem Server über einen Schedule Task die Konsolenanwendung zu einer bestimmten Uhrzeit aufgerufen, um Wetterdaten, die sich in einer CSV-Datei auf einem Netzlaufwerk befinden, in eine Datenbank zu importieren.

Es existieren aber auch genügend Konsolenanwendungen, die eine Benutzerinteraktion vorsehen. Beispielsweise dient die Anwendung sqlplus von Oracle zum Absetzen von SQL-Befehlen an eine Oracle-Datenbank.

Da C# eine objektorientierte Sprache ist, wird vom .NET-Framework für die Ein- und Ausgabe (also das Einlesen von Benutzereingaben und die Ausgabe von Text an die Konsole) ein Objekt (genauer: Klasse) namens Konsole zur Verfügung gestellt. Da die objektorientierte Programmierung erst in Kapitel 4 durchgenommen wird, werden hier die Anweisungen nur soweit kurz vorgestellt, dass in den folgenden Unterkapiteln sinnvoll damit gearbeitet werden kann.

2.3.1 Benutzereingabe

Die Benutzereingabe in einer Konsolenanwendung ist gegenüber einer Windows- Anwendung sehr einfach. Für die Benutzereingabe von Daten wird die Anweisung ConsoIe.ReadLine verwendet. Diese fordert den Benutzer auf Daten einzugeben. Die Dateneingabe wird mit der Enter-Taste beendet. ConsoIe.ReadLine liefert eine Variable (Rückgabewert) vom Datentyp string zurück.

```
string EingabeAufenthaltsdauer;
EingabeAufenthaltsdauer = Console.ReadLine();
```

Der Inhalt der eingegebenen Daten hat den Datentyp string. Dieser Datentyp muss in eine Zahl konvertiert werden. Auch diese Funktionalität wird über ein Objekt (Klasse) namens „Convert" vom .NET-Framework bereitgestellt. Die Anweisungen für die Konvertierung lauten:

```
int Ganzzahl;
Ganzzahl = Convert.ToInt32("123");
double Fließkommazahl;
Fließkommazahl = Convert.ToDouble("123,5");
```

Die Teile der Anweisungen Tolnt.32 und ToDoubIe nennt man Methoden. Um diesen Methoden (Funktionen) Daten Weiterreichen zu können, übergibt man diese Daten als sogenannte Parameter. In diesem Fall wird der Methode Tolnt.32 der Parameterwert 123 als Text übergeben und der Methode ToDouble der Wert 123,5 als Text.

2.3.2 Konsolenausgabe

Für den Benutzer einer Anwendung ist es sehr hilfreich, wenn er verschiedene Anweisungen wie „Eingabe Aufenthaltsdauer" oder „Eingabe Anzahl Personen" erhält. Bei unserem ersten Programm „HalloWelt" lautete der Befehl für die Ausgabe des Textes „Hallo Welt" ConsoIe.WriteLine. Die Variante im genannten Programm ist nur eine von mehreren Möglichkeiten, wie Ausgaben auf der Konsole erzeugt werden können. Für dieses Kapitel sind die folgenden Aufrufe wichtig.

Die erste Variante gibt den angegebenen Text am Bildschirm aus und macht einen Zeilenumbruch. D. h., ein weiterer Ausgabebefehl fängt in der nächsten Zeile an, was beispielsweise gut geeignet ist für allgemeine Ausgaben ohne Eingabeaufforderung.

```
Console.WriteLine("Berechnen eines Angebots");
```

Die nächste Variante gibt den angegebenen Text am Bildschirm ohne Zeilenumbruch aus. D.h., ein weiterer Ausgabebefehl fängt gleich nach dem letzten Zeichen an, was beispielsweise gut geeignet ist für allgemeine Ausgaben mit Eingabeaufforderung (Eingabe Aufenthaltsdauer).

```
Console.Write("Eingabe Aufenthaltsdauer:");
```

Die dritte Variante gibt den angegebenen Text und den Wert einer Variablen am Bildschirm aus. Im Text muss ein Platzhalter für den Variablenwert angegeben werden. Der Platzhalter hat das Format {0}.

Als zweiter Parameter wird der Name der Variablen angegeben, die ausgegeben werden soll.

```
Console.WriteLine("Eingabe Kilometerstand: {0}", Variable);
```

Die Ausgabe von Zahlen kann auch formatiert werden. Es gibt viele Formatierungsmöglichkeiten. Die folgende ist ein Beispiel, um eine allgemeine Zahl auf zwei Nachkommastellen zu formatieren. Nach dem Doppelpunkt wird der Formatbezeichner (hier N für allgemeine Zahlen) und 2 für die Anzahl der Nachkommastellen angegeben. Im MSDN (Microsoft Developer Network, eine Online-Hilfe von Microsoft) finden Sie unter dem Stichwort „Standardmäßige Zahlenformatzeichenfolgen" eine Auflistung aller Formatierungsmöglichkeiten.

```
Console.WriteLine("{0:N2}",2.4918);
```

2.3.3 Entwurf und Realisierung

Lösung zum Arbeitsauftrag

Der Programmablauf sieht wie folgt aus:

- *Ausgabe Begrüßung mit Zeilenumbruch*

- *Ausgabe Aufforderung ohne Zeilenumbruch*
- *Einlesen der Aufenthaltsdauer in eine Variable*
- *Ausgabe der Variable*

Damit ergibt sich folgender Programmcode:

```
private static void Main(string[] args)
{
    int Aufenthaltsdauer = 0;
    string EingabeAufenthaltsdauer;

    Console.Write("Eingabe Aufenthaltsdauer:");
    EingabeAufenthaltsdauer = Console.ReadLine();
    Aufenthaltsdauer = Convert.ToInt32(EingabeAufenthaltsdauer);
    Console.WriteLine("Die Aufenthaltsdauer beträgt: {0} ", Aufenthaltsdauer);
    Console.ReadKey();
}
```

Aufgaben

1. Implementieren Sie im Programm „Zinsenberechnung" folgende Ein- und Ausgaben:
   Eingaben : Startkapital, Zinssatz in Prozent
   Ausgabe : Zinsen
2. Implementieren Sie im Programm „Energieverbrauch" folgende Ein- und Ausgaben:
   Eingaben : elektrische Leistung, Stunden je Tag, Tage pro Jahr.
   Ausgabe : elektrische Leistung

2.4 Arithmetische Operation

Arbeitsauftrag

Berechnen Sie den Gesamtpreis eines Angebots wie folgt:

- *Nettopreis = Zimmerpreis × Aufenthaltsdauer*
- *Gesamtpreis = Nettopreis + Nettopreis × MwSt./100*

Ein Computer wie Konrad Zuse Z3 oder ENIAC aus den 1940er-Jahren wurde ursprünglich erfunden, um automatisch verschiedene Berechnungen durchzuführen. Die Reihenfolge der Berechnungen war nicht fix, sondern konnte programmiert werden, sodass sie den jeweiligen Anforderungen angepasst werden konnte. Heutzutage haben Computer einen wesentlich umfangreicheren Befehlssatz. Operatoren sind ein wichtiger Teil des Sprachumfangs einer Sprache.

Bezeichnung	Operator
Addition	+
Subtraktion	-
Multiplikation	*
Division	/
Modulo	%

Als arithmetische Operatoren werden die Grundrechenarten Addition, Subtraktion, Multiplikation und Division sowie Modulo bezeichnet.

Das Modulo liefert den Rest einer Division. Beispielsweise ist der Rest einer Division von 10 durch 3 die Zahl 1.

int Ergebnis = 0, Rest;

Ergebnis =5+10;

Rest = 10 % 3;

Lösung zum Arbeitsauftrag

Die Berechnungsvorschrift aus dem Hauptauftrag zum Ermitteln des Gesamtpreises kann nun direkt in den Programmcode übernommen werden:

Quellcode finden Sie auch in den Lösungsdateien.

```
private static void Main(string[] args)
{
    const double Zimmerpreis = 20.5;
    double Gesamtpreis, Nettopreis;
    int Anzahlpersonen = 2, Aufenthaltsdauer = 7;
```

```
    int MwSt. = 7;
    Console.WriteLine("Berechnen eines Angebots");
    Nettopreis = Zimmerpreis * Aufenthaltsdauer * Anzahlpersonen;
    Gesamtpreis = Nettopreis + Nettopreis * MwSt. / 100;
    Console.WriteLine("Der Gesamtpreis beträgt: {0}", Gesamtpreis);
    Console.ReadKey();
}
```

Aufgaben

1. Implementieren Sie im Programm „Zinsenberechnung" die Berechnung für die Zinsen:

Damit ist der Ablauf im Programm wie folgt:

- *Eingaben: Startkapital, Zinssatz in Prozent*
- *Berechnung: Zinsen = Startkapital × (Zinssatz/100)*
- *Ausgabe: Zinsen*

2. Implementieren Sie im Programm „Energieverbrauch" die Berechnung für den Energieverbrauch:

Damit ist der Ablauf im Programm wie folgt:

- *Eingaben: elektrische Leistung, Stunden je Tag, Tage pro Jahr*
- *Berechnung: Verbrauch = elektrische Leistung × Stunden je Tag × Tage pro Jahr*
- *Ausgabe: Stromverbrauch*

2.5 Weiterführende Themenbereiche

Auch der beste Programmierer macht Fehler. Visual Studio stellt dafür verschiedene Funktionalitäten zur Verfügung. Mehr Informationen dazu finden Sie in Kapitel 5.

2.5.1 Weitere mathematische Funktionen

Neben den Grundrechenarten bietet .NET auch ein paar grundlegende mathematische Funktionen wie Wurzelberechnung an. Hier sehen Sie eine kurze Auflistung. Der Aufruf erfolgt ähnlich wie bei der Funktion Convert.ToXXX. Der mathematischen Funktion werden Parameter übergeben und die Funktion liefert einen Wert zurück.

Funktionsname	Bedeutung	Beispiel
Math.Sqrt(Wert)	Wurzel	double x; x = Math.Sqrt(4); // x = 2
Math.Truncate(Wert)	Rundet ab auf Ganzzahl	double X; x = Math.Truncate(4,9); // x = 4
Math.Sin(Wert)	Sinusfunktion	double x; x = Math.Sin(90) ;
Math.Pow(x , y)	Potenziert eine Zahl x mit dem Exponenten y	double z; Z = Math.Pow(2,3); // z = 2 hoch 3

Aufgaben

1. Entwerfen und implementieren Sie ein Programm, das den Windchill (gefühlte Temperatur) berechnet. Verwenden Sie die Funktion System.Math.Pow, um beispielsweise v0,16 zu berechnen.
   - *Eingabe: Temperatur T in C°, Windgeschwindigkeit v in km/h*
   - *Berechnung: WCT = 13,12 + 0,6215 ×T-11,37 x (v ^ 0,16) + 0,3965 ×T× (v ^ 0,16)*
   - *Ausgabe: Windchill-Temperatur (WCT)*

2. In GPS-Geräten und GPS-Softwareprodukten können neben einer Adresse oft auch GPS- Koordinaten eingegeben werden. Für die Eingabe von GPS-Koordinaten stehen verschiedene Formate zur Verfügung: Grad und Dezimalminuten (N48° 10.94017 El 6° 21.62083), Grad, Minuten und Sekunden (N 48°10'56,41' O 16°2Γ37,25")

sowie Dezimalgrad.
Um von Dezimalminuten (hier: 10,94017) in Sekunden umzurechnen, wird der Wert nach dem Komma (hier: 0,94017) mit dem Wert 60 multipliziert (ergibt 56,41).

2.5.2 Literale

Literale sind Kennzeichnungen von Datentypen.

12345 ist wie Sie wissen nicht `"12345"`.

Die doppelten Hochkommas sagen dem Compiler, dass es sich um einen string handelt und nicht um eine Integer.

```
int Ganzzahl = 1;
double Gleitkommazahl = 1;
```

In obigen Codesnippet sehen Sie die Deklaration und Initialisierung einer Integer und einer Gleitkommazahl. Was Sie gar nicht mitbekommen ist, dass bei der Initialisierung der Gleitkommazahl ein impliziter Cast durchgeführt wird, da die 1 eine Integer ist und keine double. Dass heißt der Computer muss arbeiten.

Um dies zu verhindern und bei der Initialisierung einer Gleitkommazahl auch eine double zu übergeben, hängt man an die Zahl einfach ein d (oder D) an.

```
double Gleitkommazahl = 1d;
```

Natürlich ginge bei double auch folgende Lösung:

```
double Gleitkommazahl = 1.0;
```

Nachfolgend eine Liste mit Beispielen weiterer Literale.

- ***bool:*** *true oder false*
- ***char:*** *Umschließende einfache Hochkomma*
- ***decimal:*** *M oder m*
- ***double:*** *D, d, oder eine reele Zahl z. Bsp. 3.34*
- ***float:*** *F oder f*
- ***uint:*** *U, u*
- ***long:*** *L, l*
- ***ulong:*** *UL, ul, Ul, uL, LU, lu, Lu, lU*
- ***string:*** *Umschließende doppelte Hochkommas, optional mit führendem @*
- ***null****: null*

Arbeitsauftrag

Erkunden Sie weitere Literale z. Bsp. Für Hexadezimale Zahlen, Binäre Zahlen. Außerdem ermitteln Sie was das @ Zeichen vor dem String bedeutet.

3 Kontrollstrukturen, Datentypen und Operatoren

Dieses Kapitel beschreibt die grundlegenden Sprachelemente, wie Kontrollstrukturen, Datentypen und Operatoren der Sprache C# und aller anderen Programmiersprachen.

3.1 Hauptauftrag

Der Projektleiter beauftragt Sie, ein Programm zu entwickeln, das ein Angebot für Hotelübernachtungen erstellen soll.

Die Analyse des Hauptauftrags ergibt folgende wesentliche Verarbeitungsschritte:

Eingabe von:

- *Aufenthaltsdauer*
- *Anzahl der Personen*
- *Saison (Frühling = 1, Sommer = 2, Herbst = 3, Winter = 4)*
- *Zimmerkategorie (Standard = 1, Komfort = 2 und Suite = 3)*
- *Kundenkategorie (Stammkunde = 1, Firmenkunde = 2, Reisebüro = 3)*
- *Anzahl der Kinder*
- *Alter je Kind (O bis 18)*

Der Gesamtpreis wird wie folgt berechnet:

- *Der Zimmerpreis ist abhängig von Zimmerkategorie, Saison und Anzahl Personen: Nettopreis = Zimmerpreis × Aufenthaltsdauer × Anzahl Personen*
- *Wenn der Gast nur eine Übernachtung bestellt, wird ein Zuschlag berechnet.*
- *Abhängig von der Kundenkategorie werden unterschiedliche Rabatte gewährt.*
- *Für jedes eingegebene Kind werden folgende Rabatte angeboten:*
  - *0 bis 6 Jahre: 100 %*
  - *7 bis 11 Jahre: 70 %*
  - *Sonst 30 %*
- *Gesamtpreis = Nettopreis + Nettopreis × MwSt./100*

Am Bildschirm wird Folgendes ausgegeben:

- *Meldung „Berechnen eines Angebots"*
- *Preisliste je Zimmerkategorie und Saison*
- *Für jede Eingabe wird eine Eingabeaufforderung ausgegeben*
- *Anzahl der Kinder*

Gesamtpreis nur mithilfe von Kontrollstrukturen und den grundlegenden Datentypen, welche im Folgenden beschrieben werden, kann diese Berechnung durchgeführt werden.

3.2 Kontrollstrukturen

Wie in der Einleitung beschrieben, enthält ein Computerprogramm eine oder mehrere Anweisungen, die vom Computer der Reihe nach ausgeführt werden, um dem Benutzer eine bestimmte Funktionalität anzubieten.

Wie im Beispiel von Kapitel 2 zu sehen war, können im einfachsten Fall die Anweisungen der Reihe nach ausgeführt werden, um den Gesamtpreis zu berechnen. Angenommen, für Familien mit Kindern soll ein Rabatt abgezogen werden, wenn das Kind jünger als 7 Jahre ist. Ist das Kind zwischen 7 und 10 Jahren, beträgt der Rabatt 70 %, sonst wird ein anderer Rabatt gewährt. Daher müssen Sie den Programmablauf so steuern, dass abhängig von einer Bedingung (hier: Alter des Kindes) andere Anweisungen (Abziehen eines Rabatts) ausgeführt oder ausgelassen werden. Da Gäste kein, ein oder mehrere Kinder haben können, müssen Sie die Anweisungen für die Rabattberechnung pro Kind wiederholen (0, 1 oder mehrmals).

Um den Programmablauf abhängig von Bedingungen zu steuern oder Anweisungen zu wiederholen, existieren folgende Kontrollstrukturen:

- *lineare Sequenz*
- *Verzweigungen*
  - *einseitige Auswahl*
  - *Zweiseite Auswahl*
  - *mehrseitige Auswahl*
- *Schleifen*
  - *kopfgesteuerte Schleife*
  - *fußgesteuerte Schleife*
  - *Zählschleife*

Diese Kontrollstrukturen ermöglichen es, den Programmablauf eines Programms aus einer oder mehreren Kontrollstrukturen zusammenzusetzen. Sie können so einen lesbaren Source Code implementieren, der ohne Sprünge (GOTO) auskommt. Als es in den Anfängen der Programmierung noch keine Kontrollstrukturen gab, existierte lediglich das GOTO. Hier konnte abhängig von einer Bedingung an eine andere Stelle im Programmcode gesprungen werden. Das GOTO führte aber auch dazu, dass im Programmcode wie wild herumgesprungen wurde, um die Anforderungen im Programm umsetzen zu können. Das verursachte einen unübersichtlichen Code (umgangssprachlich auch Spaghetti-Code genannt). Daher wurde überlegt, wie ein Code ohne GOTO auskommt: Die Kontrollstrukturen sind das Ergebnis dieser Überlegung.

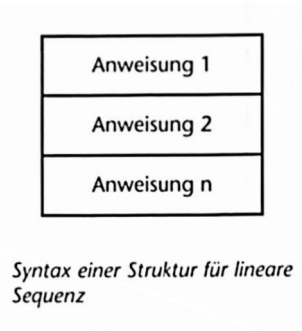
3.3 Struktogramme

Arbeitsauftrag

Zeichnen Sie ein Struktogramm für den Arbeitsauftrag aus Kapitel 2.

Nutzen Sie dazu ein Programm wie z. Bsp.
https://www.struktograf.de

Um zu erkennen, welche Kontrollstrukturen für ein zu implementierendes Programm notwendig sind, müssen Sie zuerst die Anforderungen an ein Programm analysieren und dann einen Algorithmus (Lösungsweg) entwerfen. Die einzelnen Schritte und zeitliche Abfolge eines Algorithmus werden mit einem Programmablaufdiagramm oder Struktogramm grafisch dargestellt. Neben dem Struktogramm gibt es auch noch das Flussdiagramm auch Programmablaufplan oder kurz PAP genannt. Auf dieses wird am Ende dieses Kapitels eingegangen.



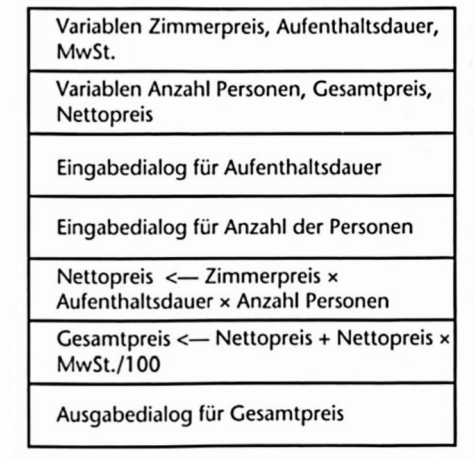
Syntax einer Struktur für lineare Sequenz

Im einfachen Fall der linearen Sequenz wird jede Anweisung in einem Rechteck dargestellt.

3.3.1 Entwurf und Realisierung

Lösung zum Arbeitsauftrag

In der Entwurfsphase für den ersten Anwendungsfall ist das nebenstehende Struktogramm entstanden. An diesem können Sie erkennen, dass selbst eine einfache Anwendung aus mehreren Anweisungen besteht. Daher ist es wichtig, in der Entwurfsphase den Programmablauf sowohl beschreibend als auch grafisch darzustellen. Erst dann sollte mit der Implementierung begonnen werden.



Struktogramm-Beispiel

Aufgaben

1. Zeichnen Sie ein Struktogramm für das Programm zur Berechnung des Windchills.

2. Zeichnen Sie ein Struktogramm für das Programm zur Umrechnung von Dezimalminuten in Sekunden.

3.4 Vergleichsoperatoren

Arbeitsauftrag

Definieren Sie einen Vergleichsoperator, der „true" liefert, wenn die Aufenthaltsdauer einem Tag entspricht.

Vergleichsoperatoren haben die Aufgabe, zwei Werte zu vergleichen. Diese Art der Operatoren können sein: größer (>), kleiner (<), gleich (==) oder eine Kombination: größer gleich (>=), kleiner gleich (<=). Ein weiterer Vergleichsoperator ist die Prüfung auf Ungleichheit (!=).

Arithmetische Operatoren liefern als Ergebnis eine Zahl, wie beispielsweise den Gesamtpreis von 234 (€). Hingegen liefern Vergleichsoperatoren einen booleschen Wert. Dieser kann nur den Wert wahr (true) oder falsch (false) annehmen.

Vergleichsoperator	Prüfung auf
>	größer
<	kleiner
==	Gleichheit
!=	Ungleichheit
>=	größer gleich
<=	kleiner gleich

Syntax

```
Wahrheitswert = Operand1 Operator Operand2;
bool Wahrheitswert = false;
int Operand1 = 1, Operand2 = 3;
Wahrheitswert = Operand1 > Operand2;
Wahrheitswert = Operand1 == Operand2;
Wahrheitswert = Operand1 != Operand2;
```

In C# muss der Gleichheitsoperator mit zwei Gleichheitszeichen geschrieben werden, damit der Compiler zwischen Gleichheit und Zuweisung unterscheiden kann. Mit den Vergleichsoperatoren lassen sich nun verschiedene Bedingungen definieren, die für die folgenden Kontrollstrukturen wie IF, FOR, WHILE und DO/WHILE wichtig sind.

3.4.1 Entwurf und Realisierung

Lösung zum Arbeitsauftrag

Für eine Prüfung auf Gleichheit wird der == Operator verwendet.

```
static void Main(string[] args)
{
    bool IstEinTag = false;
    int AnzahlTag = 0;
    IstEinTag = (AnzahlTag == 1);
}
```

3.5 IF-Abfrage: Einseitige Auswahl

Arbeitsauftrag

Schreiben Sie ein Programm, das einen Aufschlag (z.B. 20 €) berechnet, wenn der Gast nur einen Tag im Hotel übernachtet.

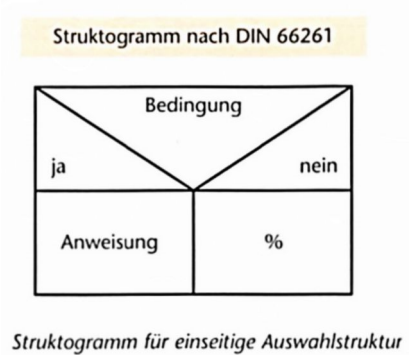
Die Analyse des Arbeitsauftrages ergibt folgende wesentliche Verarbeitungsschritte:

- *Nettobetrag = Zimmerpreis x Aufenthaltsdauer × Anzahl Personen*
- *Wenn Aufenthaltsdauer = 1, dann addiere 20 € zu Nettobetrag*

Der erste Berechnungsschritt wird immer ausgeführt. Der zweite Berechnungsschritt soll nur dann ausgeführt werden, wenn die Bedingung „Aufenthaltsdauer = 1 Tag" zutrifft. Um den Programmablauf so zu steuern, dass eine Anweisung abhängig von einer Bedingung ausgeführt wird, wird die einseitige Auswahl verwendet. Die Syntax und das Struktogramm sehen wie folgt aus:

Syntax

```
If (Bedingung)
    Anweisung;
```



Struktogramm für einseitige Auswahlstruktur

Das Sprachelement ist das Keyword if, gefolgt von der Bedingung in Klammern. In der nächsten Zeile kommt (eingerückt) die gewünschte Anweisung. Diese wird nur dann ausgeführt, wenn die Bedingung zutrifft (true ist). Das Code-Snippet kann damit wie folgt gelesen werden: Wenn die Aufenthaltsdauer 1 Tag beträgt (Bedingung), dann zähle zum Nettopreis 20 € hinzu (Anweisung1).

```
if (Aufenthaltsdauer == 1)
    Nettopreis = Nettopreis + 20;
```

Wenn Sie mehr als eine Anweisung abhängig von einer Bedingung ausführen möchten, müssen Sie geschwungene Klammern verwenden, damit der Compiler erkennt, dass diese Anweisungen zusammengehören.

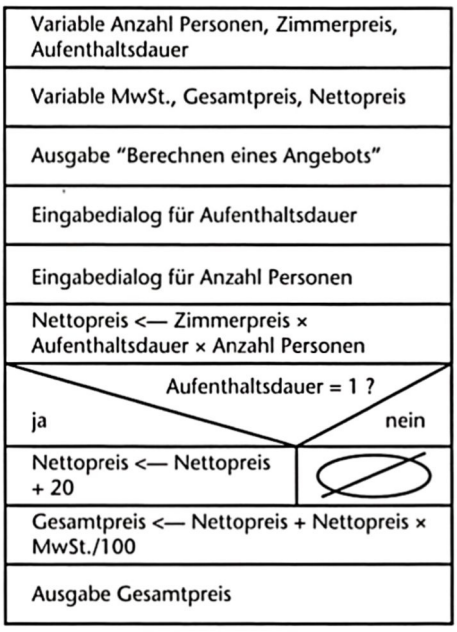
Syntax

```
If (Bedingung)
{
    Anweisung1;
    Anweisung2;
    AnweisungN;
}
```

Mit diesen Angaben lässt sich nun der Arbeitsauftrag wie folgt entwerfen und umsetzen:

3.5.1 Entwurf und Realisierung

Lösung zum Arbeitsauftrag



Struktogramm für die Entnahme der notwendigen Anweisungen

Quellcode siehe Lösungsdateien

Aufgaben

1. Entwerfen und implementieren Sie ein Programm zur Berechnung des Prämienlohns. Ob ein Arbeiter einen Prämienlohn erhält, hängt von seiner Leistung pro Schicht ä 8 Stunden ab. Unterschreitet er die Normalleistung, bekommt er nur den Basislohn.

Stellt der Arbeiter mehr Stück her und überschreitet die Normalleistung (z. B. ISO), wird er mit einem stufenförmigen Prämienmodell honoriert. Für je 10 Stück (eine Stufe) entspricht die Prämie 0,2 Cent. Der Basislohn soll 5 € pro Stunde betragen.

Eingabe: Stückzahl
Ausgabe: Gesamtlohn und Prämie

Im Fall einer Prämie (Leistungsergebnis > Normalleistung) lautet die Berechnungsformel wie folgt: Basislohn + PrämieJeStufe × GANZZAHL(Leistungsergebnis - Normalleistung)/ MengeJeStufe]

2. Entwerfen und implementieren Sie ein Rechnungsprogramm für einen Autovermieter. Wenn ein Kunde den Normaltarif (68 €) auswählt, sind 200 km Fahrtstrecke inkludiert. Erst wenn der Kunde mehr als 200 km fährt, muss er 65 Cent pro weiteren Kilometer zahlen. Die Preise sind exkl. Mehrwertsteuer. Verwenden Sie Konstanten.

   Eingabe: Gefahrene Kilometer
   Ausgabe: Rechnungsbetrag inkl. MwSt., MwSt., als Betrag

3.6 IF-Abfrage: Zweiseitige Auswahl

Arbeitsauftrag

Schreiben Sie ein Programm, das für eine Familie (zwei Erwachsene und ein Kind) ein Angebot erstellt. Kinder von 0 bis 6 Jahren übernachten kostenlos, ab 7 Jahren gibt es einen Rabatt von 70 % auf den Zimmerpreis pro Person und Nacht.

Die Analyse des Arbeitsauftrages ergibt folgende wesentliche Verarbeitungsschritte:

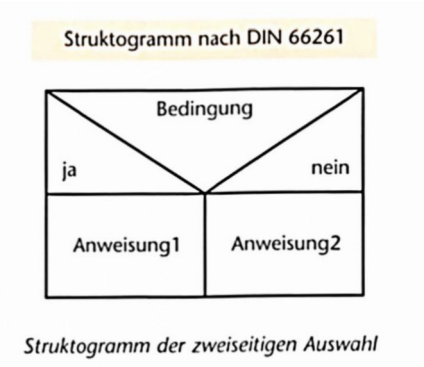
- *Wenn Alter des Kindes kleiner 7, dann beträgt der Rabatt 100 %.*
- *Sonst beträgt der Rabatt 70 %.*

Kinderpreis = Zimmerpreis × Aufenthaltsdauer × (1 - Rabatt/100). Die Berechnung des Kinderpreises ist immer gleich, jedoch wird vorher der Rabatt abhängig vom Alter des Kindes gesetzt. Die Anweisung zum Setzen des Rabatts auf 100% wird nur dann ausgeführt, wenn das Alter des Kindes weniger als 7 Jahre beträgt. In allen anderen Fällen (Kinder ab 7 Jahre) wird der Rabatt auf 70 % gesetzt.

Um den Programmablauf so zu steuern, dass entweder eine Anweisung 1 oder eine Anweisung 2 abhängig von einer Bedingung ausgeführt wird, wird die zweiseitige IF- Abfrage verwendet. Die Syntax und das Struktogramm sehen wie folgt aus:

Syntax

```
if (Bedingung1)
    Anweisung1;
else
    Anweisung2;
```



Struktogramm der zweiseitigen Auswahl

Syntax

```
if (Bedingung1)
{
    AnweisungA1;
    AnweisungA2;
    AnweisungAn;
}
else
{
    AnweisungB1;
    AnweisungB2;
    AnweisungBn;
}
```

Sprachelemente sind die Keywords if und else. Der if-Teil ist gleich wie beim einseitigen if. Dann folgt das Keyword else mit den Anweisungen zwischen den Klammern. Die Anweisungen im if-Teil (auch Zweig genannt) werden nur

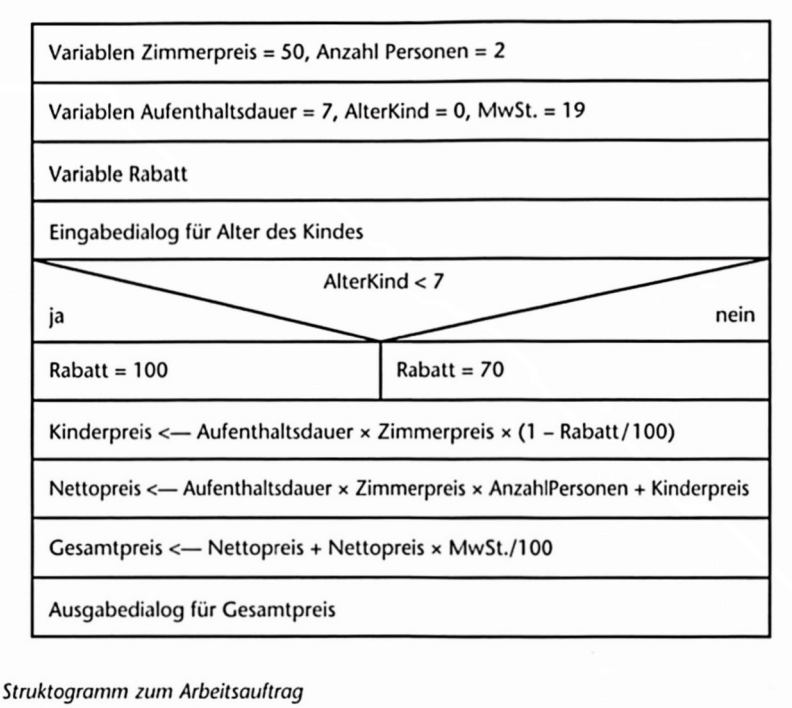
dann ausgeführt, wenn die Bedingung zutrifft (true ist). In allen anderen Fällen werden die Anweisungen im else-Zweig ausgeführt. Das Code-Snippet kann damit wie folgt gelesen werden: Wenn das Alter der Kinder weniger als 7 Jahre beträgt (Bedingung), dann setze den Rabatt auf 100 % (Anweisung1), sonst auf 70 % (Anweisung 2).

```
if (AlterKind < 7)
    Rabatt = 100; // 0 bis 6: 100%
else
    Rabatt = 70;  // sonst 70%
```

Mit diesen neuen Sprachelementen lässt sich der Arbeitsauftrag wie folgt entwerfen und umsetzen:

3.6.1 Entwurf und Realisierung

Lösung zum Arbeitsauftrag



Struktogramm zum Arbeitsauftrag

Quellcode siehe Lösungsdateien!

Aufgaben

1. Ein Hersteller von Schrauben gibt ab 1000 Stück einen Rabatt von 5% und verlangt keine Versandgebühr. Werden weniger als 1000 Stück gekauft, wird kein Rabatt gewährt und es fällt eine Versandgebühr von 5 € an. Eine Schraube kostet 20 Cent. Entwerfen und implementieren Sie ein Programm, das den Gesamtpreis berechnet.
2. Entwerfen und implementieren Sie ein Programm, das den Energieverbrauchskennwert (Schätzwert) für ein Haus berechnet. Dieser Kennwert (kWh/m²) gibt Auskunft darüber, wie optimal ein Haus Energie verbraucht und ob das Haus eventuell saniert werden sollte. Nach Eingabe des Gesamtverbrauchs (Heizöl in l oder Erdgas in m³) wird die Eingabe mit 10 multipliziert, um auf kWh zu kommen. Wird die Heizung auch für das Warmwasser verwendet, werden 1000 kWh mit der Anzahl der Personen im Haushalt multipliziert und dieser Betrag vom zuvor berechneten Gesamtverbrauch abgezogen (1 =Ja, 0 = Nein). Nun wird der tatsächliche Verbrauch durch die Anzahl der Wohnfläche dividiert. Ist die Kennzahl unter 120, hat das Haus einen guten thermischen Wert, bei darüber liegenden Werten sollte über eine Sanierung nachgedacht werden. Tipp: Eventuell sind mehrere Abfragen (hintereinander) notwendig.

3.7 IF-Abfrage: Mehrfachauswahl

Arbeitsauftrag

Schreiben Sie ein Programm, das für Familien (zwei Erwachsene und ein Kind) folgende Rabatte gibt:

- *0 bis 6 Jahre: 100%*

- *7 bis 11 Jahre: 70%*
- *sonst: 30 %*

Die Analyse des Arbeitsauftrages ergibt folgende wesentliche Verarbeitungsschritte:

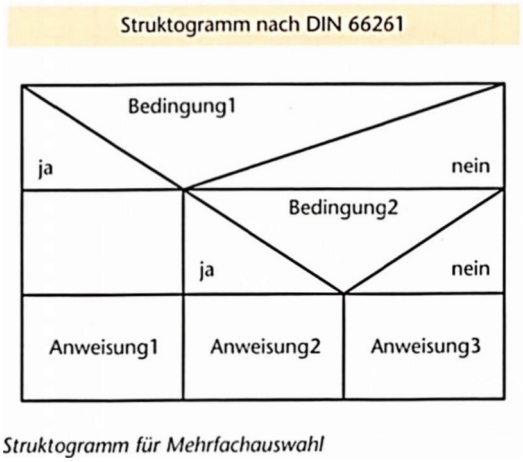
- *wenn Alter Kind kleiner 7, dann Rabatt 100%*
- *sonst*
  - *wenn Alter Kind kleiner gleich 11, dann Rabatt 70%*
  - *sonst Rabatt 30%*
- *• Kinderpreis = Zimmerpreis × Aufenthaltsdauer × (1 - Rabatt/100)*

Die Berechnung des Nettopreises ist immer gleich, jedoch wird vorher der Rabatt abhängig vom Alter des Kindes gesetzt. Die Anweisung zum Setzen des Rabatts auf 100% wir nur dann ausgeführt, wenn das Alter des Kindes weniger als 7 Jahre beträgt. In allen anderen Fällen wird der Rabatt auf 70 % gesetzt, wenn das Alter des Kindes kleiner gleich 11 ist, sonst auf 30 %.

Um den Programmablauf so zu steuern, dass unterschiedliche Anweisungen abhängig von unterschiedlichen Bedingungen ausgeführt werden, wird die Mehrfachauswahl verwendet. Die Syntax und das Struktogramm sehen wie folgt aus:

Syntax

```
if (Bedingung1)
{
    Anweisung1;
}
else
{
    if (Bedingung2)
    {
        Anweisung2;
    }
    Else
    {
        Anweisung3;
    }
}
```



Struktogramm für Mehrfachauswahl

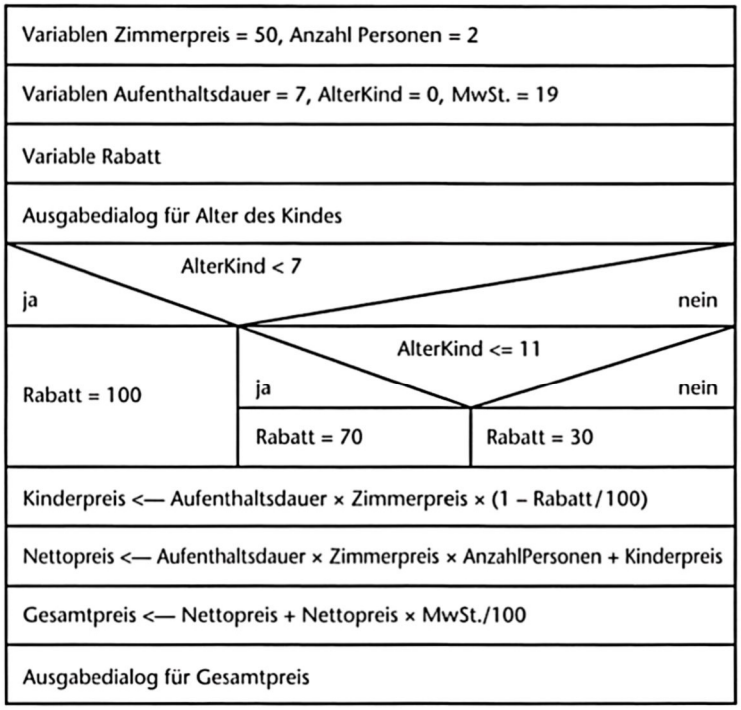
Die Mehrfachauswahl setzt sich aus zwei zweiseitigen IF-Abfragen zusammen, wobei sich die zweite IF-Abfrage im else-Zweig der ersten IF-Abfrage befindet. Diese Art der Zusammensetzung wird auch verschachtelte Kontrollstruktur genannt. Aus Gründen der Übersichtlichkeit wird empfohlen, nicht zu viele Verschachtelungen zu verwenden. Sollte dies der Fall sein, müsste man sich über den Programmentwurf Gedanken machen.

```
if (AlterKind < 7)
{
    Rabatt = 100; // 0 bis 6: 100%
}
else
{
    if (AlterKind <≡ 11)
    {
         Rabatt = 70; // 7 bis 11 Jahre: 70% Rabatt
    }
    else
    {
        Rabatt = 30; // sonst 30%
    }
}
```

Damit lässt sich der Arbeitsauftrag wie folgt entwerfen und umsetzen:

3.7.1 Entwurf und Realisierung

Lösung zum Arbeitsauftrag



Struktogramm zum Arbeitsauftrag

Quellcode siehe Lösungsdateien

Aufgaben

1. Entwerfen und implementieren Sie ein Programm, das den Body-Mass-Index errechnet. Nach Eingabe von Körpergröße h (Eingabe in cm, Umrechnung in m) und Gewicht m (kg) wird der Index wie folgt berechnet: BMI = m/h^2. Ist der Wert unter 18,5, wird von Untergewicht gesprochen, bei einem Wert ab 25 von Übergewicht. Normalgewicht gilt bei allen anderen Werten (also zwischen 18,5 und 25). Ausgabe: Untergewicht, Normalgewicht, Übergewicht.
2. Entwerfen und implementieren Sie ein Programm, das die Treueprämie für Angestellte berechnet. Ein Angestellter, der weniger als drei Jahre angestellt ist, erhält noch keine Prämie. Ab dem dritten und bis zum zehnten Jahr werden pro Jahr 150 € ausgezahlt. Ab dem zehnten Jahr werden zusätzlich zu den 150 € noch 10 € pro Jahr aufgeschlagen.

3.8 Switch

Arbeitsauftrag

Schreiben Sie ein Programm, das für verschiedene Kundenkategorien (Stammkunde = 2%, Firmenkunde = 4%, Reisebüro = 6%) unterschiedliche Rabatte gewährt.

Eingabe: Zimmerkategorie (Stammkunde = 1, Firmenkunde = 2, Reisebüro = 3).

Ausgabe: Gesamtpreis.

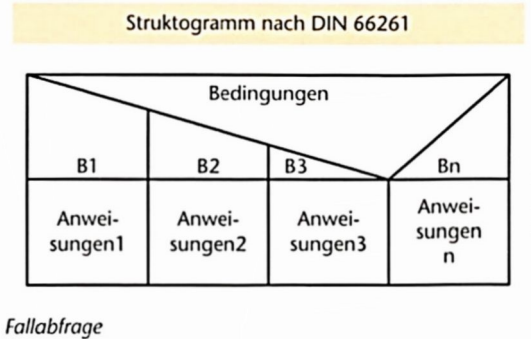
Die Analyse des Arbeitsauftrages ergibt folgende wesentliche Verarbeitungsschritte:

- *Setze Rabatt abhängig von Zimmerkategorie*
  - *Stammkunde = 2%*
  - *Firmenkunde = 4%*
  - *Reisebüro = 6%*

Um den Programmablauf so zu steuern, dass für jeden Fall (Kategorie) eine andere Anweisung ausgeführt wird, wird das Sprachelement Switch verwendet. Die Syntax und das Struktogramm sehen wie folgt aus:

Syntax

```
switch (Bedingung oder Variable)
{
Case 1:
    Anweisung1; break;
Case 2:
    Anweisung2; break;
case 3:
    Anweisung3; break;
default:
    AnweisungN; break;
}
```



Die Switch-Anweisung wird mit dem Keyword switch eingeleitet, gefolgt von einer Bedingung oder Variablen in runden Klammern. Für jede Fallunterscheidung wird (eingerückt) das Keyword case verwendet. Nach dem case wird das Ergebnis der Bedingung oder Variablen angegeben, gefolgt von einem Doppelpunkt. Danach werden die Anweisungen eingegeben und die Fallunterscheidung mit break beendet. Optional können mit dem Keyword default alle anderen Fälle behandelt werden, die nicht in den case-Zweigen behandelt wurden.

```
switch (Zimmerkategorie)
{
case 1: // Stammkunde
    Rabatt = 2;
    break;
case 2:  //Firmenkunde
    Rabatt = 4;
    break;
case 3: // Reisebüro
    Rabatt = 6;
    break;
default:
    break;
}
```

Mit diesem Wissen lässt sich der Arbeitsauftrag entwerfen und umsetzen.

3.8.1 Entwurf und Programm

Lösung zum Arbeitsauftrag

Variablen Zimmerpreis = 0, Gesamtpreis, Nettopreis		
Variablen Anzahl Personen = 2, Aufenthaltsdauer = 0, Zimmerkategorie = 0		
Variablen MwSt. = 19, Rabatt = 0		
Ausgabedialog Begrüßung		
Eingabedialog für Kundenkategorie (1–3)		
Kundenkategorie		
Fall 1	Fall 2	Fall 3
Rabatt = 2	Rabatt = 4	Rabatt = 6
Aufenthaltsdauer <— 7		
Nettopreis <— AnzahlPersonen × Aufenthaltsdauer × Zimmerpreis × (1 – Rabatt/100)		
Gesamtpreis <— Nettopreis + Nettopreis × MwSt./100		
Ausgabedialog für Gesamtpreis		

Switch-Anweisung

Quellcode siehe Lösungsdateien

Aufgaben

1. Entwerfen und implementieren Sie ein Programm, das folgende Winkelfunktionen anbietet:
   - *Cosinus*
   - *Sinus*
   - *Tangens*

   Nach dem Start des Programms wird dem Benutzer ein Menü mit obigen Winkelfunktionen angeboten. Nach Eingabe einer Zahl von 1 bis 3 wird der Winkel in Grad abgefragt. Danach wird die Winkelfunktion entsprechend der Auswahl aufgerufen und das Ergebnis angezeigt.

$$h = \left(q + \left[\frac{(m+1)\cdot 26}{10}\right] + K + \left[\frac{K}{4}\right] + \left[\frac{J}{4}\right] - 2\cdot J\right) \bmod 7$$

2. Entwerfen und implementieren Sie ein Programm, das den Wochentag zu einem Datum nach der Zeller'schen Formel berechnet. Nach Eingabe von Jahr, Monat und Tag (alles als Zahl) wird folgende Formel angewandt:
   h ist der berechnete Wochentag, wobei 1 für Sonntag, 2 für Montag usw. steht.
   q = Tag, K = die letzten beiden Stellen der 4-stelligen Jahreszahl, J = Jahrhundertzahl
   Es gelten folgende Rahmenbedingungen: Wenn der Monat Januar oder Februar ist, gilt für Januar m = 13 und für Februar m = 14, da diese Monate zum Vorjahr gezählt werden. Wenn h negativ ist, wird 7 dazugezählt.

   Ausgabe: Wochentag als Text (Sonntag, Montag etc.)

3.9 Eindimensionale Arrays

Arbeitsauftrag

Schreiben Sie ein Programm, das die Berechnung des Gesamtpreises von der eingegebenen Saison abhängig macht (beispielsweise: Frühling 60 €, Sommer 70 €, Herbst 80 € und Winter 65 €).

- *Eingabe: Saison*
- *Ausgabe: Gesamtpreis*

Die Analyse des Arbeitsauftrages ergibt folgende wesentliche Verarbeitungsschritte:

- *Saisonpreise im Programm abbilden*
- *Saison einlesen*
- *Preis zu Saison ermitteln*
- *Gesamtpreis ermitteln*

3.9.1 Arrays deklarieren

Wie würden Sie die Preisliste aus dem Arbeitsauftrag für eine Saison in einem Programm abbilden? Eine Variable, wie der Zimmerpreis, konnte bisher immer nur einen Wert annehmen. Nach dem bisherigen Wissensstand müsste man für jeden Wert in der Preisliste eine Variable definieren. In C# kann man jedoch auch eine Liste (eindimensionales Array) im Programm abbilden. Diese wird wie folgt deklariert:

Syntax

```
Datentyp[] Preisliste = new Datentyp[AnzahlElemente];
double[] Preisliste = new double[4];
```

Die Variablendeklaration beginnt wie gewohnt mit dem Datentyp. Damit der Compiler (und der Programmierer) erkennt, dass es sich um ein Array handelt, folgen zwei eckige Klammern: eine Klammer auf und eine zu. Dann folgt der Variablenname.

Diese Syntaxvariante deklariert ein Array ohne Anfangswerte (ausgenommen die Anfangswerte, die .NET vergibt). Dazu muss nach dem Gleichheitszeichen ein new angegeben werden, da ein Array ein Objekt ist (dazu später mehr in Kapitel 4). Abschließend wird zwischen den eckigen Klammern die Anzahl der Elemente angegeben (hier: 4).

3.9.2 Arraywerte setzen

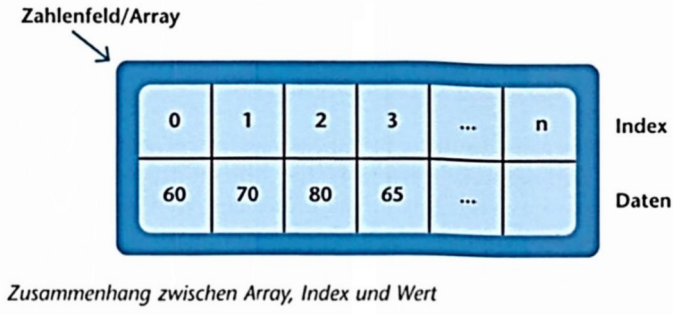
Unabhängig davon, ob das Array mit oder ohne Anfangswerte initialisiert wird, kann der Inhalt eines Arrays wie folgt verändert (oder gesetzt) werden.

Syntax

```
Variablenname[Index] = Wert;
Preisliste[0] = 60; //1.Element in Liste
Preisliste[1] = 70; //2.Element in Liste
Preisliste[2] = 80; //3.Element in Liste
Preisliste[3] = 65; //4.Element in Liste
```

Dazu gibt man zuerst den Variablennamen des Arrays an, gefolgt von einem Index zwischen zwei eckigen Klammern. Der Index in C# beginnt bei 0, das heißt 0 steht für den ersten Eintrag, 1 für den zweiten Eintrag usw.

Nach dem Gleichheitszeichen folgt der Wert, den man einem Arrayeintrag zuweisen möchte.



Zusammenhang zwischen Array, Index und Wert

3.9.3 Arraywerte auslesen

Nachdem das Array definiert wurde, kann auf diese mit folgender Syntax über einen Index zugegriffen werden.

Syntax

```
Wert = Variablenname[Index];
Umsatz = Umsatzliste[0]; // Liefert den 1. Eintrag in der Liste
Umsatz = Umsatzliste[1]; // Liefert den 2. Eintrag in der Liste
```

Den Index gibt man wieder zwischen eckigen Klammern an.

3.9.4 Arrays mit Anfangswerten initialisieren

Wenn schon von Anfang an klar ist, welche Werte ein Array haben wird, können Sie folgende Syntaxvariante verwenden

Syntax

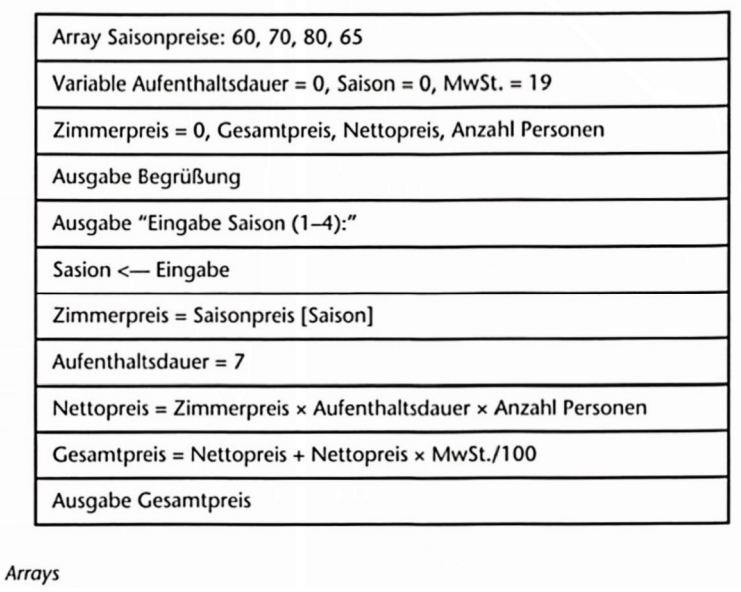
```
Datentyp[]Variablenname = new Datentyp[] {Wert1, Wert2, Wertn};
double[] Umsatzliste = new double[] { 3000, 4503, 2345, 3456 };
```

Diese Syntaxvariante initialisiert schon bei der Deklaration der Variablen das eindimensionale Array mit der Anzahl an Elementen und einer Werteliste, indem man zwischen geschwungenen Klammern die einzelnen Werte angibt.

Unabhängig davon, können Sie trotzdem die Werte eines Arrays wie oben beschrieben verändern.

3.9.5 Entwurf und Realisierung

Lösung zum Arbeitsauftrag



Arrays

Quellcode siehe Lösungsdateien

Aufgaben

1. Entwerfen und implementieren Sie ein Programmmodul für eine Lohnverrechnung, dass nach Eingabe der Vordienstzeiten (in Jahren) das Mindestgehalt ausgibt. Das Programm kann zehn Gehaltsstufen verwalten. Werden mehr als zehn Vordienstjahre eingegeben, beträgt die Vordienstzeit 10 Jahre. 0 Jahre Erfahrung = 1 Jahr. Negative Werte sind nicht erlaubt.
2. Entwerfen und implementieren Sie ein Programm, das die Versandkosten einer Warengruppe ermittelt. In einem Array werden die Versandkosten von zehn Warengruppen gespeichert. Nach Eingabe der Warengruppe (1 bis 10) werden die Versandkosten ausgegeben. Ausgabe: „Die Versandkosten für Warengruppe X betragen Y".

3.10 FOR-Schleife

Arbeitsauftrag

Schreiben Sie ein Programm, das den Gesamtpreis abhängig von der Saison berechnet. Unmittelbar nach dem Programmstart wird eine Preisliste ausgegeben (beispielsweise: Frühling 60 €, Sommer 70 €, Herbst 80 € und Winter 65 €).

- *Ausgabe: Preisliste*
- *Eingabe: Saison*
- *Ausgabe: Gesamtpreis*

Die Analyse des Arbeitsauftrages ergibt folgende wesentliche Verarbeitungsschritte:

- *Preisliste im Programm abbilden*
- *jeden Preis ausgeben*
- *Berechnung und Ausgabe des Gesamtpreises*

Wie könnten Sie mit den bisher gelernten Anweisungen jeden einzelnen Preis abbilden und ausgeben? Sie müssen folgende Anweisungen je Array-Element aufrufen:

```
double[] Saisonpreise = new double[4] { 60, 70, 80, 65 };
Console.WriteLine("Saison 1  {0}:   Euro", Saisonpreise[0]);     //     Saison 1:     6€
Console.WriteLine("Saison 2  {0}:   Euro", Saisonpreise[1]);     //     Saison 2:     70€
Console.WriteLine("Saison 3  {0}:   Euro", Saisonpreise[2]);     //     Saison 3:     80€
Console.WriteLine("Saison 4(0}:   Euro", Saisonpreise[3]);     //     Saison 4:     65€
```

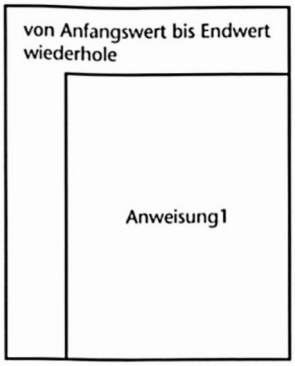
Syntax

Syntax für eine Anweisung:

```
for (Anfangswert; Abbruchbedingung; Schritt) Anweisung1;
```

Syntax für mehrere Anweisungen:

```
for (Anfangswert; Abbruchbedingung; Schritt)
{
Anweisung1;
Anweisung2;
Anweisung3;
}
```



Wiederholung durch Anweisung

```
double[] Saisonpreise = new double[] { 60, 70, 80, 65 };
for (int i = 0; i < 4; i++)
{
Console.WriteLine("Saisonpreis {0} ", Saisonpreise[i]);
}
```

Die FOR-Schleife zählt eine Variable (hier: i) von einem Anfangswert bis zu einem Endwert bei jedem Schleifendurchgang um einen definierten Schritt hoch, solange bis die Abbruchbedingung erfüllt ist. Der Anfangswert der Variablen wird in diesem Fall auf 0 gesetzt. Die Abbruchbedingung gibt an, wann die Schleife abbrechen soll. In diesem Fall soll die Schleife die Anweisungen nach der fünften Wiederholung abbrechen. Da der Index bei 0 beginnt, ist die Abbruchbedingung i < 4 (i kleiner 4) und nicht i < 5.

Der Schritt gibt an, um wie viel die Variable i erhöht werden soll. Die Anweisung i++ ist eine Abkürzung für das Erhöhen der Variablen i um den Wert 1. Innerhalb der FOR- Schleife wird mit der Zählvariablen i auf den Index der Preisliste zugegriffen und mit Console.WriteLine ausgegeben.

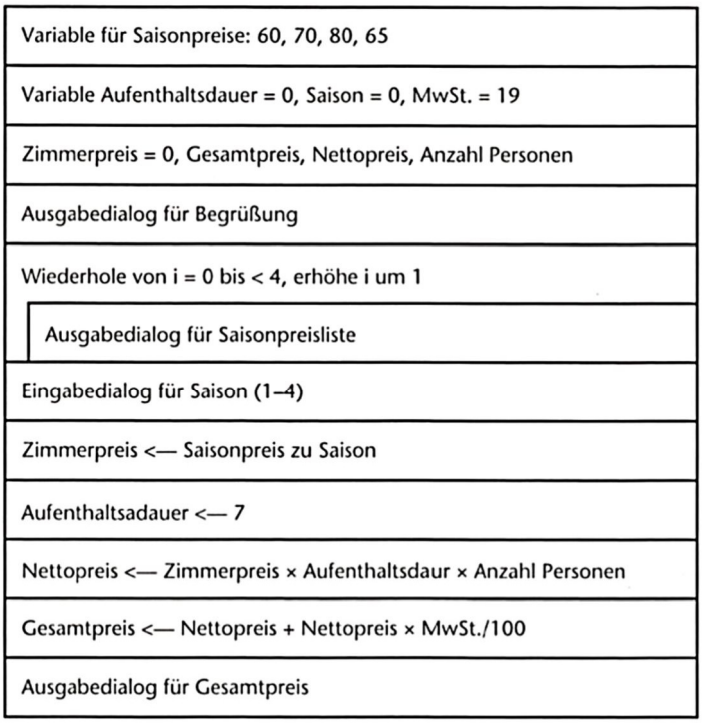
ConsoleWriteLine kann nicht nur einen Variablenwert ausgeben, sondern beliebig viele. Dazu muss man im ersten Parameter (für die Textausgabe) für jeden auszugebenden Variablenwert einen Platzhalter {x} verwenden, der durchnummeriert wird. Durch den Platzhalter {0} erkennt der ConsoIe.WriteLine-Befehl, dass der erste Parameter verwendet werden soll (hier: i). Dann folgt die zweite Variable (hier: Saisonpreis) für den zweiten Platzhalter {1}, usw.

Syntax

```
Console.WriteLine(„{0} {1}   {n}", Variable1, Variable2,..,Variable n);
Console.WriteLine("Variable 1 {0} und Variable 2 {1} ", i, Saisonpreis);
```

3.10.1 Entwurf und Realisierung

Lösung zum Arbeitsauftrag



Struktogramm zum Arbeitsauftrag

Quellcode siehe Lösungsdateien

Aufgaben

1. Entwerfen und implementieren Sie ein Programm, das von einer Netto-Preisliste (Array) sowohl den Nettopreis als auch den Bruttopreis (19 % MwSt.) ausgibt. Die Preisliste hat mindestens drei Einträge.
2. Entwerfen und implementieren Sie ein Programm, das den Bremsweg von Autos für verschiedene Geschwindigkeiten berechnet. D. h., der Benutzer muss nach dem Start des Programms zuerst die Anzahl der Geschwindigkeiten eingeben (1 bis n). Danach wird für jeden zu berechnenden Bremsweg die Geschwindigkeit abgefragt. Die Eingabe der Daten und die Berechnung der Daten erfolgt hintereinander.

   Formel für den Bremsweg: $sb = (v_0 \times v_0)/(2 \times a)$

   Bremsbeschleunigung a ist beispielsweise 8 m/s^2 bei trockener Fahrbahn.

   Beispielsweise:

   Eingabe Anzahl Geschwindigkeiten: 2 (kann 1 bis n sein)

   Eingabe 1: Geschwindigkeit 1: 30

   Eingabe 2: Geschwindigkeit 2: 60

   Ausgabe:

   Bremsweg bei Geschwindigkeit 30 km/h: xxx m

   Bremsweg bei Geschwindigkeit 60 km/h: xxx m

   Formatieren Sie die Ausgabe auf zwei Nachkommastellen.

Hinweis: Für die Formatierung der Bildschirmausgabe in der Konsole (und auch Dateien) existieren eigene Zeichen. Beispielsweise steht \n für neue Zeile und \t für Tabulator. Damit n und t nicht als Text interpretiert werden, wird ein sogenanntes Escape Zeichen (Backslash) vorangestellt.

3.11 Parameter der Main-Methode

Arbeitsauftrag

Implementieren Sie einen Konsolenparameter, mit dem von außen die Berechnungsmethode „pro Person" oder „pro Nacht" gesteuert sowie der Zimmerpreis von außen übergeben werden kann. Erstellen Sie anschließend einen Batch-Aufruf.

Die Analyse des Arbeitsauftrages ergibt folgende wesentliche Verarbeitungsschritte:

- *Übergabe des Zimmerpreises per Konsolenparameter*
- *Auswertung des Konsolenparameters*
- *Berechnung und Ausgabe des Gesamtpreises*

Wie könnten die Konsolenparameter in das Programm kommen? Was fällt Ihnen auf, wenn Sie die Main-Methode betrachten? Was könnte der Parameter args für ein Datentyp sein und wofür könnte er gut sein?

```
static void Main(string[] args)
```

Der Parameter args ist ein eindimensionales String-Array, also eine Liste von Texten. Dieser Parameter ist also dafür vorgesehen, dass einer Konsolenanwendung Parameter übergeben und im Programm ausgewertet werden können.

```
private static void Main(string[] args)
{
    Console.WriteLine(args[0]);
}
```

Wenn Sie schon in einer Konsole einen Befehl wie DIR zum Auflisten der Dateien oder Verzeichnisse eingegeben haben, haben Sie vermutlich bereits mit Konsolenparametern gearbeitet. So können Sie diesem Befehl entweder keinen oder mehrere Parameter übergeben. DIR liefert alle Dateien und Verzeichnisse, DIR *.txt alle Textdateien. Mit Konsolenparametern kann man also Daten übergeben und/oder den Programmablauf von außen steuern.

Falls die Anzahl der Parameter variabel ist, verwendet man den Aufruf args.GetLength(). GetLength ist eine Methode, die zum Objekt (Klasse) Array gehört, dazu später mehr in Kapitel 4.

Syntax

```
args.GetLength();
int Anzahl;
Anzahl = args.GetLength(0);
```

In der Visual Studio-Entwicklungsumgebung können Sie die Konsolenparameter in den Projekteigenschaften im Reiter Debug übergeben. So können Sie die Konsolenanwendung auch aus der Entwicklungsumgebung heraus starten.

Außerhalb der Entwicklungsumgebung wird die Konsolenanwendung über die Kommandozeile gestartet. Geben Sie dazu in Windows unter Start \Ausführen die drei Buchstaben cmd ein. Wechseln Sie mit dem Befehl CD (Change Directory) in das Ausführungsverzeichnis und starten Sie die EXE-Datei per Kommandozeile.

3.11.1 Entwurf und Realisierung

Lösung zum Arbeitsauftrag

In den Projekteigenschaften können Sie im Reiter Debug zum Testen die Konsolenparameter eingeben.

Erstellen Sie nach der Implementierungs- und Testphase eine Batch-Datei, welche das Konsolenprogramm mit der Berechnungsmethode (P oder N) und einem Zimmerpreis aufruft. Verlinken Sie dieses Konsolenprogramm auf den Desktop.

Ein Programm kann mit der Anweisung return vorzeitig verlassen werden. Beispielsweise wenn Werte oder Bedingungen nicht erfüllt sind, wie die Anzahl der Konsolenparameter, die dann 0 ist.

Quellcode siehe Lösungsdateien

Aufgaben

1. Entwerfen und implementieren Sie ein Programm, das die durchschnittliche Temperatur berechnet. Es kann eine beliebige Anzahl von Temperaturen (z. B. Tagesmittelwert eines Tages) als Aufrufparameter übergeben werden.

3.12 WHILE-Schleifen

Arbeitsauftrag

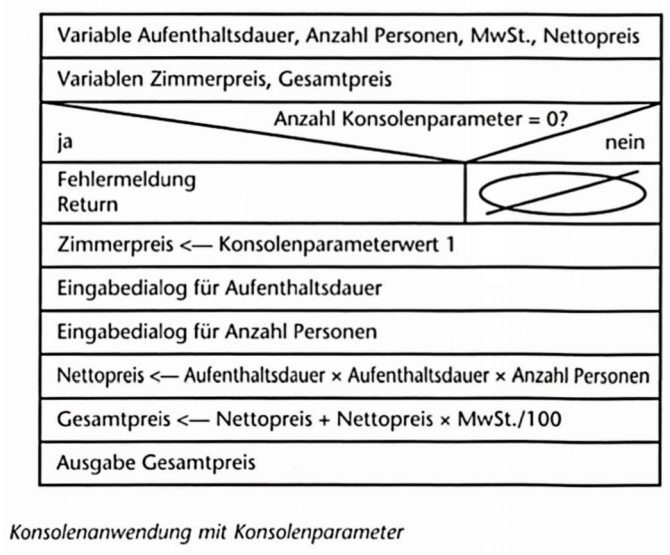
Schreiben Sie ein Programm, das für Gäste mit keinen oder maximal zwei Kindern einen Gesamtpreis erstellt. Die Rabattstaffel beträgt: 0 bis 6 Jahre 100 %, sonst 50 %.

- *Eingabe: Anzahl Kinder, Alter pro Kind*
- *Ausgabe: Gesamtpreis*

Die Analyse des Arbeitsauftrages ergibt folgende wesentliche Verarbeitungsschritte:

- *Eingabe Anzahl Kinder*
- *Berechnung Kinderpreis für jedes Kind (0 bis 2)*
    - *Eingabe Alter Kind*
    - *Setzen des Rabatts*
    - *Zimmerpreis × Rabatt/100*
- *Berechnung und Ausgabe des Gesamtpreises*

Die Berechnung des Kinderpreises wird also für jedes Kind wiederholt. Da jedoch die Anzahl der Kinder von 0 bis 2 variieren kann, ist die FOR-Schleife nicht die ideale Kontrollstruktur, denn dabei ist die Anzahl der Wiederholungen bekannt. Ist die Wiederholung jedoch abhängig von einer Bedingung, oder ist die Anzahl der Wiederholungen unbekannt, sollte die WHILE-Schleife verwendet werden. Die Syntax und das Struktogramm sehen wie folgt aus:

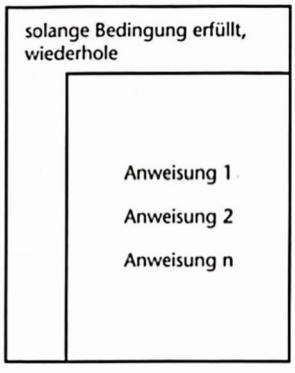


Konsolenanwendung mit Konsolenparameter

Syntax

```
while (Bedingung)
{
    Anweisung 1;
    Anweisung 2:
    Anweisung n;
}
```

Angewandt auf den Arbeitsauftrag hat die WHILE- Kontrollstruktur folgende Bedeutung: Solange noch nicht die Kinderpreise für alle Kinder berechnet wurden, berechne den Rabatt für die Kinderpreise. In diesem Fall besteht die Herausforderung darin, die Bedingung der Schleife zu definieren. Hier kann



Fallabfrage mit der while-Schleife

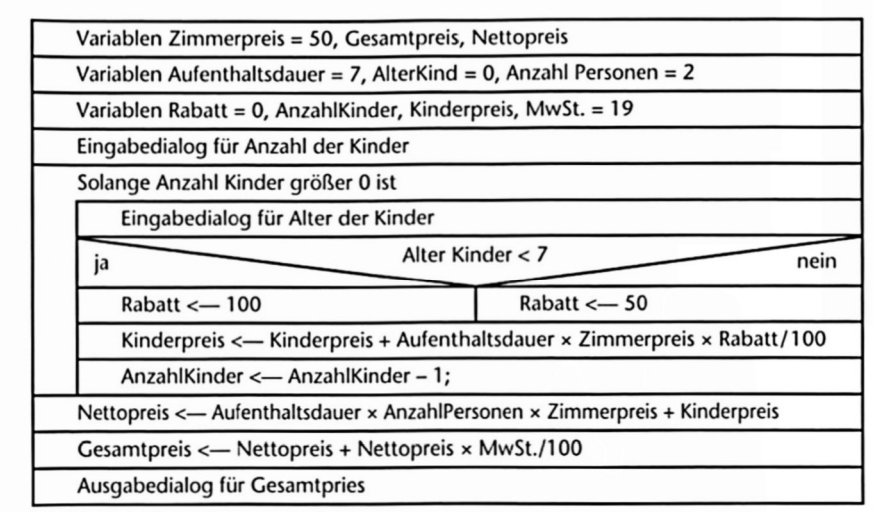
man sich mit der Anzahl der Kinder helfen, indem man innerhalb der Schleife die Variable AnzahlKinder um 1 erniedrigt. Damit kann man die Bedingung „AnzahlKinder > 0" definieren. Somit lautet die Kontrollstruktur für diesen Fall: Solange die Variable AnzahlKinder größer 0 ist, berechne den Rabatt für den Kinderpreis.

```
EingabeAnzahlKinder = Console.ReadLine();
AnzahlKinder = Convert.ToInt32(EingabeAnzahlKinder);
While (AnzahlKinder > 0)
{
    // Eingabe Alter des Kindes
    // Berechnen des Rabatts
    AnzahlKinder = AnzahlKinder - 1;
}
```

Damit lässt sich der Arbeitsauftrag wie folgt entwerfen und realisieren:

3.12.1 Entwurf und Programm

Lösung zum Arbeitsauftrag



Struktogramm zum Arbeitsauftrag

Quellcode siehe Lösungsdateien

Aufgaben

1. Entwerfen und implementieren Sie ein Programm, das den größten gemeinsamen Teiler von zwei ganzzahligen Werten nach dem klassischen Euklid'schen Algorithmus ermittelt.
   Eingabe: Zahl 1, Zahl 2
   Ausgabe: ggT
2. Entwerfen und implementieren Sie ein Programm, welches eine digitale Zahl in eine binäre Zahl umwandelt. Dabei wird eine Zahl solange durch 2 dividiert, bis 0 herauskommt. Der Rest jeder einzelnen Division entspricht dabei einer Binärzahlstelle.
   Eingabe: ganze Zahl
   Ausgabe: digitale Zahl

3.13 Exkurs: Klasse string

Der Datentyp (die Klasse) string wird in Kapitel 6 detailliert beschrieben. In diesem Kapitel wird nur kurz auf die Verwendung eingegangen. Text wird in C# (wie bei der Anweisung Console.WriteLine) zwischen doppelte Anführungszeichen gesetzt. Damit kann man einen string -Datentyp wie folgt setzen.

```
string Ort = "Berlin";
string Strasse;
char Zeichen;
Strasse = "Musterstrasse";
```

3.14 DO/While-Schleife

Arbeitsauftrag

Schreiben Sie ein Programm, mit dem ein Angebot solange berechnet wird, bis der Benutzer den Abbruch mit der Taste n bestätigt.

- *Eingabe: Aufenthaltsdauer*
- *Ausgabe: Gesamtpreis*
- *Eingabe: Eingabedialog für Abbruch*

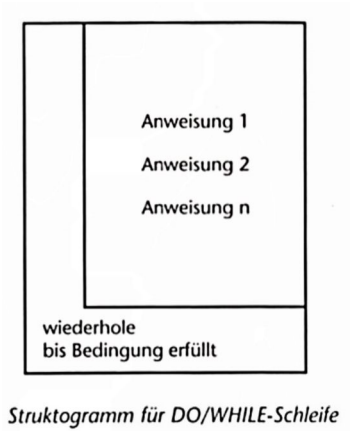
Die Analyse des Arbeitsauftrages ergibt folgende wesentliche Verarbeitungsschritte:

- *Berechne Angebot:*
  - *Eingabe Aufenthaltsdauer und Anzahl der Personen*
  - *Berechnung und Ausgabe*
  - *Eingabedialog für Abbruch*
- *bis der Benutzer mit Taste n abbricht*

Hier werden die Anweisungen mindestens einmal durchgeführt und erst danach wird geprüft, ob die Schleife verlassen werden soll. Für diese Art von Schleifen ist die DO-/ WHILE-Schleife geeignet.

Syntax

```
do
{
    Anweisung 1;
    Anweisung 2;
    Anweisung n;
}
```



Struktogramm für DO/WHILE-Schleife

Die folgende DO-/While-Schleife im Code-Snippet führt die Berechnung des Gesamtpreises so lange durch (inkl. Einlesen der Werte) bis der Benutzer die Schleife mit der Eingabe n beendet. Diese Anweisung kann man auch so lesen: Führe (Do) Anweisungen aus, solange (While) die Bedingung (Antwort != „n") erfüllt ist.

```
do
{
    // Eingaben einlesen
    // Berechnungen durchführen
    Antwort = Console.ReadLine();
} While (Antwort != "n");
```

3.14.1 Break-Anweisung

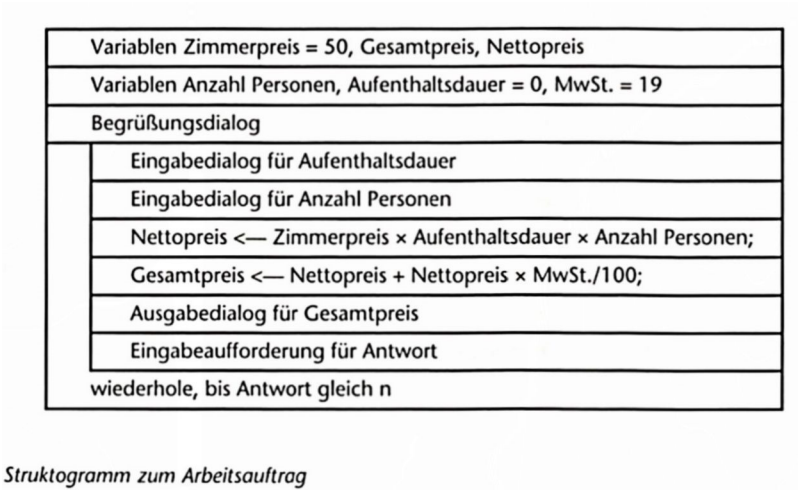
Eine Schleife (FOR, WHILE, DO/WHILE) kann vorzeitig durch die Break-Anweisung verlassen werden. Im folgenden Beispiel wird die FOR-Schleife verlassen, wenn die Bedingung „true" ist:

```
for (i = 0; i < 5; i++)
{
   if (Bedingung)
        break;
}
```

3.14.2 Entwurf und Realisierung

Lösung zum Arbeitsauftrag

Auf Basis der vorangegangenen Erklärungen lässt sich der Arbeitsauftrag wie folgt entwerfen und implementieren.



Struktogramm zum Arbeitsauftrag

Quellcode siehe Lösungsdateien

Aufgaben

1. Entwerfen und implementieren Sie ein Programm, das beliebige Zahlen so lange einliest, bis der Benutzer die Eingabe abbricht. Berechnen Sie die Summe, das Maximum und das Minimum der bisher eingegebenen Werte. Der Benutzer bricht die Eingabe mit n ab.
   Eingabe: Zahlen
   Ausgabe: Summe, Maximum, Minimum
   Eingabe: Antwort auf Abbruchabfrage
2. Passen Sie das Programm „Wochentage berechnen“ aus Kapitel 3.8 so an, dass nach der ersten Berechnung das Programm nicht beendet wird. Nach der Berechnung des Wochentages wird der Benutzer gefragt, ob er das Programm mit n abbrechen oder eine neue Berechnung durchführen möchte (Eingabe ungleich n).

3.15 MehrdimensionaleArrays

Arbeitsauftrag

Schreiben Sie ein Programm, das nach Eingabe von Zimmerkategorie (1 bis 3), Saison (1 bis 4) und Aufenthaltsdauer den Gesamtpreis ausrechnet.

Das Hotel bietet drei Zimmerkategorien (Standard, Komfort und Suite) für vier Saisons an:

	Frühling (1)	Sommer (2)	Herbst (3)	Winter (4)
Standard (1)	60	70	80	65
Komfort (2)	70	80	80	75
Suite (3)	100	110	120	105

Die Analyse des Arbeitsauftrages ergibt folgende wesentliche Verarbeitungsschritte:

- *Saisonpreise im Programm abbilden*
- *Eingabedialog für Zimmerkategorie, Saison und Aufenthaltsdauer*
- *Preis zu Saison ermitteln*
- *Berechnung und Ausgabe des Gesamtpreises*

3.15.1 Arrays deklarieren

Die Zimmerpreisliste für eine Zimmerkategorie wurde weiter vorne in Kapitel 3.7 als eindimensionales Array im Programm abgebildet. Die Preisliste im Arbeitsauftrag ist jedoch eine Tabelle und hat damit zwei Dimensionen (Spalte und Zeile). Daher werden Arrays mit mehr als einer Dimension mehrdimensionale Arrays genannt. Mit diesen Arrays kann man auch drei Dimensionen (vergleichbar mit einem Würfel) oder noch mehr Dimensionen abbilden.

Syntax

```
Datentyp[,] Variablenname = new Datentyp[Zeilenanzahl,SpaItenanzahl];
double[,] Zimmerpreise = new double[3, 4];
```

Diese Deklaration legt ein zweidimensionales Array mit drei Zeilen und vier Spalten an. Das Array wird wie bei den eindimensionalen Arrays mit eckigen Klammern gekennzeichnet. Damit das Programm erkennt, dass es sich um ein mehrdimensionales Array handelt, wird jede Dimension durch ein Komma getrennt. Bei zwei Dimensionen ist es ein Komma, bei drei Dimensionen zwei Kommas usw.

Dann kommen wie gewohnt ein Gleichheitszeichen, ein new-Operator und der Datentyp. Erst hier wird zwischen den eckigen Klammern die Dimensionsgröße angegeben. Bei dieser Syntaxvariante wurde das Array noch nicht mit Werten initialisiert, mit Ausnahme der Standardwerte, die C# vergibt. D. h., im Fall von double sind alle Arraywerte mit 0 belegt.

3.15.2 Arrays setzen

Mehrdimensionale Arrays werden ähnlich wie eindimensionale Arrays gesetzt nur, dass zusätzlich pro Dimension ein Index angegeben werden muss:

Syntax

```
Variablenname[Index Dimension 1, Index Dimension 2, Index Dimension n] = Wert;
Preisliste[0, 0] = 61;       // ändert den Preis der Zimmerkategorie
                             // Standard für die Frühlingssaison
```

3.15.3 Arrays mit Anfangswerten initialisieren

Auch ein mehrdimensionales Array kann mit Anfangswerten initialisiert werden:

Syntax

```
Datentyp[,] Variablenname = new Datentyp[ , ]
{
  {Werteliste von Zeile 1},
  {Werteliste von Zeile 2},
  {Werteliste von Zeile n},
};

double[,] Zimmerpreise = new double[ ,  ]
{
  {60 ,70 ,80 ,65},
  {70 ,80 ,90 ,75},
  {100,110,120,105}
};
```

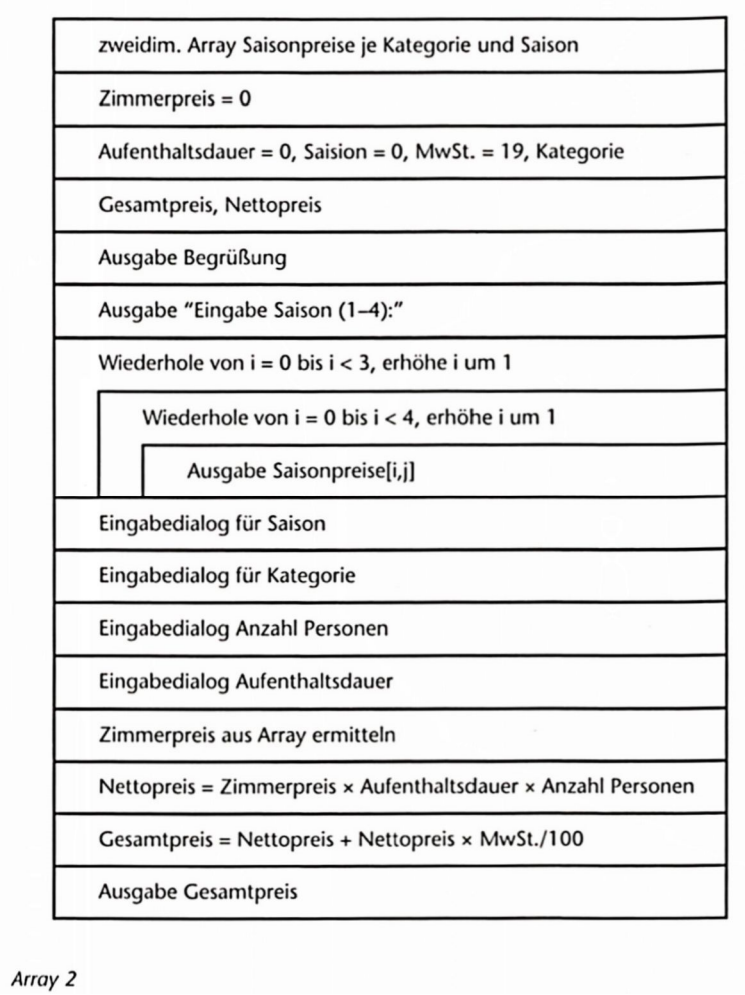
3.15.4 Zugriff mit verschachtelter FOR-Schleife

Wie in Kapitel 3.9 beschrieben, kann man mit einer FOR-Schleife auf ein Array zugreifen. Da die Preisliste ein zweidimensionales Array ist, macht man dies demnach mit zwei FOR-SchIeifen. Da zwei Indizes gleichzeitig benötigt werden, werden die FOR-SchIeifen verschachtelt.

```
for (int i = 0; i < 3; i++)
    for (int j = 0; j < 4; j++)
        Console.Write("{0}\t", Saisonpreise[i, j]); // \t = Tabulator
```

3.15.5 Entwurf und Realisierung

Lösung zum Arbeitsauftrag



Array 2

Quellcode siehe Lösungsdateien

Aufgaben

1. Entwerfen und implementieren Sie einen Tarifrechner für Strompreise eines Anbieters. Ein Energieversorgungsunternehmen bietet verschiedene Tarifmodelle an. So gibt es beispielsweise Tarife mit Grundgebühr und einen Arbeitspreis (in Cent) pro verbrauchter kWh, oder keine Grundgebühr und dafür einen höheren Arbeitspreis. Nach Eingabe des Energieverbrauchs pro Jahr (siehe z. B. letzte Stromrechnung, bzw. verbraucht ein Durchschnittshaushalt 3500 kWh/Jahr) werden von allen Tarifmodellen der Gesamtpreis, die Grundgebühr und der Arbeitspreis angezeigt. Verwenden Sie einfachheitshalber nur ganze Zahlen für den Grundpreis und den Arbeitspreis. Der Arbeitspreis wird in Cent je kWh hinterlegt.

Tarifmodel	Grundpreis €	Arbeitspreis (ct/kWh)
1	83	21
2	222	10
3	70	25
4	0	40

2. Entwerfen und implementieren Sie ein Programm für eine Bedarfsberechnung einer Kaffeehaus-Kette. Das Unternehmen stellt unterschiedliche Produkte her, siehe die folgende Produktionsmatrix. Das Programm soll den Gesamtbedarf für die unterschiedlichen Produkte berechnen. Beispielsweise werden von Kaffee mild 1000 Stück pro Tag verkauft, von Kaffee stark 1500 Stück pro Tag und von Kaffee mit Aroma 500 Stück. Das Programm soll

also die Produktionsmatrix mit einem Vektor multiplizieren. Speichern Sie das Ergebnis in einer eigenen Matrix und geben Sie diese nach der Berechnung am Bildschirm aus.
Eingabe: Mengen für jedes Produkt
Ausgabe: Produktionsmatrix, Bedarfsmatrix

3.16 Logische Operatoren

	Kaffee mild	Kaffee stark	Kaffee mit Aroma
Kaffee	100 ml	120 ml	120 ml
Milch	10 ml	10 ml	10 ml
Aroma	0	0	5 ml

Arbeitsauftrag

Schreiben Sie ein Programm, das einen Rabatt von 30 % berechnet, wenn ein Gast am Sonntag ankommt und die Aufenthaltsdauer fünf Tage beträgt.

Die Analyse des Arbeitsauftrages ergibt folgende wesentliche Verarbeitungsschritte:

- *Eingabe von Ankunftstag und Aufenthaltsdauer*
- *Wenn Aufenthaltstag gleich Sonntag UND Aufenthaltsdauer gleich fünf Tage, dann 30 % Rabatt*

Die in den bisherigen Kapiteln vorgestellten Kontrollstrukturen, wie das IF, führen nur dann die Anweisung aus, wenn eine Bedingung wie AlterKinder < 9 zutrifft. Der Rabatt im Arbeitsauftrag soll jedoch nur dann gewährt werden, wenn zwei Bedingungen (Anreisetag = Sonntag und Aufenthaltsdauer = 5 Tage) zutreffen. Damit beide Bedingungen einen Wahrheitswert (true/false) liefern, werden diese über einen logischen Operator (hier: und) verknüpft. C# kennt folgende logische Operatoren:

- *AND (UND-Verknüpfung)*
- *OR (ODER-Verknüpfung)*

3.16.1 UND-Verknüpfung

Die Syntax für die AND-Verknüpfung lautet wie folgt:

Syntax

```
Wahrheitswert = Operand1 & Operand2;
bool IstRabatt = false;
int Aufenthaltsdauer = 5;
String Anreisetag = "Sonntag";
IstRabatt = (Anreisetag == "Sonntag") & (Aufenthaltsdauer == 5);
```

Das Keyword für den AND-Operator ist das kaufmännische UND. Nur wenn beide Operanden, also Anreisetag == ''Sonntag'' und Aufenthaltsdauer == 5, true ergeben, ist auch das Gesamtergebnis true. Sobald nur ein Operand oder

	Operand 1	Operand 2
Gesamtergebnis	Anreisetag == "Sonntag"	Aufenthaltsdauer == 5
true	true	true
false	true	false
false	false	true
false	false	false

beide Operanden false sind, ist das Gesamtergebnis false. Dieser Zusammenhang lässt sich am besten in einer Wahrheitstabelle darstellen.

3.16.2 ODER-Verknüpfung

Bedingungen wie Anreisetag = Sonntag oder Montag treffen dann zu, wenn eine der beiden Teilbedingungen wahr ist. Diese Bedingung kann in Form einer ODER-Verknüpfung ausgedrückt werden.

Syntax

```
Wahrheitswert = Operand1 | Operand2;
```

Im folgenden Beispiel wird die Variable IstRabatt nur dann auf true gesetzt, wenn der Anreisetag Sonntag oder Montag ist.

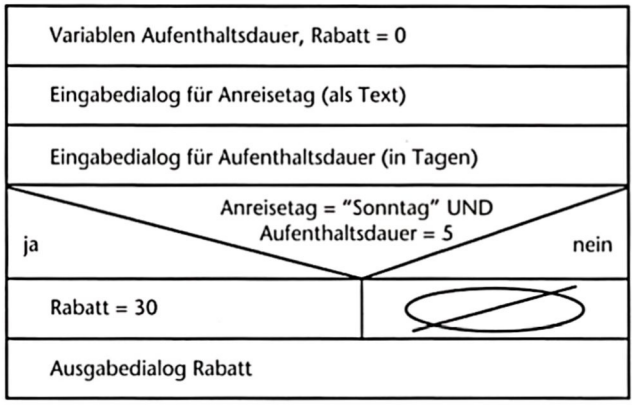
```
bool IstRabatt = false;
string Anreisetag = "Sonntag";
IstRabatt = (Anreisetag == "Sonntag") | (Anreisetag == "Montag");
```

Das Keyword für den OR-Operator ist der senkrechte Strich. Wenn einer der beiden Operanden, also Anreisetag == "Sonntag" oder Anreisetag == "Montag", true ergeben, oder beide Operanden, ist auch das Gesamtergebnis true. Sobald beide Operanden false sind, ist das Gesamtergebnis false. Dieser Zusammenhang lässt sich am besten in einer Wahrheitstabelle darstellen.

	Operand 1	Operand 2
Gesamtergebnis	Anreisetag == "Sonntag"	Anreisetag == "Montag"
true	true	false
true	false	true
true	true	true
false	false	false

3.16.3 Entwurf und Realisierung

Lösung zum Arbeitsauftrag



Struktogramm zur UND-Verknüpfung

Quellcode siehe Lösungsdateien

Aufgabe

1. Entwerfen und implementieren Sie ein Programm, das die Einheit Celsius in Fahrenheit umrechnet.

   $T_f = 32 + 180/100 \times Tc$

   Der erlaubte Wertebereich liegt zwischen T_c: -273,15 °C und etwa 15.599.726 °C, also zwischen der Temperatur des Nullpunkts und der Temperatur des Sonneninneren. Wird der Wertebereich unter- oder überschritten, erscheint eine Meldung.

3.17 Strukturen

Arbeitsauftrag

Schreiben Sie ein Programm, das zu einem Zimmer alle Preise auflistet.

Die Analyse des Arbeitsauftrages ergibt folgende wesentliche Verarbeitungsschritte:

- *Zimmerinformationen im Programm ablegen*
- *Informationen ausgeben*

Sowohl eine Variable wie Zimmerpreis als auch ein Array wie Preisliste kann immer nur einen Datentyp wie beispielsweise int speichern. Möchten Sie zusätzlich im Programm die Information Zimmerfläche abbilden, würden Sie nach bisherigem Wissensstand eine weitere Variable benötigen. In C# gibt es jedoch eine elegantere Möglichkeit einen neuen Datentyp aus anderen Datentypen zusammenzusetzen: das struct.

Syntax

```
public struct Datentypname
{
    public Datentyp Strukturvariable1;
    public Datentyp StrukturvariabIe2;
    public Datentyp StrukturvariableN;
}

public struct ZimmerPreis
{
    public int KategorieNummer;
    public double Preis;
    public int RabattKinder;
}
```

Das Schlüsselwort public ist ein sogenannter Zugriffsmodifizierer, der in Kapitel 4 beschrieben wird. Nach der Definition des eigenen Datentyps ZimmerinfoTyp kann dieser wie gewohnt als Variable definiert werden:

Syntax

```
ZusammengesetzterDatentyp Variablenname;
PreisdatenProPerson Kategorie1;
```

Das Setzen der einzelnen Variablen erfolgt nach dem folgenden Schema: Zwischen VariabIenname und Einzelvariable wird ein Punkt gesetzt.

Syntax

```
Variablenname.Strukturvariable = Wert;
Kategorie1.KategorieNummer = 1;
```

Das Lesen funktioniert ebenfalls über den Aufruf Variablenname-Strukturvariable:

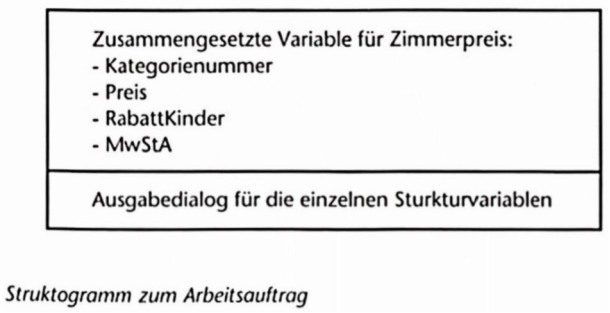
Syntax

```
Wert = Variablenname-Strukturvariable;
Preis = Kategorie1.Preis
```

Ein struct muss in C# außerhalb der Main-Methode deklariert werden. Entweder wie im Source-Code gezeigt, innerhalb der Klassendefinition (dass), oder außerhalb der Klassendefinition. Das kann auch eine eigene Datei sein.

3.17.1 Entwurf und Realisierung

Lösung zum Arbeitsauftrag



Struktogramm zum Arbeitsauftrag

Quellcode siehe Lösungsdateien

Aufgaben

1. Entwerfen und implementieren Sie ein Programm, das eine Adresse als Struktur abbildet: Straße (string), Hausnummer (string), PLZ (Ganzzahl) und Ort (string)
   Eingabe: Straße, Hausnummer, PLZ und Ort
   Prüfung: PLZ muss zwischen 10000 und 99999 liegen, sonst Eingabe von PLZ wiederholen lassen
   Ausgabe: Strukturvariablen
2. Entwerfen und implementieren Sie ein Programm, das folgende Informationen zu einem Modul als Struktur abbildet: Titel, Autor, ISBN, Erscheinungsjahr, Preis, Verlag
   Eingabe: Titel, Autor, ISBN, Erscheinungsjahr, Preis
   Prüfung: Preis muss größer 0 sein, sonst Eingabe von Preis wiederholen lassen
   Ausgabe: Strukturvariablen

3.18 Enums

Arbeitsauftrag

Schreiben Sie ein Programm, welches die möglichen Verpflegungstypen (Frühstück, Voll- und Halbpension) als Enum implementiert.

Die Analyse des Arbeitsauftrages ergibt folgende wesentliche Verarbeitungsschritte:

- *Verpflegungstyp im Programm abfragen*
- *Verpflegungstyp als Text ausgeben*

Eine Variable wie Zimmerpreis kann unterschiedliche Werte annehmen. Der Wertebereich kann je nach Datentyp und Anwendungsfall von beispielsweise 20 € bis 300 € für ein Zimmer betragen. Hingegen gibt es Variablen wie die Kundenkategorie, die nur eine bestimmte Anzahl an Werten, wie Stammkunde, Finnenkunde und Reisebüro, abbilden sollen. In Kapitel 3.8 wurde jeder Kundenkategorie eine Zahl zugeordnet. Nur über den Kommentar ist erkennbar, um welche Art von Kategorie es sich handelt. Um einerseits einem Datentyp vorgegebene Werte zuweisen zu können und andererseits den Werten einen Namen zu geben, sind Enums ein geeignetes Mittel:

Syntax

```
public enum EnumTypeName
{
    Konstante1, Konstante2, KonstanteN
}
public enum KundenkategorieTyp
{
    Stammkunde,
    Firmenkunde,
    Reisebuero
};
```

Standardmäßig beginnt der Wert der ersten Konstante mit 1. Jede weitere Konstante im Enum wird um 1 erhöht: Firmenkunde hat den Wert 2, Reisebuero den Wert 3.

Soll die Konstante 1 mit einem anderen Startwert beginnen, verwendet man folgende Syntax:

Syntax

```
public enum EnumTypeName
{
    Konstante1 = Startwert,
    Konstante2,
    KonstanteN
}

public enum KundenkategorieTyp
{
    Stammkunde = 100,
    Firmenkunde,
    Reisebuero
};
```

Des Weiteren kann jeder Konstanten im Enum ein eigener Wert gegeben werden:

Syntax

```
public enum EnumTypeName
{
    Konstante1 = Wert1,
    Konstante2 = Wert2,
    KonstanteN = WertN
}

enum VerpflegungsTyp
{
    Keiner = 0,
    Fruehstueck =10,
    HP = 20,
    VP = 30
}
```

Nachdem der Datentyp Enum deklariert wurde, kann dieser wie gewohnt als Variable deklariert und verwendet werden.

Syntax

```
EnumTypeName Variablenname;
Variablenname = EnumTypeName.Konstante1

VerpflegungsTyp Verpflegungsart;
Verpflegungsart = VerpflegungsTyp.Fruehstueck;
```

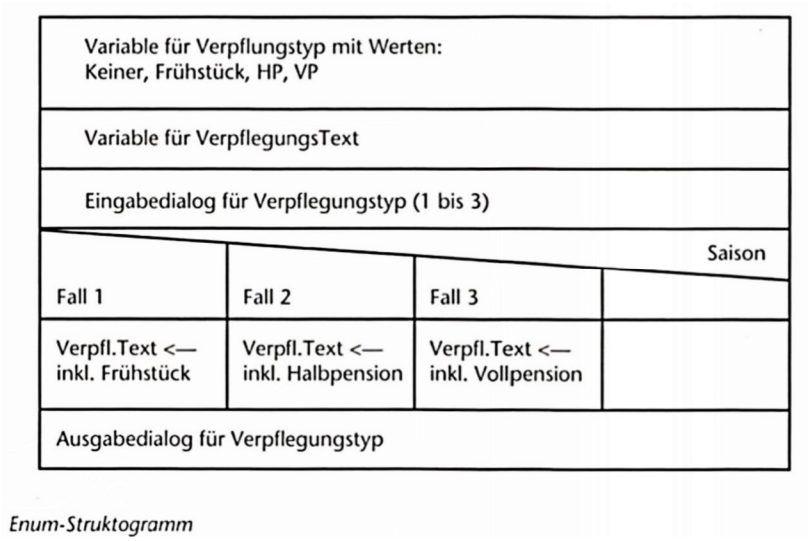
Der darunterliegende Datentyp eines Enums ist standardmäßig ein int. Abhängig vom vergebenen Startwert oder Konstantenwert kann der darunterliegende Datentyp jeder ganzzahlige Datentyp sein. Mit folgender Typkonvertierung kann zwischen einem Enum und einer ganzen Zahl konvertiert werden:

Syntax

```
Wert = (int)EnumTypName.Konstante1;
EnumTypName = (EnumTypName)Wert;

VerpflegungsTyp Verpflegungsart;
int VerpflegungInt = 1;
Verpflegung = (VerpflegungsTyp)Convert.ToInt32(VerpflegungInt);
```

Lösung zum Arbeitsauftrag



Enum-Struktogramm

Quellcode siehe Lösungsdateien

Aufgaben

1. Ändern Sie den Auftrag aus Kapitel 3.8 so, dass im Switch Enums verwendet werden.
2. Entwerfen und implementieren Sie ein Programm, das die Angabe von Grad in die vier Himmelsrichtungen (N, O, S, W) umwandelt. Entwickeln Sie eine Formel, welche die Grade direkt in die Werte 0 bis 360 umwandelt. Mithilfe eines Switches und Enums soll die Himmelsrichtung als Text ausgegeben werden.
   Eingabe: Grad von 0-360 oder n für Abbruch
   Ausgabe: Norden, Osten, Süden, Westen

3.19 Weiterführende Themenbereiche

Neben den bisher benannten Anweisungen könnten noch weitere von Interesse sein. Diese werden im Folgenden kurz erläutert.

3.19.1 Else-if-Anweisung

Arbeitsauftrag

Schreiben Sie ein Programm, das für Familien mit zwei Erwachsenen und einem Kind folgende Rabatte gibt:

- *0 bis 6 Jahre: 100%*
- *7 bis 11 Jahre: 70 %*
- *Sonst 30 %*

Neben der Vorgestellten zweiseitigen und Mehrfachauswahl steht noch die else-if- Anweisung zur Verfügung. Damit lassen sich unter Umständen verschachtelte IF- Anweisungen vermeiden. Die Syntax lautet wie folgt.

Syntax

```
if (Bedingung 1)
{
    // Anweisung 1;
}
else if (Bedingung 2)
{
    // Anweisung 2;
}
else if (Bedingung 3)
{
    // Anweisung 3;
}
else
{
    // Anweisung n;
}
```

Wenn die Bedingung 1 nicht erfüllt wird, ist eine Prüfung der Bedingung 2 erforderlich. Ist sie erfüllt, wird die Anweisung 2 ausgeführt. Falls Bedingung 2 nicht erfüllt ist, wird die Bedingung 3 geprüft usw.

Lösung zum Arbeitsauftrag

Quellcode siehe Lösungsdateien

Aufgaben

1. Erweitern Sie die Aufgabe zur Windchill-Temperatur aus Kapitel 2 so, dass das Programm nach Berechnung der Windchill-Temperatur das thermische Empfinden nach der folgenden Tabelle ausgibt. Verwenden Sie die else-if-Anweisung. Nutzen Sie für das „bis-Datum" den kleiner-Operator (Bsp. < 26).

Windchill-Temperatur in Grad Celsius	Thermisches Empfinden
unter -39	sehr kalt
-39 bis -26	kalt
-26 bis -13	kühl
-13 bis 0	leicht kühl
0 bis 20	behaglich
20 bis 26	leicht warm
26 bis 32	warm
32 bis 38	heiß
über 38	sehr heiß

   Nach Ausgabe des thermischen Empfindens wird der Benutzer gefragt, ob er das Programm beenden oder eine neue Berechnung starten möchte.

Microsoft stellt eine umfangreiche Dokumentation für C# zur Verfügung

https://docs.microsoft.com/de-de/dotnet/csharp

Nutzen Sie diese Seite um sich über foldende Themen zu informieren´.

- *break-Anweisung in Schleifen*
- *Zuweisungsoperatoren*
- *Strukturarrays*
- *bedingtes AND, OR*

Hinweis für alle Aufgaben

Bevor Sie beginnen die Aufgaben zu programmieren, sei nochmals auf die Einhaltung der Schritte Analyse, Entwurf, Realisierung und Test hingewiesen. Diese Schritte sollten Sie so verinnerlicht haben, dass Sie später auch komplexere Programme lösen können. Das Ergebnis eines Entwurfs ist daher zuerst ein Struktogramm oder PAP. Erst wenn dieses vorliegt, sollten Sie mit der Umsetzung beginnen.

2. Entwerfen und implementieren Sie die Angebotssoftware wie sie im Kapitel 3.1 beschrieben ist. Am Ende der Angebotsberechnung wird der Benutzer gefragt, ob er das Programm verlassen möchte oder die Angebotsberechnung neu durchgeführt werden soll.

3. Entwerfen und implementieren Sie ein Programm für die Bedarfsermittlung einer Firma, die PC-Gehäuse und Lüfter herstellt. Für die Prognose des Bedarfs an verschiedenen Materialien wie Schrauben, Muttern, Blech etc. soll für die nächsten drei Monate folgende Prognosemethode verwendet werden:
   - *gewichteter Durchschnitt mit exponentieller Glättung 1. Ordnung.*
     - *y(t) = Alpha × y(t) + (1 - Alpha) × y(t-1)*
     - *y(t): Prognosewert aktueller Monat, y(t-1), Prognosewert vorheriger Monat*
     - *y(t): Istwert vorheriger Monat*
     - *Alpha: Glättungsfaktor (zwischen O und 1), z. B. 0,4*
   - *Rahmenbedingungen:*
     - *Prognosewert für Monat „-n" ist Istwert*
     - *Wenn es keine Istwerte mehr gibt, dann werden die Prognosewerte verwendet.*
     - *Verwenden Sie System.Math.Round., um den Prognosewert (double) zu runden und danach auf int zu casten.*

- *Eingabe:*
  - *Anzahl Monate für vorhandene Istwerte*
  - *Stückzahl je Monat (-n bis -1, relative Monatsangabe zum aktuellen Monat)*

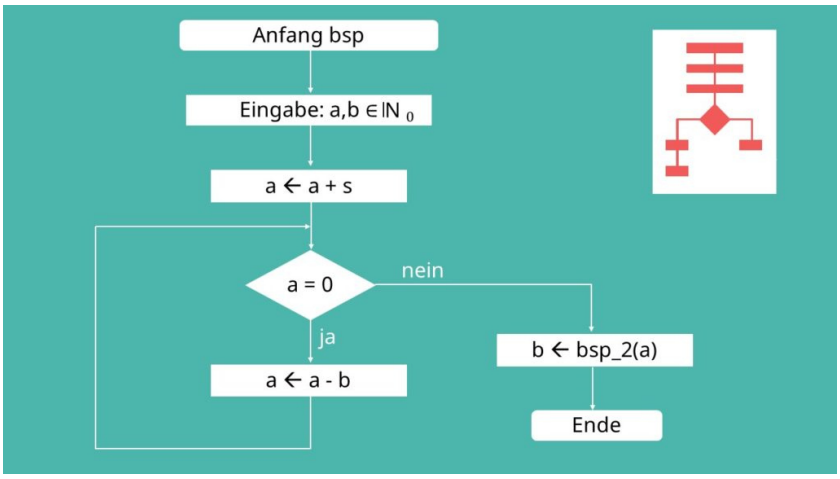
3.20 PAP Programmablaufplan

Sie haben in diesem Kapitel alle Kontrollstrukturen erlernt und wie diese in einem Struktugramm dargestellt werden. Struktugramme werden bei größeren Programmen schnell unübersichtlich, daher gibt es eine weitere Darstellung, den Programmablaufplan PAP.

3.20.1 Was sind Programmablaufpläne?

Benutzen Sie für das Erstellen der PAPs einen Editor wie z. Bsp. yED → Flussdiagramm

Eine der meistgenutzten Methoden zur Planung von Programmen ist ein Programmablaufplan. Schauen wir uns doch einmal dieses Beispiel an. Wie du siehst, besteht dieser Ablaufplan aus verschiedenen Blasen. Wir beginnen mit der abgerundeten Blase. Darin vermerken wir die Tatsache, dass das der Anfang unseres Algorithmus ist und notieren



auch dessen Namen. Nun setzen wir einen Pfeil und führen die erste Anweisung aus. Dabei handelt es sich eigentlich immer um eine Eingabe, denn ohne Werte können wir ja nichts berechnen.

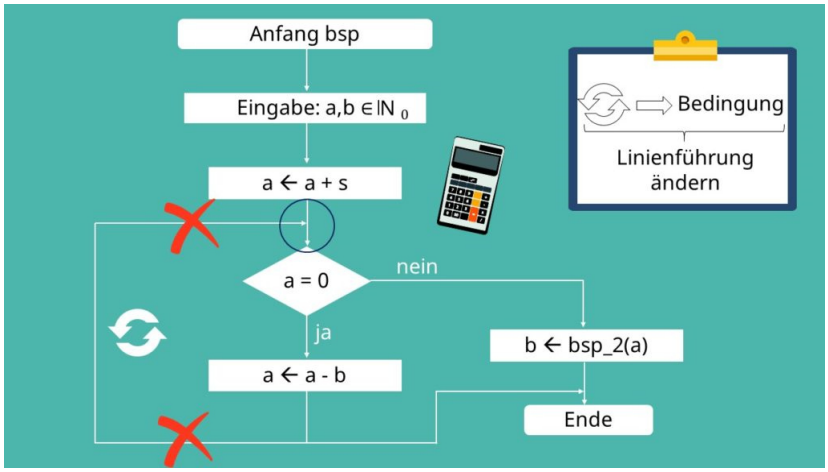
In unserem Fall geben wir zwei Variablen ein, die im Bereich der natürlichen Zahlen liegen. Hier sieht man schon den Unterschied zum sofortigen Programmieren. Denn was sonst in mehreren Zeilen Code zu lesen wäre, haben wir ganz einfach in ein Kästchen gepackt.

Jetzt, da wir unsere Start-Werte haben, können wir zu rechnen beginnen. Deswegen folgt auch gleich eine weitere Anweisung. Diesmal führen wir allerdings eine Rechnung durch, denn auch diese verpacken wir in Anweisungsblöcken.

Danach fügen wir eine Raute ein und brechen damit unser sequenzielles Programm etwas auf. Diese Rauten enthalten in einem Programmablaufplan immer Bedingungen. Sie können entweder für Schleifen oder für if-Anweisungen gedacht sein. Wofür du sie verwendest, hat keine speziellen Auswirkungen auf ihre Funktionsweise, da die Differenzierung zwischen beidem auf andere Weise erfolgt. Trifft unsere Bedingung zu, folgen wir dem mit „ja“ beschrifteten Pfeil und gelangen zu einer weiteren Anweisung. Haben wir diese ausgeführt, führt uns der nächste Pfeil – wieder zurück zu unserer ursprünglichen Bedingung?

3.20.2 Veränderung der Pfeilführung

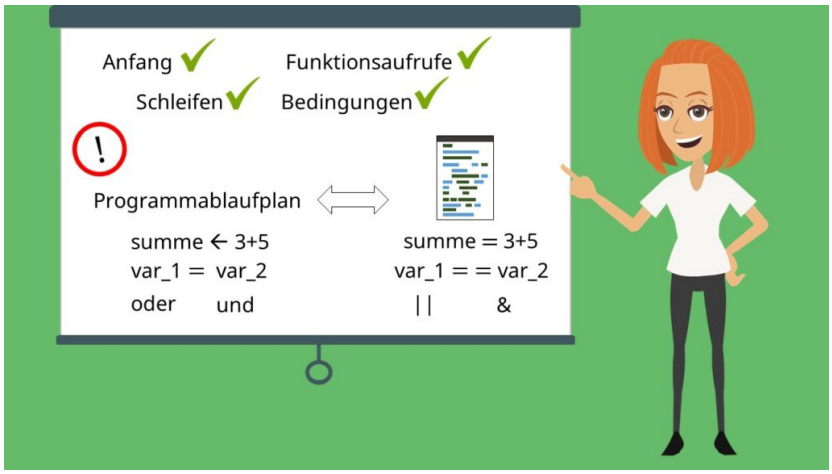
Das ist auch korrekt so, denn bei dieser Art der Notation bilden wir eine Schleife nur durch die Pfeilführung. Dabei ist es wichtig, dass der Pfeil VOR der Bedingung wieder ankommt. Wollen wir aus unserer Schleife eine Bedingung machen, so können wir das ganz einfach tun, indem wir die Linienführung unseres Pfeils ändern. Dazu biegen wir ihn so um, dass er nicht mehr über der Bedingung einmündet, sondern direkt über dem Ende des Algorithmus.



Folgen wir nun unserem nein-Pfeil kommen wir zur letzten Art von Anweisung, die hier wichtig ist: Dem Aufrufen von externen Algorithmen. Wie dir vielleicht aufgefallen ist, bilden wir im Moment nur eine Programmdatei ab. Du hast aber bei aufwendigen Programmen eigentlich immer mehrere Funktionen und Dateien. Um nun also eine andere Funktion aufrufen zu können, nennen wir einfach deren Namen und setzen die richtigen Parameter in die Klammern.

3.20.3 Wichtige Hinweise

Damit kennst du jetzt alle Arten von Befehlen. Im Gegensatz zum Quellcode wird eine Wertzuweisung nicht mit einem Gleichheitszeichen angezeigt, sondern mit einem Pfeil. Genauso verhält es sich auch mit Vergleichen, bei denen das doppelte Gleich durch ein Einfaches ersetzt wird und das „oder“ bzw. das „und“ nicht mehr durch die jeweiligen Zeichen ausgedrückt werden.



Schließlich beenden wir unseren Programmablaufplan genauso, wie wir ihn begonnen haben mit einer Runden „Blase“. Diesmal schreiben wir allerdings „Ende“ hinein.

Arbeitsauftrag

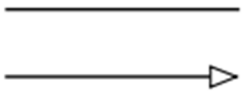
Erstelle für alle bisherigen und nachfolgenden Programme PAPs

Benutze dafür die Symbole

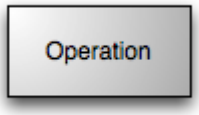
Kreis; Oval / Rechteck mit gerundeten Ecken: Terminator



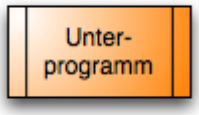
Pfeil, Linie: Verbindung zum nächstfolgenden Element



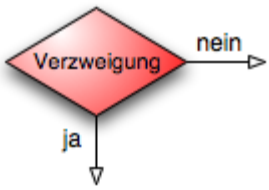
Rechteck: Operation (Tätigkeit) (auch Ein- Ausgabe)



Rechteck mit doppelten, vertikalen Linien: Unterprogramm ausführen



Raute: Verzweigung / Entscheidungen



4 Grundlagen der objektorientierten Programmierung

Nach Durcharbeiten dieses Kapitels kennen Sie die Grundbegriffe der objektorientierten Programmierung wie Objekte, Klassen, Datenkapselung, Vererbung und Polymorphismus.

4.1 Hauptauftrag

Der Projektleiter beauftragt Sie, ein Programm zu schreiben, das ein Angebot für verschiedene Unterkunftsarten berechnet.

4.2 Was bedeutet objektorientiert?

Wie würden Sie folgende Hauptschritte einer Angebotssoftware in einem Programm so abbilden, dass der SourceCode lesbar, wartungsarm (Stichwort Fehlerbehebung) und erweiterbar ist bzw. bleibt?

- *Eingabe der Daten*
- *Gesamtpreis berechnen*
- *neuen Gast anlegen oder vorhandenen auswählen*
- *Angebot berechnen*
- *Angebot per E-Mail versenden*

Bisher wurden die Anweisungen und Kontrollstrukturen für die Berechnung einfach der Reihe nach in der Main-Methode programmiert. Beispielsweise sieht der Kern der Berechnung für den Aufenthalt in einer Unterkunft wie folgt aus:

```
Nettopreis = Grundpreis * Aufenthaltsdauer;
Gesamtpreis = Nettopreis + Nettopreis * MwSt./100;
```

Hier hält sich die Anzahl der Codezeilen noch in Grenzen. Wenn jedoch auch noch andere Bedingungen bei der Berechnung und die weiteren Schritte zum Erstellen eines Angebots berücksichtigt werden sollen, dann steigt gleichzeitig die Anzahl der Codezeilen. Das hat unmittelbar Auswirkungen auf die Übersichtlichkeit. Es können dadurch leichter Fehler bei der Wartung und Erweiterung passieren, da man durch die große Anzahl an Codezeilen den Überblick verlieren kann.

Der Source Code würde so aussehen, als hätten Sie eine Bedienungsanleitung für ein Möbelstück vor sich liegen, das weder Absätze noch Überschriften hat. In den Anfängen der Programmierung wurden Programme tatsächlich so geschrieben. Daher entwickelten sich im Laufe der Zeit unterschiedliche Programmierkonzepte (Programmierparadigma), um (große) Programmieraufgaben besser in einem Programm abbilden zu können. Komplexität, Wartbarkeit und Erweiterbarkeit der Programme lassen sich so besser beherrschen.

Ein klassisches Programmierkonzept ist die prozedurale Programmierung bzw. die strukturierte Programmierung. Bei diesem Konzept teilt man eine (große) Programmieraufgabe wie die Angebotserstellung in mehrere Teilaufgaben wie „Gesamtpreis berechnen", „Gastdaten suchen", „Gastdaten anlegen" und „E-Mail versenden" auf. Eine Teilaufgabe wie „Gesamtpreis berechnen" setzt sich aus mehreren Anweisungen zusammen. Wie das folgende Code-Snippet der prozeduralen Programmiersprache C zeigt, werden diese Anweisungen in einer Prozedur (genauer: Funktion, da ein Wert retourniert wird) namens BerechneGesamtpreis zusammengefasst und im Hauptprogramm (Main) stellvertretend verwendet (aufgerufen).

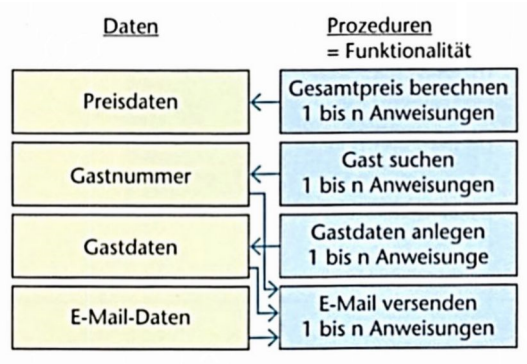
Anmerkung: Hier sehen Sie deutlich, dass die Syntax von C# und C verwandt sind, da die prozedurale Programmiersprache C der Vorvorgänger von C# ist (dazwischen gibt es noch C++, den objektorientierten Vorgänger von C#).

```
private static void Main(string[] args)
{
    double Gesamtpreis;
    double Grundpreis = 70;
    int Aufenthaltsdauer;
    int Gastnummer = -1;
    Aufenthaltsdauer = 7;
    Gesamtpreis = BerechneGesamtpreis(Grundpreis, Aufenthaltsdauer);
    Grundpreis = 66; // Grundpreis kann überschrieben werden
}
private static double BerechneGesamtpreis(double Grundpreis, int Aufenthaltsdauer)
```

```
{
    const double MwSt. = 7;
    double Nettopreis = Grundpreis * Aufenthaltsdauer;
    double Gesamtpreis = Nettopreis + Nettopreis * MwSt. / 100;
    return Gesamtpreis;
}
```

Die anderen Aufgaben wie „Gast anlegen" werden ebenfalls als Prozedur programmiert. Werden Daten, Prozeduren und die Zusammenhänge zwischen den Daten und Prozeduren grafisch dargestellt, ergibt sich folgende Abbildung:

Sowohl das Beispiel mit dem Code-Snippet als auch in der Abbildung zeigen, dass die Daten und Funktionalität nicht unmittelbar zusammengehören. Eine Verbindung zwischen den Daten Gesamtpreis, Grundpreis und Aufenthaltsdauer ist nicht auf den ersten Blick ersichtlich. In der Praxis existieren noch viel komplexere Programme mit viel mehr Codezeilen, bei denen der Zusammenhang nicht unbedingt direkt erkennbar ist. Des Weiteren existiert kein Mechanismus, der verhindert, dass eine Variable wie Grundpreis überschrieben wird (hier beispielsweise unabsichtlich durch den Wert 66).



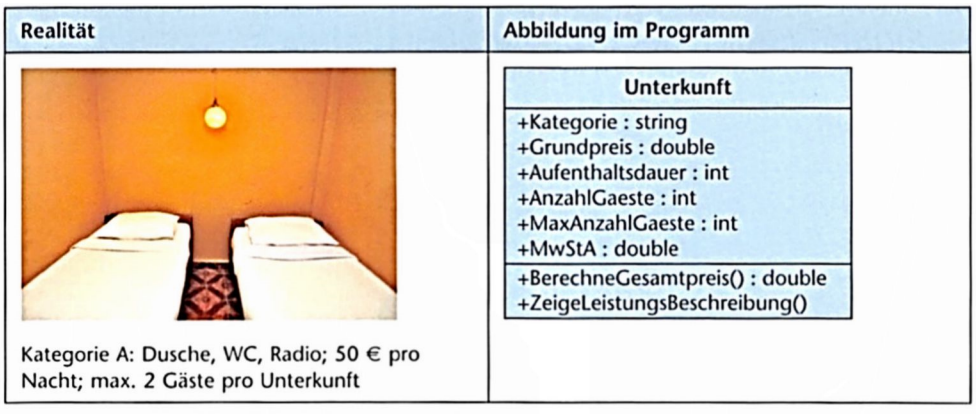
Prozedurale Programmierung: Daten und Funktionalität sind getrennt

Genau diese Nachteile versucht die objektorientierte Programmierung aufzuheben. Daten in der OOP können so implementiert werden, dass sie nicht überschrieben werden können, und es ist erkennbar, welche Daten und Prozeduren (Funktionalität) zusammengehören. Dazu verwendet die OOP Objekte, daher auch der Begriff objektorientiert.

Mithilfe dieser Objekte können auch jene aus der Realität besser abgebildet werden. Das können sein: Menschen, Häuser, Autos und andere Sachverhalte, welche in einem Programm abgebildet werden sollen. Das Angebotsmodul der Hotelsoftware soll folgende (betriebliche) Realität zeigen: Max Huber sucht für seinen Aufenthalt von drei Tagen in Berlin eine einfache Unterkunft und stellt daher eine unverbindliche Anfrage über den Preis und die Verfügbarkeit eines Zimmers. Einige Zeit später erhält er ein Angebot per E-Mail.

Bei dieser Anforderung ist die Unterkunft solch ein Objekt aus der Realität. Eine Unterkunft hat je nach Kategorie unterschiedliche Ausstattungen. So befinden sich beispielsweise in einer einfachen Unterkunft eine Dusche und das WC im selben Raum und an Unterhaltungselektronik ist nur ein Radio vorhanden. Hingegen hat die nächsthöhere Kategorie statt der Dusche eine Badewanne und es gibt zusätzlich einen kleinen Fernseher. In der Unterkunft selbst können je nach Größe ein bis maximal drei Gäste übernachten. In einem Zweibettzimmer kann aber auch ein Gast für eine bestimmte Aufenthaltsdauer übernachten. Auf Basis der Aufenthaltsdauer und des Grundpreises für eine Nacht errechnet sich der Gesamtpreis.

C#.NET ermöglicht es, ein Objekt (genauer: die Klasse) namens Unterkunft zu definieren, welche aus den Daten (Attribute) Kategorie, Grundpreis, Aufenthaltsdauer, AnzahlGaeste, Maxanzahlgaeste und der Aktion (Operation) „Berechne Gesamtpreis'' besteht. In der folgenden Tabelle wird deutlich, dass Daten und Funktionalitäten durch das Objekt Unterkunft zusammengefasst sind und so die Realität gut im Programm abgebildet werden kann. Die grafische Darstellung auf der rechten Seite (Abbildung im Programm) ist in UML-Notation verfasst (siehe Kapitel 4.3).



OOP: Daten und Funktionalität sind zusammengefasst

Noch deutlicher zeigt dies das folgende Code-Snippet: Zuerst werden dem Objekt Unterkunft alle notwendigen Daten wie Gesamtpreis und Aufenthaltsdauer übergeben, dann wird die Funktionalität BerechneGesamtpreis aufgerufen und liefert den Gesamtpreis. Über den Namen Unterkunft (genauer: Objektvariable) ist der Zusammenhang auf einen Blick erkennbar und der Code wird lesbarer als in der prozeduralen Programmierung.

```
Unterkunft Unterkunft = new Unterkunft();
Unterkunft.Kategorie = "Kategorie A";
Unterkunft.Grundpreis = 50;
Unterkunft.Aufenthaltsdauer = 7;
Unterkunft.AnzahlGaeste = 1;
Unterkunft.MaxAnzahlGaeste = 2;
double Gesamtpreis = Unterkunft.BerechneGesamtpreis();
```

Um die OOP optimal einsetzen zu können, ist das Verständnis von folgenden vier Konzepten unbedingt notwendig:

- *Abstraktion*
- *Datenkapselung*
- *Vererbung*
- *Polymorphie*

4.3 Objektorientierte Analyse, Design und UML

Arbeitsauftrag

Stellen Sie das Objekt (genauer: Klasse) Unterkunft in grafischer Form dar.

Auch in der objektorientierten Programmierung sind die Phasen Analyse, Entwurf und Implementierung sehr wichtig, um übersichtliche, wartungsarme und erweiterbare Programme realisieren zu können. Basis für die objektorientierte Analyse (OOA) sollte eine Anforderung sein, welche in einem Pflichtenheft beispielsweise wie folgt beschrieben ist:

Beispiel:

Konkretes Muster für Angebotserstellung:

Max Huber sucht für seinen Aufenthalt von drei Tagen in Berlin eine einfache Unterkunft und stellt daher eine unverbindliche Anfrage über den Preis und die Verfügbarkeit eines Zimmers. Er erhält ein Angebot per E-Mail. Der Grundpreis für eine Unterkunft gilt für eine Übernachtung. Ein Zimmer kann je nach Kategorie zwischen einem und drei Gäste aufnehmen.

In der objektorientierten Analyse geht es darum, die Anforderung zu verstehen und die Objekte wie Unterkunft und Gast aus der realen Welt zu identifizieren. Eine OOA beschreibt also die Anforderung aus objektorientierter Sicht, wobei die konkrete Lösung bzw. Implementierung oder Programmiersprache vorerst ignoriert wird. Beispielsweise wird die Anforderung wie folgt in Objekte, Personen oder Sachverhalte zerlegt:

- *Das Objekt Gast erhält ein Objekt Angebot.*
- *Das Objekt Angebot enthält ein bis mehrere Objekte Angebotspositionen. Hinweis: Auf den Zusammenhang „ein Angebot enthält mehrere Angebotspositionen" muss man nicht unbedingt im ersten Schritt kommen. Beispielsweise könnte im ersten Schritt nur das Objekt Angebot identifiziert werden und durch weitere Analyse oder Diskussion mit dem Projektleiter oder Auftraggeber stellt sich heraus, dass ein Angebot mehrere Angebotspositionen enthalten kann.*
- *Jede Angebotsposition enthält die Anzahl und ein Objekt Unterkunft.*

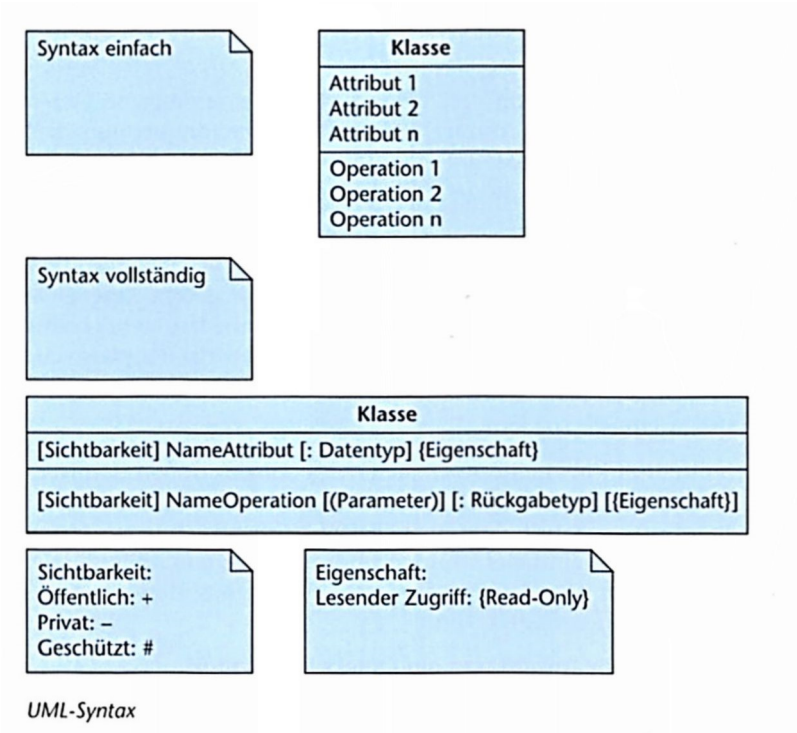
Anschließend wird für jedes Objekt untersucht, welche Daten notwendig sind, um eine bestimmte Funktionalität zu erfüllen. Für das Objekt Unterkunft sind das die Daten (Attribute) Kategorie, Grundpreis, Aufenthaltsdauer, AnzahlGaeste, Maxanzahlgaeste und die Aktion (Operation) „Berechne Gesamtpreis". Das Gastobjekt benötigt die Attribute Name und Adresse (und keine Aktion).

Nach der OOA wird das objektorientierte Design (OOD) durchgeführt. Hier wird auf Basis der objektorientierten Analyse eine Lösung entworfen, wie die Anforderung in einer objektorientierten Programmiersprache wie C#.NET umgesetzt wird. Für die grafische Darstellung und als Werkzeug zur Modellierung von Software steht die UML (Unified Modelling Language) zur Verfügung. Die UML bietet eine breite Palette von Diagrammarten an. In diesem Modul wird das Klassendiagramm verwendet. Mit dieser Diagrammart werden die Attribute (Daten) und Operationen

(Funktionalitäten) von Klassen sowie die Beziehungen von Klassen untereinander dargestellt (siehe dazu Kapitel 4.13). UML wird in der Praxis häufig genutzt, um eine objektorientierte Software zu entwerfen. Daher stammt auch der Begriff OOD für objektorientiertes Design, der englische Begriff für Entwurf.

Zunächst wird die allgemeine Syntax zur Darstellung einer Klasse erläutert:

- *Rechteck*
  - *Klassenname (fett gedruckt)*
  - *Trennstrich*
  - *1 bis n Attribute (entspricht den Daten eines Objekts, Fachbegriff in C#: Felder und Eigenschaften)*
  - *Trennstrich*
  - *1 bis n Operationen (entspricht der Funktionalität eines Objekts, Fachbegriff in C#: Methoden)*



UML-Syntax

Eine Klasse muss nicht unbedingt aus Eigenschaften und Methoden bestehen, sondern kann entweder aus nur einer oder mehreren Eigenschaften (z.B. Objekt Gast) oder aus nur einer oder mehreren Methoden (z.B. Funktionssammlung) bestehen. In diesem Fall gibt es nur einen Trennstrich zwischen Klassenname und Attribut oder Operation.

Des Weiteren haben Attribute und Operatoren eine Sichtbarkeit. Die Angabe der Sichtbarkeit (Zugriffsmodifizierer) ist optional und kann öffentlich (+), privat (-) oder protected (#) sein. Auf die Sichtbarkeit wird in den einzelnen Kapiteln eingegangen. Es gibt auch noch andere Zugriffsmodifizierer, die oben genannten sind aber die häufigsten in der Praxis.

Optional kann der Datentyp (nach dem Doppelpunkt) angegeben werden. Das betrifft Eigenschaften, Felder, Parameter und Rückgabewerte. Diese Themen werden der Reihe nach in den folgenden Kapiteln erklärt.

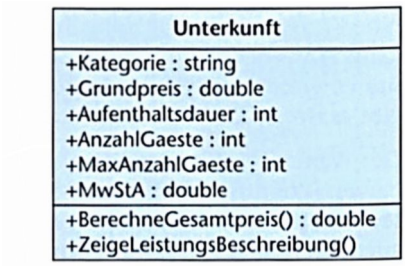
In diesem einfachen Fall können die Ergebnisse der Analyse (Objekte, Attribute und Operationen) direkt eins zu eins in ein Objekt (genauer: Klasse) übertragen werden. Hinweis: In der Praxis ist es nicht immer einfach zu erkennen, weiche Objekte aus der Realität wirklich für das Programm relevant sind und wie diese im Programm umgesetzt werden sollen. In der Praxis entsteht daher OOD nicht in einem Schritt. Auch kann es mitunter mehrere Lösungen geben.

In diesem Fall ergeben sich für das Objekt (genauer: Klasse) Unterkunft folgende Attribute und Operatoren:

- *KIassenname: Unterkunft*
- *Attribute (= Eigenschaften):*
  - *Kategorie (Sichtbarkeit öffentlich, Datentyp string). Der Datentyp (Klasse) string wird zum Speichern von Texten verwendet. Im Detail wird auf diese Klasse in Kapitel 6 eingegangen.*
  - *Aufenthaltsdauer (Sichtbarkeit öffentlich, Datentyp int).*
  - *Grundpreis (Sichtbarkeit öffentlich, Datentyp double)*
  - *Mehrwertsteuer mit Kennzeichen A (7% für Unterkunft). In diesem Kapitel werden aus Gründen der Einfachheit nur Unterkünfte ohne Verpflegung betrachtet, da die Verpflegung und andere Dienstleistungen seit dem Jahr 2010 einen anderen Steuersatz haben.*
  - *AnzahlGaeste (Sichtbarkeit öffentlich, Datentyp int)*
  - *Maxanzahlgaeste (Sichtbarkeit öffentlich, Datentyp int)*
- *Operationen bzw. Methoden (= Funktionen bzw. Datentyp):*
  - *BerechneGesamtpreis (Sichtbarkeit öffentlich, Rückgabewert double)*
  - *ZeigeLeistungsBeschreibung (Sichtbarkeit öffentlich, Rückgabewert keiner)*

Die Methode ZeigeLeistungsBeschreibung ist durch die Anforderung entstanden, dass in einer Angebotsposition aufgelistet wird, welche Leistungen im Angebot enthalten sind.

Damit lässt sich die Berechnungsmethode Zimmer mit seinen Attributen und Methoden im UML wie nebenstehend darstellen:



UML-Klassendiagramm für Klasse Unterkunft

Aufgaben

1. Entwerfen Sie ein UML-Diagramm für eine Multimedia-Verwaltung. Dort werden zunächst Musiktitel verwaltet. Die Klasse Musiktitel hat die Attribute Nummer, Name, DauerMinute, DauerSekunde, Interpret, Albumname, Genre, Label und Erscheinungsjahr. Diese Klasse stellt folgende Operatoren zur Verfügung: ZeigeDetailInfos und Zeigedauer.
2. Entwerfen Sie ein UML-Diagramm für eine Stammdaten-Verwaltung für Mitarbeiter. Die Klasse Angestellter hat folgende Attribute: PersonaINummer, Vorname, Nachname, Gehalt, Adresse, Telefonnummer, Eintrittsjahr, Eintrittsmonat und Eintrittstagtag sowie die Operatoren ZeigeBruttoArbeitsentgelt, ZeigeStammdaten.

4.4 OOP-Konzept Abstraktion

Arbeitsauftrag

Implementieren Sie die Klasse Unterkunft auf Basis des UML-Diagramms aus Kapitel 4.3.

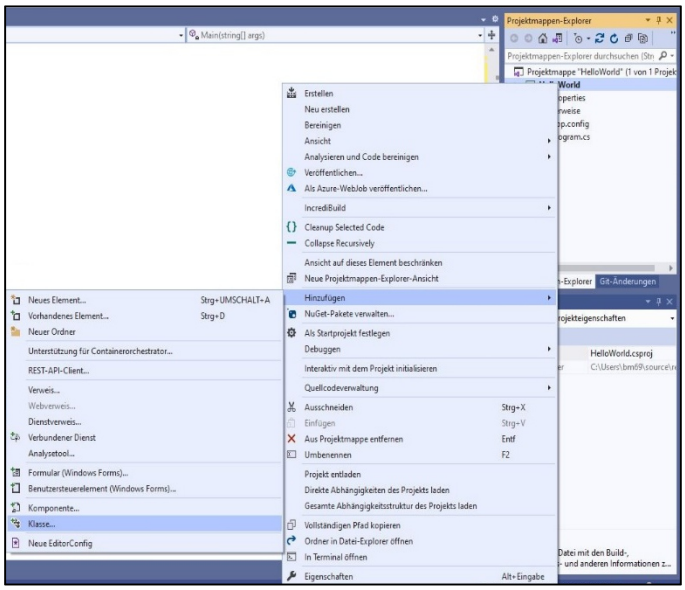
Im Code-Snippet wurde bereits gezeigt, wie das Objekt Unterkunft verwendet wird: Dem Objekt Unterkunft werden die notwendigen Daten wie Grundpreis übergeben, und die Methode BerechneGesamtpreis liefert den Gesamtpreis. Wie jedoch der Gesamtpreis berechnet wird und welche Datentypen die Eigenschaften haben, wird nicht im Objekt definiert, sondern in einer Klasse. Durch die Trennung der Verwendung des Objekts (= Deklaration der Objektvariablen und Instanzierung des Objektes, siehe Kapitel 4.4.5) und der Implementierung (= Klasse) kann man Details wie die Berechnung des Angebots weglassen. So verringert man die Komplexität eines Problems und der Umgang mit einer komplexen Aufgabe wird vereinfacht. Die Trennung von Deklaration und Implementierung wird in der OOP Abstraktion genannt.

4.4.1 Klassen in Visual Studio anlegen

Um ein Objekt verwenden zu können, muss zuerst eine Klasse in Visual Studio angelegt werden. Bevor Sie jedoch Klassen anlegen können, müssen Sie entweder ein neues Konsolenprojekt oder ein bestehendes Projekt öffnen. Für diesen Arbeitsauftrag wäre es gut, wenn Sie ein neues Projekt mit dem Projektnamen „Hotelangebotssoftware" anlegen.

Klassen werden in Visual Studio in einem neuen oder vorhandenen Projekt wie folgt angelegt:

- *Markieren Sie im Projektmappen-Explorer das Projektelement (hier: Hotelangebotssoftware).*
- *Rufen Sie im Kontextmenü (rechte Maustaste) den Menüpunkt Hinzufügen (Add) und dann den Untermenüpunkt Klasse (Class) auf.*
- *Im Dialogfeld „Neues Element hinzufügen" (Add New Element) wird automatisch das Vorlagenelement Klasse ausgewählt und*
- *Sie können direkt den Klassennamen (hier Unterkunft) eingeben und*
- *mit Hinzufügen die neue Klasse hinzufügen.*



Visual Studio legt nun eine neue Datei namens „Unterkunft.cs" im Projektmappen-Explorer an, öffnet die Datei und präsentiert folgenden Code für die Klassendefiniton.

```
class Unterkunft
{
}
```

Die Definition einer Klasse ist also ganz einfach: Sie beginnt optional mit dem Zugriffsmodifizierer, gefolgt vom Keyword, dass und einem frei wählbaren Klassennamen. In die nächsten Zeilen kommen die üblichen geschwungenen Klammern, damit der Compiler erkennt, welcher Code zur Klasse gehört.

Der Zugriffsmodifizierer auf Klassenebene steuert die Sichtbarkeit der Klasse gegenüber anderen Programmen (z.B. Konsolenanwendungen, Klassenbibliotheken, Windows- Anwendungen etc.). Wenn man den Zugriffsmodifizierer weglässt, ist der Standard privat und eine Klasse oder ein Programm können von außen nicht auf die Klasse zugreifen. Innerhalb des Programms kann immer auf private Klassen zugegriffen werden. Erst wenn die Sichtbarkeit public verwendet wird, können auch andere Programme die Klasse verwenden.

Syntax

```
Zugriffsmodifizierer class Klassenname
{
    Klassenmember Felder;
    Klassenmember Eigenschaften;
    Klassenmember Methoden;
}
```

In der Klasse selbst sollen die Attribute und Operatoren aus dem OOP- bzw. UML- Diagramm implementiert werden. Dies geschieht in C# innerhalb der geschwungenen Klammern durch sogenannte Klassenmitglieder (class members). .NET unterscheidet unter anderem zwischen folgenden Arten von Klassenmitgliedern: In Feldern (fields) und Eigenschaften (properties) werden die Attribute bzw. Daten gespeichert und die Methoden implementieren die Operatoren. Innerhalb der Methoden wird noch zwischen Prozeduren und Funktionen unterschieden.

4.4.2 Klassenmember Feld

In Feldern werden die Daten eines Objekts gespeichert. Die Felder einer Klasse werden ähnlich wie eine Variable definiert:

Syntax

```
Zugriffsmodifizierer Datentyp Feldname
```

Auch hier gibt es wieder zuerst einen Zugriffsmodifizierer, der die Sichtbarkeit des Feldes steuert, ähnlich wie auf Klassenebene. Da wir in diesem Fall von außen auf das Feld zugreifen wollen, müssen wir public verwenden. Die Zugriffsmodifizierer private werden in Kapitel 4.7 beschrieben, protected in Kapitel 4.8.

```
public double Grundpreis;
public int Aufenthaltsdauer;
```

Hier werden die bekannten Felder wie Preis und Aufenthaltsdauer deklariert. Wie bei einer normalen Variablen kann man auch hier das Feld auf einen Anfangswert setzen. In diesem Fall wird das Feld Aufenthaltsdauer mit 7 vorbelegt.

```
public int Aufenthaltsdauer = 7;
```

4.4.3 Klassenmember Funktion

Wer bei der Bezeichnung Funktion an mathematische Funktionen denkt, liegt nicht ganz falsch. Beispielsweise liefert die Sinusfunktion eines Taschenrechners das Seitenverhältnis von Gegenkathete zu Hypotenuse zu einem eingegebenen Winkel zurück. In C# gibt es ebenfalls mathematische Funktionen, die benutzt werden können. Darüber hinaus können eigene Funktionen implementiert werden, wenn ein berechneter Wert benötigt wird. Dazu wird innerhalb der Klasse eine Funktion wie folgt definiert:

Syntax

```
Zugriffsmodifizierer RueckgabewertDatentyp Funktionsname()
{
    Datentyp Ergebnis;
    Anweisung 1;
    Anweisung n;
    return Ergebnis;
}
```

Nach der Deklaration des Zugriffsmodifizierers folgt der Datentyp des Rückgabewerts, da eine Funktion wie BerechneGesamtpreis den berechneten Wert liefert. Danach kommt der Name der Funktion, gefolgt von einer offenen und einer geschlossenen runden Klammer. Zwischen diesen Klammern können 0 oder mehrere Daten der Funktion in Form von Parametern übergeben werden (siehe Kapitel 4.5). In diesem Fall benötigt diese Funktion keine Parameter. Eine oder mehrere Anweisungen werden zwischen zwei geschwungenen Klammern (auf und zu) angegeben, damit der Compiler erkennt, dass der Code zur Funktion gehört. Mit dem Keyword return wird der ermittelte Wert an den Aufrufer weitergegeben.

Wird kein Zugriffsmodifizierer angegeben, ist die Sichtbarkeit standardmäßig auf privat eingestellt. Private Methoden können verwendet werden, wenn man eine öffentliche Methode mit vielen Codezeilen in mehrere kleinere Methoden aufteilen möchte oder Hilfsmethoden benötigt, aber diese Methoden nicht nach außen zur Verfügung stellen möchte.

Das folgende Code-Snippet zeigt eine typische Funktion. Hier wird der Gesamtpreis berechnet und abschließend mit return retourniert. Die Methode selbst kann auf andere Klassenmember wie AnzahlErwachsene zugreifen und auch verändern.

```
public double BerechneGesamtpreis()
{
    double Gesamtpreis = 0, GesamtpreisBrutto = 0, MwStABetrag = 0;
    Gesamtpreis = Grundpreis * Aufenthaltsdauer;
    MwStABetrag = Gesamtpreis * MwSt. / 100;
    GesamtpreisBrutto = Gesamtpreis + MwStABetrag;
    return GesamtpreisBrutto;
}
```

4.4.4 Klassenmember Prozedur

Wird kein Rückgabewert benötigt, sondern sollen lediglich eine oder mehrere Anweisungen ausgeführt werden, ist die Prozedur geeignet. Die Syntax lautet wie folgt:

Syntax

```
public void Prozedurename()
{
    Anweisung 1;
```

```
    Anweisung n;
}
```

Da kein Rückgabewert an den Aufrufer zurückgeliefert wird, ist kein return nötig. Als Rückgabedatentyp muss aber void deklariert werden. Das folgende Code-Snippet zeigt die Leistungsbeschreibung an.

```
public void ZeigeLeistungsBeschreibung()
{
    Console.WriteLine("Beherbergung ohne Verpflegung");
}
```

Ein return wird nur dann benötigt, wenn man die Prozedur vorzeitig verlassen möchte. Hinweis: Auch eine Funktion kann vorzeitig mit einem return verlassen werden. Da eine Prozedur keinen Rückgabewert liefert, gibt man bei return auch keine Variable an. Im folgenden Beispiel wird die Prozedur vorzeitig verlassen, wenn das Feld XYZ gleich 0 ist. (Die IF-Abfrage könnte natürlich auch ohne return auskommen, wurde aber zu Demonstrationszwecken mit return realisiert.)

```
public void TueEtwas()
{
    Console.WriteLine("Zeile 1");
    if (XYZ == 0)
        return;
    Console.WriteLine("Zeile 2");
}
```

4.4.5 Objekte instanzieren

Nachdem die Klasse mit ihren Feldern und/oder Methoden deklariert ist, kann sie im Programm verwendet werden. Bevor dies geschieht, muss sie deklariert werden. Da die Syntax für die Deklaration der Syntax einer Variablen ähnlich ist, wird auch von einer Objektvariablen gesprochen. Gleichzeitig wird mit der Implementierung einer Klasse auch ein neuer Datentyp definiert.

Syntax

```
Klassenname Objektvariable;
Unterkunft Kategorie;
```

Mit obiger Syntax wird allerdings nur die Objektvariable namens Kategorie deklariert. Um das Objekt verwenden zu können, muss sie mit new instanziiert werden, d.h. die .Net-Laufzeitumgebung reserviert Speicher im Computer.

Syntax

```
Objektvariable = new Klassenname();
```

Die Deklaration und Instanzierung kann entweder in einer oder zwei Zeilen implementiert werden: Variante 1 wird verwendet, wenn noch nicht klar ist, ob das Objekt von Haus aus instanziiert werden soll. Beispielsweise kann die Instanzierung von einer oder mehreren Bedingungen bzw. Kontrollstrukturen abhängen.

```
//Variante 1
Unterkunft Kategorie = null;
Kategorie = new Unterkunft();

// Variante 2
Unterkunft Kategorie = new Unterkunft();
```

Danach kann auf die einzelnen Klassenmember wie Felder und Methoden zugegriffen werden:

Syntax

```
Objektname.Feld = Wert;
KategorieA.Grundpreis = 70;
```

Es kann wie folgt ausgelesen werden:

Syntax

```
Wert2 = Objektname-Feld;
```

In diesem Code-Snippet wird der Grundpreis ausgelesen:

```
double Grundpreis2;
```

```
Unterkunft Kategorie = new Unterkunft();
KategorieA.Kategorie = "Komfortzimmer";
KategorieA.Grundpreis = 70;
Grundpreis2 = KategorieA.Grundpreis;
```

Von einer Klasse kann es nicht nur eine Instanz (= Objekt) geben, sondern beliebig viele. Die Klasse selbst implementiert die Eigenschaften und Methoden, welche ein Objekt beschreibt. Erst das Objekt selbst ist eine konkrete Ausprägung (= Instanz) der Klasse. Angewandt auf den Hotelbetrieb bedeutet das Folgendes: Die Klasse Unterkunft beschreibt die Attribute wie Kategorie und die maximale Anzahl der Gäste einer Unterkunft, sowie die Berechnungsvorschrift zum Berechnen eines Gesamtpreises. Die Klasse entspricht daher eher der Beschreibung einer Unterkunft, wie man sie in einem Prospekt oder auf einer Homepage im Internet finden würde.

Hingegen ist ein Objekt eine konkrete Ausprägung einer Klasse. Einem Gast kann mehr als eine Unterkunft angeboten werden. Im folgenden Beispiel werden einem Gast sowohl eine Unterkunft der Kategorie A als auch eine der Kategorie B angeboten. Für jede Unterkunft wird ein Objekt instanziiert und in einer eigenen Objektvariablen gespeichert. So repräsentiert das Objekt Unterkunft1 ein(e) Angebot(sposition) für eine bestimmte Aufenthaltsdauer und Unterkunftskategorie.

```
double Gesamtpreis = 0;
Unterkunft Unterkunft1 = new Unterkunft();
Unterkunft1.Preis = 80;
Unterkunft1.Aufenthaltsdauer = 7;
Gesamtpreis = Gesamtpreis + Unterkunft1.BerechneGesamtpreis();

Unterkunft Unterkunft2 = new Unterkunft();
Unterkunft2.Preis = 60;
Unterkunft2.Aufenthaltsdauer = 7;
Gesamtpreis = Gesamtpreis + Unterkunft2.BerechneGesamtpreis();
```

4.4.6 Entwurf und Realisierung

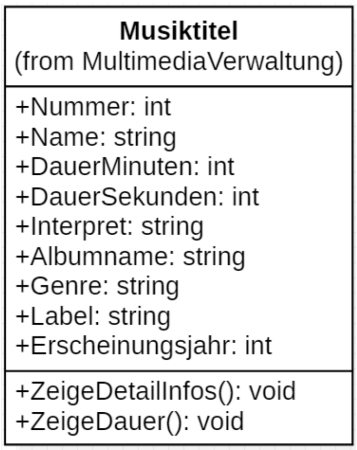
Lösung zum Arbeitsauftrag

Mit diesem Wissen kann das erste objektorientierte Programm entwickelt und realisiert werden. Der Entwurf dieser Klasse wurde bereits in Kapitel 4.3 beschrieben. Daher folgt hier nur die Implementierung.

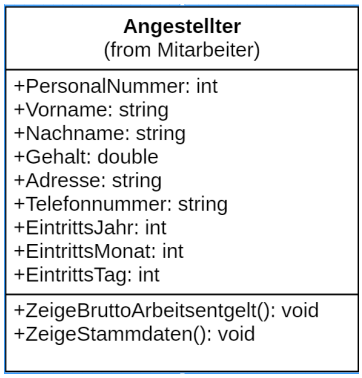
Quellcode siehe Lösungsdateien

Aufgaben

1. Implementieren Sie die Klasse Musiktitel laut UML-Diagramm. Instanzieren Sie die Klasse in der Main-Methode und befüllen Sie die Objektvariable mit Werten. Rufen Sie anschließend die Operatoren Zeigedauer und ZeigeDetailInfos auf. Zeigedauer greift auf die Felder DauerMinute und DauerSekunde zu und gibt diese im Format mm:ss aus, beispielsweise 1:20 für 1 Minute und 20 Sekunden.



2. Implementieren Sie die Klasse Angestellter laut UML-Diagramm. Instanziieren Sie die Klasse in der Main-Methode und befüllen Sie die Objektvariable mit Werten. Rufen Sie anschließend die Operatoren Zeigestammdaten und ZeigeBruttoArbeitsentgeIt auf. ZeigeBruttoArbeitsentgelt gibt das Gehalt aus. Instanziieren Sie in der Main-Methode eine Objektvariable, befüllen Sie diese mit Testwerten und rufen Sie die beiden Methoden hintereinander auf.



4.5 Methoden mit Parametern

Arbeitsauftrag

Implementieren Sie eine Prozedur mit Parameter.

Wie in Kapitel 4.4.3 erwähnt, kann einer Prozedur oder Funktion Null oder mehrere Daten in Form von Parametern übergeben werden:

Syntax

Syntax für Funktionen mit Parametern:

```
Zugriffsmodifizierer Datentyp Methodenname(Datentyp Parametername1, Datentyp
Parametername2, Datentyp ParameternameN)
{
    Anweisung(en);
    return (Datentyp);
}
```

Syntax für Prozeduren mit Parametern:

```
Zugriffsmodifizierer void Methodenname(Datentyp Parametername1, Datentyp Parametername2,
void ParameternameN)
{
    Anweisungen);
    return ; // return ist optional, muss nicht angegeben werden
}
```

Die Deklaration der Methode beginnt wie bei einer Funktion oder Prozedur ohne Parameter, nur dass zusätzlich zwischen den Klammern die Parameterliste deklariert wird. Diese kann aus einem oder mehreren Parametern bestehen. Mehrere Parameter werden durch Kommas getrennt. Auch hier wird der Parameter wie eine Variable deklariert: zuerst der Datentyp, dann der Parametername.

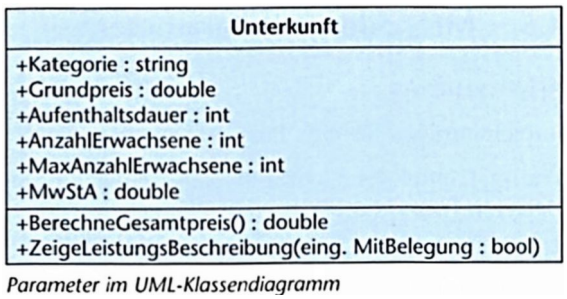
Die folgende Methode hat einen Parameter namens Mitbelegung vom Datentyp Boolean. Dieser Parameter steuert, ob zusätzlich neben der Leistungsbeschreibung auch die Belegung ausgegeben wird oder nicht.

```
public void ZeigeLeistungsBeschreibung(Bool Mitbelegung)
{
}
```

Im UML-Klassendiagramm werden die Parameter im Operator zwischen zwei runden Klammern angegeben und der Datentyp wie gewohnt mit Doppelpunkt eingeleitet.

Der Aufruf der Methode im Programm (z.B. Main) funktioniert wie folgt: Zwischen den Klammern wird der Wert übergeben, in diesem Fall true.

Unterkunft1.ZeigeLeistungsBeschreibung(true);



Parameter im UML-Klassendiagramm

Auch einer Funktion können 1 bis n Parameter übergeben werden.

4.5.1 Parameter out und ref

Wenn eine Methode einen Rückgabewert liefern soll, wurde bisher eine Funktion in Kombination mit dem Keyword return verwendet. Eine Methode kann auch einen oder mehrere Werte über Parameter zurückgeben. Bisher wurden Parameter nur als sogenannte Eingabeparameter verwendet.

Wenn ein Parameter in einer Prozedur oder Funktion auch einen Wert zurückliefern soll, muss dieser mit dem Keyword ref oder out gekennzeichnet werden.

ref und out verhalten sich fast gleich, nur mit dem Unterschied, dass bei ref die Variable auf Aufrufseite initialisiert werden muss und bei out nicht.

Syntax

```
Zugriffsmodifizierer RueckgabewertDatentyp Funktionsname(out int Parameter1, ref double
Parameter2)
{
    Datentyp Ergebnis;
    Parameter1 = 3;
    Parameter2 = 4;
    Anweisung 1;
    Anweisung n;
    return Ergebnis;
}
```

Möchte man vom Gesamtpreis auch die Mehrwertsteuer zurückbekommen, könnte man entweder eine eigene MwSt.-Funktion implementieren oder diese per out oder ref Parameter retournieren lassen.

```
public double BerechneGesamtpreis(out double MwStABetrag)
{
    double Gesamtpreis = O, GesamtpreisBrutto = 0;

    Gesamtpreis = Grundpreis * Aufenthaltsdauer;
    MwStABetrag = Gesamtpreis * MwSt. / 100;
    GesamtpreisBrutto = Gesamtpreis + MwStABetrag;
    return GesamtpreisBrutto;
}
```

Der Aufruf sieht damit wie folgt aus:

```
double MwStABetrag;
double Gesamtpreis = EinfacheUnterkunft.BerechneGesamtpreis(out MwStABetrag);
```

4.5.2 Entwurf und Realisierung

Lösung zum Arbeitsauftrag

Erweitern Sie die vorhandene Methode ZeigeBelegung in der Klasse Unterkunft in der Datei Unterkunft.cs wie folgt:

```
public void ZeigeLeistungsBeschreibung(bool Mitbelegung)
{
    Console.WriteLine("Beherbergung ohne Verpflegung");
    if (Mitbelegung)
        Console.WriteLine("für {0} Personen", AnzahlGaeste);
}
```

Fügen Sie der Klasse Programm die statische Methode EingabeDialogFuerlnt hinzu (statische Methoden siehe Kapitel 4.12). Diese Methode übernimmt die Aufgabe für die Eingabe von int-Werten. Damit lässt sich die Anzahl an Codezeilen für die Eingabe reduzieren, da nunmehr die Methode aufgerufen werden muss.

```
class Programm
{
    static void Main(string[] args)
    {
        Unterkunft EinfacheUnterkunft = new Unterkunft();
        EinfacheUnterkunft.Kategorie = "Einfache Unterkunft";
        EinfacheUnterkunft.Grundpreis = 50;
        EinfacheUnterkunft.Aufenthaltsdauer = EingabeDialogFuerInt("Eingabe
                                                                   Aufenthaltsdauer:");
```

```
        EinfacheUnterkunft.AnzahlGaeste = EingabeDialogFuerInt("Eingabe AnzahlGaeste:");
        EinfacheUnterkunft.MaxAnzahlGaeste = 2;

        double Gesamtpreis = EinfacheUnterkunft.BerechneGesamtpreis();
        Console.WriteLine("Gesamtpreis = {0}", Gesamtpreis);
        EinfacheUnterkunft.ZeigeLeistungsBeschreibung(true);
        Console.ReadKey();
    }

    private static int EingabeDialogFuerInt(string EingabeAufforderung)
    {
        string Eingabe;
        Console.Write("{0}:", EingabeAufforderung);
        Eingabe = Console.ReadLine();
        return Convert.ToInt32(Eingabe);
    }
}
```

Aufgaben

1. Erweitern Sie die Klasse Musiktitel um eine neue Methode namens SetzeDauer. Diese hat die Parameter pDauerMinute und pDauerSekunde, welche die entsprechenden Feldvariablen setzt, wobei der Wert von pDauerSekunden 59 nicht überschreiten darf.
2. Erweitern Sie die Klasse Angestellter um eine neue Methode namens SetzeEintrittsDatum. Diese hat die Parameter pJahr, pMonat und pTag, welche die entsprechenden Feldvariablen setzen. Rufen Sie in Main anstelle der Properties die neue Methode auf.

4.6 Konstruktoren

Arbeitsauftrag

Erweitern Sie die Klasse Unterkunft um Konstruktoren.

Mit dem Keyword new wird ein Objekt instanziiert. Während des Instanziierungsvorgang ruft .NET den Konstruktor auf. Das ist eine Methode, die den gleichen Namen wie die Klasse hat. Der Konstruktor ist standardmäßig vorhanden, auch wenn er in der Klasse nicht definiert ist. Im Konstruktor können verschiedene Initialisierungsanweisungen aufgerufen werden. Beispielsweise kann man die Felder auf einen Anfangswert setzen. Dem Konstruktor können auch ein oder mehrere Parameter übergeben werden, um dem Objekt schon beim Instanziierungsvorgang verschiedene Werte übergeben zu können.

Syntax

Syntax ohne Parameter:

```
public Klassenname()
{
    Anweisungen;
}
```

Syntax mit Parameter:

```
public Klassenname(Datentyp Paramter1, Datentyp Parameter2, Datentyp ParameterN)
{
    Anweisungen;
}
```

Nach dem Zugriffsmodifizierer public wird der Name der Klasse wiederholt, gefolgt von zwei runden Klammern. Jedoch ist hier kein Datentyp nötig, auch nicht void. Zwischen den runden Klammern werden keine, ein oder mehrere Parameter definiert. Die Deklaration erfolgt wie bei den Methoden: Mehrere Parameter werden durch Kommas getrennt. Innerhalb der geschwungenen Klammern wird der Initialisierungscode implementiert.

Soll der Standardkonstruktor, der Konstruktor ohne Parameter, nicht verwendet werden, muss dieser explizit auf private gesetzt werden, siehe Arbeitsauftrag. Damit muss man beim Instanziieren Anfangswerte mitgeben.

4.6.1 Entwurf und Realisierung

Lösung zum Arbeitsauftrag

Fügen Sie der Klasse Unterkunft folgende Konstruktoren hinzu:

```
private Unterkunft() { }

public Unterkunft(string pKategorie, double pGrundpreis, int pMaxAnzahlGaeste)
{
    Kategorie = pKategorie; Grundpreis = pGrundpreis;
    MaxAnzahlGaeste = pMaxAnzahlGaeste;
}
```

Ändern Sie den Source Code in der Methode Main wie folgt:

```
private static void Main(string[] args)
{
    Unterkunft Unterkunft1 = new Unterkunft("Kategorie", 50, 2);
    Unterkunft1.Aufenthaltsdauer = EingabeDialogFuerInt("Aufenthaltsdauer");
    Unterkunft1.AnzahlGaeste = EingabeDialogFuerInt("Anzahl Gäste");
    Console.WriteLine(Unterkunft1.BerechneGesamtpreis());
    Console.ReadKey();
}
```

Aufgaben

1. Erweitern Sie die Klasse Musiktitel um einen Konstruktor. Dieser hat die Parameter pNummer und pName, welche die entsprechenden Feldvariablen setzen. Setzen Sie den Zugriffsmodifizierer des Standardkonstruktors auf private und passen Sie den Code in Main an.
2. Erweitern Sie die Klasse Angestellter um einen Konstruktor. Dieser hat die Parameter pPersonalNummer, pVorname und pNachname, welche die entsprechenden Feldvariablen setzen. Setzen Sie den Zugriffsmodifizierer des Standardkonstruktors auf private und passen Sie den Code in Main an.

4.7 OOP-Konzept Datenkapselung

Arbeitsauftrag

Erweitern Sie die Klasse Unterkunft so, dass bestimmte Felder nicht überschrieben werden können.

Auf die Eigenschaften wie Grundpreis kann aufgrund des Zugriffsmodifizierers public von außen unkontrolliert zugegriffen werden. Im vorletzten Arbeitsauftrag wurde der Grundpreis auf 50 € gesetzt. Der Grundpreis kann jedoch auch (unabsichtlich) auf andere Werte gesetzt werden.

4.7.1 Zugriffsmodifizierer private

Daher gibt es neben public auch noch folgende wichtige Zugriffsmodifizierer:

Zugriffsmodifizierer	Bedeutung
public	Zugriff von außen, auch von anderen Programmen
private	Kein Zugriff von außen, nur innerhalb der eigenen Klasse
protected	Zugriff innerhalb der eigenen und abgeleiteten Klassen, siehe Kapitel 4.8
internal	Der Zugriff ist auf die aktuelle Assembly beschränkt.

Da das Klassenmember Grundpreis nicht von außen gesetzt werden soll, muss das Feld Grundpreis auf private gesetzt werden. Verwenden Sie dafür das Programm aus dem letzten Arbeitsauftrag oder kopieren Sie das Visual-Studio-Projekt über den File Explorer in ein neues Verzeichnis und arbeiten Sie mit einer Kopie.

```
private double Grundpreis;
```

Wenn Sie nun das Programm aus dem vorherigen Arbeitsauftrag so ändern, dass der Grundpreis überschrieben wird,

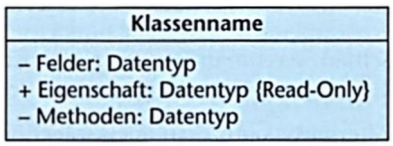
```
private static void Main(string[] args)
{
    Unterkunft Unterkunft1 = new Unterkunft("Kategorie", 50, 2);
    Unterkunft1.Grundpreis = 66;
    Unterkunft1.Aufenthaltsdauer = EingabeDialogFuerInt("Aufenthaltsdauer");
    Unterkunft1.AnzahlGaeste = EingabeDialogFuerInt("Anzahl Gäste");
    Console.WriteLine(Unterkunft1.BerechneGesamtpreis());
    Console.ReadKey();
}
```

erhalten Sie folgende Fehlermeldung in der Fehlerliste:

„Der Zugriff auf „Hotelangebotssoftware-Unterkunft-Grundpreis" ist aufgrund der Sicherheitsebene nicht möglich."

Die Fehlermeldung hat insofern ihre Richtigkeit als in der Main-Methode der Grundpreis auf 66 € gesetzt wird. Wie kann nun von außen auf dieses Feld zugegriffen werden? Für den Zugriff auf private Felder können Eigenschaften (Properties) verwendet werden.

In UML wird der Zugriffsmodifizierer private als Minuszeichen dargestellt, das ReadOnly-Property (siehe Kapitel 4.7.4) wird mit {ReadOnly} gekennzeichnet. Manche UML-Programme verwenden auch das Wort (unveränderlich).



UML-Datenkapselung

4.7.2 Klassenmember Eigenschaften

Mit Eigenschaften kann man lesend und/oder schreibend auf gekapselte Daten in der Klasse zugreifen. Die Syntax dafür lautet wie folgt:

Syntax

```
public double PropertyName
{
    get { return Feldname; }
    set { Feldname = value; }
}
```

Die Deklaration einer Property hat Ähnlichkeiten mit einer gewöhnlichen Felddeklaration. Um Daten zu setzen und auslesen zu können gibt es noch zusätzlich die Keywords set und get. Im get-Teil wird der ermittelte Wert mit return an den Aufrufer zurückgegeben. In diesem Teil können, wie in einer Funktion, beliebige Anweisungen implementiert werden. D. h., der Rückgabewert muss nicht unbedingt ein Feld sein, sondern kann auch anders ermittelt werden. Im set-Teil bekommt man den von außen gesetzten Wert mit dem Keyword value. Aber auch hier könnte man eine oder mehrere Anweisungen implementieren, z.B. ob der Wertebereich erlaubt ist.

Möchte man für das Property den gleichen Namen verwenden wie für das Feld, gibt es verschiedene Namenskonventionen, in diesem Modul wird der Präfix m für private Felder verwendet. Die von außen zugänglichen Properties haben keine Präfixe im Namen, damit sie besser lesbar sind.

```
private int mAufenthaltsdauer;
public int Aufenthaltsdauer
{
    get {return mAufenthaltsdauer;}
    set
    {
        Aufenthaltsdauer = -1;
        if (value > 0)
            mAufenthaltsdauer = value;
    }
}
```

Im obigen Code-Snippet liefert der get-Teil den Inhalt des Feldes Aufenthaltsdauer zurück. Im set-Teil wird das Feld nur gesetzt, wenn die Aufenthaltsdauer größer als 0 ist.

Von außen betrachtet ändert sich für den Zugriff auf den Preis nichts, da sich aus Verwendungssicht ein Feld und eine Eigenschaft gleich verhalten, nur mit dem Unterschied, dass man bei einer Eigenschaft den Zugriff auf die Daten kontrollieren kann. Genau diese Kapselung ist eines der vier Konzepte der OOP.

Alternativ kann man auch folgende Kurzschreibweise verwenden:

Syntax

```
Zugriffsmodifizierer Datentyp PropertyName {get; set;}
```

Wie im Code-Snippet zu sehen ist, verschmelzen Feld und Property. Der get-Teil benötigt kein return mehr und der set-Teil kein value. Diese Kurzschreibweise funktioniert jedoch nur, wenn man keine weiteren Anweisungen in einer Property verwenden möchte. Eine Überprüfung des value, wie weiter oben beschrieben (if (value > 0)), ist mit dieser Syntax nicht möglich.

```
public int Aufenthaltsdauer{ get; set; }
```

In der Property kann man entweder den get- oder den set-Teil weglassen, falls man nur setzende oder nur lesende Properties verwenden möchte.

4.7.3 WriteOnly-Property

Wenn der get-Teil der Property weggelassen wird, erhält man eine WriteOnly-Property. Dies ist beispielsweise nützlich, wenn man ein Passwort setzen, aber verhindern möchte, dass man es wieder auslesen kann.

Syntax

```
public double PropertyName
{set
    {
    FieldName = value;
    }
}
```

Alternativ ist folgende Syntax möglich:

Syntax

```
Zugriffsmodifizierer Datentyp PropertyName {private get; set;}
```

Mit dem Keyword private set kann der Code nur innerhalb der Klasse zugreifen.

```
public int Passwort {private get; set; }
```

4.7.4 ReadOnly-Property

Wenn der set-Teil der Property weggelassen wird, erhält man eine ReadOnly-Property. Damit kann man verhindern, dass man Werte von außen verändert.

Syntax

```
public double PropertyName
{
    get { return FieldName;}
}
```

Liefert die Property ein Field zurück, muss das Field entweder in der Deklaration mit einem Anfangswert initialisiert und/oder im Konstruktor gesetzt werden.

Alternativ ist folgende Syntax möglich:

Syntax

```
Zugriffsmodifizierer Datentyp PropertyName (get; private set;)
```

Mit dem Keyword private set kann der Code nur innerhalb der Klasse die Property setzen.

```
public int Grundpreis {get; private set;}
```

4.7.5 Entwurf und Realisierung

Lösung zum Arbeitsauftrag

Die Klasse Unterkunft wird wie folgt angepasst:

```
public class Unterkunft
```

Quellcode siehe Lösungsdateien

Aufgaben

1. Ändern Sie in der Klasse Musiktitel folgende Felder auf ReadOnly: Nummer und Name, DauerSekunde, DauerMinute.

2. Ändern Sie in der Klasse Angestellter folgende Felder auf ReadOnly: PersonaINummer, Vorname und Name sowie EintrittsJahr, -Monat und -Tag.

4.8 OOP-Konzept Vererbung

Arbeitsauftrag

Erweitern Sie das Programm um folgende Unterkunftsarten: Zimmer (Berechnungsbasis pro Person) und Apartment (Endreinigung wird extra berechnet).

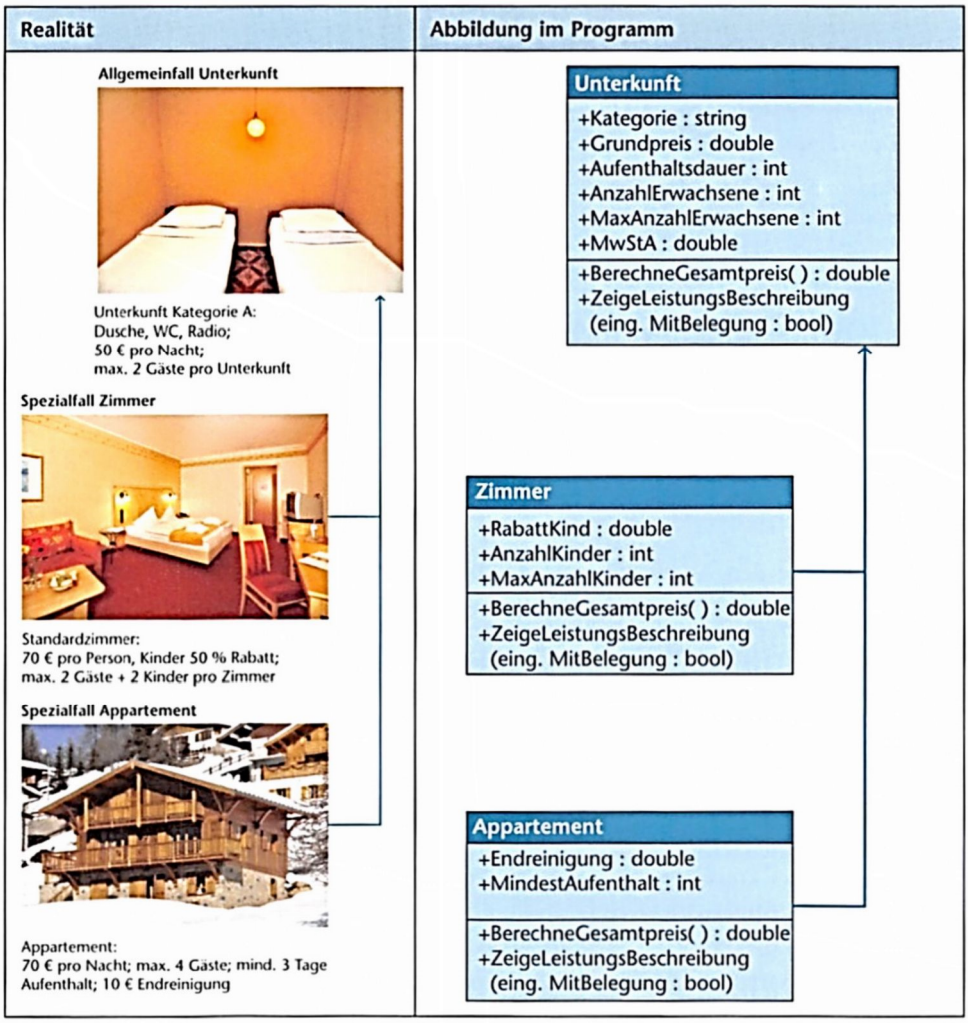
Wie würden Sie den Arbeitsauftrag nach bisherigem Wissensstand umsetzen? Wahrscheinlich würden Sie für jede neue Unterkunftsart eine eigene Klasse implementieren. Dabei würde sich die Unterkunftsart Apartment lediglich dadurch unterscheiden, dass ein bestimmter Preis für die Endreinigung dazukommt. Ergäbe es daher Sinn, den kompletten Source Code der Klasse Unterkunft zu duplizieren, nur um den Spezialfall Endreinigung im Programm abzubilden?

Vermutlich nicht. Viel eher würde es Sinn ergeben, die Klasse Unterkunft wieder zu verwenden und aus dem Allgemeinfall einen Spezialfall zu machen. Genau hier kommt die Vererbung ins Spiel. Mithilfe der Vererbung kann aus einer Generalisierung eine Spezialisierung gemacht werden. Eine weitere Klasse namens Apartment erbt die Klassenmitglieder der Klasse Unterkunft und kann diese erweitern und verändern. Die neue Klasse namens Apartment verwendet die Methode BerechneGesamtpreis aus der Klasse Unterkunft und addiert lediglich den Preis für die Endreinigung.

Die Unterkunftsart Zimmer basiert hingegen auf einer anderen Berechnungsmethode: Hier wird der Preis pro Person berechnet. Trotzdem kann auch bei diesem Spezialfall die Vererbung verwendet werden, da das Verhalten einzelner Klassenmitglieder wie BerechneGesamtpreis komplett überschrieben werden kann. Mit dem Konzept der Vererbung werden auch folgende neue Begriffe eingeführt: Basisklasse, Unterklasse und Ableiten. Die Basisklasse repräsentiert den Allgemeinfall, hier die Klasse Unterkunft. Unterklassen sind jene Klassen (wie Apartment oder Zimmer), welche die Klassenmitglieder von der Basisklasse erben. Um die Vererbung zu realisieren muss die Unterklasse von der Basisklasse ableiten.

Mithilfe der Vererbung lassen sich auch Hierarchien darstellen, daher spricht man im Zusammenhang mit der Vererbung auch von Klassenhierarchien. So lassen sich neben Generalisierung und Spezialisierung auch verschiedene natürliche oder künstliche Hierarchien mithilfe der Vererbung realisieren.

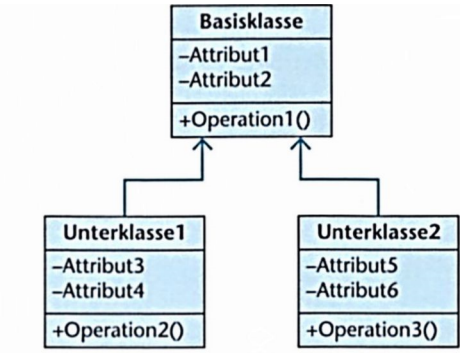
- *Unterkunft (Basisklasse)*
  - *Zimmer (Unterklasse: erbt Klassenmitglieder von Basisklasse)*
  - *Apartment (Unterklasse: erbt Klassenmitglieder von Basisklasse)*



Vererbung: Generalisierung und Spezialisierung

4.8.1 UML-Diagramm

Im UML-KIassendiagramm wird die Vererbung durch einen Pfeil symbolisiert, wobei dieser von der Unterklasse in Richtung Basisklasse zeigt.



UML-Syntax für Vererbung

4.8.2 Klassen ableiten

Die Syntax für die Vererbung bzw. Ableitung lautet wie folgt:

Syntax

```
Zugriffsmodifizierer Unterklasse : Basisklasse
```

Nach dem Namen der Unterklasse werden ein Doppelpunkt und der Name der Basisklasse angegeben. Durch die Vererbung kann nun die Unterklasse auf alle öffentlichen (public) und geschützten (protected) Klassenmember zugreifen:

```
class Zimmer : Unterkunft
```

4.8.3 Klassenmember der Basisklasse verwenden

Durch die Vererbung können bestehende Klassen um neue Funktionalitäten erweitert und bestehende Funktionalität wiederverwendet werden. Dabei wird die Funktionalität entweder neu implementiert oder die bestehende erweitert.

Beispielsweise kann die Methode BerechneGesamtpreis in der Unterklasse Apartment die Basismethode aufrufen und um die neue Funktionalität (Endreinigung addieren) erweitern. Durch die Angabe des Keywords base erkennt der Compiler, dass ein Klassenmember der Basisklasse aufgerufen werden soll (und nicht das der Unterklasse). Ist das Klassenmember eindeutig in der Klassenhierarchie (wie beispielsweise Grundpreis), kann das Keyword base weggelassen werden.

```
public double BerechneGesamtpreis()
{
    double Gesamtpreis = -1;
    if (Aufenthaltsdauer >= MindestAufenthalt)
        Gesamtpreis = base.BerechneGesamtpreis() + Endreinigung;
    return Gesamtpreis;
}
```

Man könnte jedoch auch die Funktionalität aus der Basisklasse komplett (oder teilweise) außer Acht lassen und neu implementieren (hier werden nur die gemeinsamen Eigenschaften aus der Basisklasse verwendet, aber nicht die Methode BerechneGesamtpreis).

```
public double BerechneGesamtpreis()
{
    double Gesamtpreis = 0, GesamtpreisBrutto = 0;
    Gesamtpreis = Grundpreis * Aufenthaltsdauer;
    GesamtpreisBrutto = Gesamtpreis + BerechneMwStA(Gesamtpreis);
    return GesamtpreisBrutto;
}
```

4.8.4 Zugriffsmodifizierer protected

Wenn Sie aus einer abgeleiteten Klasse wie Apartment auf ein privates Klassenmember der Basisklasse wie Aufenthaltsdauer zugreifen wollen, kommt die folgende Fehlermeldung:

„Der Zugriff auf "Hotelangebotssoftware.Unterkunft.mAufenthaltsdauer" ist aufgrund der Sicherheitsebene nicht möglich."

Der Grund liegt darin, dass der Setter der Eigenschaft Grundpreis private ist. Daher muss in der Basisklasse der Zugriffsmodifizierer von private auf protected geändert werden:

```
protected int mAufenthaltsdauer;
```

Mit protected können auch abgeleitete Klassen auf Klassenmember (also auch Eigenschaften und Methoden) zugreifen. Von außen ist jedoch ein Zugriff nicht möglich. Protected kann nicht nur für Eigenschaften, sondern auch für andere Klassenmitglieder wie Felder und Methoden verwendet werden.

4.8.5 Konstruktoren in Unterklassen

Verfügt eine Basisklasse über einen Konstruktor mit Parameter, kann die Unterklasse den Konstruktor wie folgt aufrufen.

Syntax

```
Zugriffsmodifizierer Konstruktorname (Parameterliste Unterklasse) : base(Parameterliste
Basisklasse)
{
}
```

Nach dem Konstruktor der Unterklasse wird ein Doppelpunkt und das Keyword base angegeben. Zwischen runden Klammern werden dem Konstruktor die notwendigen Parameter übergeben. Die Anzahl der Parameter zwischen

Unter- und Basisklasse kann unterschiedlich sein. Die Unterklasse sollte jedoch die Parameter der Basisklasse enthalten.

4.8.6 Typumwandlung und späte Bindung

Um zwei Unterkunftsarten gleichzeitig anzubieten, würden nach dem bisherigen Wissensstand Objekte der Klassen Zimmer und Apartment in jeweils eigene Objektvariablen gespeichert werden. Andererseits ist erst zur Laufzeit bekannt, ob ein Objekt Zimmer oder Apartment instanziiert wird, da es Kunden gibt, die nur eine Unterkunftsart anfragen oder mehrere. Daher wäre es praktisch, wenn man jede Unterkunftsart einer Klassenhierarchie in eine Objektvariable speichern und erst zur Laufzeit den Typ bestimmen könnte. Genau hier kommt das Casten von Objekten (Typumwandlung) ins Spiel. Die Vererbung erlaubt die Speicherung von Objekten einer Klassenhierarchie in eine Objektvariable vom Typ der Basisklasse (oder Typ einer darunterliegenden Hierarchie, wenn die Klassenhierarchie mehrere Ebenen haben sollte). Das hat den Vorteil, dass man zur Deklarationszeit noch nicht wissen muss, welches Objekt der Klassenhierarchie instanziiert wird. Dazu sind die Anweisungen laut folgendem Code-Snippet notwendig. Zuerst wird eine Objektvariable vom Typ einer Basisklasse deklariert. Entweder existiert bereits ein Objekt vom Typ Unterklasse, dann braucht man es nur einfach zuweisen, oder man instanziiert die Variable beim Deklarieren.

Syntax

```
Basisklasse Objektvariable = Objekt vom Typ Unterklasse;
```

Syntax

```
Unterklasse ObjektUnterklasse = new Unterklasse(...);
Basisklasse Objektvariable = ObjektUnterklasse;
```

Im folgenden Code-Snippet hat die Objektvariable UnterkunftsObjekt den Datentyp der Basisklasse Unterkunft. Dem Unterkunftsobjekt kann nun jede Unterklasse einer Klassenhierarchie zugewiesen werden. In diesem Fall wird das UnterkunftsObjekt schon bei der Deklaration mit dem ApartmentObjekt instanziiert:

```
Unterkunft UnterkunftsObjekt = new Apartment(50, 10, 3, 7, "Apartment 35m2", 2, 2);
```

In diesem Code-Snippet wird das Objekt Apartment implizit in das Objekt UnterkunftsObjekt umgewandelt. Dieser Fall wird auch downcast genannt (da von einer Unterklasse auf eine Basisklasse herunter gewandelt wird). Hier geschieht die Typumwandlung automatisch. Da das UnterkunftsObjekt vom Typ Unterkunft ist, stehen auch nur die Klassenmitglieder der Unterklasse zur Verfügung. Möchte man auf die Klassenmember der Unterklasse zugreifen, muss die Basisklasse mit einer expliziten Typumwandlung (upcast) in die Unterklasse umgewandelt werden.

Syntax

```
Unterklasse Objektvariable = (Unterklasse)Basisklasse;
```

Zwischen zwei runden Klammern wird der Typ angegeben, in den umgewandelt werden soll.

```
Apartment ApartmentObjekt2 = (Apartment)UnterkunftsObjekt;
double Endreinigung = ApartmentObjekt2.Endreinigung;
```

Im Code-Snippet oben haben wir den Datentyp der Objektvariable UnterkunftsObjekt schon bei der Deklaration festgelegt. Dies wird auch frühe Bindung oder early binding genannt. Da noch nicht bekannt ist, welche Unterkunftsart der Gast haben möchte, müssen wir die späte Bindung verwenden. Diese wird auch late binding genannt. Hierbei entscheiden wir erst bei Laufzeit, welche Unterklasse benannt werden soll.

Die späte Bindung funktioniert wie folgt:

- *Deklaration eines Objekts vom Typ Basisklasse (ohne Zuweisung eines Objekts vom Typ Unterklasse; es wird jedoch empfohlen das Objekt mit null zu initialisieren)*
- *eine oder mehrere Anweisungen, die ermitteln, welche Unterklasse verwendet werden soll*
- *Zuweisung oder Instanzierung des Objekts vom Typ Basisklasse mit einem Objekt vom Typ Unterklasse*

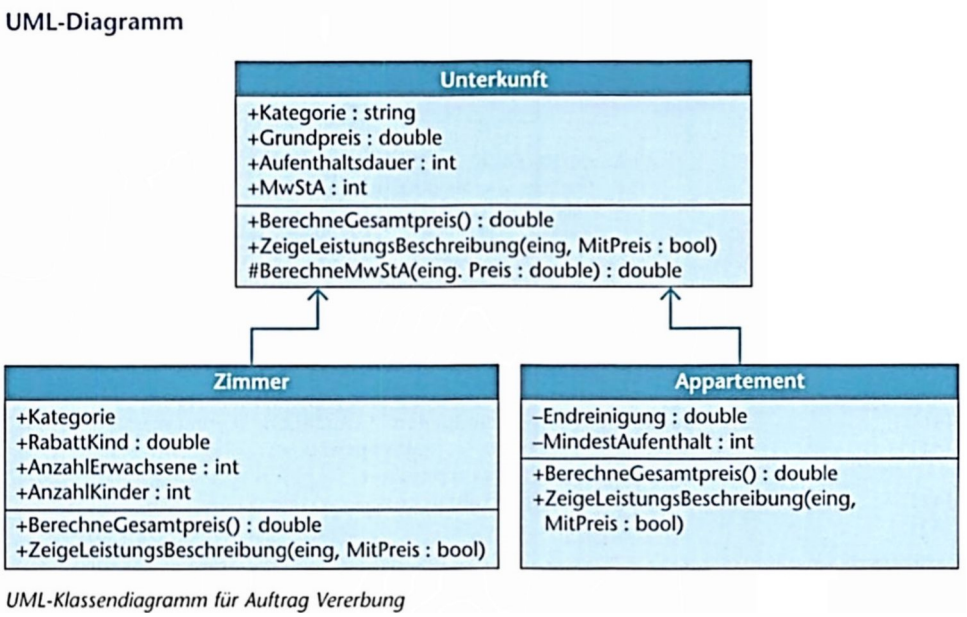
Folgendes Code-Snippet soll den Vorgang verdeutlichen:

```
Unterkunft UnterkunftsObjekt = null;
// ... Entscheidung, welches Objekt verwendet werden soll
UnterkunftsObjekt = new Apartment(50, 10, 3, 7, "Apartment 35m2", 2, 2);
```

Wird im Programm nur mit einer Objektvariablen vom Typ Basisklasse gearbeitet, hat das den Vorteil, dass abhängig von unterschiedlichen Bedingungen verschiedene Unterklassen distanziert werden können. Damit lässt sich der Arbeitsauftrag wie folgt realisieren.

4.8.7 Entwurf und Realisierung

Lösung zum Arbeitsauftrag



UML-Klassendiagramm für Auftrag Vererbung

Quellcode siehe Lösungsdateien

In der Methode BerechneGesamtpreis wird der Gesamtpreis nur dann berechnet, wenn der Mindestaufenthalt erreicht wurde. Wird der Mindestaufenthalt unterschritten, retourniert die Methode -1, Der Aufrufer der Methode AngebotBerechnen, entsprechend eine Ebene höher in der Main-Methode, muss verhindern und prüfen, ob Angebote mit -1 übernommen werden und darauf reagieren. Ein Beispiel dafür wäre, dass der Benutzer die Eingabe wiederholt.

Es können aber auch Exception verwendet werden (siehe Kapitel 5.2). In diesem Source Code wird die Prüfung aus Platzgründen ausgelassen.

Die Unterkunftsklassen berechnen den Preis für eine Unterkunftsart. Ein Gast kann jedoch beispielsweise zwei Zimmer der Kategorie Standard und ein Zimmer der Kategorie Komfort anfragen. D.h., eine Angebotsposition besteht aus einer Anzahl und einer Unterkunftsart. Daher wird eine eigene Klasse namens Angebotsposition mit den Klassenmembern Anzahl und Unterkunftsobjekt implementiert.

Hier zeigt sich nun die Stärke der Vererbung bzw. des Castens: Das Klassenmember UnterkunftsObjekt kann nun jedes von Unterkunft abgeleitete Objekt wie Zimmer und Apartment speichern. Erst zur Laufzeit entscheidet der Benutzer (oder die Programmlogik), welche Unterklasse erzeugt und zugewiesen wird.

Die Angebotspositionen werden in einer Liste vom gleichnamigen Typ gespeichert. Die Anzahl wird auf 20 beschränkt. In den nachfolgenden Kapiteln werden dynamische und generische Listen vorgestellt. Bei dieser Listenart ist die Anzahl der Listenelemente nicht starr, sondern es können fast beliebige Listenelemente hinzugefügt oder entfernt werden.

Die Methode Console.WriteLine bietet bei Platzhaltern für die Variablen eine weitere Option an: die maximale Anzahl an Zeichen. Damit können die Daten für die Angebotsposition etwas besser formatiert werden.

Wenn Sie nun ein Beispiel wie im Screenshot durchspielen, wird nach der Eingabe der Einzelpreis der eingegebenen Unterkunft angezeigt und nach der Eingabe E werden alle Angebotspositionen aufgelistet. Bei genauerer Analyse können Sie erkennen, dass die Einzelpreise nach der Eingabe und in der Auflistung der Angebotspositionen unterschiedlich sind. Auch werden in der Angebotsposition 1 (Standardzimmer) nicht die Anzahl der Kinder aufgelistet, obwohl die Methode BerechneGesamtpreis in der Klasse Zimmer angepasst wurde.

Eine Begründung für diese Differenz finden Sie in Kapitel 4.9.

Aufgaben

1. Erweitern Sie die Multimedia-Verwaltung um die Verwaltung von Hörbüchern.
   Hörbücher haben folgende Attribute: Nummer, Name, DauerMinuten, DauerSekunden Autor, Genre, Verlag und Erscheinungsjahr. Die Operatoren sind die gleichen wie für die Klasse Musiktitel.
   Implementieren Sie die Gemeinsamkeiten in der Klasse Medium. Fügen Sie zum Testen in der Main-Methode in einer Liste vom Typ Medium jeweils einen Musiktitel und ein Hörbuch hinzu und geben Sie die Detailinfo und Dauer über eine FOREACH-Schleife aus.
2. Erweitern Sie die Stammdatenverwaltung für Angestellte um die Verwaltung von Arbeitern.
   Arbeiter haben folgende Attribute: PersonaINummer, Vorname, Nachname, Stundenlohn, Stunden, Adresse, Telefonnummer, Eintrittsjahr, -Monat und -Tag sowie die Operatoren ZeigeBruttoArbeitsentgeIt, Zeigestammdaten. Der Operator ZeigeBruttoArbeitsentgeIt zeigt das Produkt von Stundenlohn und Stunden an.
   Implementieren Sie die Gemeinsamkeiten in der Klasse Mitarbeiter. Fügen Sie zum Testen in der Main-Methode in einer Liste vom Typ Mitarbeiter jeweils einen Angestellten und einen Arbeiter hinzu und geben Sie die Stammdaten und das Arbeitsentgelt über eine FOREACH- Schleife aus.

4.9 OOP-Konzept Polymorphie

Arbeitsauftrag

Schreiben Sie ein Programm, welches die Polymorphie bei den Methoden Berechne.Gesamtpreis anwendet.

In manchen Programmen entscheidet sich erst zur Laufzeit, welche Unterklasse verwendet werden soll. Erst wenn ein Gast ein Angebot anfordert, steht fest, für wie viele und welche Unterkunftsarten ein Preis berechnet werden soll. Wie in Kapitel 4.8 gezeigt wurde, wird das konkrete Unterkunftsobjekt einer Instanz der Klasse Angebotsposition übergeben. Das Klassenmember UnterkunftsObjekt kann jede Instanz, die von der Basisklasse Unterkunft abgeleitet ist, aufnehmen. Beim Durchspielen eines Angebotsbeispiels wurde jedoch festgestellt, dass die Angebotspositionen andere Ergebnisse liefern als die Zwischenergebnisse nach der Eingabe.

Setzen Sie zu Demonstrationszwecken einen Breakpoint in die Methode BerechneGesamtpreis der Unterklasse Zimmer und starten Sie die Anwendung. Beim ersten Aufruf für das Zwischenergebnis stoppt die Entwicklungsumgebung beim Breakpoint. Sobald Sie jedoch „Angebot erstellen" ausgewählt haben und das Programm die FOR-Schleife durchläuft, kommen Sie nicht mehr zu diesem Breakpoint. Setzen Sie nun den Breakpoint in der Berechnungsmethode der Basisklasse Unterkunft. Hier ist es nun umgekehrt: Beim ersten Aufruf für das Zwischenergebnis stoppt die Entwicklungsumgebung nicht beim Breakpoint in der Unterklasse, sondern erst beim Aufruf der Berechnungsmethode in der Basisklasse. Warum ist das so?

Auch wenn das Klassenmember UnterkunftsObjekt vom Typ Unterkunft ist, kann es jede Instanz aufnehmen, die von Unterkunft ableitet. Die Laufzeitumgebung sieht jedoch nur den Typ der Variable UnterkunftsObjekt, was in diesem Fall die Klasse Unterkunft ist. Daher wird die Methode BerechneGesamtpreis der Klasse Unterkunft aufgerufen. Das hat seine Richtigkeit. Wie kann nun der Laufzeitumgebung mitgeteilt werden, dass die Methode der Unterklasse aufgerufen wird?

Dies geschieht mithilfe des Polymorphismus. Damit wird der Aufruf an die Unterklasse weitergeleitet. Um den Polymorphismus zu implementieren, sind folgende Schritte notwendig:

- *implizite Typumwandlung auf Basisobjekt*
- *späte Bindung*
- *Methoden Virtualisieren und überschreiben*

Die Typumwandlung und späte Bindung wurden bereits besprochen, daher folgt hier die Beschreibung des letzten Schritts: Methoden Virtualisieren und überschreiben.

4.9.1 Methoden Virtualisieren und überschreiben

Damit der Aufruf der Methode aus der Basisklasse an die Unterklasse weitergeleitet wird, muss diese überschrieben werden.

Jene Eigenschaft oder Methode in der Basisklasse, welche an die Unterklasse weitergeleitet werden soll, wird mit dem Keyword virtual gekennzeichnet:

Syntax

```
Zugriffsmodifizierer virtual Datentyp Methodenname( 0 bis n Parameter)
{
}
public virtual double BerechneGesamtpreis()
```

In der Unterklasse muss die Methode mit override gekennzeichnet werden, damit diese überschrieben werden kann.

Syntax

```
Zugriffsmodifizierer override Datentyp Methodenname(0 bis n Parameter)
{
}
```

Erst dadurch können gleichartige Objekte, die unterschiedliche Verhaltensweisen haben, gleich verwendet werden. Daher kommt auch der Begriff Polymorphie (Vielgestaltigkeit): Verschiedene Objekte (Apartment und Zimmer) können gleich verwendet werden, verhalten sich aber dennoch unterschiedlich.

```
public override double BerechneGesamtpreis()
```

4.9.2 Entwurf und Realisierung

Lösung zum Arbeitsauftrag

Die Unterklassen bleiben gegenüber dem Vorkapitel unverändert, daher wird hier nur der geänderte Source Code aufgelistet.

Anpassung der Methode BerechneGesamtpreis in der Basisklasse Unterkunft in Datei Unterkunft.cs um das Keyword virtual:

```
public virtual double BerechneGesamtpreis()
{
    double Gesamtpreis = 0, GesamtpreisBrutto = 0;

    Gesamtpreis = Grundpreis * Aufenthaltsdauer;
    GesamtpreisBrutto = Gesamtpreis + BerechneMwstA(Gesamtpreis);
    return GesamtpreisBrutto;
}
```

Anpassung der Methode BerechneGesamtpreis in der Unterklasse Zimmer in Datei Zimmer.cs um das Keyword override:

```
public override double BerechneGesamtpreis()
{
    double Gesamtpreis = 0, ErwachsenenPreis = 0, KinderPreis = 0;
    double GesamtpreisBrutto = 0, MWstABetrag = 0;

    ErwachsenenPreis = Grundpreis * Aufenthaltsdauer * AnzahlGaeste;
    KinderPreis = Grundpreis * Aufenthaltsdauer * AnzahlKinder * RabattKinder / 100;
    Gesamtpreis = ErwachsenenPreis + KinderPreis;
    GesamtpreisBrutto = Gesamtpreis + BerechneMwstA(Gesamtpreis);
    return GesamtpreisBrutto;
}
```

In der Klasse Apartment muss die Methode AngebotBerechnen ebenfalls mit override gekennzeichnet werden. Der Source Code wurde aus Platzgründen weggelassen.

Main-Methode in Klasse Programm:

Der Source Code in der Main-Klasse muss nicht angepasst werden.

Aufgaben

1. Erweitern Sie die Operatoren ZeigeDetailinfos und Zeigedauer in der Multimedia-Verwaltung um die Keywords virtual und override.
2. Erweitern Sie die Operatoren ZeigeBruttoArbeitsentgelt und Zeigestammdaten in der Stammdaten-Verwaltung für Angestellte um die Keywords virtual und override.

4.10 Methoden überladen und ausblenden

Neben den bisher erläuterten Hauptkonzepten bietet die OOP noch zahlreiche andere Möglichkeiten, um das Verhalten einer Klasse bzw. der Klassenmember zu steuern. Beispielsweise können Methoden überladen oder ausgeblendet werden.

4.10.1 Methoden überladen

Arbeitsauftrag

Implementieren Sie zwei Methoden mit dem Namen ZeigeLeistungsBeschreibung. Die erste Methode hat keinen Parameter, die zweite Methode hat einen Parameter namens Mitbelegung.

Ein gutes Beispiel, das bis jetzt verwendet wurde, ist Console.WriteLine. Dieser Methode können unterschiedliche Anzahlen an Parametern übergeben werden:

```
Console.WriteLine();
Console.WriteLine("Willkommen");
Console.WriteLine("Preis = {0}", Preis);
```

Methoden können ganz einfach überladen werden, indem die gleiche Methode mehrfach mit unterschiedlichen Parametern deklariert wird:

```
public void ZeigeLeistungsBeschreibung()
{
}
public void ZeigeLeistungsBeschreibung (Bool Mitbelegung)
{
}
```

Eine überladene Methode kann zusätzlich auch virtualisiert und überschrieben werden.

Ändern Sie die Methoden ZeigeLeistungsBeschreibung in der Klasse Unterkunft in der Datei Unterkunft.cs wie folgt:

```
public virtual void ZeigeLeistungsBeschreibung()
{
    Console.WriteLine("Unterkunft ohne Verpflegung");
}

public virtual void ZeigeLeistungsBeschreibung(Bool Mitbelegung)
{
    ZeigeLeistungsBeschreibung();
    if (Mitbelegung)
        Console.WriteLine("für {0} Personen“, AnzahlGaeste);
}
```

In der Main-Methode könnten nun beide Varianten aufgerufen werden.

4.10.2 Methoden ausblenden

Arbeitsauftrag

Implementieren Sie die Methode Aufenthaltsdauer in der Klasse Apartment so, dass nicht die Methode der Basisklasse verwendet wird.

In der Basisklasse Unterkunft wurde die Eigenschaft Aufenthaltsdauer implementiert, die prüft, dass keine negativen Werte eingegeben wurden. Die Apartmentklasse benötigt allerdings eine andere Prüfung: Es muss ein Mindestaufenthalt berücksichtigt werden. Daher muss die Eigenschaft in der Methode ausgeblendet werden. Das funktioniert mit dem Keyword new:

Syntax

```
Zugriffsmodifizierer new RueckgabewertDatentyp Methodenname()
{
}
```

Arbeitsauftrag

Ändern Sie die Eigenschaft Aufenthaltsdauer wie folgt:

```
public new int Aufenthaltsdauer
{
    get { return mAufenthaltsdauer; }
    set
    {
        Aufenthaltsdauer = -1;
        if (value >= MindestAufenthalt)
            mAufenthaltsdauer = value;
    }
}
```

4.11 Abstrakte Klassen

Arbeitsauftrag

Ändern Sie das Programm so, dass von der Basisklasse kein Objekt instanziiert werden kann.

Die Basisklasse Unterkunft kann ebenfalls instanziiert werden:

```
Unterkunft UnterkunftsObjekt = new Unterkunft();
```

In der Entwurfsphase wurde erkannt, dass die Klasse Unterkunft der Allgemeinfall ist und daher als Basisklasse implementiert wurde. In manchen Fällen hat diese Basisklasse für konkrete Objekte keine Bedeutung, sondern ist Basis einer künstlichen Klassenhierarchie. Soll verhindert werden, dass das Basisobjekt instanziiert wird, weil es in der realen Welt keine Entsprechung hat, kann eine Klasse abstrakt gemacht werden:

Syntax

Zugriffsmodifizierer abstract class Klassenname

```
public abstract class Unterkunft
```

Wird ein Objekt von einer abstrakten Klasse instanziiert, kommt folgende Fehlermeldung:

„Es konnte keine Instanz der abstrakten Klasse oder Schnittstelle "Hotelangebotssoftware.Unterkunft" erstellt werden."

4.11.1 Entwurf und Realisierung

Lösung zum Arbeitsauftrag

```
public abstract class Unterkunft
{
// unveränderte Properties und Methoden aus Platzgründen ausgelassen
}
```

4.12 Statische Klassen und Klassenmember

4.12.1 Statische Eigenschaften und Felder

Arbeitsauftrag

Ändern Sie die Eigenschaft MwSt. auf statisch.

Derzeit hat jede Instanz der Klasse Zimmer eine Variable für die Mehrwertsteuer. Wenn einem Gast zwei verschiedene Zimmerkategorien angeboten werden, wird die Mehrwertsteuer zweimal im Speicher abgelegt. Soll eine Eigenschaft oder ein Feld nur einmal je Klasse im Speicher gelten, bietet die OOP statische Eigenschaften und Felder an.

Die Syntax für statische Eigenschaften und statische Felder ist die der normalen Eigenschaften und Felder, mit dem Unterschied, dass nach dem Zugriffsmodifizierer das Keyword static verwendet wird. Hier wird beispielhaft die Syntax für Felder beschrieben:

Syntax

```
Zugriffsmodifizierer static Datentyp Feldname;
public static double MwSt.;
```

Die statische Eigenschaft wird nicht über den Objektnamen, sondern über den Klassennamen gesetzt, da statische Klassenmitglieder zur Klasse gehören:

Syntax

```
Klassenname.Feld = Wert;
Unterkunf.MwStA = 10;
```

In nicht statischen Klassenmembern wie BerechneGesamtpreis kann wie auf ein nicht statisches Klassenmember zugegriffen werden:

Arbeitsauftrag

Ändern Sie die Eigenschaft MwSt. in der Klasse Unterkunft wie folgt:

```
public static int MwstA { get; private set; }
```

Um die Mehrwertsteuer auszugeben, kann nur über den Klassennamen zugegriffen werden.

Ändern Sie im Case H den Variablennamen Zimmer auf ZimmerObjekt und geben Sie die Mehrwertsteuer wie folgt aus:

```
case "H":
    Kategorie = EingabeDialogFuerInt("Eingabe Kategorie: Standard = 0
                                                        , Komfort = 1, Suite = 2");
    AnzahlGaeste = EingabeDialogFuerInt("Anzahl Erwachsene");
    AnzahlKinder = EingabeDialogFuerInt("Anzahl Kinder");
    AnzahlZimmer = EingabeDialogFuerInt("Anzahl Zimmer");

    Zimmer ZimmerObjekt = new Zimmer(
    Preisdatenzimmer[Kategorie].Preis, Preisdatenzimmer[Kategorie].RabattKinder,
    Preisdatenzimmer[Kategorie].Kategorie, 7,
    Preisdatenzimmer[Kategorie].MaxAnzahlGaeste,
    Preisdatenzimmer[Kategorie].MaxAnzahlKinder);

    ZimmerObjekt.Aufenthaltsdauer = Aufenthaltsdauer;

    ZimmerObjekt.AnzahlGaeste = AnzahlGaeste;
    ZimmerObjekt.AnzahlKinder = AnzahlKinder;
    Console.WriteLine("Einzelpreis Komfortzimmer {0}", ZimmerObjekt.BerechneGesamtpreis());
    Console.WriteLine("Die MwSt.- beträgt {0}", Zimmer.MwstA);

    AngebotspositionsListe[AnzahlAngebotspositionen] =
                                          new Angebotsposition(AnzahlZimmer, ZimmerObjekt);

    AnzahlAngebotspositionen = AnzahlAngebotspositionen + 1;
    break;
```

4.12.2 Statische Methoden und statische Klassen

Auch Methoden können als statisch gekennzeichnet werden. In den bisherigen Kapiteln haben Sie bereits mit statischen Methoden gearbeitet: Console.Write, Console.WriteLine und Console.ReadLine sind statische Methoden. Um diese Art der Methoden verwenden zu können, wird wie bei den statischen Eigenschaften oder Feldern kein Objekt instanziiert, sondern über den Namen der Klasse angesprochen (z. B. Console).

Syntax

```
Zugriffsmodifizierer static Datentyp Methodenname(0 bis n Parameter);
```

Besteht eine Klasse nur aus statischen Klassenmitgliedern, spricht man von einer statischen Klasse und sie kann mit dem Keyword static versehen werden. Mithilfe von statischen Klassen können Funktionssammlungen realisiert werden. Statische Klassen sind sinnvoll, wenn sich das Objekt keine Werte bzw. Zustände durch Eigenschaften oder Felder merken muss.

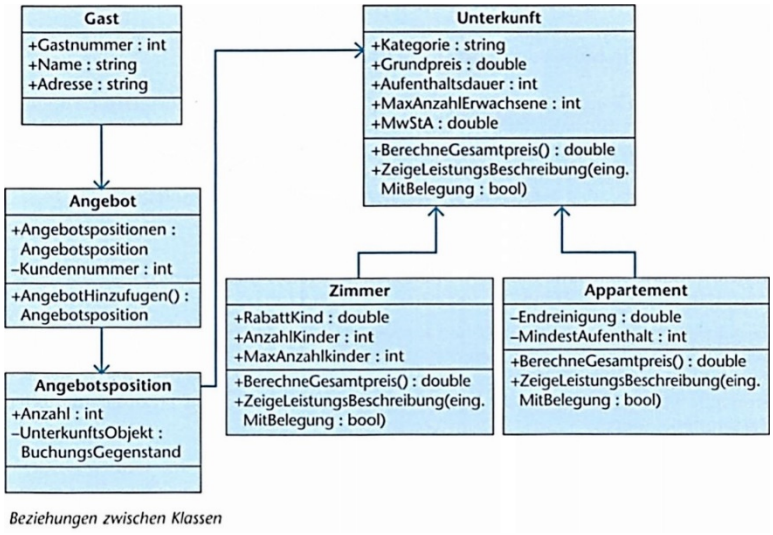
4.13 Beziehungen zwischen Klassen

In der objektorientierten Analyse wurden die für das Programm relevanten Objekte identifiziert: Gast, Angebot, Angebotsposition, Unterkunft. In Kapitel 4.7 kamen die Spezialfälle Zimmer und Apartment hinzu. Aus didaktischen Gründen wurde zuerst mit der Generalisierung begonnen. In der Praxis ist es meistens umgekehrt: Die Analyse ergibt

beispielsweise verschiedene Objekte wie Zimmer, Apartment usw. Das ist zunächst nicht weiter problematisch, da die Analyse nur herausfinden soll, welche Objekte, Eigenschaften und Aktionen notwendig sind, um eine bestimmte Anforderung zu erfüllen.

Erst im objektorientierten Design macht man sich nähere Gedanken darüber, wie die Lösung umgesetzt werden soll. Hier stellt sich beispielsweise die Frage, ob Vererbung sinnvoll ist. Wenn auf Objekte wie Zimmer und Apartment das Kriterium Generalisierung/Spezialisierung zutrifft und/oder verschiedene Klassen gemeinsame Klassenmitglieder besitzen, ist es sehr wahrscheinlich, dass es sich um eine Vererbung handelt. Ein weiteres Kriterium ist die Art der Beziehung zwischen den Klassen. Hier gibt es die sogenannte „ist ein(e)" und „hat ein(e") Beziehung.

Kann jetzt auch noch die Beziehungsart „ist ein(e)" angewandt werden, ist es fast schon sicher, dass es sich um eine Vererbung handelt. Denn mithilfe der Beziehungsart „ist ein(e)" kann man eine Vererbungshierarchie ausdrücken: Ein Zimmer „ist eine" Unterkunft, aber kein Gast. Ein Gast kann ein Zimmer anfragen („hat ein" Zimmer), aber ist kein Zimmer oder Unterkunft. Es gibt zwar eine Abhängigkeit zwischen Gast und dem Buchungsgegenstand, aber nicht im Sinne einer Vererbung. Erst durch die Analyse der Anforderung und durch das Entwerfen von einer oder mehreren Varianten von Klassendesigns kann man zu einer Lösung wie in der Abbildung unten kommen.



Beziehungen zwischen Klassen

Hinweis: Vererbung ist nicht das „Heilmittel" der objektorientierten Programmierung. Es existieren in der Softwareentwicklung noch andere Methoden, um ein Klassendesign zu ermitteln. Methoden wie Entwurfsmuster und komponentenorientierte Entwicklung versuchen nun der Komplexität von Programmen Herr zu werden. Diese Themen werden hier nicht weiter behandelt, da sie für den hier dargestellten Inhalt zu umfangreich wären.

4.14 Weitere Themenbereiche

4.14.1 Klasse Objekt, Boxing und Unboxing

Des Weiteren basiert jede Klasse in .NET direkt oder indirekt auf der Basisklasse Objekt. Diese stellt verschiedene Methoden wie Equals, ToString, GetHashCode usw. zur Verfügung. Mit ToString kann beispielsweise ein Wertetyp (der ebenfalls indirekt auf Objekt basiert) in einen string konvertiert werden.

```
int i = 3;
object o = i;
string s = i.ToString();
```

Mit dem Boxing kann ein Wertetyp in einen Referenztyp umgewandelt werden bzw. mit dem Unboxing in umgekehrter Reihenfolge. Durch diese Methode kann eine Zahl in ein Objekt umgewandelt werden und beispielsweise einer Collection (siehe Kapitel 6) übergeben werden.

4.14.2 Werte- und Referenztypen

Wie in diesem und in Kapitel 3 erwähnt, speichert eine Variable verschiedene Datentypen wie Zahlen, Enums, Structs und Objekte. Dabei wird zwischen Werte- und Referenztypen unterschieden. Zu den Wertetypen zählen einfache Datentypen, Enums und Structs. Hier wird der Wert (z.B. 3) direkt in der Speicherzelle (Variable) abgelegt. Dieser Wert wird auf dem sogenannten Stack gespeichert. Den Stack kann man sich als Stapel vorstellen. Das oberste

Element im Stapel enthält die lokalen Wertetypen der aktuellen Methode (siehe auch Aufrufliste in Kapitel 5). Erst wenn die aktuelle Methode beendet wird (zurückkehrt) wird das oberste Element (Variable) vom Stack entfernt.

Hingegen werden Objekte im sogenannten Heap gespeichert, sobald das Objekt instanziiert wird. Die Speicherzelle (bzw. die Variable) enthält dann nur eine Referenz auf das Objekt im Speicher. Das Objekt wird im Gegensatz zum Stack erst aus dem Heap entfernt, wenn das Objekt nicht mehr verwendet wird. Im Normalfall braucht man sich um die Freigabe nicht zu kümmern, da dies der Garbage-Collector (GC) übernimmt. In unbestimmten Intervallen (z.B. wenn der PC weniger ausgelastet ist) prüft der GC, ob noch eine Variable auf das Objekt verweist.

Aufgaben

1. Entwerfen und implementieren Sie ein Programm zur Verwaltung von Points of Interests (POIs, Orte von Interesse). POIs können Sehenswürdigkeiten, Museen, Autowerkstätten usw. sein, welche in einer digitalen Karte angezeigt werden. Die Basisklasse POI hat die Attribute Name, Latitute (ReadOnly, Wertebereich: 0 bis +/-90°) und Longitude (ReadOnly, Wertebereich: 0 bis +/-180°) in Dezimalgrad, z.B. 10,234°. Die Operatoren heißen: ZeigeDetaiIs und SetzeKoordinaten (zwei Parameter, virtuell). Leiten Sie davon folgende Unterklassen ab: Sightseeing mit dem Attribut Beschreibung. Die Unterklasse Autowerkstatt hat die Attribute Postadresse und Telefonnummer. Im Programm können beispielsweise fünf oder zehn POIs eingegeben und abgefragt werden. Wenn das Programm startet, werden die Menüpunkte Sightseeing anlegen/bearbeiten, Autowerkstätte anlegen/bearbeiten, POIs auflisten angezeigt sowie die aktuelle Anzahl an POIs.

   Hinweise: Verwenden Sie statische Hilfsfunktionen in der Klasse Main. Mit Console.Clear() wird der Bildschirm gelöscht. Beim Löschen kann ein beliebiger Platz gelöscht werden (auf Null gesetzt werden). Berücksichtigten Sie das beim Ermitteln eines freien Platzes im Array. Sie können einer Methode (z. B. in Main) als Parameter auch eine Objektvariable übergeben oder sie retournieren lassen:

   ```
   static void Auflisten(POI[] poi) { }

   static POISightseeing SightseeingAnlegen()
   { // Daten einlesen
       POISightseeing Sightseeing = new POISightseeing(); // – return Sightseeing;
   }
   ```

   Für das Bearbeiten ist es wichtig, den Typ wegen des Castens zu überprüfen. Das geschieht wie folgt: TypeOf liefert den Typ einer Klasse und GetType den Typ des Objekts.

   ```
   if(typeof(POIAutowerkstatt) == poi[Index].GetType())
   ```

5 Fehlersuche und Fehlerbehandlung

In diesem Kapitel lernen Sie Werkzeuge für die Fehlersuche kennen. Darüber hinaus werden neue Anweisungen vorgestellt, um unvorhergesehene Laufzeitfehler behandeln zu können.

5.1 Werkzeuge für die Fehlersuche

Arbeitsauftrag

Verwenden Sie folgende Werkzeuge für die Fehlersuche: Breakpoint, Variablen überwachen und Aufrufliste.

5.1.1 Fehlerarten

In den verschiedenen Phasen der Softwareentwicklung können Fehler auftreten, wodurch sich die Software nicht wie gewünscht verhält. Auch wenn Sie streng alle Phasen der Softwareentwicklung (Analyse, Entwurf, Programmierung und Test) einhalten, kann ein Programm fehlerhaft sein. Das kann unterschiedliche Ursachen haben: Die Programmaufgabe wurde falsch oder unklar definiert. Man kann nicht immer alles eindeutig in Worte fassen oder dem Auftraggeber ist nicht alles klar.

Selbst wenn die Programmaufgabe klar definiert ist, kann es aus verschiedenen Gründen zu Fehlern in der Analyse- und Entwurfsphase kommen. Beispielsweise kann ein falscher oder langsamer (Unperformanter) Algorithmus gewählt werden oder das Programm kann falsch oder zu kompliziert entworfen worden sein. Oder es kann schlicht etwas übersehen werden.

Auch wenn die Phasen Analyse und Entwurf korrekt abgelaufen sind, können sich durch Tippfehler oder falsch angewandte Anweisungen Fehler einschleichen. Selbst wenn ein Programm umfangreich getestet wird, kann dieses so komplex sein, dass nicht alle Situationen getestet werden können und so oft erst im laufenden Betrieb sogenannte Bugs auftreten können. Diese Fehler erzeugen meistens ein nicht gewünschtes Verhalten und müssen daher vom Entwickler behoben werden. Daher sollte ein Programmierer auch die Fehlersuche beherrschen, um Fehler lokalisieren und beheben zu können.

Daher stellt eine moderne Entwicklungsumgebung folgende Werkzeuge zur Fehlersuche zur Verfügung:

5.1.2 Breakpoint setzen

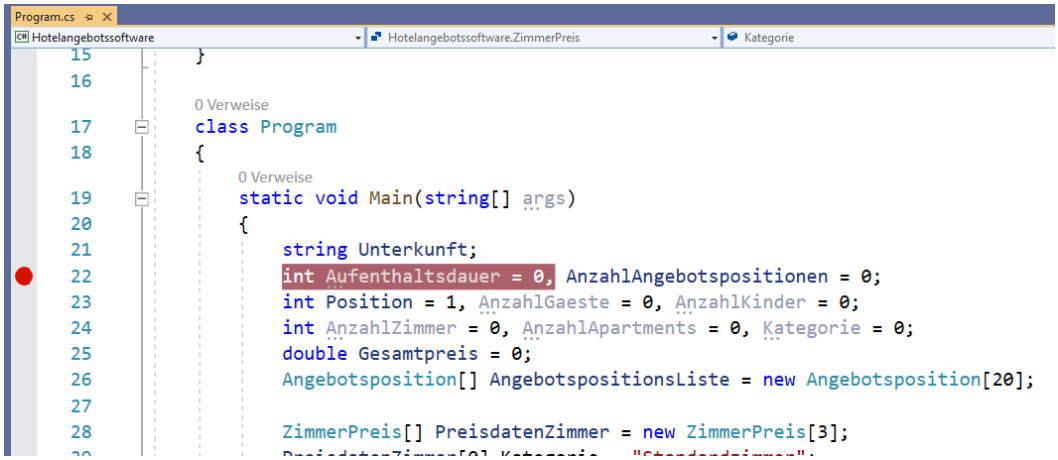
Ein Programmierfehler wird meistens entweder vom Programmierer selbst, vom Tester oder vom Anwender gefunden. Kann der Fehler nachvollzogen werden und wird festgestellt (oder vermutet), dass es sich um einen Programmierfehler handelt, ist es Aufgabe des Programmierers, den Fehler zu finden und zu beheben.

Im Idealfall kann durch die Art der Fehlermeldung oder ein (zu implementierendes) Protokoll (Datei, Datenbank) der Ort des Fehlers eingegrenzt werden (z. B. in welcher Methode, Klasse oder Datei der Fehler passiert). Es gibt aber auch Fälle, wo der Fehler nicht genau lokalisiert werden kann. Hier muss die Fehlersuche an einer geeigneten Stelle, beispielsweise in der Main-Methode beginnen. In beiden Fällen muss jedoch ein Breakpoint (Haltepunkt) gesetzt werden, damit das Programm Codezeile für Codezeile ausgeführt werden kann und die Werte im Programm beobachtet werden können.

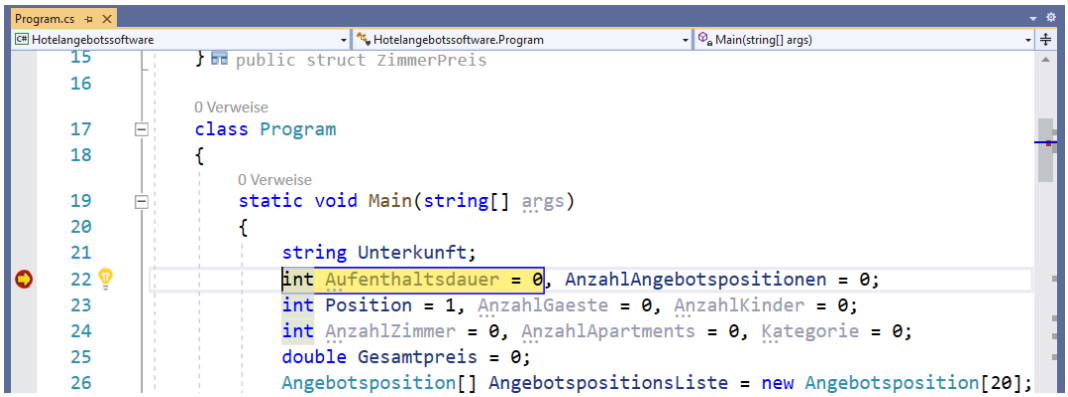
Ein Haltepunkt (Breakpoint) wird in Visual Studio wie folgt gesetzt:

- *Der Cursor wird im Source Code (z. B. Program.cs) auf eine ausführbare Anweisung gesetzt.*
- *Breakpoint setzen über Menü Debuggen\Haltepunkt umschalten (Debug\ToggIe Breakpoint) oder durch Drücken der Taste F9 (Zeile wird wie in der Abbildung dargestellt rot markiert).*

- *Das Debugging wird gestartet über Menü Debuggen\Debuggen starten (Debug\Start Debugging) oder durch Drücken von F5.*



- Nach dem Start des Debuggings kann der Source Code Zeile für Zeile mit folgenden Befehlen ausgeführt werden (die aktuelle Zeile wird gelb markiert, siehe auch Abbildung*):*



- *Einzelschritt (Menü Debuggen\Einzelschritt (Debug\Step Into) oder F11): Hier wird jede einzelne Zeile ausgeführt. Enthält die Codezeile eine Methode, wird auch jede einzelne Zeile in der Methode ausgeführt.*
- *Prozedurschritt (Menü Debuggen\Prozedurschritt (Debug\Step Over) oder F10): Oft möchte man aus verschiedenen Gründen nicht den Source Code von Prozeduren untersuchen, beispielsweise weil aufgrund der Fehlermeldung klar ist, dass diese Methode den Fehler nicht verursacht. In diesem Fall ist die Funktion Prozedurschritt geeignet, um den Code zu untersuchen.*
- *Die Funktionen Einzelschritt und Prozedurschritt können gemischt ausgeführt werden.*

Ist der Fehler gefunden, geht man wie folgt vor:

- *Debugging stoppen über Menü Debuggen\Debuggen beenden (Debug\Stop Debugging) oder Tastenkombination Umschalt + F5*
- *Fehler korrigieren*
- *Haltepunkt ausschalten über Debuggen\Haltepunkt umschalten (Debug\Toggle Breakpoint) oder Taste F9*
- *Programm testen*
- *Gegebenenfalls mit der Fehlersuche erneut beginnen*

5.1.3 Variablen überwachen

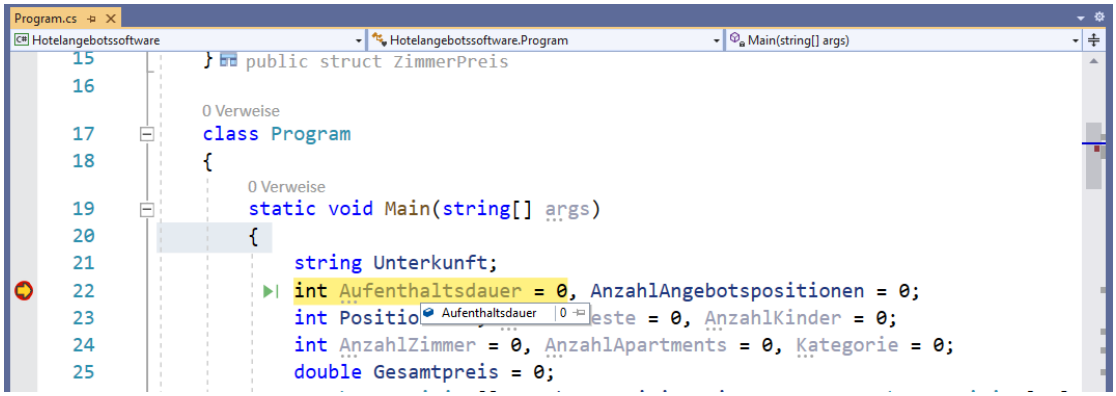
Arbeitsauftrag

Überwachen Sie die Werte von Variablen wie Gesamtpreis.

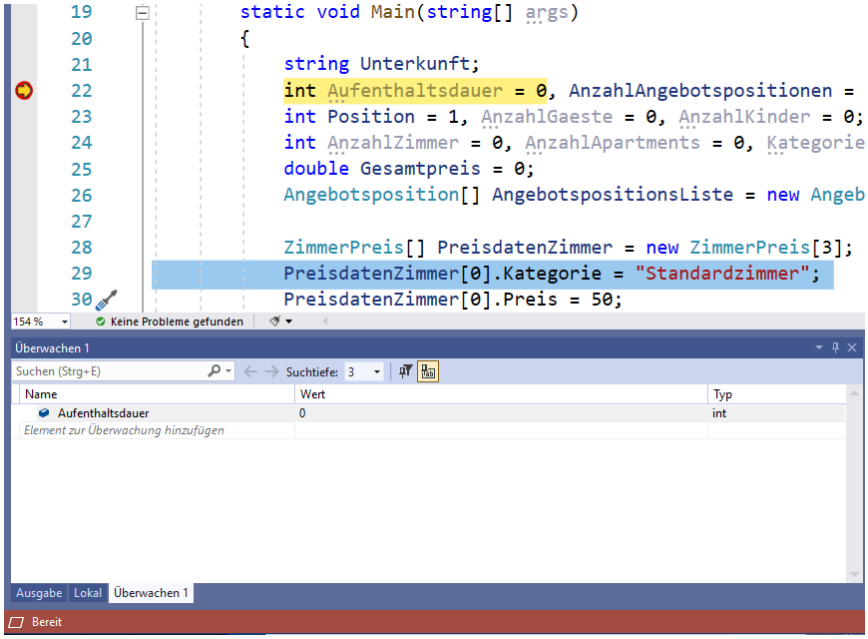
Breakpoints setzen ist nur ein Teil der Fehlersuche. Um das Ergebnis von Anweisungen, z.B. Berechnungen, zu prüfen, muss man den Wert einer Variablen überwachen. Bisher haben wir Variablen mit Console.Write(Line) ausgegeben. Das ist eine Möglichkeit, um zur Laufzeit Werte von Variablen auszugeben. In der Praxis wird man jedoch nicht alle Variablen ausgeben, sondern nur jene, die den Endanwender interessieren. Um auch an die Werte der anderen Variablen heranzukommen, bietet Visual Studio die Funktionalität „Variablen überwachen" an.

Wenn Sie ein Programm schrittweise über Breakpoints ausführen, können Sie die Werte von Variablen wie folgt überwachen:

- *Mithilfe der DataTips: Wenn Sie mit der Maus über die Variable fahren, erscheint ein kleines Popup-Fenster, das den Wert anzeigt. Handelt es sich bei der Variablen um eine Objektvariable, kann man ein Pluszeichen anklicken und man erhält die Werte der Klassenmember.*



- *Variablen überwachen: Über das Kontextmenü „Überwachung hinzufügen" wird jene Variable, die markiert ist oder auf die der Cursor steht, in das Überwachungsfenster hinzugefügt.*



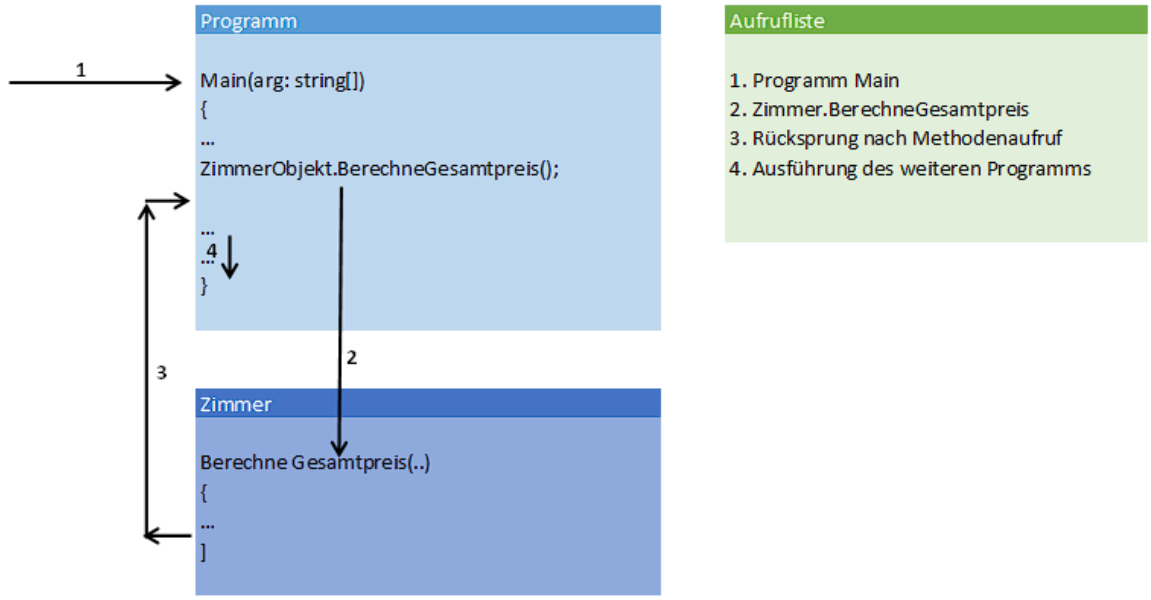
Lokale Variablen sind Variablen, die nur in der aktuellen Methode sichtbar sind, wie Zimmerpreis, Gesamtpreis oder Nettopreis. Die Methode AngebotBerechnen selbst kann auf die Variablen Zimmerpreis und Aufenthaltsdauer aus der Main-Methode nicht zugreifen. Nur durch die Übergabe von Parametern (Weitergabe der Werte) kann die Methode AngebotBerechnen mit den Werten aus der Main-Methode rechnen (oder durch Setzen von Properties oder Konstruktorparametern im Falle einer Klasse).

Wird eine lokale Variable überwacht oder eine Variable zur Überwachung hinzugefügt, kann der Wert der Variablen in der Spalte Wert geändert werden.

5.1.4 Aufrufliste

Während des Programmablaufs werden verschiedene Methoden der Reihe nach aufgerufen. In einem umfangreicheren Programm können mehrere Methoden und auch Klassen beteiligt sein. Wird in einer Methode ein Breakpoint gesetzt, kann es oft von Interesse sein, welche anderen Methoden (Klassen) diese Methode aufrufen. Eine Methode kann von einer oder mehreren Methoden aufgerufen werden.

Die folgende Abbildung zeigt vereinfacht dargestellt, wie die Aufrufliste funktioniert.



Funktionsweise der Aufrufliste

- *Das Programm startet in der Methode Program.main. --> Wird als erstes der Aufrufliste hinzugefügt.*
- *Die Berechnung wird in der Methode AngebotBerechnen durchgeführt. --> Wird als zweites der Aufrufliste hinzugefügt.*

Über die Aufrufliste (Menü Debuggen\Fenster\Aufrufliste) erhält man die Liste der aufgerufenen Methoden. Durch Doppelklick auf eine Methode kann Man in den Code dieser Methode springen.

Arbeitsauftrag

Setzen Sie einen Breakpoint auf die erste ausführbare Anweisung in der Main-Methode und führen Sie das Programm Zeile für Zeile aus. Überwachen Sie die Variablen und die Aufrufliste.

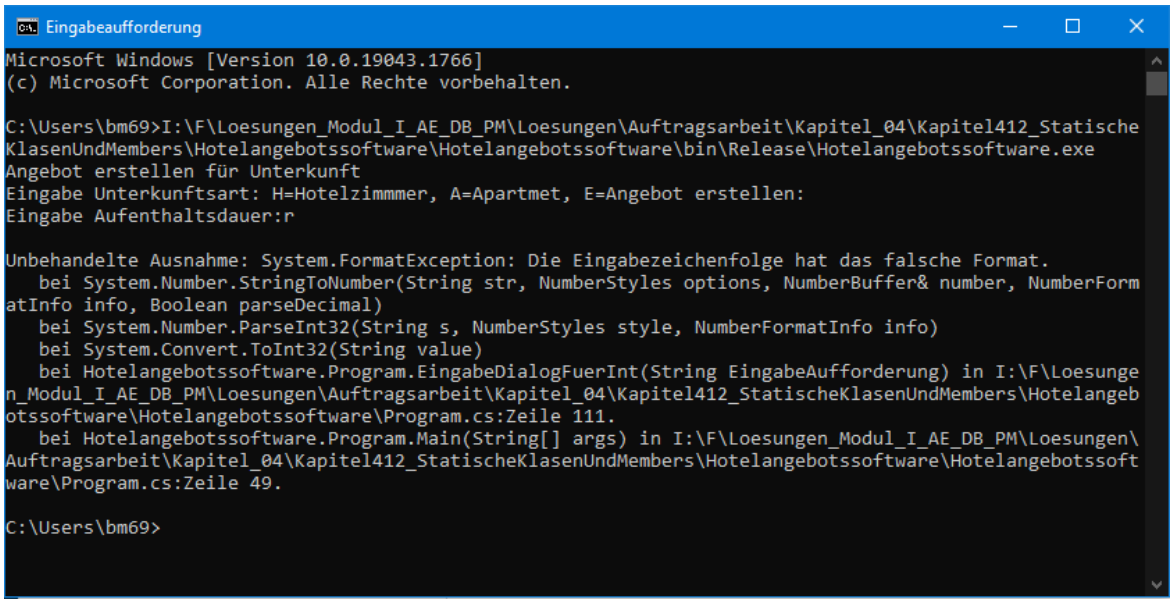
5.2 Fehlerbehandlung (Exception Handling)

Arbeitsauftrag

Implementieren Sie ein Exception Handling, um unvorhergesehene Laufzeitfehler abzufangen.

5.2.1 Unvorhergesehene Fehler abfangen

Starten Sie das Programm aus Kapitel 4 über die Kommandozeile. Geben Sie in der Eingabeaufforderung „Aufenthaltsdauer" anstelle einer Zahl einen Buchstaben ein und beobachten Sie das Verhalten der Anwendung.



Wie Sie sehen, erscheint zuerst eine MeIdung des Betriebssystems und dann eine Fehlermeldung in der Konsole. Fehlermeldungen dieser Art sind für einen Endanwender sehr verwirrend und sollten nach Möglichkeit vermieden werden. Andererseits ist diese Fehlermeldung für den Programmierer sehr hilfreich, da diese die Art und Quelle des Fehlers verrät. Wie kann man daher unerwartete Laufzeitfehler abfangen und sie sowohl für den Endanwender als auch für den Programmierer besser aufbereiten?

Die Lösung des Problems lautet Exception Handling. Um Unvorhergesehene Ausnahmen abzufangen, wird die try/catch-Anweisung verwendet. Die Syntax für die Fehlerbehandlung lautet wie folgt:

Syntax

```
try
{
    Anweisung(en);
}
catch[(Exception ex)]
{
    Anweisung(en) für Fehlerbehandlung;
}
```

Nach dem Keyword try folgen die gewohnten geschwungenen Klammern, zwischen denen eine oder mehrere Anweisungen stehen. Tritt ein Laufzeitfehler auf, wird die „Anweisung(en) für Fehlerbehandlung" ausgeführt, der Fehler wird also (auf)gefangen. Daher kann man obige Syntax auch wie folgt lesen: Versuche (try) die Anweisung(en) auszuführen. Fange (catch) einen möglichen Laufzeitfehler auf und führe die „Anweisung(en) für Fehlerbehandlung" aus.

In diesem Code-Snippet wird der Fehler einfach abgefangen und eine benutzerdefinierte Fehlermeldung ausgegeben.

```
try
{
    Console.WriteLine("Berechnen eines Angebots“);
    // Eingabe Aufenthaltsdauer
    Console.Write("Eingabe Aufenthaltsdauer:");
    EingabeAufenthaltsdauer = Console.Readline();
    Aufenthaltsdauer = Convert.ToInt32(EingabeAufenthaltsdauer);
}
catch
{
    Console.WriteLine("Die Eingabe ist ungültig“);
}
```

Dem Endanwender mag diese Fehlermeldung genügen, für die interne Fehlersuche ist sie jedoch nicht optimal. Daher kann im catch-Teil optional die Fehlerquelle abgefragt werden.

Syntax

```
catch (Exception ex)
```

Exception selbst ist eine Klasse (bzw. ein Objekt), das von der .NET-Laufzeit befüllt wird und einige interessante Informationen über den Fehler enthält. Die Variable ex kann im catch-Teil wie eine gewöhnliche Objektvariable verwendet werden.

Mit dem Aufruf der Methode ToString() erhält man alle Informationen über den Fehler. Beispielsweise kann man in Kombination mit ConsoIe.WriteLine sowohl Fehlermeldungen für den Endanwender als auch für den Support generieren. Die detaillierte Fehlermeldung kann auch in eine Protokolldatei geschrieben werden.

```
catch (Exception ex)
{
    Console.WriteLine("Unerwarteter Fehler bei Eingabe“);
    Console.WriteLine("Information für den Support:");
    Console.WriteLine(ex.ToString());
}
```

Die Klasse Exception stellt noch folgende wichtige Informationen zur Verfügung:

Property/Methode	Bedeutung
Message	Enthält die Fehlermeldung als Text. Beispiel: Die Eingabezeichenfolge hat das falsche Format.
Source	Enthält oder setzt den Namen der Anwendung oder des Objekts der Fehlerquelle.
Stacktrace	Enthält die Aufrufliste, also die Liste der bisher aufgerufenen Methoden.
InnerException	Wenn eine neue Ausnahme gefangen und geworfen wird (siehe Kapitel 5.23), kann man hier die ursprüngliche Fehlerklasse hinterlegen.

Neben der Message ist vor allein der Stacktrace (Aufrufliste) interessant. Letzterer gibt an, in welcher Methode (und gegebenenfalls Zeilennummer) der Fehler passiert ist. Im folgenden Code-Snippet gibt die erste Zeile den Namen der Methode an, in der der Fehler passiert ist.

```
bei System.Number.StringToNumber(String str, NumberStyles options,
NumberBuffer& number, NumberFormatInfo info, Boolean parseDecimal)
bei System.Number.ParseInt32(String s, NumberStyles style, NumberFormat-
Info info)
bei System.Convert.ToInt32(String value)
bei ExceptionHandling.Program.Main(String[] args) in C:\Georg\
ProgrammentwicklungMitCSharp\Auflage_04\Programm\Kapitel_5\Kapitel_52_
Exception\ExceptionHandling\ExceptionHandling\Program.cs:Zeile 26.
```

Dieser Stacktrace zeigt, dass die Methode Convert.ToInt32 auch noch andere Klassen und Methoden wie System.Number.ParseInt32 des .NET-Frameworks aufruft. Da wir den .NET-Code nicht ändern können, müssen wir die Methode in unserem Programm suchen. In dem Fall ist das der vierte Eintrag im Stacktrace (bei ExceptionHandIing. Program.Main). Wird das Programm mit Debug-Information übersetzt, wird sogar die Zeilennummer mit ausgegeben (Zeile 26).

5.2.2 Mehrfaches catch

Im vorherigen Kapitel wurde gezeigt, wie Laufzeitfehler allgemein abgefangen werden können, indem beispielsweise die Klasse Exception im catch-Teil verwendet wurde, um die detaillierte Fehlermeldung zu erhalten. Die Klasse Exception ist eine allgemeine Basisklasse für Laufzeitfehler. In der Praxis gibt es jedoch eine Vielzahl an verschiedenen Laufzeitfehlern (beispielsweise System.DivideByZeroException für Division durch 0 oder System.IO.FileNotFoundException für nicht gefundene Dateien). Möchte man auf unterschiedliche Laufzeitfehler unterschiedlich reagieren, verwendet man einfach mehrere catch-Teile. Dabei ist die Reihenfolge der catch-Teile wichtig. Soll zuerst die Exception für die Division durch Null gefangen werden, muss diese zuerst deklariert werden, erst dann darf die (allgemeine/andere) Exception gefangen werden.

Syntax

```
try
{
    //Anweisung(en);
}
catch (Exception1 ex)
{
    //Anweisung für diesen catch-Teil 1;
{
catch (Exception2 ex)
}
    //Anweisungen für catch-Teil 2;
}
```

5.2.3 Fehler werfen

In manchen Fällen kann es sinnvoll sein, dass man den gefangenen Fehler nicht an der Stelle behandelt, wo man ihn fängt, sondern beispielsweise an übergeordneter Stelle weiterleitet, um beispielsweise die Fehlermeldungen zentral in einer Datenbank oder Datei zu protokollieren oder andere Anweisungen für den Fehlerfall durchzuführen.

Dabei können Fehler im Code wie folgt geworfen werden, wobei man dem Konstruktur einen Fehlertext mitgeben kann:

Syntax

```
throw new Exception("Fehlertext");
```

oder im catch-Teil (mit oder ohne Angabe der Exception Variable ex):

```
throw [ex];
```

Im folgenden Code-Snippet wurde die Eingabe der Aufenthaltsdauer in einer eigenen Methode mit Try/catch-Block implementiert. Wird beim Aufruf von Convert.ToInt32 statt einer Zahl ein Text eingegeben, wird eine System-FormatException geworfen. Beispielsweise könnte man die Fehlerbehandlung so gestalten, dass die FormatException behandelt wird und eine benutzerfreundliche Fehlermeldung ausgegeben wird. Hingegen werden andere Laufzeitfehler mit throw an die übergeordnete Methode weitergeworfen. Beispielsweise wird eine allgemeine Exception geworfen, wenn die Aufenthaltsdauer negativ ist.

```
private static int EingabeAufenthaltsdauer()
{
    try
    {
        int Aufenthaltsdauer;
        string EingabeAufenthaltsdauer;
        Console.Write("Eingabe Aufenthaltsdauer: ");
        EingabeAufenthaltsdauer = Console.ReadLine();
        Aufenthaltsdauer = Convert.ToInt32(EingabeAufenthaltsdauer);
        if (Aufenthaltsdauer < 0)
            throw new Exception("Zahl negativ");
        return Aufenthaltsdauer;
    }
    catch (System.FormatException ex) {
        Console.WriteLine("Eingabe ist keine Zahl");
    } catch (Exception ex)
    {
        throw ex;
    }
}
```

5.2.4 Finally

Mit dem Keyword catch können unvorhergesehene Laufzeitfehler behandelt werden. Unabhängig davon, ob die Anweisungen im try-Teil einen Laufzeitfehler werfen oder ohne Laufzeitfehler durchlaufen, möchte man oft trotzdem einen gemeinsamen Code ausführen. Wenn ein Programm mit anderen Ressourcen wie Dateien oder Datenbanken (siehe nachfolgende Kapitel) zusammenarbeitet, müssen diese Ressourcen wieder freigegeben werden. Daher ist es wichtig, dass diese Anweisungen ausgeführt werden. Für abschließende Arbeiten kann der finally-Teil verwendet werden:

Syntax

```
try
    Anweisung(en);
catch [(Exception»
    Anweisung(en) im Fehlerfall;
finally
    Anweisung(en) für Aufräumarbeiten;
```

5.2.5 Eigene Exception definieren

Wird von der Klasse Exception abgeleitet, kann man seine eigene Exception deklarieren, werfen und fangen.

Syntax

```
Zugriffsmodifizierer class OwnException : Exception
```

Das folgende Code-Snippet definiert eine eigene Exception namens NegativNumberException.

```
public class NegativNumberException : Exception
```

Dein Konstruktor können verschiedene Parameter wie Messages oder andere Exceptions mitgegeben werden, indem man den Konstruktor beispielsweise wie folgt überschreibt: Soll nur ein fixer Text (z.B. „Negative Zahlen sind nicht erlaubt.") verwendet werden, implementiert man einen Konstruktor ohne Parameter und setzt die Message der Basisklasse.

```
public class NegativNumberException : Exception
{
    public NegativNumberException(string message) : base(message)
    { }

    public NegativNumberException(string message, Exception inner) : base(message, inner)
    { }
}
```

Lösungen zum Arbeitsauftrag

Klasse MindestAufenthaltunterschrittenException in Datei MindestAufenthaltUnterschrittenException.cs

```
public class MindestAufenthaltUnterschrittenException : Exception
{
    public MindestAufenthaltUnterschrittenException(string pMessage) :
        base(pMessage) { }
}
```

Eigenschaft Aufenthaltsdauer in Klasse Apartment

Nun kann die Exception in der Eigenschaft Aufenthaltsdauer wie folgt angewandt werden:

```
public new int Aufenthaltsdauer
{
    get { return mAufenthaltsdauer; }
    set
    {
        Aufenthaltsdauer = -1;
        if (value >= MindestAufenthalt)
            mAufenthaltsdauer = value;
        else
            throw new MindestAufenthaltUnterschrittenException("");
    }
}
```

Klasse Programm in Datei Program.cs

Siehe Lösungsdateien

6 Grundlegende .NET-Klassen

Das .NET-Framework stellt für verschiedene Aufgaben unterschiedliche Klassen zur Verfügung, welche die tägliche Programmiertätigkeit vereinfachen. Dieses Kapitel stellt die grundlegenden .NET-Klassen wie string, DateTime und Collections vor.

6.1 Texte verarbeiten mit Klasse string und StringBuilder

Seit Kapitel 2 haben wir mit der Klasse string gearbeitet. Diese Klasse stellt verschiedene Methoden und Eigenschaften zur Verfügung, um Text zu speichern, zu vergleichen und zu bearbeiten.

6.1.1 Textvariable deklarieren und initialisieren

Arbeitsauftrag

Initialisieren Sie eine Textvariable mit beliebigem Text bzw. geben Sie 20 Mal ein Zeichen aus.

Bisher wurde zum Instanziieren einer String-Variablen folgende Variante verwendet: Die Klasse wird durch die Zuweisung eines beliebigen Textes entweder gleich bei der Deklaration oder erst später im Code instanziiert.

Syntax

```
string Variable = "beliebiger Text";
```

Es gibt jedoch auch noch weitere Varianten, um eine String-Variable zu initialisieren. Beispielsweise kann man dem Konstruktor ein Zeichen und eine Anzahl übergeben. In der Variablen wird das Zeichen entsprechend der Anzahl wiederholt.

Syntax

```
string VariablenName = new string(char c,20);
```

```
string Text1 = "beliebiger Text";
string Text2 = new string('*', 20);
Console.WriteLine(Text1);
Console.WriteLine(Text2);
Console.ReadKey();
```

6.1.2 Texte zusammensetzen

Arbeitsauftrag

Setzen Sie zwei Texte zu einem Text zusammen.

Texte werden mit einem Plus-Operator (+) zusammengesetzt. Das betrifft sowohl Textvariablen als auch den Inhalt der Variablen, also die Texte selbst.

Im folgenden Code-Snippet werden zwei Texte mit dem Plus-Operator zusammengesetzt. Leerzeichen muss man extra angeben, siehe Code-Snippet unten.

```
string Variable1 = "Hallo" + "Europa"; // liefert HalloEuropa (ohne Leerzeichen)
```

Dieses Code-Snippet addiert zwei Variablen. In der letzten Zeile wird gezeigt, wie Leerzeichen verwendet werden.

```
private static void Main(string[] args)
{
    string Variable2 = "Hallo";
    string Variable3 = "Deutschland";
    string Variable4;
    Variable4 = Variable2 + Variable3; // liefert HalloDeutschland (ohneLeerzeichen)
    Console.WriteLine(Variable4);
    Variable4 = Variable2 + " " + Variable3; // liefert Hallo Deutschland (mit Leerzeichen)
    Console.WriteLine(Variable4);

    double Wetterwert = 15.7;
    string Einheit = "°";
    string WetterwertMitEinheit = Wetterwert + Einheit;
    Console.WriteLine(WetterwertMitEinheit);
    Console.ReadKey();
```

```
}
```

6.1.3 Texte teilen

Arbeitsauftrag

Zerlegen Sie einen Text, der durch (ein) bestimmte(s) (Sonder-)Zeichen getrennt ist, in seine Einzelteile.

Ein Text, der beispielsweise durch Leerzeichen oder Semikolon getrennt ist, kann in seine Einzelteile aufgeteilt werden. Damit kann man beispielsweise eine Adresse (im Routenplaner) in ihre Adressteile wie Postleitzahl, Ort, Straße und Hausnummer zerlegen, um damit die Adresse in der Datenbank finden zu können. Die Methode zum Teilen von Texten lautet Split.

Syntax

```
string Text = "Teil1; Teil2; Teil3";
string[] ZerteiIterText = Text.Split(';');
```

In diesem Beispiel werden Wetterdaten (z.B. Temperatur), die in einer csv-Datei liegen (dazu später mehr), in ihre Einzelteile zerlegt. In der csv-Datei (String-Variable Wetterdaten) sind hinterlegt: Datum, Uhrzeit, Wetterwert, Einheit. In diesem FaII handelt es sich um Grad (es kann auch Niederschlag oder Globalstrahlung sein).

```
private static void Main(string[] args)
{
    string Wetterdaten = "10.06.2015;7:30;15.7;°";
    string[] Datenteile;
    int Anzahl;
    Datenteile = Wetterdaten.Split(';');
    Anzahl = Datenteile.GetLength(0); for (int i = 0; i < Anzahl; i++)
    {
        Console.WriteLine(Datenteile[i]);
    }
    Console.ReadKey();
}
```

6.1.4 Anzahl von Zeichen ermitteln und auf Einzelzeichen zugreifen

Arbeitsauftrag

Schreiben Sie ein Programm, das den eingegebenen Text umdreht.

Um den Arbeitsauftrag zu erfüllen, müssen folgende Schritte durchgeführt werden:

- *Anzahl der Zeichen ermitteln*
- *rückwärts zählende Schleife über alle Zeichen, welche jedes einzelne Zeichen ausgibt*

Die Anzahl der Zeichen wird mit der Eigenschaft (Property) Lenght ermittelt:

```
int Anzahl = Eingabe.Length;
```

Über eine indizierte Eigenschaft kann auf einzelne Zeichen zugegriffen werden. Damit gemeint ist eine Eigenschaft mit dem Standardnamen this, dem ein Parameter (Index) übergeben werden kann. Damit kann man eine Klasse wie ein Array verwenden.

```
char Zeichen = Eingabe[i];
```

Mit diesen Anweisungen lässt sich der Arbeitsauftrag wie folgt Umsetzen:

```
private static void Main(string[] args)
{
    string Eingabe;
    int Anzahl;
    char Zeichen;
    Console.Write("Eingabe Text:");
    Eingabe = Console.ReadLine();
    Anzahl = Eingabe.Length;
    for (int i = Anzahl - 1; i >= 0; i--)
    {
        Zeichen = Eingabe[i];
        Console.Write(Zeichen);
```

```
    }
    Console.ReadKey();
}
```

6.1.5 Texte vergleichen

Arbeitsauftrag

Schreiben Sie ein Programm, das Wörter in einer Stichwortliste sucht. Dabei stehen folgende Suchoptionen zur Verfügung: Genaue Suche, Suchtext am Anfang, Suchtext am Ende, Suchtext irgendwo im Text.

Die Klasse string stellt verschiedene Methoden und Operatoren für die Suche innerhalb eines Textes zur Verfügung:

- *Vergleichsoperator ==*

  Damit können Texte in der gleichen Weise wie Zahlen verglichen werden.
- *Vergleichsoperator !=*

  Damit können zwei Texte auf ungleich geprüft werden.
- *Methode Contains prüft, ob der übergebene Text bzw. die String-Variable im string enthalten ist.*
- *Methode StartsWith prüft, ob der übergebene Text bzw. die String-Variable am Anfang des strings enthalten ist.*
- *Methode EndWith prüft, ob der übergebene Text bzw. die string-Variable am Ende des strings enthalten ist.*

Der Operator wird wie gewohnt verwendet:

```
if (Wetterdaten[i] == Suchwort)
```

Den letzten drei Methoden wird der Vergleichstext als Parameter übergeben und liefert als Rückgabewert bool zurück, das angibt, ob der Vergleichstext enthalten ist. Die Rückgabewerte Gefunden1, Gefunden2 und Gefunden3 im Code-Snippet enthalten den Wert true, da der Suchtext an der entsprechenden Stelle (Anfang, irgendwo, Mitte) vorkommt.

```
private static void Main(string[] args)
{
    string[] Wetterdaten = new string[] { "Temperatur-Prognose Dresden"
        , "Temperatur-Prognose Köln","Temperatur-Istwerte Dresden"
        , "Temperatur-Istwerte Köln","Niederschlags-Prognose Dresden"
        , "Niederschlags-Prognose Köln","Niederschlags-Istwerte Dresden"
        , "Niederschlags-Istwerte Köln", };

    string Suchart = "", Suchwort = "";

    Console.WriteLine("Suche Wetterdaten");
    Console.WriteLine("1: Genaue Suche");
    Console.WriteLine("2: Suchwort am Anfang");
    Console.WriteLine("3 : Suchwort am Ende");
    Console.WriteLine("4: Suche");
    Console.Write("Eingabe Suchart (1-4):");
    Suchart = Console.ReadLine();
    Console.Write("Eingabe Suchwort:");
    Suchwort = Console.ReadLine();
    Console.WriteLine("Suchergebnisse: ");

    for (int i = 0; i < Wetterdaten.GetLength(0); i++)
    {
        bool Gefunden = false; switch (Suchart)
        {
            case "1":
                if (Wetterdaten[i] == Suchwort) Gefunden = true;
                break;

            case "2":
                if (Wetterdaten[i].StartsWith(Suchwort)) Gefunden = true;
                break;

            case "3":
                if (Wetterdaten[i].EndsWith(Suchwort)) Gefunden = true;
                break;
```

```
            case "4":
                if (Wetterdaten[i].Contains(Suchwort)) Gefunden = true;
                break;

            default: break;
        }
        if (Gefunden)
            Console.WriteLine(Wetterdaten[i]);
    }
    Console.ReadKey();
}
```

6.1.6 Groß-/Kleinschreibung

Arbeitsauftrag

Schreiben Sie ein Programm, das einen Text sucht, wobei die Groß-/Kleinschreibung ignoriert werden soll.

Bei dem vorherigen Suchprogramm muss das Suchwort genauso eingegeben werden, wie es intern (im Array) abgespeichert ist. Ein Anwender gibt jedoch meistens ein Suchwort ungeachtet der Groß-/KIeinschreibung ein. Daher müssen für die genaue Suche Suchwort und Stichwort auf gleich gebracht werden, wenn die Groß-/Kleinschreibung ignoriert werden soll. Mit den Methoden ToLower bzw. ToUpper kann ein Text auf Klein- bzw. Großschreibung konvertiert werden. Das ist ein beliebter „Trick", um eine Suche oder den Vergleich von Texten zu ermöglichen. Es gibt aber auch noch andere Anwendungsfälle, wo ToLower und ToUpper verwendet werden. Die Syntax lautet wie folgt:

Syntax

```
public string ToLower();
public string ToUpper();
```

Das folgende Code-Snippet konvertiert den Text auf Kleinschreibung:

```
string Test = "KleinGrossschreibung";
string Klein = Test.ToLower();
```

Das aktuelle Stichwort in der FOR-Schleife wird in die Variable Stichwort gespeichert. Sowohl das Suchwort (im switch) als auch das Stichwort werden mit ToLower auf gleich gebracht. So ist es möglich, dass man Texte unabhängig von der Groß-/KIeinschreibung vergleichen kann.

```
static void Main(string[] args)
{
    string[] Wetterdaten = new string[]
        { "Temperatur-Prognose Dresden", "Temperatur-Prognose Köln",
            "Temperatur-Istwerte Dresden", "Temperatur-Istwerte Köln",
            "Niederschlags-Prognose Dresden", "Niederschlags-Prognose Köln",
            "Niederschlags-Istwerte Dresden", "Niederschlags-Istwerte Köln",
        };
    string Suchart = "", Suchwort = "";
    string Stichwort = "";

    Console.WriteLine("Suche Wetterdaten");
    Console.WriteLine("1: Genaue Suche");
    Console.WriteLine("2: Suchwort am Anfang");
    Console.WriteLine("3: Suchwort am Ende");
    Console.WriteLine("4: Suche");

    Console.Write("Eingabe Suchart (1-4):");
    Suchart = Console.ReadLine();

    Console.Write("Eingabe Suchwort:");
    Suchwort = Console.ReadLine();
    Suchwort = Suchwort.ToLower()

    Console.WriteLine("Suchergebnisse:");
    for (int i = 0; i < Wetterdaten.GetLength(0); i++)
    {
        bool Gefunden = false;
```

```
        Stichwort = Wetterdaten[i].ToLower();
        switch (Suchart.ToLower())
        {
            case "1":
                if (Stichwort == Suchwort)
                    Gefunden = true;
                break;
            case "2":
                if (Stichwort.StartsWith(Suchwort))
                    Gefunden = true;
                break;
            case "3":
                if (Stichwort.EndsWith(Suchwort))
                    Gefunden = true;
                break;
            case "4":
                if (Stichwort.Contains(Suchwort))
                    Gefunden = true;
                break;
            default:
                break;
        }
        if (Gefunden)
            Console.WriteLine(Wetterdaten[i]);
    }
    Console.ReadKey();
}
```

6.1.7 Texte ersetzen

Arbeitsauftrag

Ersetzen Sie Umlaute durch ae, oe, ue bzw. ß durch ss.

Die Methode Replace ersetzt einen Text(teil) durch einen anderen Text. Achtung: Dabei wird der ersetzte Text als Rückgabewert geliefert, der Originalstring bleibt unverändert.

Syntax

```
public string Replace(string oldValue, string newValue);
```

Das folgende Code-Snippet ersetzt den Text(teil) „Anfang" durch das Wort „Start" und retourniert das Ergebnis in die Variable NeuerString.

```
string StringVariable = "AnfangMitteEnde";
string NeuerString = StringVariable.Replace("Anfang", "Start");
```

Klasse SearchTerm in Datei SearchTerm.cs

```
public class SearchTerm
{
    public static string GetSearchTerm(string SearchTerm)
    {
        string Term = SearchTerm;
        Term = Term.ToUpper();
        Term = SearchTerm.Replace("Ä", "AE");
        Term = SearchTerm.Replace("Ö", "OE");
        Term = SearchTerm.Replace("Ü", "UE");
        Term = SearchTerm.Replace("ß", "SZ");
        return Term;
    }
}
```

Main-Methode in Datei Program.cs

```
private static void Main(string[] args)
{
    string Suchstring = "bfw";
    string NeuerSuchstring = SearchTerm.GetSearchTerm(Suchstring);
}
```

6.1.8 Weitere Themenbereiche: Klasse StringBuilder

Mit der Klasse Stringbuilder lassen sich einfache und komplexe Texte zusammensetzen. Generell wird empfohlen, anstelle der Klasse string die Klasse Stringbuilder zu verwenden, wenn viel Text zusammengesetzt werden soll, da die Performance der Klasse StringBuilder besser ist. Hintergrund: Werden mehrere Texte mit der Klasse string über den Plusoperator zusammengesetzt, wird jedes Mal neuer Speicher angelegt. Hingegen optimiert die Klasse Stringbuilder den Speicher und damit auch die Performance.

Die Klasse stellt folgende wichtige Methoden zur Verfügung:

- *Append: Fügt einen einfachen Text an.*
- *AppendLine: Fügt eine Zeile mit Zeilenumbruch ein.*
- *AppendFormat: Fügt eine oder mehrere Variablen in den Text ein.*
- *ToString: Gibt einen string zurück (für Ausgabe in Konsole oder Weiterverarbeitung).*

Im folgenden Code-Snippet werden drei Variablen deklariert, die über den Stringbuilder unterschiedlich formatiert werden. Dabei funktioniert die Methode AppendFormat wie die Methode Console.WriteLine: Es werden 1 bis n Platzhalter definiert und anschließend 1 bis n Variablen übergeben. Ist der Text im gewünschten Format zusammengesetzt, kann er mit der Methode ToString in eine String-KIasse zur Weiterverarbeitung umgewandelt werden.

```
int Variable1 = 100;
double Variable2 = 200;
DateTime Datum = new DateTime(2015, 6, 1);
Stringbuilder Text = new Stringbuilder();
Text.Append("Das ist eine Zeile ohne Werte ");
Text.AppendLine();
Text.AppendFormat("und das mit 2 Variablen: {0) {1}",Variable1,Variable2);
Text.AppendLine("In neuer Zeile:");
Text.AppendFormat("Das Datum: {0}", Datum);
Console.WriteLine(Text.ToString());
Console.ReadKey();
```

Aufgaben

1. Entwerfen und implementieren Sie eine AdressKlasse, welche folgende Eigenschaften aufweist: Vor- und Nachname, Postleitzahl, Ort, Straße und Hausnummer. Die Methode BriefAdresse liefert einen string, der die Adresse für einen Brief enthält, also im Format:
   - *Vorname Nachname*
   - *Straße Hausnummer*
   - *PLZ Ort*

Verwenden Sie die Methode AppendLine der Klasse StringBuiIder, um eine neue Zeile in einem String zu erzeugen. Die Ausgabe der Adresse in der Konsole erfolgt in der Methode Main.

2. Ein Text ist beispielsweise wie folgt aufgebaut: B;20.4

Der Buchstabe (B) repräsentiert die Qualität des nachfolgenden Wertes (20.4): B steht für Beobachtungswert und P für Prognosewert. Entwerfen und implementieren Sie eine Klasse namens Wetterdaten. Dem Konstruktor wird als Parameter der oben genannte Text (B;20.4 oder auch ein anderer Wert wie P;19.7) übergeben.

Diese Klasse besitzt folgende zwei readonly-Properties:

- *Wert (Datentyp Integer) liefert den Wert nach dem Semikolon.*
- *Qualität (Enum: Prognosewert, Beobachtungswert) liefert die Qualität des Wertes als Enum.*

6.2 Datum und Uhrzeit verarbeiten mit Klasse DateTime und TimeSpan

Die Klasse DateTime speichert ein Datum und stellt verschiedene Methoden und Eigenschaften zur Manipulation des Datums zur Verfügung. Mithilfe der Klasse Timespan wird die Differenz von zwei Datumswerten gebildet.

6.2.1 DateTime initialisieren

Arbeitsauftrag

Implementieren Sie eine Klasse namens Wetterdaten, welche folgende Eigenschaften besitzt: Zeitstempel, Wert und Einheit (string). Instanziieren Sie ein Objekt, das anschließend in der Konsole ausgegeben wird.

Die KIasse DateTime kann sowohl zum Speichern von Datumswerten ohne Zeitangabe (13.2.2015) als auch Datumswerten mit Zeitangaben (13.2.2015 21:20) verwendet werden. Daher existieren u.a. folgende zwei überladenen Konstruktoren. Der ersten Syntaxvariante werden Jahr, Monat und Tag übergeben. Hingegen werden der zweiten Syntaxvariante Jahr, Monat, Tag, Stunde, Minute und Sekunde übergeben.

Syntax

```
DateTime Datum = new DateTime(Jahr, Monat, Tag);
DateTime Datum = new DateTime(Jahr, Monat, Tag, Stunde, Minute, Sekunde);

DateTime Eintrittsdatum = new DateTime(14, 12, 2015);
DateTime Zeitstempel = new DateTime(2015, 12, 30, 16, 30, 0);
```

Lösung zum Arbeitsauftrag

Die Datumsvariable Zeitstempel kann der Methode ConsoIe.Write() als Parameter übergeben werden. Sofern kein anderes Format definiert wird, wird das Format aus der Systemeinstellung verwendet.

Methode Main in Datei Program.cs

```
static void Main(string[] args)
{
    DateTime Zeitstempel = new DateTime(2011, 6, 13, 21, 42, 0);
    Wetterdaten WetterAktuell = new Wetterdaten(Zeitstempel, 22, "°C");
    Console.Write("Die Temperatur um {0} beträgt {1}{2}",
        WetterAktuell.Zeitstempel, WetterAktuell.Wert, WetterAktuell.Einheit);
    Console.ReadKey();
}
```

Klasse Wetterdaten in Datei Wetterdaten.cs

```
public class Wetterdaten
{
    public Wetterdaten(DateTime pZeitstempel, double pWert, string pEinheit)
    {
        Zeitstempel = pZeitstempel;
        Wert = pWert;
        Einheit = pEinheit;
    }

    public DateTime Zeitstempel { get;  set; }
    public double Wert { get; set; }
    public string Einheit { get;  set; }
}
```

6.2.2 Vergleichsoperatoren

Arbeitsauftrag

Schreiben Sie ein Programm (Klasse), welches die meteorologische Jahreszeit zum übergebenen Datum ausgibt.

Wie bei Zahlen oder Strings können auch zwei Datumswerte miteinander verglichen werden. Dazu stehen folgende Vergleichsoperatoren zur Verfügung:

- *kleiner: <*
- *größer: >*
- *gleich: =*
- *kleiner gleich: <=*
- *größer gleich: >=*

Lösung zum Arbeitsauftrag

Klasse MeteorologischerFruehling in Datei MeteorologischerFruehling.cs

Die Datumsgrenzen (1. März, 1. Juni etc.) werden nur temporär für den Vergleich benötigt, daher müssen für die Datumsgrenzen keine Datumsvariablen verwendet werden.

Damit immer das korrekte Jahr des Parameters Datum verwendet wird, wird auf die Eigenschaft Jahr (siehe Kapitel 6.2.3) zugegriffen.

```
public enum JahreszeitEnum {Fruehling, Sommer, Herbst, Winter};

public class MeterologischerFruehling
{
    public static JahreszeitEnum GetJahreszeitZuDatum(DateTime Datum)
    {
        if (Datum >= new DateTime(Datum.Year, 3, 1) &
            Datum < new DateTime(Datum.Year, 6, 1))
        {
            return JahreszeitEnum.Fruehling;
        }
        else if (Datum >= new DateTime(Datum.Year, 6, 1) &
            Datum < new DateTime(Datum.Year, 9, 1))
        {
            return JahreszeitEnum.Sommer;
        }
        else if (Datum >= new DateTime(Datum.Year, 9, 1) &
            Datum < new DateTime(Datum.Year, 12, 1))
        {
            return JahreszeitEnum.Herbst;
        }
        else
        {
            return JahreszeitEnum.Winter;
        }
    }
}
```

Der Aufruf in der Methode Main lautet wie folgt:

```
static void Main(string[] args)
{
    DateTime Datum = new DateTime(2011, 12, 6);
    JahreszeitEnum Jahreszeit = MeterologischerFruehling.GetJahreszeitZuDatum(Datum);
    Console.WriteLine("Datum {0} fällt in Jahreszeit {1}", Datum, Jahreszeit);
    Console.ReadKey();
}
```

6.2.3 Auf Datumselemente zugreifen

Arbeitsauftrag

Erweitern Sie die Klasse Wetterdaten so, dass nur halbe und ganze Stunden (16:00, 16:30) dem Konstruktor übergehen werden können. Der Sekundenwert muss 0 sein.

Um den Arbeitsauftrag erfüllen zu können, müssen Sie im Konstruktor auf die Datumselemente wie Stunde oder Sekunde zugreifen können. Die Klasse DateTime stellt folgende lesende Eigenschaften für den Zugriff auf die Datumselemente zur Verfügung:

- *Year*
- *Month*
- *Day*
- *Hour*
- *Minute*
- *Second*
- *Millisecond (es gibt eine Konstruktorvariante, welcher Millisekunden übergeben werden können)*

- *DayOfYear (Anzahl Tage im Jahr: 365 oder 366)*
- *DayOfWeek (liefert ein Enum, welches den Wochentag enthält, beispielsweise Montag)*

Quellcode siehe Lösungsdateien.

Wird eine nicht erwartete Stunde oder Sekunde übergeben, wird eine Exception geworfen.

6.2.4 Aktuelles Datum

Arbeitsauftrag

Ermitteln Sie das aktuelle Datum.

Das aktuelle Datum wird mit der statischen Eigenschaft Now ermittelt, welches ein DateTime-Objekt retourniert.

Quellcode siehe Lösungsdateien.

6.2.5 Datumselemente verändern

Arbeitsauftrag

Schreiben Sie ein Programm, das alle Tage eines Monats ausgibt.

Auf die oben genannten Eigenschaften Year, Month etc. kann nur lesend zugegriffen werden. Möchte man die einzelnen Datumselemente verändern, stehen folgende Methoden zur Verfügung:

Syntax

```
public DateTime AddYears(int value)
public DateTime AddMonths(int value);
public DateTime AddDays(double value);
public DateTime AddHours(double value);
public DateTime AddMinutes(doubIe value);
public DateTime AddSeconds(double value);
public DateTime AddMilliseconds(double value);
```

So addiert im Code-Snippet beispielsweise die Methode AddMonths einen Monat zum Datumswert in der Variablen StartDatum.

```
DateTime EndeDatum = StartDatum.AddMonths(1);
```

Wie bei fast allen DateTime-Methoden wird hier nicht die Originalvariable verändert, sondern der veränderte Wert als Rückgabewert zurückgegeben. Soll ein Datumsintervall (Jahr, Monat, Tag etc.) subtrahiert werden, werden negative Werte übergeben. Übergibt man beispielsweise der AddMonths-Methode -2, werden zwei Monate abgezogen.

Quellcode siehe Lösungsdateien

6.2.6 Datum formatieren

Arbeitsauftrag

Formatieren Sie das Datum aus dem vorherigen Arbeitsauftrag so, dass nur der Datumsteil ohne Uhrzeit ausgegeben wird.

Wenn Sie den Arbeitsauftrag aus dem vorherigen Kapitel ausgeführt haben, werden Sie bemerkt haben, dass sowohl Datum als auch Uhrzeit (Mitternacht) ausgegeben werden. Diese Formatierung ist abhängig von den „Regions- und Sprachoptionen" des Betriebssystems. Gehen Sie in die Systemeinstellungen und ändern Sie die regionale Einstellung beispielsweise auf „Englisch" (Großbritannien). Starten Sie den Arbeitsauftrag aus dem vorherigen Kapitel erneut. Ohne dass Sie etwas umprogrammieren mussten, wird das Datums- und Uhrzeitformat der jeweiligen Region angezeigt.

Das Format von Datum und Uhrzeit lässt sich jedoch auch beeinflussen. Dafür stellt die DateTime-Klasse u.a. folgende Methoden zur Verfügung.

Methodenname	Beschreibung/Ausgabebeispiel
public string ToLongDateString();	Gibt das Datum im langen Format aus: Donnerstag, 16. |uni 2015
public string ToLongTimeString();	Gibt die Uhrzeit im langen Format aus: 13:26:16 (Stunde, Minute, Sekunde)
public string ToShortDateString();	Gibt das Datum im kurzen Format aus: 11.06.2015 (Jahr, Monat, Tag)
public string ToShortTimeString();	Gibt die Uhrzeit im kurzen Format aus: 13:26 (Stunde, Minute)
public string G-String(string format);	Enthält ein Standard- oder benutzerdefiniertes Datumsformat, siehe nächste Tabelle

Die Methoden, welche mit ToLong und ToShort beginnen, sowie das Standarddatumsformat der ToString-Methode sind von den Ländereinstellungen des Betriebssystems abhängig.

Für ein benutzerdefiniertes Datumsformat können folgende Formatbezeichner verwendet werden. Hier finden Sie die wichtigsten:

Formatbezeichner	Beschreibung/Beispiele
d	Tag. Anzeige 1- oder 2-stellig (Bsp. 1 oder 13)
dd	Tag. Anzeige immer 2-stellig (Bop: 01 oder 13)
ddd	Tag als Kurzname, Bsp.: Mo für Montag
dddd	Tag als Name, Bsp.: Montag, Dienstag
h	Stunde im 12-Stunden-Format, 1- oder 2-stellig
hh	Stunde im 12-Stunden-Format, immer 2-stellig
H	Stunde im 24-Stunden-Format, 1- oder 2-stellig
HH	Stunde im 24-Stunden-Format, immer 2-stellig
m	Minute 1- oder 2-stellig
mm	Minute immer 2-stellig
M	Monat 1- oder 2-stellig
MM	Monat immer 2-stellig
MMM	Monat als Kurzname, Bsp.: Jun für Juni
MMMM	Monat als Name, Bsp.: Juni
S	Sekunden 1- oder 2-stellig
SS	Sekunden immer 2-stellig
y	Jahr 1- oder 2-stellig
yy	Jahr immer 2-stellig
yyyy	Jahr 4-stellig

Das folgende Code-Snippet zeigt ein Beispiel für ein übliches Format im deutschsprachigen Raum im 24-Stunden-Format. Die Datumselemente werden mit Punkt getrennt, die Uhrzeit mit Doppelpunkt.

```
StartDatum.ToString("dd.MM.yyyy HH:mm:ss")
```

Da nur das Datum ausgegeben werden soll, werden die Formatbezeichner für die Uhrzeit ausgelassen.

Quellcode siehe Lösungsdateien

6.2.7 Weitere Themenbereiche

Methode ToUniversalTime

Ihre Uhr haben Sie höchstwahrscheinlich auf die lokale Zeit eingestellt. Die Welt ist in verschiedene Zeitzonen eingeteilt. Deutschland befindet sich beispielsweise in der Zeitzone der Mitteleuropäischen Sommerzeit (MESZ). Zusätzlich zur Zeitzone haben wir noch einen Sommerzeit-Winterzeit-Wechsel (letzter Sonntag im Oktober: Die Stunde 2 gibt es doppelt) und einen Winterzeit-Sommerzeit-Wechsel (letzter Sonntag im März, die Stunde 2 wird ausgelassen).

Darüber hinaus gibt es neben Zeitzonen mit Wechsel auch eine sogenannte koordinierte Weltzeit (UTC, Universal Time Coordinated), welche keinen Sommerzeit-Winterzeit-Wechsel und Winterzeit-Sommerzeit-Wechsel enthält, und die der Zeitzone entlang des Nullmeridians (auch Greenwich-Meridian genannt) entspricht. Die Winterzeit (MEZ) hat eine Differenz von zwei Stunden gegenüber UTC, hingegen hat die Sommerzeit (MESZ) eine Stunde Zeitdifferenz.

Die UTC wird mithilfe der Methode ToUniversalTime ermittelt. Diese Methode gibt ebenfalls ein DateTime-Objekt zurück.

```
DateTime Lokal = new DateTime(2015, 6, 1);
DateTime UTC = Lokal.ToUniversalTime();
Console.WriteLine("Lokal {0}, UTC: {1}", Lokal, UTC);
Console.ReadKey();
```

Klasse Timespan

In der Klasse Timespan können Sie eine Zeitspanne abbilden. Sie entsteht beispielsweise, wenn Sie zwei Datumswerte subtrahieren. Auch können Sie zu einem Datumswert einen Zeitbereich addieren und zahlreiche andere Operationen durchführen.

Das folgende Code-Snippet errechnet die Zeitspanne zwischen zwei Datumswerten:

```
DateTime Datum1= new DateTime(2011,6,1);
DateTime Datum2 = new DateTime(2011,9,1);
Timespan Zeitdifferenz = Datum2 - Datum1;
Console.WriteLine("Differenz in Tage: {0}",Zeitdifferenz.Days);
Console.ReadKey();
```

Aufgaben

1. Entwerfen und implementieren Sie ein Programm, welches von einem Jahr den Monatsletzten aller Monate ausgibt. Beispiel: 31.Januar, 28. Februar etc.; Eingabe: Jahr. Verwenden Sie AddMonths und AddDays.
2. Viele Programme in Unternehmen oder auch in Privathaushalten arbeiten oft mit anderen Programmen zusammen und tauschen dabei Daten aus. Der Datenaustausch kann dabei auf unterschiedliche Art und Weise erfolgen. Das betrifft sowohl die Art des Transports (Datei, http) als auch die Struktur und das Format der Daten. Das Datenformat kann sowohl herstellerspezifisch als auch standardisiert sein oder ein sonstiges bekanntes Format sein. So gibt es etwa auch für die Darstellung der UTC ein genormtes Format, wenn Programme Daten mit Zeitangaben austauschen möchten.
   Das UTC-Format nach ISO 8601 bietet einige Formatvariationen. In dieser Aufgabe soll aus einem Datum folgendes Format generiert werden (entwerfen und implementieren Sie eine eigene Klasse):

   JJJJmmddthhmmssz

   JJJJ = Jahr, MM = Monat, DD = Tag, T = Trennzeichen zwischen Tag und Uhrzeit HH = Stunde, MM = Minute, SS = Sekunde, Z = Kennung für UTC

6.3 Textdateien lesen mit Klasse StreamReader

Arbeitsauftrag

Implementieren Sie ein Programm, welches Daten im CSV-Format aus einer Datei liest.

Die Klasse StreamReader stellt Funktionalitäten zum Lesen von Textdateien zur Verfügung. Dabei kann der Inhalt einer Textdatei auf einmal oder zeilenweise ausgelesen werden. In beiden Fällen wird beim Instanziieren dem Konstruktor der Pfad inklusive Dateiname übergeben. Der Vollständigkeit halber sei erwähnt, dass es auch noch andere überladene Konstruktoren gibt.

Syntax

```
public StreamReader(string path);
```

In C# ist der Backslash (\) ein sogenanntes Escape-Zeichen. Ein Escape-Zeichen gibt an, dass der umgekehrte Schrägstrich kein druckbares Zeichen ist, sondern eine Sonderfunktion einleitet. Eine Sonderfunktion ist beispielsweise der Tabulator (\t) oder eine neue Zeile (\n). Möchte man einen Backslash trotzdem ausdrucken, muss man ihn verdoppeln (W) oder alternativ einen verbatim string einfügen. Beim verbatim string wird vor dem String-Wert ein Klammeraffe (@ angegeben, wodurch die Escape-Zeichen ignoriert und 1:1 ausgegeben werden.

```
StreamReader Datei = new StreamReader(@"C:\Testtext.txt");
```

Nachdem der Textinhalt auf einmal oder zeilenweise ausgelesen wurde (siehe nächste Kapitel), sollte der Zugriff mit der Methode Close wieder geschlossen werden, damit die Ressource freigegeben wird und andere Programme darauf zugreifen können.

Syntax

```
public override void Close( );
```

Wird eine Fehlerbehandlung verwendet (siehe Kapitel 5.2), wäre der finally-Block ein typischer Ort, wo die Datei geschlossen werden sollte (da der finally-Block auch im Fehlerfall durchlaufen wird).

Um den StreamReader nutzen zu können, muss man den NamespaceSystem.IO importieren Fügen Sie dazu die using-Direktive using System.IO wie folgt hinzu:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.IO;

namespace Readline
{
    class Programm
    {
        static void Main(string[] args)
        {
        }
    }
}
```

6.3.1 Textdatei auf einmal auslesen

Arbeitsauftrag

Schreiben Sie ein Programm, das den Inhalt einer Textdatei ausgibt.

Nach dem Instanziieren der Klasse StreamReader wird die Datei mit der ReadToEnd komplett ausgelesen. Der Inhalt der Textdatei wird als String-Objekt retourniert.

Syntax

```
public override string ReadToEnd();

string Textinhalt = Datei.ReadToEnd();
```

Nach dem Auslesen sollte die Datei mit Close geschlossen werden, siehe Source Code des Arbeitsauftrags.

Arbeitsauftrag

Öffnen Sie Notepad++ und schreiben Sie einen beliebigen Text hinein. Speichern Sie die Textdatei namens Testtext.txt auf der Festplatte im Rootverzeichnis C ab.

```
static void Main(String[] args)
{
    StreamReader Datei = new StreamReader(@"C:\Testtext.txt");
    string Textinhalt = Datei.ReadToEnd();
    Console.WriteLine(Textinhalt);
    Console.ReadKey();
    Datei.Close();
}
```

6.3.2 Textdatei zeilenweise auslesen

Arbeitsauftrag

Schreiben Sie ein Programm, das den Inhalt einer Textdatei zeilenweise ausliest.

Angenommen, Sie haben Wetterwerte als CSV-Datei vorliegen. Eine CSV-Datei (Dateiendung .csv) wird verwendet, um Daten zwischen verschiedenen Systemen auszutauschen. Der Inhalt sind ASCII-Zeichen, um etwa Tabellen oder Listen zu übertragen. Die Spalten oder Listenwerte werden beispielsweise mit Semikolon getrennt.

Beispiel

Mögliche Textinhalte für Temperaturwerte:

6.1.2O15;16:00;21.7;Grad

6.1.2015;17:00;21.3,Grad

In dieser Textdatei enthält die erste Spalte das Datum, die zweite die Uhrzeit, die dritte den Wert und die vierte die Einheit. Sollen nun diese Werte in der Textdatei weiterverarbeitet werden, beispielsweise in eine Datenbank importiert werden, ist es oft einfacher die Textdatei zeilenweise auszulesen und die Zeilen einzeln zu verarbeiten.

Um eine Textdatei zeilenweise zu lesen sind folgende Schritte notwendig:

- *Prüfen, ob das Dateiende erreicht wurde (Eigenschaft EndOfStream)*
- *Textzeile auslesen (Methode Readline)*

Wird eine Datei geöffnet, zeigt die Klasse StreamReader auf die erste Zeile. Nachdem die erste Zeile mit der Methode ReadLine ausgelesen wurde, zeigt die Klasse StreamReader auf die zweite Zeile usw. Das Ganze geht so lange, wie es Zeilen in der Textdatei gibt. Diese liefert so lange true, wie Zeilen vorhanden sind. Daher muss man mit der Eigenschaft EndOfStream prüfen, ob Dateien vorhanden sind. Sind keine Zeilen mehr vorhanden, liefert die Eigenschaft false.

Lösung zum Arbeitsauftrag

Legen Sie eine Textdatei namens Wetterdaten.csv an und kopieren Sie die Werte aus dem obigen Beispiel. In diesem Beispiel werden die Textzeilen mit einer While-Schleife so lange ausgelesen, bis EndOfStream false liefert. Mit Split werden die Werte der Datenzeile extrahiert, damit sie weiterverarbeitet werden können. Beispielsweise können die Werte in eine Datenbank importiert werden.

Quellcode siehe Lösungsdateien

Aufgaben

1. Erweitern Sie das Programm „POIs verwalten“ um eine neue Funktion namens „Sightseeing POI laden". Diese Funktion liest eine CSV-Datei ein und fügt sie der internen POI-Liste hinzu.
2. Entwerfen und implementieren Sie ein Programm, das Wetterdaten aus einer CSV-Datei ausliest und in der Konsole auflistet. Verwenden Sie die Klasse Wetterdaten aus Kapitel 6.1. Erweitern Sie diese um die Eigenschaft Von, d. h., eine Zeile in der Datei besteht aus Datum (inkl. Uhrzeit), Wert und Qualität.

6.4 Textdateien schreiben mit Klasse StreamWriter

Arbeitsauftrag

Schreiben Sie ein Programm, das Texte in eine Datei schreibt.

Die Klasse StreamWriter stellt mehrere Konstruktoren zur Verfügung. Für dieses Beispiel ist folgender Konstruktor wichtig:

Syntax

```
public StreamWriter(string path, bool append);
```

Diesem überladenen Parameter wird einerseits der Pfad inklusive Dateiname übergeben. Andrerseits bestimmt ein boolescher Wert namens append, ob bei einer bestehenden Datei der Textinhalt überschrieben (append = false) oder angehängt (append = true) wird.

```
StreamWriter Datei = new StreamWriter(@"C:\Test.txt", false);
```

Mit den Methoden WriteLine und Write kann ein Text zeilenweise (WriteLine) oder auf einmal (Write) in die Textdatei geschrieben werden.

Syntax

```
public virtual void WriteLine(string value);
public override void Write(string value);
```

Die Datei sollte wieder mit der Methode Close geschlossen werden.

Quellcode siehe Lösungsdateien

Aufgaben

1. Schreiben Sie eine Klasse Logging, die sowohl die Programminformationen als auch Exceptions in eine Datei schreibt. Folgende statische Methoden schreiben automatisch immer einen Zeitstempel (Datum + Uhrzeit) als Erstes in die Textdatei. Erst dann folgt der Info- oder Fehlertext.
   - *Methode Info: Parameter Message (Typ string)*
   - *Methode Fehler: Parameter ExceptionInfo (Typ Exception)*

Testen Sie die Klasse, erzeugen Sie absichtlich Fehler. Lesen Sie beispielsweise mit Konsole. Readline Werte ein, die in eine Zahl konvertiert werden. Geben Sie einmal eine Zahl und einmal einen Text ein.

2. Erweitern Sie das Programm „POIs verwalten" um eine neue Funktion namens „Sightseeing POI in Datei speichern". Diese Funktion speichert einen auszuwählenden Sightseeing-POI aus der internen Liste in eine CSV-Datei.

6.5 Dateien- und Verzeichnisoperationen mit Klassen Directory, Path und File

Arbeitsauftrag

Schreiben Sie ein Programm, das Dateien aus einem Verzeichnis in ein Archivverzeichnis verschiebt.

Angenommen, in einem Verzeichnis (z.B. C:\Import) liegen Textdateien (z.B. CSV- Dateien, die Wetterdaten enthalten). Nachdem diese Textdateien verarbeitet wurden (z. B. von einem Datenimport), sollen sie in ein Archivverzeichnis verschoben werden. Dabei wird pro Monat ein neues Archivverzeichnis angelegt (Format: Jahr_Monat, z. B. 2015_Jun). Für diese Aufgabe sind die Klassen Directory und File gut geeignet.

6.5.1 Prüfen, ob ein Verzeichnis existiert

Die Klasse Directory stellt die statische Methode Exists zur Verfügung.

Syntax

```
public static bool Exists(string path);
```

Dieser Methode wird das Verzeichnis als Parameter übergeben. Die Methode liefert true, wenn das Verzeichnis existiert.

```
bool Exists = Directory.Exists(@"C:\");
```

6.5.2 Verzeichnis anlegen

Zum Anlegen eines Verzeichnisses kann die statische Methode CreateDirectory der Klasse Directory verwendet werden:

Syntax

```
public static DirectoryInfo CreateDirectory(string path);
```

Das folgende Code-Snippet legt das Verzeichnis Import an, in dem das Verzeichnis als Parameter übergeben wird.

```
Directory.CreateDirectory(@"C:\Import");
```

6.5.3 Verzeichnisnamen zusammensetzen

Wenn ein Verzeichnisname dynamisch zusammengesetzt wird, gibt es mehrere Möglichkeiten, den gesamten Verzeichnisnamen zusammenzusetzen:

- *Klasse string: string Verzeichnis = „C:\Import" + „2015_Juni"*
- *Klasse Stringbuilder: Methode AppendFormat*
- *Klasse Path: Methode Combine*

Letztere Möglichkeit funktioniert wie folgt:

```
string Verzeichnis = Path.Combine("C:", "Import");
```

Der statischen Methode Combine kann man beliebige Verzeichnisteile übergeben (1 bis n, ähnlich wie bei Console.Write, bei der man mehrere Variablen übergeben kann). Bei dieser Methode braucht man sich auch nicht darum zu kümmern, ob ein Backslash angegeben ist oder nicht. Das erledigt die Methode Combine. Das Ergebnis des obigen Code-Snippets ist C:\Import.

6.5.4 Dateinamen extrahieren

Mit Combine kann man Verzeichnisnamen zusammensetzen. Liegt jedoch ein Verzeichnis oder ein kompletter Dateiname inklusive Pfad vor, gibt es in der Path-Klasse unterschiedliche Methoden zum Extrahieren der einzelnen Datei- oder Verzeichnisteile.

Syntax

```
public static string GetFileName(string path);
```

Folgender Aufruf liefert den Dateinamen inklusive Dateiendung:

```
string FileNameRelativ = Path.GetFileName(@"C:\Import\File1.txt");
```

6.5.5 Dateien suchen

Die statische Methode Path.GetFileName liefert ein Array der gesuchten Dateien zurück.

Syntax

```
public static string[] GetFiles(string path, string SearchPattern);
```

Der Methode werden das Verzeichnis und ein Suchmuster übergeben. Das Suchmuster kann Wildcards enthalten, ähnlich wie bei der Dateisuche im File Explorer. Das folgende Code-Snippet liefert alle Dateien im Verzeichnis mit der Dateiendung .txt. Sind keine Dateien enthalten, ist das Array leer.

```
string[] FilesToMove = Directory.GetFiles(@"C:\Import", "*.txt");
```

6.5.6 Dateien verschieben

Die statische Methode Move der Klasse File verschiebt eine Datei.

Syntax

```
public static void Move(string sourceFileName, string destFileName);
```

Das folgende Code-Snippet verschiebt die Datei C:\Import\Test.txt nach C:\Import\ Archiv\Test.txt

```
File.Move(@"C:\Import\Test.txt", @"C:\Import\Archiv\Test.txt");
```

6.5.7 Weitere Themenbereiche

Es gibt noch weitere Methoden in den Klassen Directory, Path und File, wie beispielsweise zum Löschen oder Umbenennen von Dateien oder Verzeichnissen. Die weitere Auflistung dieser Methoden und Klassen würde jedoch über den Modulumfang hinausführen.

6.5.8 Entwurf und Realisierung

Arbeitsauftrag

Die Analyse ergibt für diesen Arbeitsauftrag folgenden Ablauf:

- *Name des Monatsarchiv-Verzeichnisses ermitteln (Format Jahr_Monat)*
- *Wenn das Monatsarchiv noch nicht existiert, wird dieses angelegt.*
- *zu archivierende Dateien ermitteln*
- *jede ermittelte Datei ins Monatsarchiv verschieben*

Legen Sie ein Verzeichnis namens Import im Rootverzeichnis C an. Als nächstes legen Sie im Importverzeichnis ein paar Textfiles an (im File Explorer über Kontextmenü), die keinen Inhalt haben müssen. Die Textfiles müssen lediglich die Endung .txt haben.

Mithilfe der ToString-Methode der DateTime-KIasse wird der Verzeichnisname des Monatsarchivs ermittelt und in der Variablen MonatsarchivReIativ gespeichert. Die Combine-Methode der Klasse setzt den Verzeichnisnamen zusammen und speichert ihn in der Variablen MonatsarchivFullname. Anschließend wird geprüft, ob das Verzeichnis schon existiert oder gegebenenfalls noch angelegt werden muss. Die zu archivierenden Dateien werden mit GetFiles ermittelt und einzeln in einer FOR-Schleife verschoben. Da in dem Array FilesToMove der komplette Dateiname mit Verzeichnis steht, muss der Dateiname mit GetFiIeName extrahiert werden, um den Namen der Zieldatei zusammensetzen zu können. Erst dann kann die Datei endgültig verschoben werden.

Quellcode siehe Lösungsdateien

Aufgabe

1. In einem Verzeichnis liegen Dateien (z.B. Berichte oder Aggregatsdaten), deren Dateinamen folgendes Format haben: YYYYMM_LieferantZZZ.csv. In diesem Verzeichnis können Dateien aus unterschiedlichen Monaten liegen, da verschiedene Sachbearbeiter Dateien für unterschiedliche Monate erstellen. Erstellen Sie ein Programm, das nach Eingabe eines Datums (Monat/Jahr) die Dateien des eingegebenen Monats in ein Archiv verschiebt. Die Ordnerstruktur des Archivs besteht aus zwei Ebenen: Jahr (im Format YYYY) und Monat (MM).

6.6 Auflistungen mit Klasse List

Arbeitsauftrag

Schreiben Sie ein Programm, das in einer Liste beliebig viele Objekte (einer Klassenhierarchie) verwaltet. Dabei gibt der Benutzer die Zahlen ein. Am Ende werden die Summe und der Durchschnitt berechnet.

In den vorherigen Kapiteln wurden ein- oder mehrdimensionale Arrays verwendet. Der Nachteil dieser Arrays ist, dass die Anzahl an Elementen fix definiert werden muss. Dies geschieht entweder schon bei der Deklaration oder später im Source Code. Die Anzahl der Elemente in einer Liste kann jedoch auch unbekannt sein und zur Laufzeit variieren, also dynamisch sein. Dynamische Listen lassen sich in .NET mit generischen Collections realisieren.

Eine Liste kann wie auch in der Realität verschiedene Datentypen haben. Beispielsweise kann man eine Zahlenliste bilden, um eine Summe von unterschiedlichen Werten zu bilden, oder eine Einkaufsliste, welche Namen von Artikeln enthält. Der gemeinsame Nenner dieser Listen ist jedoch, dass man einer Liste Elemente hinzufügen, ansehen oder wieder streichen bzw. entfernen kann. Damit man eben nicht für jeden Datentyp eine eigene Listenklasse programmieren muss, gibt es die generischen Collections. Hier legt man bei der Deklaration per Parameter den Datentyp fest. Für die Methoden Add, Remove usw. ist es damit unerheblich, um welchen Datentyp es sich handelt.

6.6.1 Collection deklarieren

Eine generische Collection wird wie folgt deklariert und instanziiert:

Syntax

```
List<Datentyp> Zahlenliste = new List<Datentyp>();
```

Im folgenden Code-Snippet wird eine Liste vom Typ double definiert:

```
List<double> Zahlenliste = new List<double>();
```

6.6.2 Objekte hinzufügen

Ein Objekt wird mit der Methode Add hinzugefügt:

Syntax

```
public void Add(T item);
```

Im folgenden Code-Snippet wird die Zahl 3 der Zahlenliste hinzugefügt:

```
ZahlenListe.Add(3.0);
```

6.6.3 Auf Objekte zugreifen

Über indizierte Eigenschaften kann man ähnlich wie bei einem Array auf die Elemente in einem Objekt zugreifen.

Syntax

```
public T this[int index] { get; set; }
```

Im folgenden Code-Snippet wird auf das erste Element zugegriffen:

```
double Wert = Zahlenliste[0];
```

6.6.4 Liste durchlaufen

Um alle Elemente einer Liste zu durchlaufen gibt es zwei Varianten:

- *FOR-Schleife*
- *FOREACH-Schleife*

Die FOR-Schleife sollte bereits bekannt sein. Über die Eigenschaft Count wird die Anzahl der Elemente bestimmt und über die Zählvariable (i) kann auf die Elemente in der Liste zugegriffen werden.

Syntax

```
for (int i = 0; i < ZahlenListe.Count; i++)
{
    gesamt = gesamt + Zahlenliste[i];
}
```

Die Syntax für die FOREACH-SchIeife lautet wie folgt:

Syntax

```
foreach (Datentyp item in collection)
{
Anweisung(en);
}
```

Dabei enthält die Variable item den Inhalt eines Listenelements der Collection. Beim ersten Schleifendurchgang enthält die Variable item den ersten Eintrag der Collection. Beim zweiten Durchgang das zweite Element, so lange bis die Liste kein Element mehr enthält. Innerhalb der geschwungenen Klammer kann auf die Variable item zugegriffen werden.

Im folgenden Code-Snippet werden die einzelnen Listenelemente einfach ausgegeben:

```
foreach (double item in Zahlenliste)
{
    Console.WriteLine("Wert = {0}", item);
}
```

6.6.5 Weitere Themenbereiche

Die Klasse List stellt noch weitere Methoden wie Remove oder Sort zur Verfügung.

Der Vollständigkeit halber sei erwähnt, dass noch weitere Collections im Namespace System.IO.Collections oder System.IO.Collections.Generic wie zum Beispiel das zweispaltige Dictionary existieren.

6.6.6 Anwendung

Lösung zum Arbeitsauftrag

Mit den genannten Methoden und Eigenschaften lässt sich der Arbeitsauftrag wie folgt umsetzen:

```
private static void Main(string[] args)
{
    List<double> Zahlenliste = new List<double>();
    string Eingabe = "";
    double zahl, gesamt = 0, durchschnitt;

    do
    {
        Console.Write("Eingabe Zahl (Ende mit e):");
        Eingabe = Console.ReadLine();
        if (Eingabe != "e")
        {
```

```
            zahl = Convert.ToDouble(Eingabe);
            ZahlenListe.Add(zahl);
        }
    } While (Eingabe != "e");

    foreach (double item in Zahlenliste)
    {
        gesamt = gesamt + item;
    }
    durchschnitt = gesamt / ZahlenListe.Count;
    Console.WriteLine("Der Durchschnitt beträgt: {0}, die Summe {1}", durchschnitt,
gesamt);
    Console.ReadKey();
}
```

Aufgaben

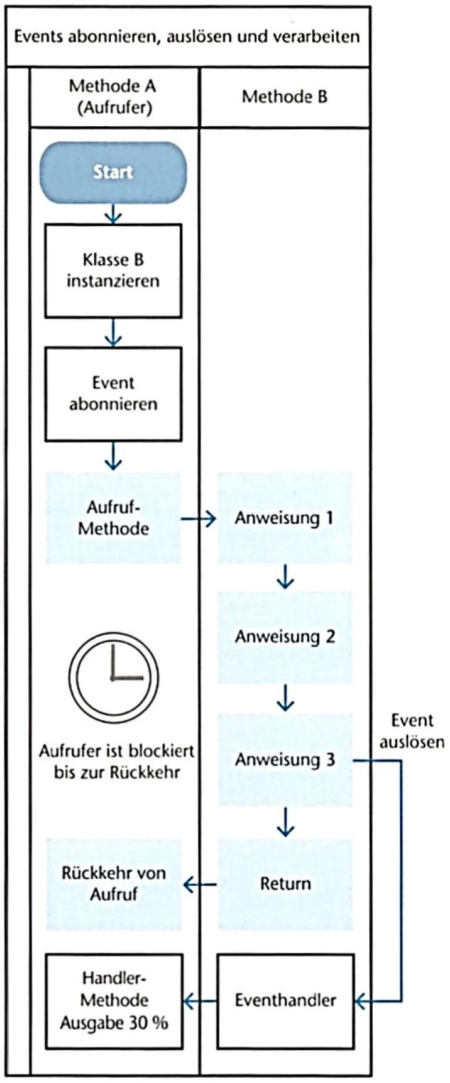
1. Ändern Sie das Programm „POIs verwalten" aus Kapitel 6.4 so, dass anstelle von Arrays eine Collection verwendet wird.
2. Ändern Sie den Arbeitsauftrag „Suche“ aus Kapitel 6.1.6 so, dass anstelle eines String-Arrays eine List verwendet wird.

6.7 Delegaten und Events

Arbeitsauftrag

Schreiben Sie ein Programm, das Events auslöst und verarbeitet.

Wird wie in der folgenden Abbildung im Programm eine Methode (Methode B) aufgerufen, wird der Code in dieser Methode abgearbeitet und kehrt erst nach der letzten Anweisung in der Methode zum Aufrufer zurück (blau eingefärbt). Solange diese Methode abgearbeitet wird, ist der Aufrufer blockiert und dieser kann daher nicht den Zustand bzw. Fortschritt der abzuarbeitenden Methode abfragen. Der Zustand einer laufenden Methode kann in vielen Fallen von Interesse sein. Beispielsweise kann ein Datenimport länger dauern (von mehreren Sekunden bis mehreren Stunden). Um von außen den Status bzw. den Fortschritt nachzuvollziehen, müssen Events verwendet werden.



Events

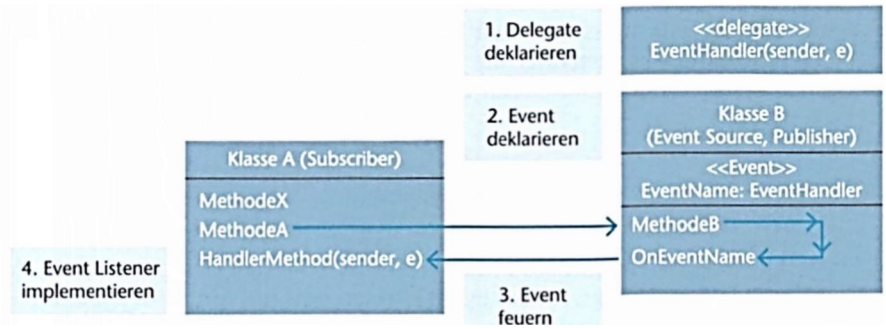
Events bestehen aus mehreren Teilen: Eventsender (auch Event Source oder Publisher genannt), Event Handler und Eventempfänger (auch Handlermethode oder Subscriber genannt). Auf der einen Seite löst die Event Source das Event aus. Auf der anderen Seite gibt es einen oder mehrere Empfänger (auch Subscriber genannt), die das Event empfangen und darauf entsprechend reagieren können.

Um Events empfangen zu können, muss jeder Empfänger eine Methode (Handler oder Listenermethoden genannt) implementieren. Dabei ruft die EventSource nicht direkt die Handlemethoden auf, sondern verwendet einen Eventhandler, da der Eventsender nicht wissen kann, welche und wie viele Eventempfänger es zur Laufzeit geben wird. Der Eventhandler basiert auf Delegaten. Mithilfe von Delegaten ist es möglich, dass erst zur Laufzeit bestimmt wird, welche Methode zugewiesen und aufgerufen wird. Anstelle des direkten Aufrufs wird ein Stellvertreter (Delegat) verwendet. Nur so ist es für die Event Source unerheblich, welche Handlermethoden zur Laufzeit zugewiesen werden.

Um ein Event zu implementieren, sind folgende Schritte notwendig, siehe folgende Abbildung:

- *• Auf Senderseite:*
  - *Delegaten deklarieren*
  - *Event deklarieren*
  - *Event auslösen (feuern)*
- *• Auf Empfängerseite:*
  - *Eventhandler implementieren*
  - *Event in Empfängerklasse(n) abonnieren*

Eventhandler implementieren



Die Syntax für die Deklaration lautet wie folgt: Der Delegat wird nicht als Klassenmember deklariert, sondern als eigener Typ außerhalb der Klassendeklaration. In .NET sind die Parameter sender (Klasse object) und e (Klasse EventArgs) üblich, es können aber generell auch andere Parameter verwendet werden. Die Klasse EventArgs kann als Basisklasse verwendet werden, um eigene Parameter zu definieren.

Syntax

```
Delegat DatentypRückgabewert DelegateName[(Parameter)];
```

Danach kann das Event deklariert werden:

Syntax

```
public event DelegateName EventName;
```

Das Event wird auf Senderseite ausgelöst, indem einfach das oben deklarierte Event wie eine Methode aufgerufen wird:

Syntax

```
EventName[(Parameter)];
```

Da sich an das Event 0 bis n Empfänger anhängen können, kann das Event auch den Wert null haben. Daher sollte auf Null abgefragt werden. Des Weiteren ist es üblich, den Aufruf der Methode in eine eigene Methode zu packen, die den Präfix On hat. Der Vorteil liegt darin, dass das Event innerhalb der Klasse von überall aufgerufen werden kann, ohne jedes Mal auf Null abfragen müssen.

```
public void OnInfoEvent(string Message)
{
    if (Infoevent != null)
        Infoevent(Message);
}
```

Auf der Empfängerseite muss der Eventhandler die gleiche Signatur wie der Delegat haben. Eine Signatur kennzeichnet die Eindeutigkeit einer Methode und besteht aus dem Namen der Methoden und dessen Parameterliste.

Syntax

```
DatentypRückgabewert EventHandIer[(Parameter)];
```

Das Event wird wie folgt abonniert: Mit += new wird über den Delegaten die Handlermethode an das Event angehängt. An das Event können mithilfe von += new weitere HandIermethoden angehängt werden.

Syntax

```
Objektvariable.EventName += new DelegateName(EventHandler);
```

In der Klasse simuliert die MethodeB einen langandauernden Prozess, in dem die Anweisung Sleep aufgerufen wird. Der statischen Methode Sleep wird die abzuwartende Zeit in Millisekunden übergeben. Jede Sekunde wird in der FOR-Schleife das Event über die OnInfoEvent Methode ausgelöst.

Klasse KlasseB in Datei KlasseB.cs

```
public delegate void InfoEventHandler(string Message);

public class KlasseB
{
    public event InfoEventHandler Infoevent;

    public void MethodeB()
    {
        for (int i = 0; i < 10; i++)
        {
            System.Threading.Thread.Sleep(1000);
            OnInfoEvent(i.ToString());
        }
    }

    public void OnInfoEvent(string Message)
    {
        if (Infoevent != null)
            Infoevent(Message);
    }
}
```

In der Main-Methode wird das Event abonniert, indem dem Konstruktor die Handle- Methode als Parameter übergeben wird.

Main-Methode und Handlermethode in Datei Program.cs

```
class Programm
{
    static void Main(string[] args)
    {
        Klasse B = new Klasse();
        B.InfoEvent += new InfoEventHandler(InfoEventHandlerMethod);
        B.MethodeB();
        Console.ReadKey();
    }

    static void InfoEventHandlerMethod(string Message)
    {
        Console.Write(Message);
    }
}
```

Aufgaben

1. Entwerfen und implementieren Sie eine Klasse, welche ein Metronom simuliert. Von außen ist einstellbar, in welchem Intervall (Millisekunden) ein Event abgefeuert wird. Verwenden Sie die Thread.Sleep-Funktion. Da die Thread.Sleep-Methode den Prozess blockiert, reagiert die Anwendung nicht mehr auf Tastendruck und das Metronom kann nicht gestoppt werden. Um das Metronom doch stoppen zu können, muss ein eigener Thread gestartet werden. Das geschieht ganz einfach mit der Thread-Klasse. Dem Konstruktor wird der Name der Methode übergeben, welche in einem eigenen Thread starten soll. Schließlich startet die Start-Methode den Thread.

   ```
   System.Threading.Thread Thread = new System.Threading.Thread(MetronomObjekt.Start);
   Thread.Start();
   ```

2. Die Feiertage sind in jedem Staat anders geregelt. In Deutschland sind sie sogar von einem zum anderen Bundesland unterschiedlich. Beispielsweise hat nur Sachsen den Buß- und Bettag.

   Entwerfen und implementieren Sie ein Programm zur Verwaltung von Feiertagen. Dieses Programm erzeugt für eine beliebige Region (Bundesland) und bei gegebenem Jahr eine Liste der Feiertage. Die Feiertagsverwaltung unterscheidet zwischen folgenden Arten von Feiertagen:

   - *Fixe Feiertage (wie Neujahr, Maria Himmelfahrt)*
   - *Bewegliche Feiertage, die von Ostersonntag abhängen*

- *Buß- und Bettag*

Fixe Feiertage: Entwerfen und implementieren Sie eine Klasse, welche einen fixen Feiertag repräsentiert. Ein fixer Feiertag ist ein Feiertag, der immer am selben Tag und Monat eines Jahres begangen wird, wie beispielsweise Neujahr (01.01.) oder Maria Himmelfahrt (15.08.).

Übergeben Sie dem Konstruktor Jahr, Monat, Tag und Name des Feiertags. Auf das absolute Datum und den Feiertag soll nur lesend zugegriffen werden können.

Buß- und Bettag: Entwerfen und implementieren Sie eine Klasse, welche den Buß- und Bettag berechnet. Übergeben Sie beim Instanziieren das Jahr. Die Eigenschaft Datum liefert den berechneten Wert. Der Buß- und Bettag ist 32 Tage vor dem vierten Advent. Der vierte Advent selbst ist der letzte Sonntag vor dem 25. Dezember, wobei dieser Tag auf einen beliebigen Wochentag fallen kann. Um den Wochentag zu ermitteln, bietet die DateTime- Klasse die Eigenschaft DayOfWeek an, welches ein Enum liefert. Dieses kann per Cast (int) in eine Zahl umgewandelt werden, um so den Wochentag als Zahl zu bekommen. Beachten Sie, dass Sonntag bei 0 zu zählen beginnt.

Osterfeiertage: Entwerfen und implementieren Sie eine Klasse, welche Feiertage repräsentiert, die vom Ostersonntag abhängig sind. Ausgangspunkt für einen Kirchenfeiertag ist der Ostersonntag, welcher nach der Gauß'schen Formel berechnet wird. Der Pseudocode für die Formel lautet wie folgt, wobei mod für Modul Operator steht (% in C#):

a = Jahr Mod 19

b = Jahr \ 100

c = (8 * b + 13) \ 25 - 2

d = b - (Jahr \ 400) - 2

e = (19 * (Jahr Mod 19) + ((15 - c + d) Mod 30)) Mod 30

Wenn e = 28 und a > 10, dann e = 27

Wenn e = 29, dann e = 28

f = (d + 6 * e + 2* (Jahr Mod 4) + 4 * (Jahr Mod 7) + 6) Mod 7

Tag =e + f + 22

Monat = 3

Wenn Tag > 31, dann; Tag = Tag -31, Monat = 4

Ostersonntag = { Tag, Monat, Jahr)

Die weiteren kirchlichen Feiertage sind relativ zum Ostersonntag zu sehen:

Kirchlicher Feiertag	Relativ zu Ostersonntag
Karfreitag	- 2 Tage
Ostersonntag	0
Ostermontag	+ 1 Tag
Christi Himmelfahrt	+ 39 Tage
Pfingstmontag	+ 50 Tage
Fronleichnam	+ 60 Tage

Programmfunktionen: Nach dem Start fordert das Programm den Benutzer auf, folgende Eingaben zu tätigen:

- *Dateiname inklusive Pfadangabe*
- *Jahr*

Danach erfolgt die Ausgabe der absoluten Feiertage im eingegebenen Jahr.

In der angegebenen Datei wird eine Feiertagsliste unabhängig vom Jahr im CSV-Format gespeichert. So kann man für jede Region (Bundesland) die Feiertage unabhängig verwalten.

In der Feiertagsdatei hat jede Zeile folgenden Aufbau:

XX;P1 ;P2;Name

Die Bedeutung ist wie folgt:

Art des Feiertags	Spalte 1 (XX)	Spalte 2 (Pl)	Spalte 3(P2)	Spalte 4 (Name)	Bedeutung PI, P2
fixer Feiertag	FF	1	S	01. Mai	Pl = Tag P2 = Monat
beweglicher Feiertag relativ zu Ostern	OF	0	-	Ostersonntag	Pl = relative Tage zu Ostersonntag
Buß- und Bettag	BB	-	-	Buß- und Bettag	nicht genutzt

Bevor die Datei ausgegeben wird, werden die einzelnen Feiertage in einer generischen Liste gespeichert.

7 Die Verwaltung von Daten mithilfe von Datenbanken planen und umsetzen



Durch Probleme wie z. B. Datenredundanz, welche bei der Speicherung von Daten in Dateien auftreten, wurden schon sehr früh neue Formen der Verwaltung von Daten entwickelt. Dazu entstanden in den 1960er-Jahren erste Programme, die zwischen den Daten und den Anwendungsprogrammen die Verwaltung der Daten übernahmen. Daraus entwickelten sich später die Datenbankmanagementsysteme, welche ein wesentlicher Teil von Datenbanksystemen sind.

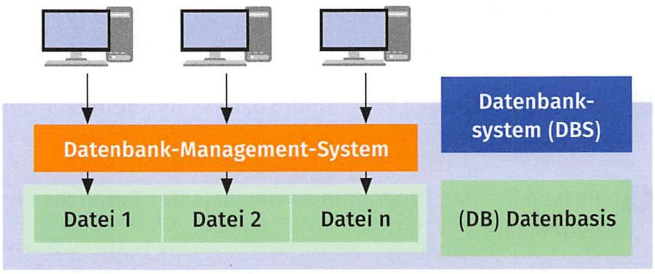
Datenbanksysteme, landläufig auch einfach als Datenbanken bezeichnet, sind heutzutage für viele Unternehmen und Behörden ein zentraler und unersetzlicher Bestandteil bei der Speicherung von Daten. Dabei müssen die Verfügbarkeit, die Vollständigkeit, die Richtigkeit sowie die Sicherheit und der Schutz der gespeicherten Informationen gewährleistet werden.

7.1 Datenbanksysteme beschreiben

Datenbanksysteme sind ein weitverbreitetes technisches Hilfsmittel zur effizienten, rechnergestützten Organisation, Speicherung, Manipulation, Integration und Verwaltung großer Datensammlungen.

7.1.1 Bestandteile

Ein Datenbanksystem besteht aus zwei Teilen: einer Datenbasis und einem Datenbank-managementsystem. Datenbanksysteme kapseln die Datenverwaltung, d. h. sie überlagern die Daten mit dem Datenbankmanagementsystem, welches den Zugriff auf die Daten organisiert. Über eine definierte Schnittstelle erhalten die Anwendungsprogramme so den Zugriff auf die Daten.



Als „Zugriffssprache" wird in vielen Fällen SQL verwendet. Die Datenverwaltung geschieht somit unabhängig von den Anwendungsprogrammen. Datenbanksysteme haben einen standardisierten Aufbau, der dem ANSI/SPARC-Architekturmodell folgt.

7.1.1.1 Datenbasis

Die Datenbasis ist der eigentliche Datenbestand, welcher eine strukturierte Sammlung von Daten darstellt. Der Datenbestand wird grafisch oft noch als Zylinder dargestellt, was an den rotierenden Stapel von magnetisierbaren Oberflächen aus einem Festplattenlaufwerk erinnern soll.

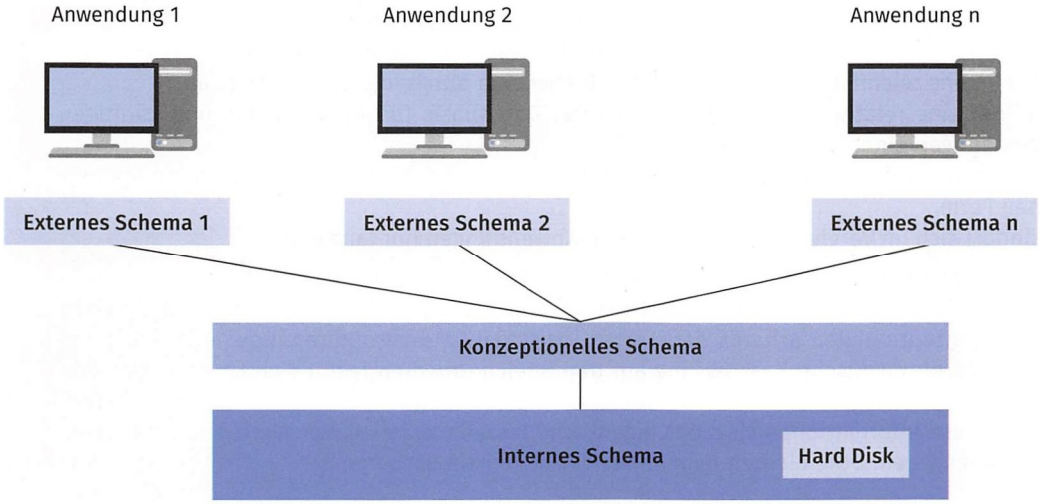
7.1.1.2 Datenbankmanagementsystem

Das Datenbankmanagementsystem dient der Verwaltung der Zugriffe auf den Datenbestand. Es umschließt (kapselt) den Datenbestand. Aufgaben des Datenbankmanagementsystems sind u.a. die Durchführung der Datenspeicherung, die Verwaltung der Daten auf Basis des zugrunde gelegten Datenmodells, die Gewährleistung des Mehrbenutzerzugriffs auf die Daten über eine entsprechende Kommunikationsschnittstelle, die Sicherstellung des „optimalen" Zugriffs auf die Daten und die Gewährleistung der Datensicherheit sowie des Datenschutzes. Beispiele für Datenmanagementsysteme sind: MS Access, MySQL, MS-SQLServer, SAP HANA, IBM DB2, PostgreSQL oder Oracle DB.

7.1.2 ANSI/SPARC-Architektur für Datenbanksysteme

Die ANSI/SPARC-Architektur wurde 1975 vom „Standards Planung and Requirements Committee“ (SPARC) des „American National Standards Institute" (ANSI) entwickelt. Das American National Standards Institute ist der nationale Normenausschuss der USA, der dem DIN in Deutschland entspricht. Mit dem Architekturmodell unterbreitete ANSI einen Vorschlag für die prinzipielle Architektur von Datenbanksystemen, an dem sich sowohl die Hersteller von Datenbanksoftware als auch die Betreiber von Datenbanken orientieren sollten. Ziel war die Entkopplung von Anwendung und Datenspeicherung, um einerseits den Benutzer einer Datenbank vor nachteiligen Auswirkungen von Änderungen in der Datenbankstruktur zu schützen und andererseits den Entwicklern von

Datenbankmanagementsystemen den Raum für kreative Lösungen zu geben.



Im Zentrum dieser Architektur liegt das konzeptionelle Schema (Ebene). In diesem Schema werden alle Daten und ihre Beziehungen zueinander modelliert. Hier liegen nicht die Daten, hier liegen die Informationen über die Struktur der Daten. Der Inhalt des konzeptionellen Schemas ist unabhängig von der eingesetzten Hardware zur Datenspeicherung und unabhängig von den Anforderungen einzelner Benutzer. Dadurch erreicht man die Entkopplung der Anwendungsprogramme von der physischen Datenspeicherung.

Über bzw. unter dem konzeptionellen Schema liegen das externe und das interne Schema. Im internen Schema wird die Speicherung der Daten mit ihren Zugriffspfaden auf den physischen Speicher festgelegt. Das externe Schema regelt, welche Daten bestimmte Benutzer bzw. Programme sehen und bearbeiten können. Außerdem definiert das ANSI-Architekturmodell noch die Zuordnung von personellen Instanzen, die als Anwendungs-, Unternehmens- und Datenbankadministrator bezeichnet werden und die für das externe, konzeptionelle oder interne Schema zuständig sind.

7.1.2.1 Externes Schema (Ebene)

Das externe Schema beschreibt die Benutzersichten auf die Daten. Das auf die Bedürfnisse einer Benutzergruppe abgestimmte externe Schema soll die Dateneinheiten und Beziehungen nicht enthalten, die diese Benutzer nicht sehen wollen oder nicht sehen sollen. In einem Unternehmen existieren in der Regel mehrere Benutzergruppen. Es sind daher mehrere unterschiedliche Schemata zu entwickeln - je ein Schema pro Benutzergruppe. Mehrere unterschiedliche externe Subschemata ermöglichen den Multi-User-Betrieb, mehrere Anwender können aus mehreren unterschiedlichen Anwendungen gleichzeitig auf die Daten in dem Datenbanksystem zugreifen. Das stellt einen gewaltigen Fortschritt gegenüber der Arbeit mit Dateien dar, verlangt aber zusätzlich die Einrichtung eines Berechtigungskonzepts, welches sicherstellt, dass nur Anwender auf die Daten zugreifen, die hierzu aus ihrer Tätigkeit heraus auch berechtigt sind.

7.1.2.2 Konzeptionelles Schema (Ebene)

Das konzeptionelle Schema benennt und beschreibt alle logischen Dateneinheiten (Entity) sowie die Beziehungen (Relationship) zwischen den Dateneinheiten. Das konzeptionelle Schema enthält keine Datenwerte (Values). Es beschreibt lediglich deren Struktur in anwendungsübergreifender Form.

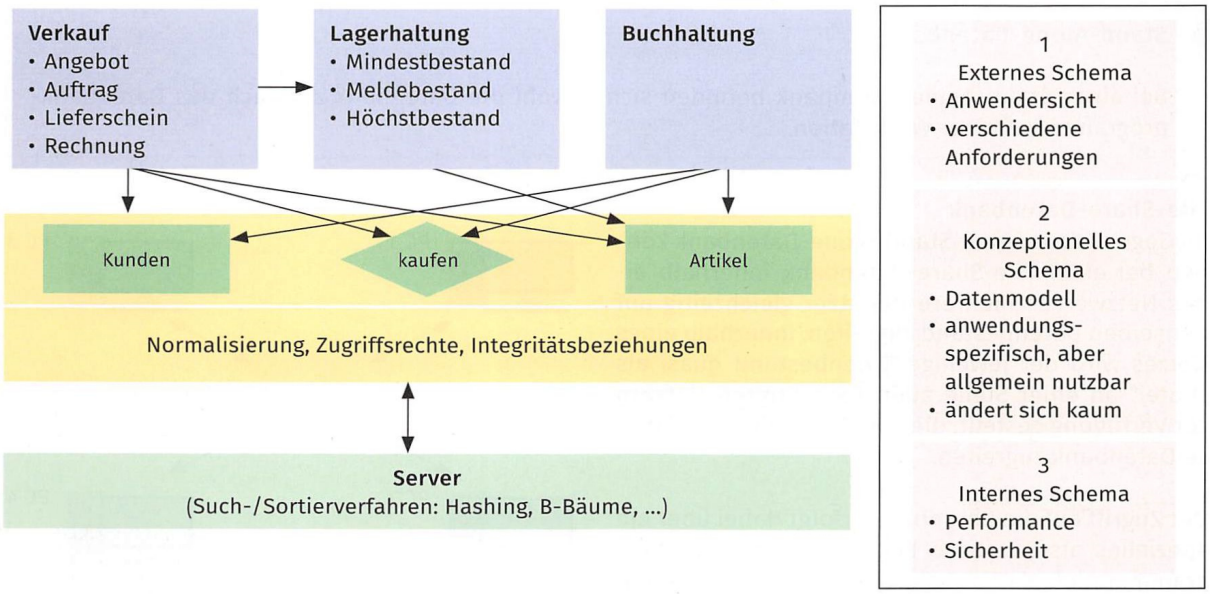
Die Form der Beschreibung und die Beschreibungsmöglichkeiten werden durch das Datenmodell festgelegt, das zur Erstellung des konzeptionellen Schemas herangezogen wird. So kann ein hierarchisches, ein netzwerkartiges oder ein relationales Datenmodell infrage kommen. Keinesfalls enthält ein konzeptionelles Schema Angaben zur physischen Organisation der Speicherung. Angaben dieser Art bleiben dem internen Schema vorbehalten.

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

- *Es bildet eine relativ stabile Beschreibung der Datenbasis für alle aktuellen und künftigen Anwendungen eines Unternehmens.*
- *Es dokumentiert die Informationszusammenhänge eines Unternehmens in einer einheitlichen Form.*
- *Es ändert sich im Vergleich zu den einzelnen Anwendungen nur langsam.*

7.1.2.3 Internes Schema (Ebene)

Als internes Schema bezeichnet man die Organisation der physischen Datenspeicherung zu den in einem konzeptionellen Schema definierten logischen Datenstrukturen. Das interne Schema legt also die physikalische Realisierung auf den Speichermedien fest. Es enthält die Algorithmen zur Speicherung, zur Indizierung, Sortierung und Selektion der Daten aus dem Datenbestand. Diese Informationen sind das eigentliche Know-how des Entwicklers des Datenbankmanagementsystems. Sie bleiben dem Anwender allgemein verborgen.



Beispiel für das externe, konzeptionelle und interne Schema

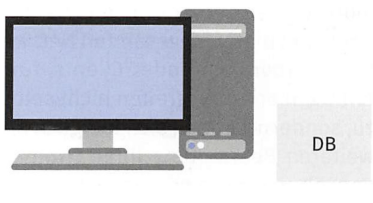
Trotz der einheitlichen ANSI/SPARC-Architektur wurden in der Vergangenheit von den Softwarefirmen und wissenschaftlichen Instituten eine Vielzahl von Datenbankmanagementsystemen entwickelt, denen unterschiedliche Anforderungen und Konzepte zugrunde liegen. Diese Grundzüge sollte man kennen und die Datenbankmanagementsysteme dahingehend unterscheiden können, um den Anforderungen entsprechend das richtige Datenbankmanagementsystem auszuwählen.

7.1.3 Unterscheidung nach dem Betriebskonzept

Datenbankmanagementsysteme lassen sich in verschiedene Kategorien einteilen. Eine mögliche Kategorisierung ist die nach dem Betriebskonzept, d. h. auf welche Art auf die Datenbank zugegriffen wird und wie das Backend und das Frontend Zusammenarbeiten. Eine Einteilung von Datenbanken nach ihrem Betriebskonzept ist in der folgenden Übersicht dargestellt.

Einteilung nach Betriebsart		
Stand-Alone-Datenbank	**File-Share-Datenbank**	**Client-Server-Datenbank**

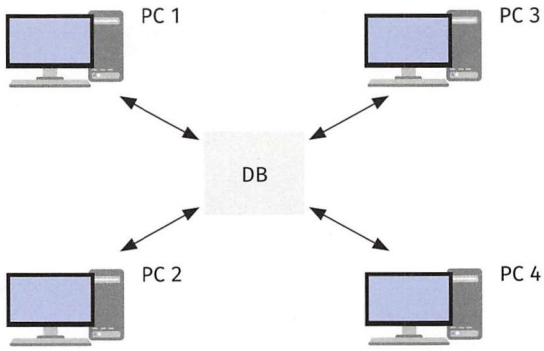
7.1.3.1 Stand-Alone-Datenbank



Sie stellt die einfachste Form einer Datenbank dar. In der Regel handelt es sich um eine lokale Datenbank, die weder für Mehrfachzugriffe noch ausgefeilte Zugangsberechtigungen für unterschiedliche Benutzer ausgelegt ist. Sie ist lediglich dazu da, von einem Benutzer immer über das gleiche Programm angesteuert zu werden. Typische Beispiele dafür sind Adresskarteien, elektronische Telefonbücher oder Buchhaltungsprogramme, die mit dBase, Access, Filemaker, FoxPro, Paradox oder ähnlichen Programmen erstellt wurden und ihrem Benutzer auf einfache Weise einen mehr oder weniger guten elektronischen Karteikasten zur Verfügung stellen. Sie können teilweise sogar als einfaches Dateisystem realisiert werden, besitzen somit ein sehr einfaches internes Schema und haben mit komplexen Datenbanken wenig gemein.

Bei einer Stand-Alone-Datenbank befinden sich sowohl die Datenbank als auch das Datenbankprogramm auf einer Workstation.

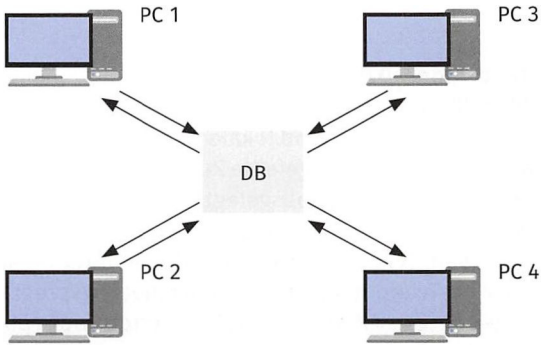
7.1.3.2 File-Share-Datenbank



Im Gegensatz zu einer Stand-Alone-Datenbank können bei einer File-Share-Datenbank innerhalb eines Netzwerkes mehrere Benutzer gleichzeitig auf denselben Datenbestand zugreifen. Innerhalb eines Netzes wird der jeweilige Datenbestand quasi als „Datei" an einer Stelle allen berechtigten Nutzern Zur Verfügung gestellt, die von ihren Clients auf diese Datenbank zugreifen. Der Zugriff auf die Datenbank erfolgt dabei über ein spezielles als Database-Engine bezeichnetes Programmmodul, das auf der jeweiligen Workstation ausgeführt werden muss. Die Datenbank liegt zwar auf dem Server, der jedoch bezüglich der Verwaltung dieser Datenbank keine eigene „Intelligenz" aufbringt. Die gesamte dafür erforderliche Kompetenz befindet sich separiert auf jedem einzelnen Client. Der Hauptvorteil solcher Datenbanken besteht darin, dass damit die redundante (mehrfach vorhandene) Datenhaltung überwunden wird und Datenänderungen sofort allen Benutzern redundanzfrei zur Verfügung stehen. Probleme gibt es allerdings dann, wenn viele simultane und vor allem ändernde Zugriffe auf den Datenbestand erfolgen (z. B. bei der Auftragserfassung) oder häufig umfangreiche Auswertungen erstellt werden müssen. Da bei diesem Konzept die Datenbank letztlich von jeder Workstation im lokalen Arbeitsspeicher Satz für Satz betrachtet und ausgewertet werden muss, führt dies zu einem umfangreichen Netzwerkverkehr mit all seinen negativen Folgen. Bei konkurrierenden Zugriffen lassen sich dabei nur sehr ineffiziente Mechanismen zur Wahrung der operationalen Integrität realisieren, sodass letztlich immer recht große Datenbereiche so lange gesperrt werden müssen, bis der nächste lokale Client an der Reihe ist.

Bei der File-Share-Datenbank befindet sich die Datenbank auf dem Server, das Datenbankprogramm läuft auf jedem einzelnen Client.

7.1.3.3 Client-Server-Datenbank

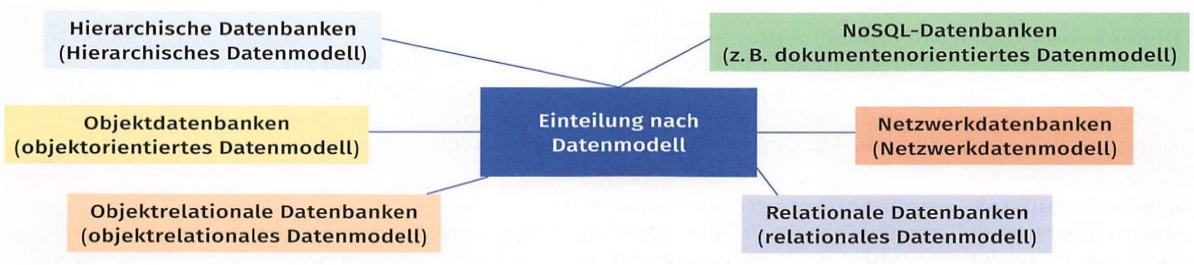


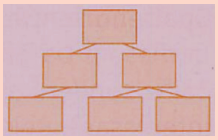
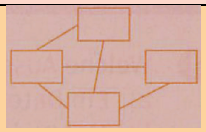
Der Kern einer solchen Datenbank ist ein sog. Datenbankserver (Softwaresystem), der innerhalb eines Netzwerkes auf einer dedizierten Maschine läuft. Dieser Server kapselt die komplette Datenbank und bietet im gesamten Netzwerk seine Dienste an. Programme oder Clients, für die der Server Anfragen erledigt, greifen nicht selbst auf die Daten zu, sondern wenden sich lediglich an den Server zur weiteren Bearbeitung ihrer „Wünsche". Die Datenverwaltung ist hier konsequent vom Datenbank zugriffgetrennt. Die Clients wenden sich lediglich an die standardisierte Schnittstelle des Servers. Wo und wie der Server seine Daten verwaltet, ist für die Clients nicht transparent. Client-Server-Datenbanken bieten viele Sicherheits-, Leistungs- und Flexibilitätsvorteile, erfordern allerdings auch die Betreuung durch einen Datenbankadministrator. Zu den Produkten aus diesem Bereich gehören u.a. Oracle, DB2, Microsoft SQL-Server, MySQL oder MAXDB.

Bei einer Client-Server-Datenbank befinden sich Datenbank und Datenbankmanagementsystem auf dem Server, während auf den Clients nur ein Programm als Zugriffsoberfläche ausgeführt wird.

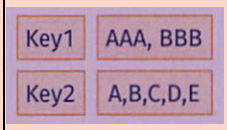
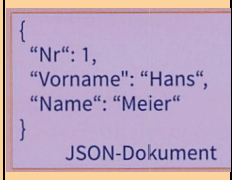
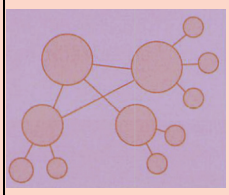
7.1.4 Unterscheidung nach dem Datenmodell

Eine weitere mögliche Kategorisierung ist die nach dem zugrundeliegenden Datenmodell Hier wird das Konzept betrachtet, nach dem die Daten in der Datenbank organisiert sind. Eine Einteilung von Datenbanken nach dem zugrundeliegenden Datenmodell ist in der folgenden Übersicht dargestellt.



Arten von Datenbanken und deren zugrundeliegenden Datenmodelle	
Hierarchische Datenbank (hierarchisches Datenmodell) Das hierarchische Datenmodell ist das älteste Datenmodell. Dabei werden die Daten in einer Baumstruktur verwaltet. Bis auf die Wurzel hat jeder Datensatz genau einen Vorgänger. In diesem Datenmodell lässt sich Datenredundanz nicht vermeiden. Die Dateisysteme von Betriebssystemen nutzen noch teilweise dieses Modell um ihre Daten abzubilden. Im Bereich der XML-Entwicklung hat dieses Modell heutzutage aber wieder etwas an Bedeutung gewonnen. Beispiel: IMS DB von IBM	
Netzwerkdatenbank (Netzwerkdatenmodell) Das Netzwerkdatenmodell ist eine Erweiterung des hierarchischen Datenmodells. Dabei ist es möglich, dass ein Datensatz auch mehrere Vorgänger hat. Dadurch können m:n Beziehungen abgebildet werden und Datenredundanz wird i.d. R. vermieden.	
Relationale Datenbank (relationales Datenmodell) Im relationalen Datenmodell werden die Daten in verschiedenen Tabellen gespeichert, welche untereinander in Beziehung stehen. Dieses Modell wurde im Jahre 1970 von E.F. Codd vorgestellt und hat sich von da an als Standard in der Datenbankwelt etabliert. Allerdings unterliegt auch das relationale Datenmodell einigen Einschränkungen in Bezug auf der Speicherung und schnellen Verarbeitung von sehr großen Datenmengen wie sie z. B. im Big-Data-Bereich auftreten. Beispiele. MySQL, MS Access, MS SQL Server, SAP MaxDB, SQLite	
Objektdatenbank (objektorientiertes Datenmodell) Beim Objektorientieren Datenmodell werden die Daten mit den dazugehörigen Funktionen abgelegt. Das Datenmodell basiert auf den objektorientierten Programmierparadigma. Dieses Datenmodell ist recht mächtig, aber auch sehr komplex. Durch die Entwicklung der NoSQL-Datenmodelle hat dieses Datenmodell aber an Bedeutung verloren. Beispiele: db4o, Gemstone	
Objektrelationale Datenbank (objektrelationales Datenmodell) Bei dem objektrelationalen Datenmodell wird versucht die Vorteile des relationalen Datenmodells mit denen des objektorientierten Datenmodells zu verbinden. Hier werden z. B. die Daten als Relation von komplexen Objekten gespeichert. Beispiel: PostgreSQL	

NoSQL-Datenbanken (Not only SQL) wurden entwickelt, um sehr große Datenmengen wie sie z.B. im Big- Data Bereich anfallen effizient zu speichern und zu verarbeiten. Sie lassen sich für Anwendungen nutzen, bei denen relationale Datenbanken an ihre Grenzen stoßen und sind sehr gut skalierbar. So sind sie im Laufe der Zeit zu einer etablierten Alternative für relationale Datenbanken geworden. Die meisten von diesen Datenbanken sind aber auf eine bestimmte Aufgabe spezialisiert, für die sich hervorragend eigenen. Bei anderen Aufgaben bieten sie in der Regel eine schlechtere Performance oder eigenen sich gar nicht.

NoSQL-Datenbanken und deren zugrundeliegenden Datenmodelle	
Key-Value Datenbank (Key-Value Datenmodell) In diesem Datenmodell werden die Daten in Form von Schlüssel-Wertepaaren abgespeichert. Der Schlüssel dient dabei als Index über den auf die Daten zugegriffen werden kann. Die Werte selber bestehen aus strukturierten oder unstrukturierten Daten (siehe auch S. 482) Beispiele: Amazon Dynamo, Google BigTable	
Dokumenten-Datenbank (dokumentenorientiertes Datenmodell) Das dokumentenorientierte Datenmodell nutzt Dokumente als Grundeinheit zur Speicherung der Daten. Ein Dokument können strukturierte als auch unstrukturierte Daten enthalten, (siehe auch S. 482) Eingesetzt werden solchen Datenbaken z.B. bei Blogs, Wikis oder Content-Management-Systemen (CMS). Beispiele: BaseX, MongoDB	
Graph-Datenbank (graphenorientiertes Datenmodell) Dieses Datenmodell stellt eine Art Weiterentwicklung des Netzwerkdatenmodells dar. Es benutzt Graphen zur Darstellung und Speicherung stark vernetzter Informationen. Hier spielen neben den eigentlichen Daten auch deren Beziehungen untereinander eine große Rolle, wie z.B. bei Twitter die Beziehungen der Follower untereinander. Beispiele: Neo4j, HyperGraphDB	
Spaltenorientierte Datenbank (spaltenorientiertes Datenmodell) In diesem Datenmodell werden die Daten, anders als beim relationalen Datenmodell, nicht Zeile für Zeile, sondern spaltenweise im Speicher abgelegt. Das macht sie besonders für analytische Aufgaben interessant. Beispiele: Apache Cassandra, Sybase IQ	

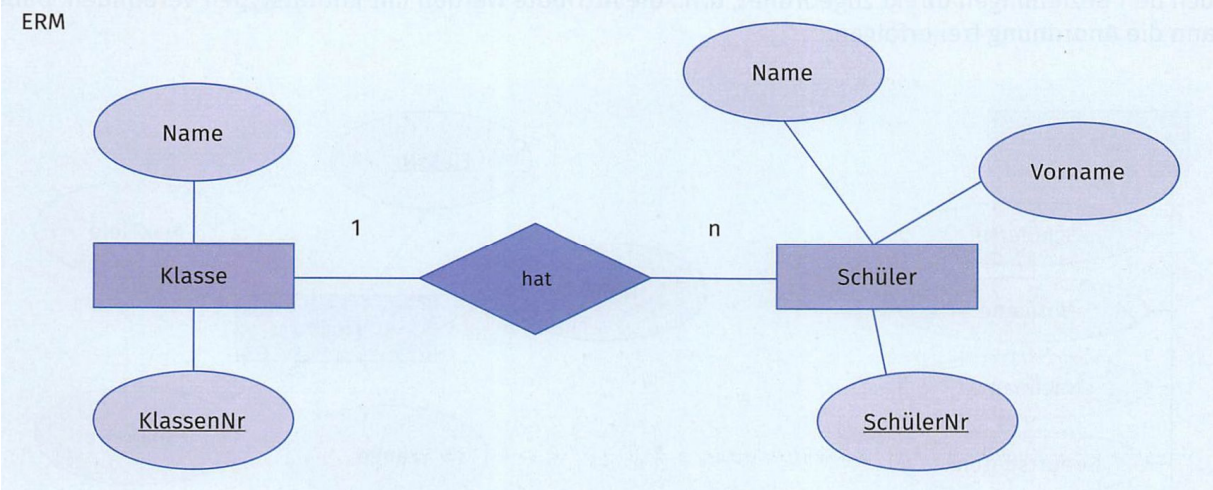
Kompetenzcheck

1. Welche Aussagen sind richtig?

   a. Ein Datenbanksystem besteht aus einer Datenbasis und einem Datenbankmanagementsystem.
   b. MySQL ist eine Datenbasis.
   c. PostgreSQL ist ein DMBS.
   d. Ziel der ANSI/SPARC-Architektur ist die Entkopplung der Anwendung von der Datenspeicherung.
   e. Das interne Schema ist Bestandteil der ANSI/SPARC-Architektur.
   f. Das externe Schema ermöglicht die Bildung unterschiedlicher Benutzergruppen.
   g. Das externe und das interne Schema hängen stark voneinander ab.
   h. Das interne Schema beschreibt, auf welchen Speichermedien die Daten gespeichert werden.
   i. Das konzeptionelle Schema beschreibt die Beziehungen zwischen den Daten.
   j. Bei Stand-Alone-Datenbanken befinden sich Datenbank und -programm auf einem System.
   k. File-Share-Datenbanken erzeugen redundante Daten.
   l. Bei Client-Server-Datenbanken befinden sich die Datenbasis und das Datenbankmanagementsystem auf dem Client.
   m. MySQL ist ein Client-Server-Datenbanksystem.
   n. Relationale Datenbanken sind sehr weitverbreitet.
   o. Spaltenorientierte Datenbanken gehören zu den NoSQL-Datenbanken.
   p. Dokumentenorientierte Datenbanken sind eine Unterart von relationalen Datenbanken.
   q. MongoDB ist eine Graphdatenbank.
   r. Bei hierarchischen Datenbanken werden die Daten in einer Baumstruktur verwaltet.

2. Bearbeiten Sie im Handout die Aufgaben 1, 2 und 3 und überprüfen Sie u.a. Ihr Wissen über Datenbankmanagementsysteme.

7.2 Daten und deren Beziehungen mithilfe eines ER-Modells beschreiben

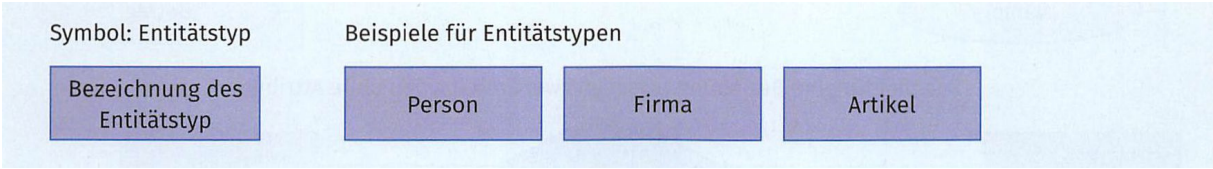


Bei der Datenanalyse für eine neue Datenbank oder Datenbankerweiterung werden zuerst alle projektrelevanten Informationen ermittelt. Das können beispielsweise Informationen über Arbeitsprozesse, Personen oder verschiedene Objekte sein, die später in der Datenbank erfasst werden sollen. Die Erhebung der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

Das Entity-Relationship-Modell (ER-Modell) wird benutzt, um Daten darzustellen, die bei der Datenanalyse erhoben wurden. Dieses Modell kommt u.a. in der Planungsphase zum Einsatz und dient zur Verständigung zwischen dem Anwender und dem Entwickler der Datenbank. Das ER-Modell wurde im Jahre 1976 das erste Mal von Peter Chen vorgestellt und ist inzwischen zum De-facto-Standard für die Datenmodellierung geworden. Nachfolgend werden der Aufbau und die Bestandteile eines ER-Modells näher betrachtet.

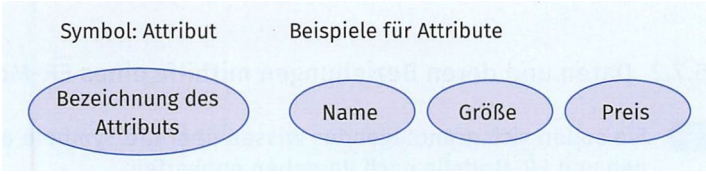
7.2.1 Entität und Entitätstypen

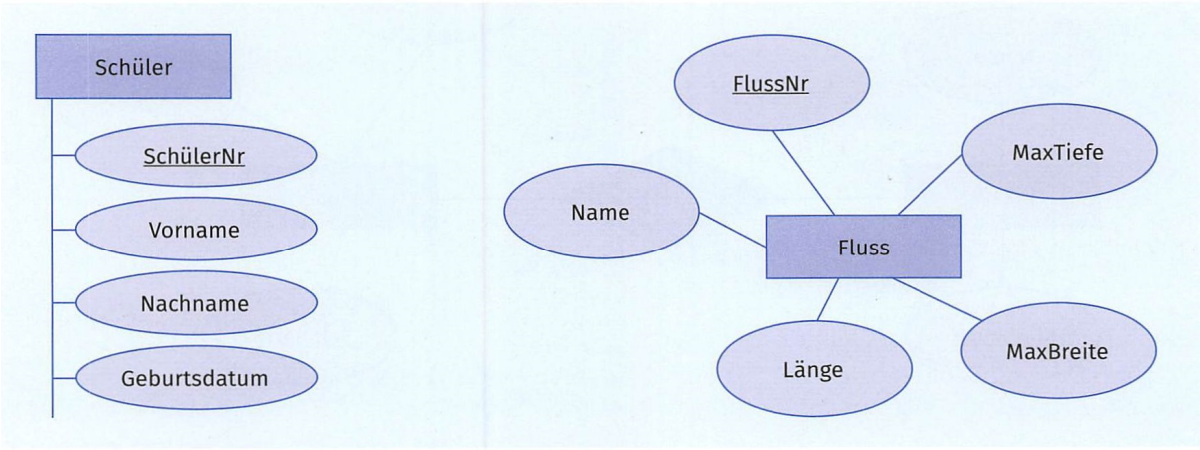
Eine Entität (Entity) ist vergleichbar mit einem Objekt. Die konkreten Daten eines Kunden beispielsweise sind eine solche Entität. Eine Datenbank besteht demnach aus vielen Entitäten, wobei die Entitäten gleichen Typs zu einem Entitätstyp zusammengefasst werden. Entitäten sind z. B. Herr Maier, Frau Schulz oder Herr Müller. Der dazugehörige Entitätstyp wäre z. B. Person.



7.2.2 Attribute

Attribute werden als „Daten" der Entität bezeichnet. Wenn die Entität ein eindeutig identifizierbares Attribut besitzt (z.B. Kundennummer), dann wird dieses Attribut unterstrichen. Die Attribute werden den entsprechenden Entitätstypen oder auch den Beziehungen direkt zugeordnet, d.h., die Attribute werden mit Entitätstypen verbunden. Dabei kann die Anordnung frei erfolgen.

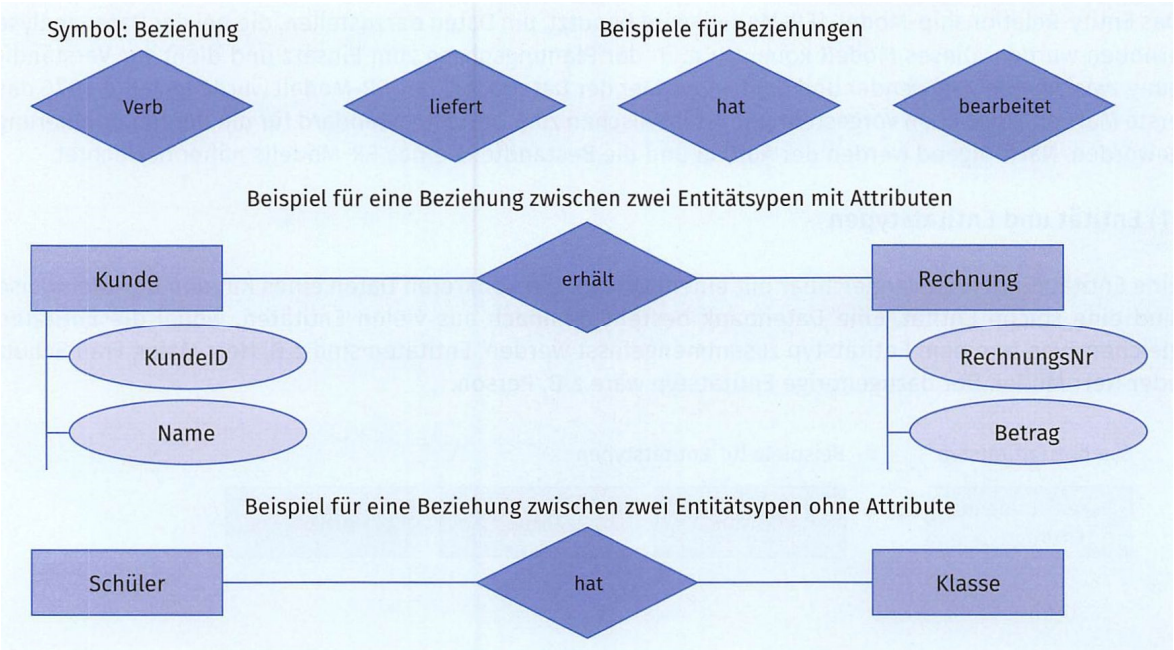




Beispiele für Entitätstypen mit Attributen

7.2.3 Beziehungstyp

Durch eine Beziehung, auch Relation genannt, wird eine semantische Verbindung zwischen zwei Entitätstypen dargestellt. Das ER-Modell kennzeichnet die Beziehungen der Entitätstypen mithilfe einer Raute. Im Falle einer m:n-Beziehung können der Beziehung auch Attribute direkt zugeordnet werden.



7.2.4 Kardinalitäten

Die Kardinalität dient der genauen Charakterisierung von Beziehungen und gibt an, in welchem Verhältnis die beteiligten Entitätstypen zueinander in Beziehung stehen.

Kardinalität	Beschreibung	Beispiel
1:1	Ein Mitarbeiter hat genau eine Personalakte und jede Personalakte ist genau einem Mitarbeiter zugeordnet.	
1:n	Ein Gebäude hat einen oder mehrere Räume. Aber ein Raum kann nur in einem Gebäude sein.	
m:n	Ein Kunde kauft mehrere Artikel. Aber ein Artikel kann auch von mehreren Kunden gekauft werden.	

7.2.5 Beispiele für die Erstellung eines ER-Modells

Die Erstellung eines ER-Modells soll nun anhand von zwei Beispielen erläutert werden.

Beispielauftrag 1

Für einen Zoo soll eine kleine Datenbank entwickelt werden. Dazu haben im Vorfeld Gespräche stattgefunden, in denen folgenden Informationen gesammelt wurden:

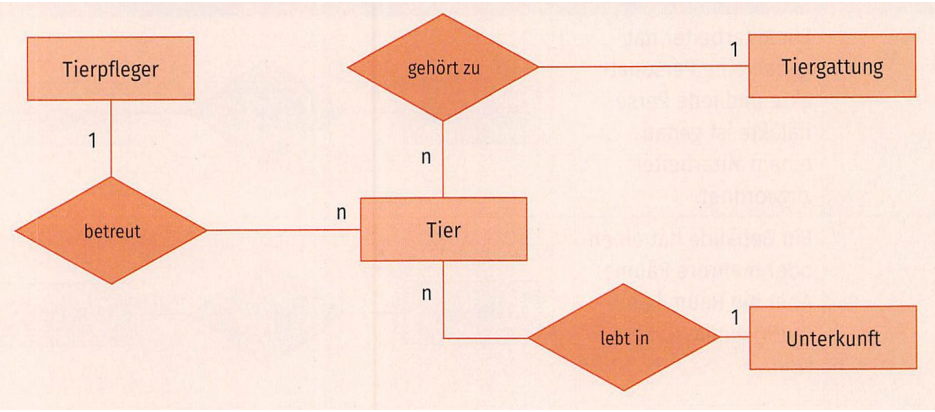
- *Der Zoo hat verschiedene Tiere, die jeweils einer bestimmten Tiergattung (z. B. Säugetier, Reptilien) zugeordnet werden können.*
- *Die Tiere im Zoo werden durch Tierpfleger versorgt. Dabei ist für ein Tier immer genau ein Pfleger verantwortlich. Ein Pfleger versorgt immer mehrere Tiere.*
- *Jedes Tier ist in einer Unterkunft untergebracht, in der es in der Regel mit mehreren Tieren zusammenlebt.*

Diese Informationen sollen nun in einem ersten Schritt in einem ER-Modell dargestellt werden. Dabei werden zunächst nur die Entitätstypen und deren Beziehungen berücksichtigt.

Lösung

Zunächst werden alle im Text vorhandenen Entitätstypen herausgesucht. In diesem Fall sind es: Tier, Tiergattung, Tierpfleger und Unterkunft. Der Zoo als Entitätstyp spielt in dieser Aufgabe keine Rolle. Anders würde es sich verhalten, wenn z. B. alle Zoos aus Deutschland verwaltet werden sollten.

Die Entitätstypen werden nun aufgezeichnet und es wird gleichzeitig geklärt, welche Entitätstypen miteinander in Beziehung stehen. Diese werden dann auch in das Modell mit übernommen. Laut Aufgabenstellung steht der Entitätstyp „Tier“ mit Tiergattung, Tierpfleger und Unterkunft in Beziehung. Zwischen den anderen Entitätstypen besteht keine weitere Beziehung. Das ER-Modell für diese Aufgabe sieht demnach wie folgt aus:



Zum Schluss werden noch die Kardinalitäten an die Beziehungen geschrieben. In diesem Fall sind es alles 1:n-Beziehungen.

Beispielauftrag 2

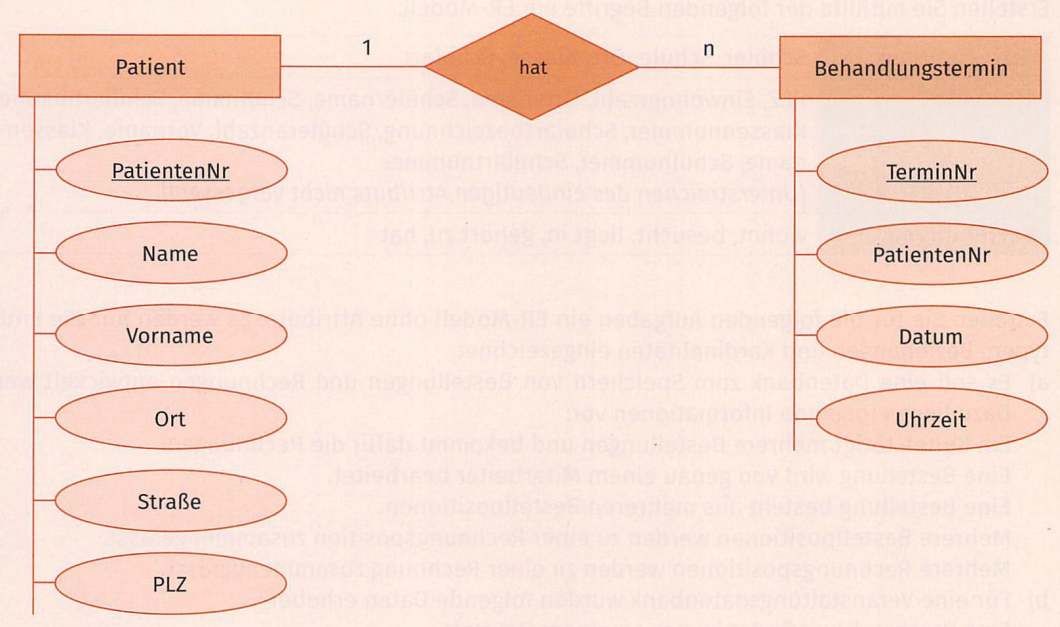
Eine Arztpraxis hat bisher ihre Patientendaten und die Behandlungstermine in Excel Tabellen verwaltet. Zukünftig sollen diese Daten nun in einer kleinen Datenbank gespeichert werden. Als Erstes soll nun aus den vorhandenen Excel-Tabellen ein ER-Modell erstellt werden.

Excel-Tabellen mit fiktiven Beispieldaten:

Patienten						
PatientenNr	**Name**	**Vorname**	**Geburtsdatum**	**Ort**	**Straße**	**PLZ**
1	Maier	Hans	12.12.1999	Berlin	Brunnenstraße	10115
2	Müller	Klara	01.07.1981	Berlin	Gartenstraße	10115
3	Schulz	Simone	29.03.2004	Berlin	Bergstraße	10115

Behandlungstermine				
TerminNr	**Datum**	**Uhrzeit**	**Name**	**Vorname**
1	20.02.2019	15:00	Maier	Hans
2	20.02.2019	15:30	Müller	Klara
3	21.02.2019	15:00	Schulz	Simone

Zunächst müssen die Entitätstypen ermittelt werden. Das ist in diesem Fall relativ einfach, weil diese den Exceltabellen entsprechen: Patient und Behandlungstermin. Diesen werden nun die entsprechenden Attribute zugeordnet. Da in der Exceltabelle „Behandlungstermine" die Patientennamen redundant vorkommen, werden diese durch einen Verweis auf die Patiententabelle ersetzt. Da beide Entitätstypen in Beziehung miteinander stehen, wird diese eingezeichnet und die Kardinalität bestimmt. Es handelt sich dabei um eine Im-Beziehung, da ein Patient mehrere Behandlungstermine haben kann, aber ein Termin nur für einen Patienten vergeben wird. Die Lösung der Aufgabe sieht dann wie folgt aus:



Kompetenzcheck

1. Welche Aussagen sind richtig?

   a. Das ER-Modell dient der Darstellung von Algorithmen.
   b. Entitätstypen werden im ER-Modell als Ellipse dargestellt.
   c. mm ist eine Kardinalität.
   d. Durch eine Relation wird die semantische Verbindung zwischen zwei Entitätstypen dargestellt.
   e. Die Anordnung der Attribute zum Entitätstyp kann frei erfolgen.

2. Untersuchen und diskutieren Sie, welche Kardinalitäten die folgenden Beziehungen haben und zeichnen Sie jeweils ein ER-Modell

   f. Postleitzahl-Ort
   g. Planet-Mond
   h. Kunde-Versicherung
   i. Volleyballspiel-Mannschaft
   j. Buch-Verlag
   k. Kino-Film
   l. Land - Fluss
   m. Klasse - Schüler
   n. Mitarbeiter-Gehalt
   o. Auto - Automarke

3. Zeichnen Sie ein ER-Modell für folgende Entitätstypen und deren Beziehungen.

Entitätstypen:

- *Artikel; Attribute: ArtikelNr, Bezeichnung, Listenpreis, Bestand, Mindestbestand*
- *MwStSatz; Attribute: MwStNr, Prozent, Beschreibung*
- *Kunde; Attribute: KundenNr, Name, Straße, PLZ, Ort*

Beschreibung der Beziehungen:

- *Ein Artikel hat genau einen MwSt-Satz.*

- *Ein Kunde kauft eine bestimmte Menge von verschiedenen Artikeln an unterschiedlichen Tagen.*

4. Erstellen Sie mithilfe der folgenden Begriffe ein ER-Modell

Entitätstypen	Schüler, Schule, Ort, Klasse, Schulart
Attribute	PLZ, Einwohnerzahl, Ortsname, Schülername, Schulname, Schülernummer, Klassennummer, Schulartbezeichnung, Schüleranzahl, Vorname, Klassenname, Schulnummer, Schulartnummer *(Unterstreichen des eindeutigen Attributs nicht vergessen!)*
Beziehungen	wohnt, besucht, liegt in, gehört zu, hat

5. Erstellen Sie für die folgenden Aufgaben ein ER-Modell ohne Attribute. Es werden nur die Entitätstypen, Beziehungen und Kardinalitäten eingezeichnet.

   a. Es soll eine Datenbank zum Speichern von Bestellungen und Rechnungen entwickelt werden.
   Dazu liegen folgende Informationen vor:
   Ein Kunde tätigt mehrere Bestellungen und bekommt dafür die Rechnungen.
   Eine Bestellung wird von genau einem Mitarbeiter bearbeitet.
   Eine Bestellung besteht aus mehreren Bestellpositionen.
   Mehrere Bestellpositionen werden zu einer Rechnungsposition zusammengefasst.
   Mehrere Rechnungspositionen werden zu einer Rechnung zusammengefasst.

   b. Für eine Veranstaltungsdatenbank wurden folgende Daten erhoben:
   Eine Veranstaltung findet immer an einem Ort statt.
   Es gibt verschiedene Veranstaltungsarten und jede Veranstaltung kann einer Veranstaltungsart immer eindeutig zugeordnet werden.
   Es können mehrere Künstler gleichzeitig bei einer Veranstaltung auftreten. Künstler werden auch für mehrere Veranstaltungen gebucht.

   c. Es soll eine Datenbank für Stundenpläne erstellt werden, die folgende Informationen enthält: Welche Klasse hat in welchem Raum mit welchem Lehrer in welchem Fach Unterricht?
   Es wird erst einmal davon ausgegangen, dass eine Klasse pro Unterrichtsstunde nur von einem Lehrer in einem Fach unterrichtet wird. Dafür steht immer genau ein Raum zur Verfügung. Erweitern Sie das Modell um sinnvolle Zeitangaben.
   Wie sieht das Modell aus, wenn davon ausgegangen wird, dass pro Unterrichtsstunde auch zwei Lehrer in einer Klasse unterrichten können oder dass die Klasse geteilt und in zwei Räumen in verschiedenen Fächern unterrichtet wird?

6. Bearbeiten Sie im Handout die Aufgaben 4 und 5 und überprüfen Sie Ihr Wissen über ER-Modelle.

7.3 Grundlagen des relationalen Datenmodells erläutern

In der Gegenwart werden meist „relationale Datenbanken" genutzt. Anfang der 1970er-Jahre entwickelte der Mathematiker Edgar F. Codd das relationale Datenmodell, das bereits Anfang der 1980er-Jahre unter der Bezeichnung „dBase" auf Personal Computern implementiert wurde und sich bis heute allgemein für Datenbanken durchgesetzt hat. In relationalen Datenbanken werden die Daten meist in mehreren Tabellen, die miteinander in Beziehung stehen, unabhängig voneinander gespeichert.

7.3.1 Relationales Datenmodell

Ein relationales Datenmodell ist eine Sammlung von Tabellen (Relationen), die miteinander verknüpft sind, d.h. miteinander in Beziehung stehen.

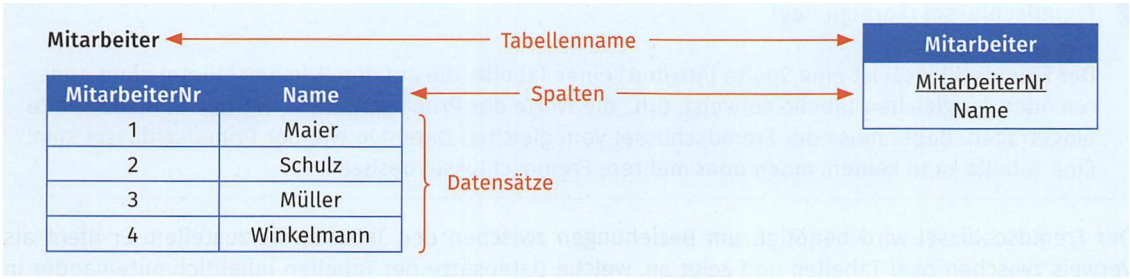
7.3.2 Tabellen

Eine Tabelle (Relation) besteht aus einem Tabellennamen (Relationsname) sowie den Spalten (Attribute, Felder) und den Datensätzen (Zeilen, Tupel).

7.3.3 Datensatz

Ein Datensatz besteht aus Datenfeldern (= entsprechend der Spaltenbezeichnung). In einem Datenfeld wird eine bestimmte Eigenschaft (Attributwert) des Objektes beschrieben.

Es ist möglich, die Tabellen auf zwei Arten darzustellen. Einmal als Tabelle mit Datensätzen (siehe linke Seite in Abbildung) oder als Tabelle nur mit Attributen (siehe rechte Seite in Abbildung). Die erste Variante wird meistens nur zu Erklärungszwecken verwendet. Für die eigentliche Darstellung des Modells wird immer die zweite Variante verwendet.



Beispiel einer Tabelle

7.3.4 Schlüssel

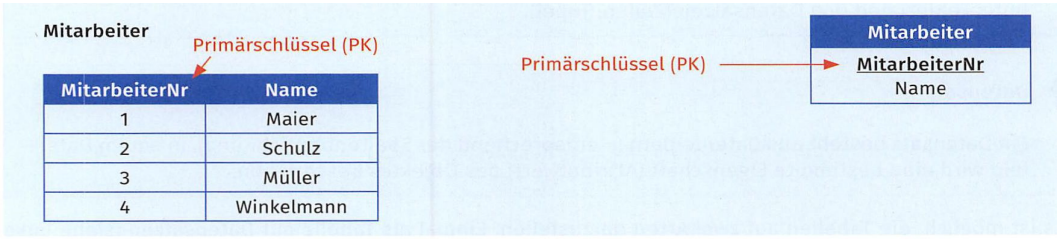
7.3.4.1 Primärschlüssel (Primary Key)

Der Primärschlüssel (Primary Key) einer Tabelle ist eine minimale Kombination von Spalten (Attributen), mit dessen Hilfe jeder Datensatz eindeutig identifiziert werden kann. Jede Tabelle muss genau einen Primärschlüssel besitzen. Der Primärschlüssel muss folgende Bedingungen erfüllen:

- ***Eindeutigkeit:*** *Die Werte des Primärschlüssels müssen eindeutig sein.*
- ***Minimalität:*** *Der Primärschlüssel sollte aus so wenig wie möglich Spalten bestehen. In der Regel reicht eine Spalte aus.*
- ***Unveränderbarkeit:*** *Die Werte des Primärschlüssels sollten sich nicht ändern.*

Im relationalen Datenmodell werden in den Tabellen nur die reinen Daten gespeichert. Wie diese Werte miteinander in Beziehung stehen, muss in Form von Verknüpfungen realisiert werden. Damit eine Verknüpfung von zwei Tabellen überhaupt funktionieren kann, muss jeder einzelne Datensatz eindeutig identifiziert werden. Dieses Merkmal nennt man Primärschlüssel (PK).

Im Prinzip kann jedes natürliche Attribut verwendet werden. Dadurch entstehen aber oft Probleme mit der Eindeutigkeit. Gleiche Namen existieren oft mehrfach für verschiedene Personen. Deshalb verwendet man meist eine Nummer (Nr) oder einen sog. Identifier (ID). Dieser künstliche Schlüssel kann stets so gewählt werden, dass er eindeutig ist. Der künstliche, willkürlich gewählte Schlüssel besitzt noch einen weiteren Vorteil, denn er dient ausschließlich der Identifikation des Datensatzes und enthält keinerlei weitere Informationen. Anders verhält es sich, wenn beispielsweise die Postleitzahl ein Bestandteil des Schlüssels ist. Dieser „sprechende Schlüssel" enthält eine zusätzliche Information. Ändert sich aber das Merkmal, weil der Kunde umzieht, so wird der Schlüssel fortan mit „gespaltener Zunge" sprechen, weil ja der Schlüssel nicht geändert werden darf. Wird jedoch bei einem Kunden der Schlüssel nachträglich geändert, gehen alle bisherigen Verknüpfungen verloren bzw. müssen alle Verknüpfungen erneut angepasst werden.

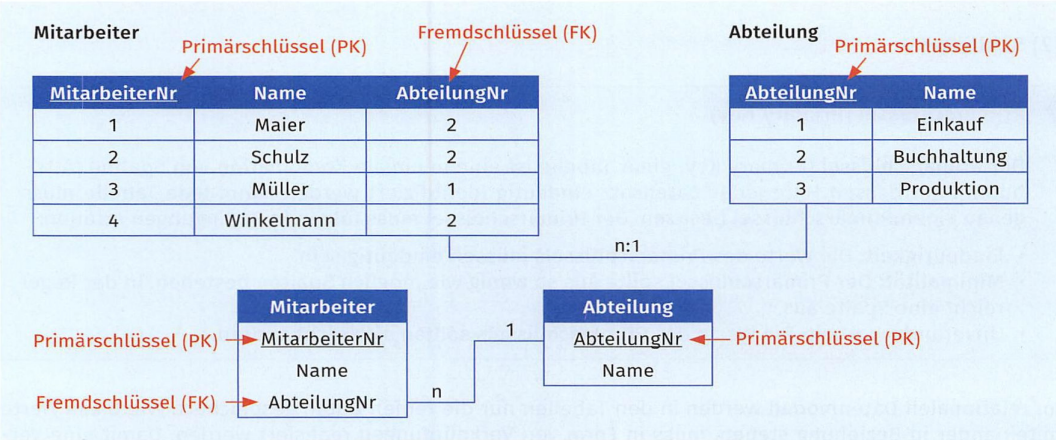


Beispiel für Primärschlüssel

7.3.4.2 Fremdschlüssel (Foreign Key)

Der Fremdschlüssel ist eine Spalte (Attribut) einer Tabelle, die auf den Primärschlüssel einer anderen oder der gleichen Tabelle verweist, d.h., die Werte des Primärschlüssels werden in diese Spalte eingetragen. Dabei muss der Fremdschüssel vom gleichen Datentyp wie der Primärschlüssel sein. Eine Tabelle kann keinen, einen oder mehrere Fremdschlüssel besitzen.

Der Fremdschlüssel wird benötigt, um Beziehungen zwischen den Tabellen herzustellen. Er dient als Verweis zwischen zwei Tabellen und zeigt an, welche Datensätze der Tabellen inhaltlich miteinander in Verbindung stehen.



Beispiel für Primär- und Fremdschlüssel

7.3.4.3 Kardinalitäten

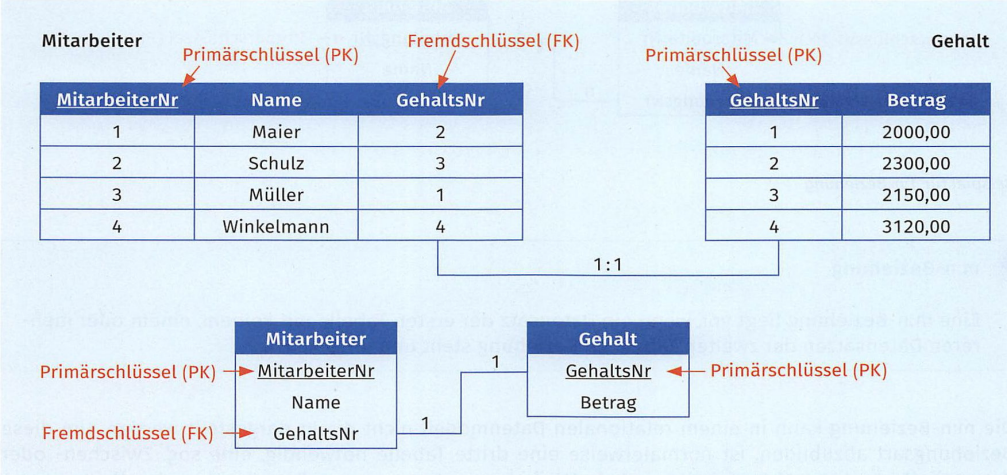
Die Kardinalität dient der genauen Charakterisierung von Beziehungen. Sie gibt an, in welchem Verhältnis die beteiligten Tabellen zueinander in Beziehung stehen. Im relationalen Datenmodell werden folgende Kardinalitäten unterschieden:

- *1:1 (eins zu eins)*
- *1:n (eins zu viele)*
- *m:n (viele zu viele)*

Nachfolgend werden die einzelnen Kardinalitäten anhand von Beispielen näher erläutert.

7.3.4.3.1 1:1-Beziehung

Eine 1:1-Beziehung liegt vor, wenn jedem Datensatz der ersten Tabelle genau ein Datensatz der zweiten Tabelle zugeordnet ist.

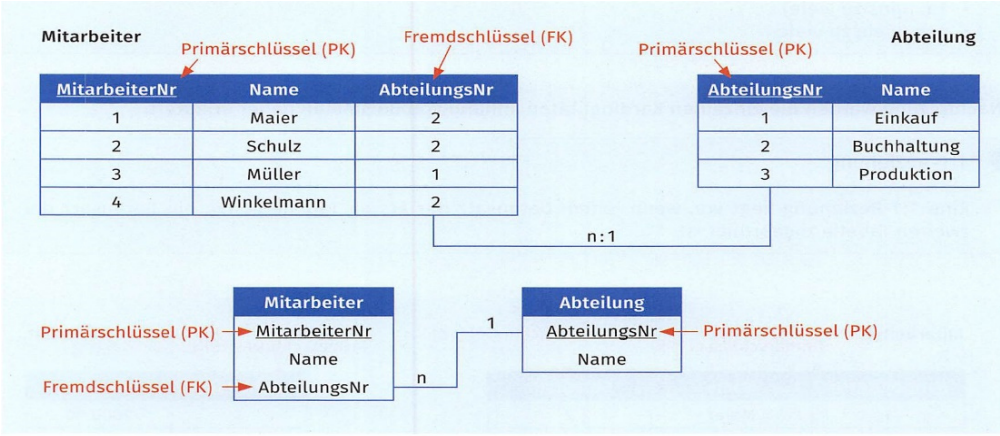


Beispiel für eine 1:1 Beziehung

Im Beispiel erhält jeder Mitarbeiter ein individuelles Gehalt, nicht eine Lohngruppe, wo eine andere Kardinalität zutreffen würde. Die meisten Daten, die auf diese Weise miteinander in Beziehung stehen, befinden sich normalerweise nur in einer Tabelle, weshalb diese Art der Beziehung selten ist. Zum Beispiel könnte das Gehalt auch direkt in die Tabelle „Mitarbeiter" eingetragen werden. Sinnvoll kann die Kardinalität verwendet werden, wenn eine Tabelle beispielsweise aus Gründen der Rechtevergabe in mehrere Tabellen aufgeteilt werden muss. Im Beispiel kann der Zustand hinterlegt sein, dass ein Benutzer die Daten der Gehaltstabelle nur lesen, während ein anderer Benutzer sie zusätzlich noch bearbeiten kann. Beide Benutzer besitzen aber die gleichen Rechte an der Tabelle „Mitarbeiter". Ein anderer Grund für eine 1:1-Beziehung besteht, wenn Attribute wegen der Übersichtlichkeit in andere Tabellen ausgelagert werden müssen.

7.3.4.3.2 1:n-Beziehung

Eine 1:n-Beziehung liegt vor, wenn ein Datensatz der ersten Tabelle mit keinem, einem oder mehreren Datensätzen der zweiten Tabelle in Beziehung steht. Im nachfolgenden Beispiel ist jeder Mitarbeiter genau einer Abteilung zugeordnet. Jede Abteilung hat aber mehrere Mitarbeiter.

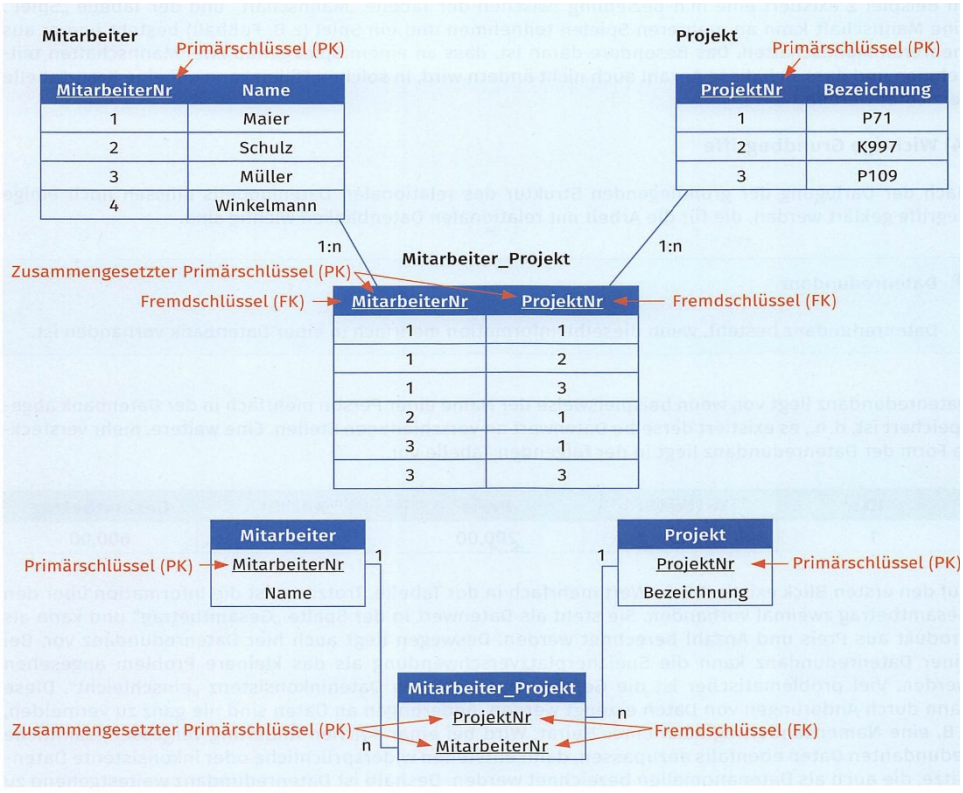


Beispiel für 1:n-Beziehung

7.3.4.3.3 m:n-Beziehung

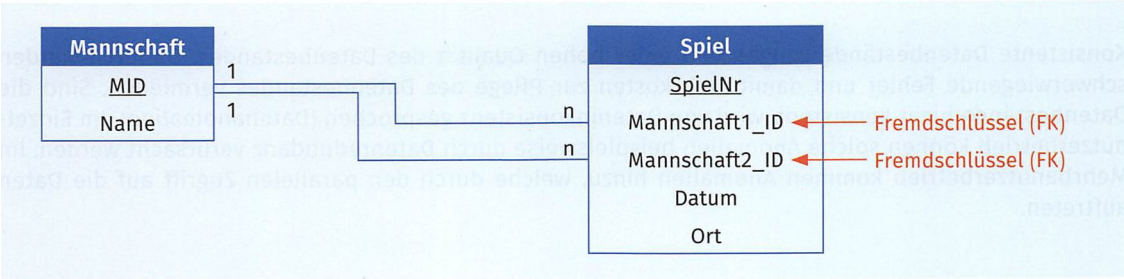
Eine m:n-Beziehung liegt vor, wenn ein Datensatz der ersten Tabelle mit keinem, einem oder mehreren Datensätzen der zweiten Tabelle in Beziehung steht und umgekehrt.

Die m:n-Beziehung kann in einem relationalen Datenmodell nicht direkt dargestellt werden. Um diese Beziehungsart abzubilden, ist normalerweise eine dritte Tabelle notwendig, eine sog. Zwischen- oder Kreuztabelle. In dieser Kreuztabelle sind als Attribute meistens nur zwei Fremdschlüssel enthalten, wor über sie mit den beiden anderen Tabellen verbunden wird. Die beiden Fremdschlüssel ergeben in Kom bination auch einen zusammengesetzten Primärschüssel. Daraus resultiert, dass die m:n-Beziehung in zwei 1:n-Beziehungen aufgelöst werden kann. Im Beispiel 1 (siehe S. 587) gehören zu einem Projekt meh rere Mitarbeiter, aber ein Mitarbeiter kann auch an verschiedenen Projekten beteiligt sein. Findet sich für die Kreuztabelle kein geeigneter Name, wird er aus den Namen der beiden anderen Tabellen gebildet (z. B. Mitarbeiter_Projekt).



Beispiel 1 für m:n-Beziehung

Normalerweise wird jede m:n-Beziehung durch die Einführung einer Kreuztabelle dargestellt. In einigen Fällen ist es aber auch möglich, darauf zu verzichten und eine andere Abbildungsform zu wählen, wie im Beispiel 2 zu sehen ist.



Beispiel 2 für m:n-Beziehung

Im Beispiel 2 existiert eine m:n-Beziehung zwischen der Tabelle „Mannschaft" und der Tabelle „Spiel". Eine Mannschaft kann an mehreren Spielen teilnehmen und ein Spiel (z.B. Fußball) besteht immer aus mehreren Mannschaften. Das Besondere daran ist, dass an einem Spiel genau zwei Mannschaften teilnehmen und dass sich diese Anzahl auch nicht ändern wird. In solchen Fällen kann auf eine Kreuztabelle verzichtet werden.

7.3.4.4 Wichtige Grundbegriffe

Nach der Darlegung der grundlegenden Struktur des relationalen Datenmodells müssen noch einige Begriffe geklärt werden, die für die Arbeit mit relationalen Datenbanken wichtig sind.

7.3.4.4.1 Datenredundanz

Datenredundanz besteht, wenn dieselbe Information mehrfach in einer Datenbank vorhanden ist.

Datenredundanz liegt vor, wenn beispielsweise der Name einer Person mehrfach in der Datenbank abgespeichert ist, d. h., es existiert derselbe Datenwert an verschiedenen Stellen. Eine weitere, mehr versteckte Form der Datenredundanz liegt in der folgenden Tabelle vor.

ID	ArtikelNr	Preis	Anzahl	Gesamtbetrag
1	45	200,00	3	600,00

Auf den ersten Blick existiert kein Wert mehrfach in der Tabelle. Trotzdem ist die Information über den Gesamtbetrag zweimal vorhanden. Sie steht als Datenwert in der Spalte „Gesamtbetrag" und kann als Produkt aus Preis und Anzahl berechnet werden. Deswegen liegt auch hier Datenredundanz vor. Bei einer Datenredundanz kann die Speicherplatzverschwendung als das kleinere Problem angesehen werden. Viel problematischer ist die Gefahr, dass sich eine Dateninkonsistenz „einschleicht". Diese kann durch Änderungen von Daten erzeugt werden. Änderungen an Daten sind nie ganz zu vermeiden, z.B. eine Namensänderung nach einer Heirat. Wird bei einer solchen Änderung vergessen, sämtliche redundanten Daten ebenfalls anzupassen, dann entstehen widersprüchliche oder inkonsistente Datensätze, die auch als Datenanomalien bezeichnet werden. Deshalb ist Datenredundanz weitestgehend zu vermeiden.

7.3.4.4.2 Datenkonsistenz

Als Datenkonsistenz wird die Korrektheit der in der Datenbank gespeicherten Daten bezeichnet. Daten sind konsistent, wenn sie widerspruchsfrei sind.

Konsistente Datenbestände zeugen von einer hohen Qualität des Datenbestandes. Dadurch werden schwerwiegende Fehler und damit Mehrkosten zur Pflege des Datenbestandes vermieden. Sind die Datenbestände nicht konsistent, wird von Dateninkonsistenz gesprochen (Datenanomalien). Im Einzel nutzerbetrieb können solche Anomalien beispielsweise durch Datenredundanz verursacht werden. Im Mehrbenutzerbetrieb kommen Anomalien hinzu, welche durch den parallelen Zugriff auf die Daten auftreten.

Bezeichnung	Beschreibung
Änderungsanomalie	Wenn dieselben Daten an verschiedenen Stellen gespeichert wurden (Datenredundanz) und diese Daten geändert werden sollen, können beim Ändern einige Datensätze vergessen werden. In diesem Fall spricht man von einer Änderungsanomalie. Ändert etwa ein Mitarbeiter seinen Namen, muss diese Änderung an allen Stellen vorgenommen werden.
Einfügeanomalie	Eine Einfügeanomalie liegt vor, wenn bestimmte Daten nur in Verbindung mit anderen Daten erfasst werden können. Soll beispielsweise ein Mitarbeiter erfasst werden, muss ihm auch eine Abteilung zugewiesen werden. Steht die Abteilung zum Erfassungszeitpunkt aber noch nicht fest, so muss dem Mitarbeiter zunächst eine „Dummy-Abteilung“ zugewiesen werden.
Löschanomalie	Eine Löschanomalie tritt auf, wenn beim Löschen bestimmter Daten ungewollt andere Daten gelöscht werden. Wird beispielsweise eine Abteilung gelöscht, wer den ggf. auch alle Daten der Mitarbeiter dieser Abteilung gelöscht, obwohl diese Mitarbeiter in andere Abteilungen aufgeteilt werden.

7.3.4.4.3 Referenzielle Integrität

Referenzielle Integrität besagt, dass die Werte des Fremdschlüssels der Fremdschlüsseltabelle auch als Werte beim Primärschlüssel der Primarschlüsseltabelle vorhanden sein müssen. Dadurch wird Dateninkonsistenz vermieden.

Um die referenzielle Integrität zu gewährleisten, muss ein Datensatz zuerst mit einem entsprechenden Wert in der Tabelle mit dem Primärschlüssel vorhanden sein, bevor dieser Wert in die Tabelle mit dem Fremdschüssel eingetragen wird. Beim Löschen eines Datensatzes aus der Primärschlüsseltabelle verhält es sich umgekehrt. Zuerst müssen alle Datensätze aus der Fremdschlüsseltabelle entfernt werden, deren Wert mit dem des Primärschlüssels im Datensatz der Primärschlüsseltabelle übereinstimmt.

7.3.4.4.4 Normalisierung

Normalisierung ist ein Verfahren zur Verringerung von Datenredundanz in relationalen Datenmodellen, verbunden mit den Zielen Anomalien zu vermeiden und die Datenkonsistenz zu erhöhen.

Die Einhaltung der Integrität bei einer relationalen Datenbank erzwingt bestimmte Anforderungen an die Form der Tabellen, die über den Weg der Normalisierung erreicht werden kann. Am Ende der Normalisierung stehen Relationen, die einer vorgegebenen Normalform entsprechen. Je höher die gewählte Normalform definiert wird, desto höher sind die Anforderungen an die innere Struktur der Relation. Durch die Normalisierung sollen einerseits die Relationen aufgebaut und andererseits die Redundanzen verringert bzw. ganz vermieden werden.

7.3.4.5 Beispiele für die Erstellung eines relationalen Datenmodells

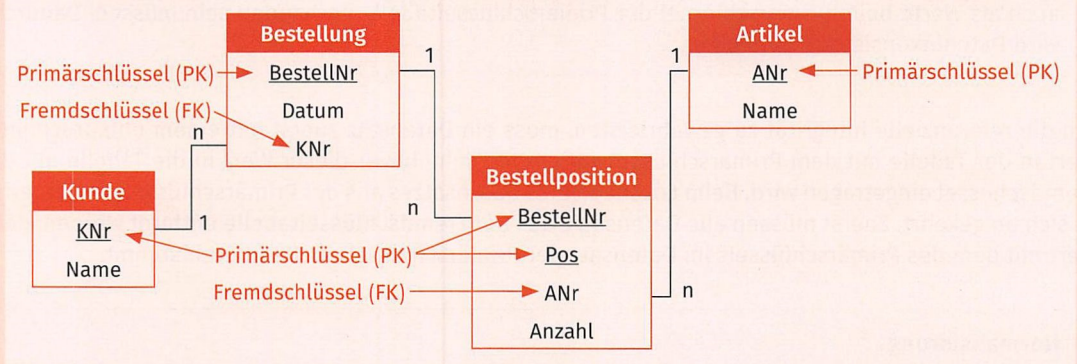
Beispielaufgabe:

Es soll ein relationales Datenmodell für die Verwaltung von Kundenbestellungen erstellt werden. Folgende Fakten sind bekannt:

- *Ein Kunde (Name, KundenNr) macht mehrere Bestellungen.*
- *Jede Bestellung (BestellNr, Datum) kann mehrere Bestellpositionen haben, in denen der Artikel (ArtikelNr, Name) und die bestellte Anzahl aufgeführt werden.*

Lösung:

Zunächst werden die Tabellen „Kunde", „Bestellung", „Bestellposition" und „Artikel" mit den entsprechenden Attributen angelegt. Danach wird in jeder Tabelle der Primärschlüssel festgelegt. Wenn kein Attribut oder eine Kombination aus Attributen sich als Primärschlüssel eignet, dann wird ein künstlicher Primärschlüssel eingefügt. In diesem Fall ist zu jeder Tabelle ein Primärschlüssel vorhanden, z. B. die BestellNr. In der Tabelle „Bestellposition" wird ein zusammengesetzter Primärschlüssel aus BestellNr und Pos gebildet. Danach werden die Beziehungen zwischen den Tabellen festgelegt und dementsprechende Fremdschlüssel in die Tabellen eingetragen. Es sind alles 1:n-Beziehungen, z.B. ein Kunde kann mehrere Bestellungen machen oder ein Artikel kann in mehreren Bestellpositionen vorkommen. Die Lösung für die Aufgabe ist in folgender Abbildung zu sehen.

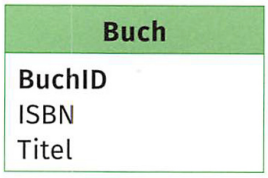


Kompetenzcheck

1. Welche Aussagen sind richtig?

   a. Das relationale Datenmodell ist nicht weitverbreitet.
   b. Die Werte eines Primärschlüssels müssen immer eindeutig sein.
   c. m:n-Beziehungen erfordern immer eine Kreuztabelle.
   d. Dateninkonsistenz ist ein gewollter Zustand der Datenbank.
   e. Ein Primärschlüssel besteht immer nur aus einem Attribut.
   f. Jede Tabelle muss mindestens einen Fremdschlüssel enthalten.
   g. Änderungsanomalien können durch Datenredundanz entstehen.

2. Erweitern Sie nachfolgendes Datenmodell dahingehend, dass Sie eine Tabelle mit den Autoren (Name, Vorname) hinzufügen und diese mit der Tabelle Buch in Beziehung setzten. Hinweis: Ein Buch kann auch von mehreren Autoren geschrieben worden sein.



3. Erweitern Sie die Tabelle „Auto" um folgende Attribute: Kennzeichen, Farbe, Schäden, Kraftstoffart, Hersteller und kmStand. Versuchen Sie mögliche Datenredundanzen zu vermeiden, indem Sie die entsprechenden Attribute in neue Tabellen auslagern und diese mit der Tabelle „Auto" in Beziehung setzen.



4. Es sollen die Anbaupläne für Felder (Feldname) verwaltet werden. Dabei soll erfasst werden, welche Pflanze (Pflanzenname) von wann bis wann, auf welchem Feld angebaut wird. Entwickeln Sie ein relationales Datenmodell.

5. Bearbeiten Sie im Handout die Aufgaben 6 und 7.

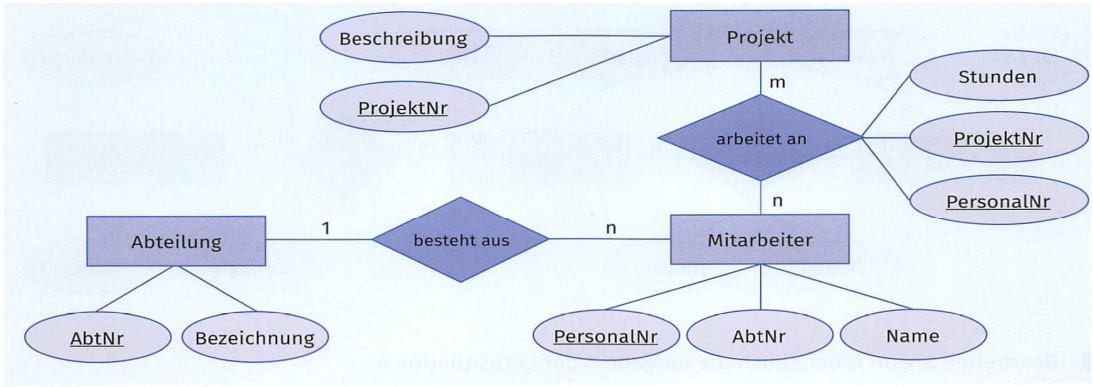
7.4 ER-Modelle in relationale Datenmodelle überführen

Nach der Datenanalyse und der Erstellung eines ER-Modells muss dieses in ein relationales Datenmodell überführt werden. Zur Unterstützung dieses Prozesses gibt es mehrere Regeln.

Transformationsregeln

1. Ein Entitätstyp wird mit all seinen Attributen zu jeweils einer Tabelle zusammengefasst.
2. Jede Tabelle erhält einen Primärschlüssel aus einem Attribut oder einer Kombination aus Attributen, welche im ER-Modell gekennzeichnet wurden. Sind solche Attribute nicht vorhanden, dann wird ein künstlicher Primärschlüssel eingeführt.
3. Jede m:n-Beziehung wird in einer eigenen Tabelle abgebildet. Diese Tabelle enthält die Primärschlüssel der beteiligten Entitätstypen als Fremdschlüssel und die Attribute, welche der Beziehung direkt zugeordnet sind.
4. Die Umsetzung der 1:n-Beziehungen erfolgt, indem der Primärschlüssel der einen Tabelle als Fremdschlüssel in die andere Tabelle eingefügt wird.

Der gesamte Vorgang wird nachfolgend erläutert. Als Beispiel liegt folgendes ER-Modell vor:



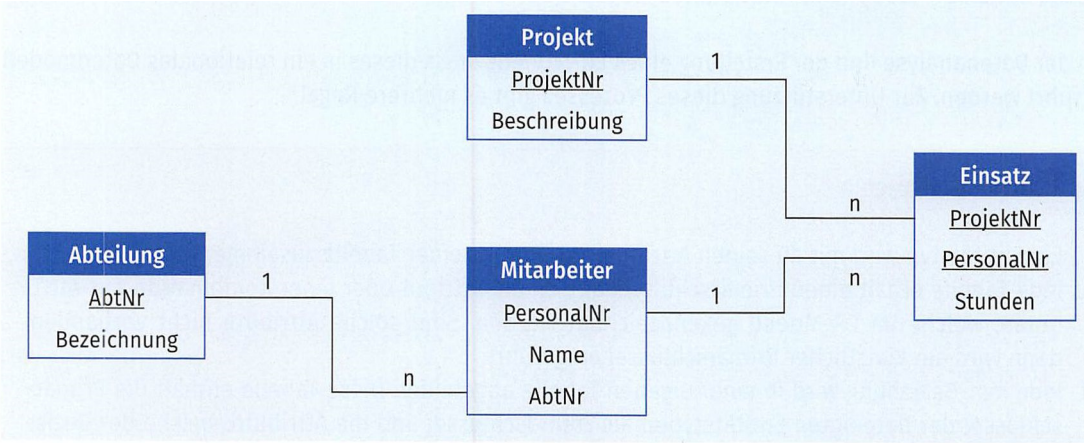
ER-Modell

In diesem Beispiel sind die Entitätstypen „Abteilung", „Mitarbeiter" und „Projekt" mit entsprechenden Attributen abgebildet. Eine Abteilung besitzt mehrere Mitarbeiter und die Mitarbeiter arbeiten in verschiedenen Projekten mit. Durch die m:n-Beziehung zwischen Mitarbeiter und Projekt können auch Attribute direkt an der Beziehung angegeben werden.

Man kann wie folgt vorgehen, um daraus ein relationales Datenmodell zu entwickeln:

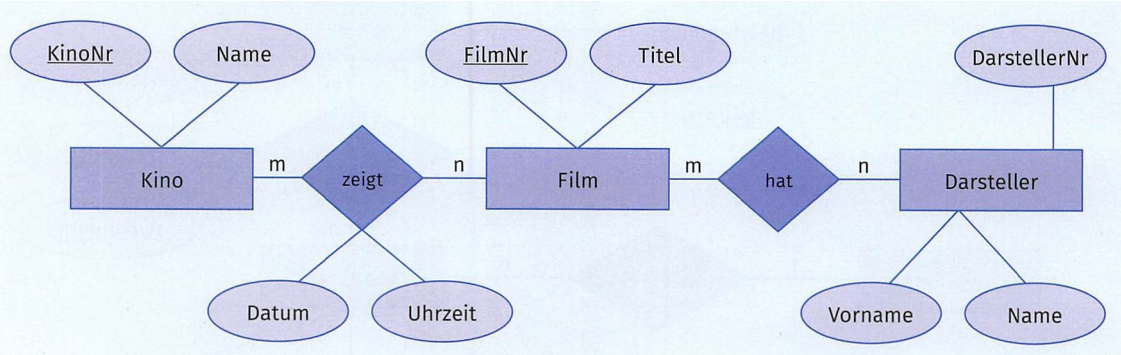
1. Aus jedem Entitätstyp des ER-Modells wird eine Tabelle mit dem entsprechenden Namen erstellt. Die Attribute der einzelnen Entitätstypen werden zu Attributen der jeweiligen Tabelle.
2. Die Primärschlüssel werden festgelegt. Diese können in diesem Fall direkt aus dem ER-Modell übernommen werden.
3. Die 1:n-Beziehung zwischen Abteilung und Mitarbeiter wird ohne Veränderung übernommen. Der Primärschlüssel AbtNr ist schon als Attribut in der Mitarbeitertabelle vorhanden und wird zum Fremdschlüssel.
4. Die n:m-Beziehung zwischen Mitarbeiter und Projekt wird aufgelöst und in zwei 1:n-Beziehungen überführt. Dazu wird die n:m-Beziehung und deren Attribute in einer neuen Tabelle dargestellt, der sog. Kreuztabelle.

Das relationale Datenmodell entspricht im Ergebnis der nachfolgenden Abbildung.



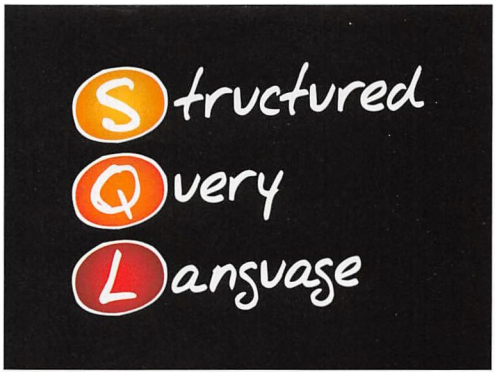
Kompetenzcheck

1. Wandeln Sie folgendes ER-Modell in ein relationales Modell um.



2. Bearbeiten Sie im Handout die Aufgabe 8.

7.5 Die Datenbanksprache SQL anwenden



Gegenwärtig ist SQL (Structured Query Language) unbestritten die verbreitetste Sprache für relationale Datenbanken. Auch in Zukunft wird diese Sprache keine Konkurrenz bekommen, da mittlerweile alle Hersteller relationaler Datenbanken ihr System auf diese Sprache umgestellt haben.

Die Entwicklung von SQL begann Anfang der 1970er-Jahre. Zu dieser Zeit begann in den IBM-Forschungslaboratorien in San Jose im US-Bundesstaat Kalifornien ein Forschungsprojekt, das sich „System R“ nannte. Es sollte die Praktizierbarkeit der relationalen Theorien untersuchen.

SQL (Structured Query Language)

SQL ist eine Datenbanksprache, mit der in relationalen Datenbanken die Datenstrukturen definiert und die Datenbestände bearbeitet bzw. abgefragt werden können.

Die IBM-Mitarbeiter R.F. Boyce und D. D. Chamberlain entwickelten die Sprache SEQUEL, die später in SQL umbenannt wurde. Die Syntax wurde an die Begriffe der englischen Umgangssprache angelehnt, erkenn bar an SELECT, FROM oder WHERE. Seither entwickelten fast alle Datenbankhersteller die erforderlichen SQL-Schnittstellen zu ihren relationalen und nicht relationalen Datenbanksystemen. SQL ist als Standardsprache für alle relationalen Datenbanksysteme zu betrachten, da eine ANSI- und eine ISO-Definition der Sprache SQL vorliegt.

SQL-Standards

SQL-87: ISO 9075:1987
SQL-89: ISO/IEC 9075:1989
SQL-92: ISO/IEC 9075:1992
SQL-1999: ISO/IEC 9075:1999
SQL-2003: ISO/IEC 9075:2003
SQL-2006: ISO/IEC 9075:2006
SQL-2008: ISO/IEC 9075:2008
SQL-2011: ISO/IEC 9075:2011
SQL-2016: ISO/IEC 9075:2016

Trotz ihrer Standardfunktion wird SQL von den gängigen Datenbanksystemen in unterschiedlichem Maße unterstützt. Außerdem können die verschiedenen „Dialekte" von SQL leicht vom Standard abweichen. SQL besitzt verschiedene Sprachelemente, die nach ihrer Funktion in Kategorien eingeteilt werden.

Sprache	Beschreibung
Data Definition Language (DDL) - Datendefinitionssprache	Die DDL umfasst alle Anweisungen, die verwendet werden, um Datenstrukturen und verwandte Elemente zu beschreiben, zu ändern oder zu entfernen. Zum Beispiel können damit Tabellen erzeugt oder vorhandene Tabellenstrukturen verändert werden. Zu den SQL- Befehlen der DDL gehören u.a. CREATE TABLE oder DROP VIEW.
Data Manipulation Language (DML) - Datenbearbeitungssprache	Die DML umfasst alle Anweisungen an das Datenbanksystem, die dazu dienen, grundlegende Operationen an Datensätzen auszuführen, z. B. das Auswählen von Datensätzen aus einer Tabelle, das Ein fügen neuer Datensätze, das Löschen von Datensätzen oder die Veränderung von bestehenden Datensätzen. Zu den SQL-Befehlen der DML gehören u.a. SELECT, UPDATE oder INSERT INTO.
Data Control Language (DCL) - Datenüberwachungssprache	Die DCL enthält Befehle, um an Nutzer der Datenbank Berechtigungen zu vergeben oder ihnen Berechtigungen zu entziehen. Typische Befehle der DCL sind u.a. GRANT und REVOKE.
Transaction Control Language (TCL) – Transaktions-überwachungssprache	Die Transaktionsbefehle der SQL stellen die Datenintegrität sicher, indem logisch zusammenhängende Anweisungen entweder komplett oder gar nicht ausgeführt werden. Zu den SQL-Befehlen der TCL gehören u.a. COMMIT oder ROLLBACK.

Im ersten Lehrjahr werden vor allem die Befehle der DDL und DML in sehr einfacher Form betrachtet. Im zweiten bzw. dritten Lehrjahr werden diese vertieft und es kommen die Befehle der DCL und TCL hinzu.

7.5.1.1 Syntaktische Grundlagen und Operatoren

SQL ist bewusst sehr einfach gehalten. Es wird nicht zwischen Groß- und Kleinschreibung unterschieden. Es ist also unerheblich, ob in der SQL-Anweisung alles groß, alles klein oder als Kombination geschrieben wird. Allerdings existiert ein üblicher Standard, wonach SQL-Befehle nur mit Großbuchstaben geschrieben werden. Im Normalfall wird eine SQL-Anweisung mit einem Semikolon beendet. Allerdings gibt es auch hier Ausnahmen. Im Zweifelsfall sollte immer ein Semikolon gesetzt werden. In der folgenden Tabelle sind weitere Sprachelemente und Operatoren aufgeführt.

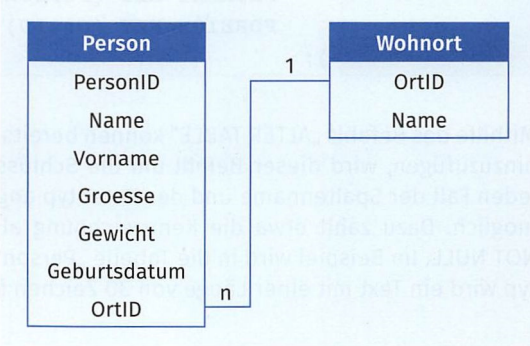
Sprachelement	**Beschreibung**		
Anführungszeichen und Hochkommata	Sie werden immer dann genutzt, wenn in der SQL-Anweisung Zeichenketten enthalten sind. Der entsprechende Text erscheint in einfachen Hochkommata oder in normalen Anführungszeichen oben, z. B.: WHERE Name = ‘Maier’ oder WHERE Name = “Maier".		
Vergleichsoperatoren	Sie kommen hauptsächlich in WHERE-Klauseln vor, die in verschiedenen Zusammenhängen verwendet werden. Mit den Operatoren werden die Bedingungen beschrieben, unter denen die SQL-Anweisung ausgeführt wird.	**Operator**	**Beschreibung**
		=	gleich
		>	größer
		>=	größer gleich
		<	kleiner
		<=	kleiner gleich
		<>	ungleich
Logische Operatoren	Für die logischen Operatoren gilt wie für die Vergleichsoperatoren, dass sie bevorzugt in der WHERE-Klausel verwendet werden und verschiedene Bedingungen miteinander verknüpfen. Mithilfe von Klammern können diese Operatoren zu logischen Einheiten gebündelt werden.	**Operator**	**Beschreibung**
		AND	logisches UND
		OR	logisches ODER
		NOT	logisches NICHT

Sprachelement	**Beschreibung**		
Rechenoperatoren	Sie können an verschiedenen Stellen einer SQL- Anweisung angewendet werden. Durch den Einsatz von Klammern wird ein mathematischer Ausdruck gezielt beschrieben.	**Operator**	**Beschreibung**
		+	Addition
		-	Subtraktion
		*	Multiplikation
		/	Division
Zuweisungsoperator	Er ist identisch mit dem Vergleichsoperator, wird hauptsächlich benutzt, um die Werte einer Spalte zu verändern.	**Operator**	**Beschreibung**
		=	Zuweisung
NULL	Der Wert NULL steht für einen nicht definierten Wert und ist nicht vergleichbar mit der Ziffer „0" oder einem Leerstring "". Man verwendet den Vergleich IS NULL, um zu testen, ob eine Spalte einen nicht definierten Wert enthält, z. B.: WHERE Ort IS NULL.		

Des Weiteren werden beispielsweise für das Anlegen von Tabellen oder für das Hinzufügen von Spalten Datentypen benötigt. Auch hier gibt es Standardvorgaben, die von den meisten Datenbankmanagementsystemen unterstützt werden. Darüber hinaus hat aber fast jeder Datenbankhersteller seine eignen Erweiterungen geschaffen, welche sich teilweise erheblich unterscheiden. Es ist zu empfehlen, sich die Datentypen für das Datenbankmanagementsystem, welches man benutzt, zu erarbeiten.

ANSI-SQL-Datentyp	Beschreibung
INTEGER, BIGINT	Ganze Zahlen
NUMERIC, NUMBER(n,m)	Festkommazahlen
FLOAT, DOUBLE	Gleitkommazahlen
VARCHAR(n)	Zeichenkette variabler Länge
CHARACTER(n), CHAR(n)	Zeichenkette mit fester Länge
DATE, TIME, TIMESTAMP	Datums- und Zeitangaben
BOOLEAN	Wahrheitswerte

In den nachfolgenden Abschnitten werden die wichtigsten Befehle der SQL näher betrachtet und an Beispielen erläutert. Als Grundlage für alle Bei spiele dient ein relationales Datenmodell der Beispieldatenbank „Personenverwaltung" Zum Testen der Beispiele für die SELECT-Anweisungen müssen die Tabellen noch mit entsprechenden Datensätzen gefüllt werden.



Beispieldatenbank „Personenverwaltung“

7.5.2 Anlegen, Ändern, Löschen von Tabellen

Zum Anlegen einer Tabelle wird der Befehl “CREATE TABLE” verwendet. Dessen Syntax kann anhand des folgenden Beispiels beschrieben werden, indem die Tabelle „Person“ mit den entsprechenden Spalten erzeugt wird. Es bestehen zwei Möglichkeiten, um den Primär- und die Fremdschlüssel festzulegen: beim Definieren der Spalte oder nach der Definition aller Spalten. Im Beispiel „Personalverwaltung" muss die Tabelle „Wohnort" für das Anlegen des Fremdschlüssels bereits existieren, d.h., die Tabelle muss schon vorab mithilfe des Befehls „CREATE TABLE" erzeugt werden.

Anlegen einer Tabelle	
Syntax	*CREATE TABLE Tabellenname (* *Spaltenname1 Datentyp1 [NOT NULL] [AUTOINCREMENT] [PRIMARY KEY]* *Spaltenname2 Datentyp2 [FOREIGN KEY REFERENCES]* *);*
Erläuterung	*Tabellenname = Name der Tabelle* *Spaltenname = Name der jeweiligen Spalte* *Datentyp = Datentyp der jeweiligen Spalte (z. B. INT, VARCHAR(50) etc.)* *NOT NULL = Wert der Spalte darf nicht NULL sein* *AUTOINCREMENT = Zahl wird bei jedem neuen Datensatz automatisch um eins erhöht* *PRIMARY KEY = Spalte wird zum Primärschlüssel* *FOREIGN KEY = Spalte wird zum Fremdschlüssel* *REFERENCES = Verknüpfung mit dem Primärschlüssel einer anderen Tabelle*

Beispiele	CREATE TABLE Person (PersonID INT NOT NULL AUTOINCREMENT PRIMARY KEY, Name VARCHAR(50) NOT NULL, Vorname VARCHAR (50), Groesse FLOAT, Gewicht FLOAT, Geburtsdatum DATE OrtID INT FOREIGN KEY REFERENCES Ort(OrtID));
	Alternative
	CREATE TABLE Person (PersonID INT NOT NULL AUTOINCREMENT, Name VARCHAR(50) NOT NULL, Vorname VARCHAR (50), Groesse FLOAT, Gewicht FLOAT, Geburtsdatum DATE, OrtID INT, PRIMARY KEY (PersonID), FOREIGN KEY (OrtID) REFERENCES Ort(OrtID));

Mithilfe des Befehls „ALTER TABLE" können bereits existierende Tabelle angepasst werden. Um eine Spalte hinzuzufügen, wird dieser Befehl um die Schlüsselwörter „ADD COLUMN" erweitert. Dabei müssen auf jeden Fall der Spaltenname und der Datentyp angegeben werden. Die Angabe von weiteren Optionen ist möglich. Dazu zählt etwa die Kennzeichnung als Primär- oder Fremdschlüssel oder die Angabe von NOT NULL. Im Beispiel wird in die Tabelle „Person" die neue Spalte „Bemerkungen" eingefügt. Als Daten typ wird ein Text mit einer Länge von 30 Zeichen festgelegt. Weitere Optionen existieren nicht.

Ändern einer Tabelle: Spalte hinzufügen	
Syntax	ALTER TABLE Tabellenname ADD COLUMN Spaltenname Datentyp Optionen;
Erläuterung	Tabellenname = Name der Tabelle, in welcher eine neue Spalte eingefügt werden soll Spaltenname = Name der neuen Spalte Datentyp = Datentyp der neuen Spalte (z. B. INT, VARCHAR(50) etc.) Optionen = Angabe von zusätzlichen Optionen wie NOT NULL, FOREIGN KEY etc.
Beispiel	ALTER TABLE Person ADD COLUMN Bemerkungen VARCHAR(30);

Wird ALTER TABLE in Verbindung mit MODIFY COLUMN verwendet, kann einer bereits vorhandenen Spalte ein neuer Datentyp zugewiesen werden. Im Beispiel wird der Datentyp der Spalte „Bemerkungen" auf eine Länge mit 250 Zeichen verändert.

Ändern einer Tabelle: Datentyp einer Spalte ändern	
Syntax	ALTER TABLE Tabellenname MODIFY COLUMN Spaltenname Datentyp;
Erläuterung	Tabellenname = Name der Tabelle, in welcher eine Spalte bearbeitet werden soll Spaltenname = Name der zu bearbeitenden Spalte Datentyp = neuer Datentyp der ausgewählten Spalte
Beispiel	ALTER TABLE Person MODIFY COLUMN Bemerkungen VARCHAR(250);

Die Anweisung „DROP COLUMN" löscht in Verbindung mit ALTER TABLE eine Spalte aus der Tabelle.

Ändern einer Tabelle: Spalte löschen	
Syntax	`ALTER TABLE Tabellenname DROP COLUMN Spaltenname;`
Erläuterung	`Tabellenname = Name der Tabelle, in der eine Spalte gelöscht werden soll.` `Spaltenname = Name der zu löschenden Spalte`
Beispiel	`ALTER TABLE Person DROP COLUMN Bemerkungen;`

Der Befehl zum Löschen einer Tabelle ist einfach, aber wirkungsvoll. Nach DROP TABLE wird der Name der zu löschenden Tabelle angeben. Im Beispiel wird die Tabelle „Person" gelöscht.

Löschen einer Tabelle	
Syntax	`DROP TABLE Tabellenname;`
Erläuterung	`Tabellenname = Name der zu löschenden Tabelle`
Beispiel	`DROP TABLE Person;`

7.5.3 Einfügen, Ändern, Löschen von Datensätzen

Zum Einfügen von Datensätzen dient der Befehl „INSERT". Im Beispiel wird ein neuer Datensatz in die Tabelle „Person" geschrieben, wobei nur die Spalten „Vorname", „Name" und „Gewicht" Werte erhalten. Da der Primärschlüssel „PersonID" als Autowertgekennzeichnet ist, wird dieser Wert automatisch erzeugt und muss nicht extra vergeben werden.

Einfügen von Datensätzen	
Syntax	`INSERT INTO Tabellenname (Spalte1, Spalte2, ... )` `  VALUES (Wertl, Wert2,  );`
Erläuterung	`Tabellenname = Name der Tabelle, in die etwas geschrieben werden soll` `Spalte = Name der Spalte, in die ein Wert geschrieben werden soll` `Wert = Wert, der in die entsprechende Spalte geschrieben werden soll`
Beispiel	`INSERT INTO Person (Vorname, Name, Gewicht)` `  VALUES ("Hans", "Müller", 80.5);`

Sollen die Werte von vorhandenen Datensätzen verändert werden, kommt der Befehl „UPDATE" zum Einsatz. Mithilfe der Bedingungen in der WHERE-Klausel werden jene Datensätze ausgewählt, bei denen der Wert geändert werden soll. Entfällt die WHERE-Klausel, erfolgt die Änderung bei allen Datensätzen der Tabelle. Im Beispiel soll das Gewicht einer bestimmten Person geändert werden, welche durch die PersonID bestimmt wird. Der Befehl „UPDATE" kann auch zum Löschen von einzelnen Werten verwendet werden. Dazu wird der Wert der entsprechenden Spalte auf NULL gesetzt. Der Datensatz bleibt dabei aber erhalten.

Ändern von Daten	
Syntax	`UPDATE Tabellenname SET Spalte = Wert WHERE Bedingungen;`
Erläuterung	`Tabellenname = Name der Tabelle, welche aktualisiert werden soll` `Spalte = Name der Spalte, in welcher der Wert geändert werden soll` `Wert = neuer Wert, der in die Spalte geschrieben wird` `Bedingungen = Einschränkung der Datensätze, die einen neuen Wert erhalten`
Beispiele	`Ändern des Wertes eines Datensatzes:` `UPDATE Person SET Gewicht = 79.25 WHERE PersonID = 10;`

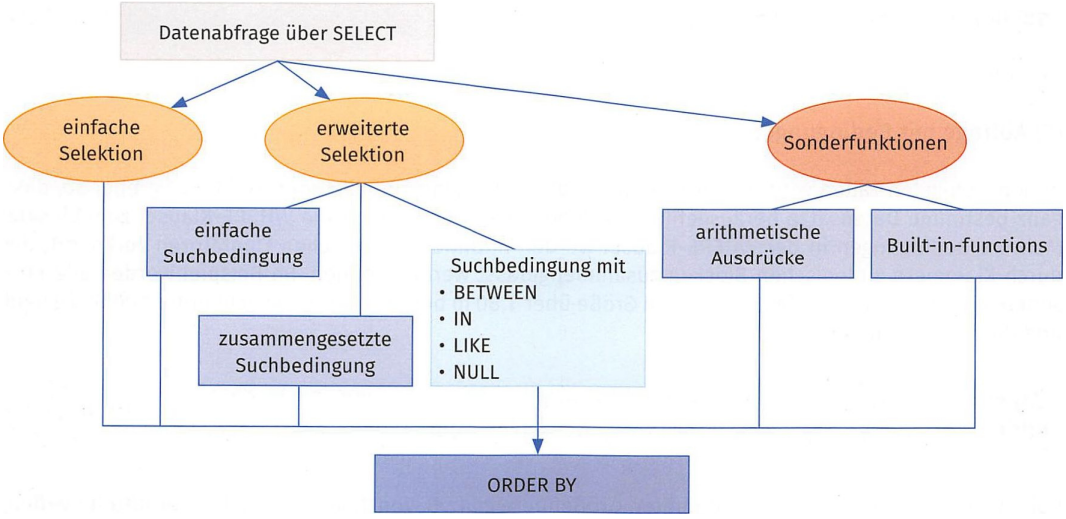
	Löschen (auf NULL setzen) von Werten eines Datensatzes: *UPDATE Person SET Gewicht = NULL WHERE PersonID = 21 OR PersonID = 43;*

Zum Löschen von Datensätzen wird der Befehl „DELETE" verwendet. Mit diesem Befehl werden immer komplette Datensätze gelöscht. Deswegen ist es auch nicht nötig, nach DELETE Spalten anzugeben. Auf den Befehl „DELETE" folgt stattdessen das Schlüsselwort „FROM". Mithilfe der Angabe von Bedingungen in der WHERE-Klausel können die zu löschenden Datensätze eingegrenzt werden. Fehlt die Angabe von Bedingungen, werden alle Datensätze der Tabelle gelöscht. Im Beispiel werden alle Datensätze aus der Tabelle „Person" gelöscht, in denen kein Name enthalten ist.

Löschen von Datensätzen	
Syntax	*DELETE FROM Tabellenname WHERE Bedingungen;*
Erläuterung	*Tabellenname = Name der Tabelle, aus der Datensätze gelöscht werden sollen* *Bedingungen = Einschränkung auf jene Datensätze, die gelöscht werden*
Beispiel	*DELETE FROM Person WHERE Name IS NULL;*

7.5.4 Abfragen von Daten

Die Datenabfrage erfolgt mithilfe des Befehls „SELECT". Dabei reicht das Spektrum der Abfragen von „sehr einfach" bis „sehr komplex", weil der Befehl „SELECT" durch verschiedene Befehle erweitert werden kann. Im ersten Lehrjahr werden erst einmal nur einfache SQL-Abfragen betrachtet. Diese werden dann im zweiten bzw. dritten Lehrjahr komplexer. Die Möglichkeiten für die Befehlserweiterungen werden in der folgenden Abbildung veranschaulicht.



Die Erläuterungen zur allgemeinen Syntax des Befehls „SELECT" sollen die Verständlichkeit erhöhen.

Datenabfrage mit SELECT	
Syntax	*SELECT [DISTINCT] Spalte1 [AS Alias], Spalte2,* *FROM Tabelle1 [Alias], Tabelle2 [Alias]* *[WHERE Bedingungen]* *[GROUP BY Spalte [HAVING Bedingungen]]* *[ORDER BY Spalte1 [ASC | DESC], Spalte2..];*
Erläuterung	*Spalte = Angabe der Spalten, deren Werte abgefragt werden sollen* *Alias = Temporärer Name der Spalte oder der Tabelle* *Tabelle = Angabe der Tabellen, aus denen die Datensätze abgefragt werden* *WHERE Bedingung = dient zum Filtern der Datensätze* *GROUP BY und HAVING = dient zum Gruppieren der Datensätze* *ORDER BY = dient zum Sortieren der Datensätze*

In unserem Beispiel werden für eine einfache SQL-Abfrage „Name", „Vorname" und „Geburtsdatum" aller Personen

abgefragt. Dabei wird immer der Tabellenname vor dem Spaltennamen angegeben. Es kann auch ein Alias für den Tabellennamen vergeben und benutzt werden, um Schreibarbeit zu sparen. Die entsprechende SQL-Anweisung sieht wie folgt aus:

Beispiele	`SELECT Person.Name, Person.Vorname, Person.Geburtsdatum from Person;` `oder alternativ:` `SELECT P.Name, P.Vorname, P.Geburtsdatum FROM Person P;`

Die Ergebnismenge innerhalb einer SELECT-Abfrage kann mit dem Befehl „DISTINCT" zusätzlich reduziert werden, sodass mehrfach auftretende Werte nur einmal angezeigt werden. Mit der SQL-Anweisung

```
SELECT Person.Name FROM Person
```

werden die Namen aller Personen angezeigt. Dabei ist die Wahrscheinlichkeit groß, dass Namen in der Ergebnismenge mehrfach auftreten, z. B. die Namen „Müller", „Maier" oder „Schulz". Wenn man aber nur wissen möchte, welche unterschiedlichen Namen es gibt, dann muss die Ergebnismenge entsprechend eingeschränkt werden. Hier kommt der Befehl „DISTINCT" zum Einsatz. Mit

```
SELECT DISTINCT Person.Name FROM Person
```

werden alle redundanten Namen aus der Ergebnismenge entfernt.

7.5.5 Abfrage mit Bedingungen

In den wenigsten Fällen wird die Anzeige aller Datensätze einer Tabelle angestrebt. Es ist eher so, dass ganz bestimme Datensätze herausgefiltert werden sollen. Hier kommt die WHERE-Klausel zum Einsatz. Mehrere Bedingungen in der WHERE-Klausel werden mithilfe von logischen Operatoren verknüpft, die durch Klammern zu logischen Blöcken zusammengefasst werden können. Im Beispiel werden alle Personen mit ihren Namen aufgelistet, deren Größe über 1,80 m beträgt, deren Gewicht unter 100,00 kg liegt und die „Müller" heißen.

```
SELECT P.Name, P.Vorname FROM Person P
WHERE P.Groesse > 1.80 AND P.Gewicht < 100 AND P-Name = 'Müller';
```

Sollen alle Personen in einem bestimmten Größenbereich (z. B. von 1,60 m bis 1,80 m) ermittelt werden, bestehen zwei Möglichkeiten für die Gestaltung der Abfrage. Die erste Möglichkeit entspricht der Verfahrensweise einer Abfrage mit Bedingungen (siehe vorheriges Beispiel). Die zweite Möglichkeit besteht in einer SQL-Abfrage mit dem **Operator „BETWEEN".**

```
SELECT P.Name, P.Vorname FROM Person P
WHERE P.Groesse BETWEEN 1.60 AND 1.80;
```

Der **Operator „LIKE"** ermöglicht den Vergleich eines Strings mit einem Muster. Das Muster kann ein einzelnes Zeichen oder ein Teilstring sein. Dazu können Platzhalter verwendet werden. Im Beispiel einer SQL-Abfrage mit LIKE werden alle Personen abgefragt, die ein „m" im Namen tragen.

Platzhalter	
%	Prozentzeichen: steht für kein Zeichen, ein Zeichen oder mehrere Zeichen.
_	Unterstrich: repräsentiert immer genau ein Zeichen.

```
SELECT P.Name, P.Vorname FROM Person P WHERE P.Name LIKE '%m%';
```

Weitere Beispiele	
P.Name LIKE 'm%'	Der Name beginnt mit einem „m".
P.Name LIKE '_m%'	Der Name hat an der zweiten Stelle ein „m".
P.Name LIKE '%m'	Der Name endet mit einem „m".
P.Name LIKE 'st_m%'	Der Name beginnt mit „st", dann folgt erst ein beliebiges Zeichen, darauf ein „m", gefolgt von beliebigen Zeichen.

7.5.6 Rechnen in Abfragen

In SQL-Abfragen können auch Berechnungen mit den bereits erwähnten Rechenoperatoren ausgeführt werden. Sollen beispielsweise alle Personen und der dazugehörige Body-Mass-Index (BMI) ausgegeben werden, besteht zunächst das Problem, dass in der Tabelle „Person“ keine Spalte „BMI" mit entsprechenden Werten existiert. Dieses Problem kann aber durch die Berechnung des BMI aus dem Gewicht und der Größe nachfolgender Formel gelöst werden:

BMI = Gewicht / (Groesse * Groesse)

Das Gewicht wird in Kilogramm und die Größe in Metern angegeben. In der SQL-Anweisung für die Berechnung erscheint zusätzlich der vergebene Alias „BMI".

```
SELECT P.Name, P.Vorname, (P.Gewicht / (P.Groesse * P.Groesse)) AS BMI FROM Person P;
```

7.5.7 Sortieren der Ergebnismenge

Das Schlüsselwort „ORDER BY" wird zum Sortieren der Ergebnismenge verwendet. Dabei kann die Reihenfolge der Sortierung angegeben werden. Mit dem Schlüsselwort „ASC" (ASCENDING) wird aufsteigend und mit dem Schlüsselwort „DESC" (DESCENDING) absteigend sortiert. ASC ist standardmäßig voreingestellt und muss nicht angegeben werden. Es ist auch möglich, mehrere Spalten anzugeben, nach denen sortiert werden soll. So kann beispielsweise zuerst nach „Name“ und innerhalb der Namen nach „Groesse" sortiert werden. Im Beispiel der SQL-Abfrage mit Sortierung werden alle Personen absteigend nach Namen ausgegeben.

```
SELECT P.Name, P.Vorname FROM Person P ORDER BY P.Name DESC;
```

7.5.8 Aggregatfunktionen und Gruppen

SQL verfügt über Aggregatfunktionen bzw. arithmetische Funktionen. In der folgenden Übersicht sind diese Funktionen, deren Bedeutung und jeweils ein Beispiel dargestellt.

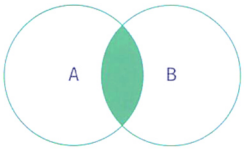
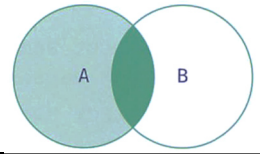
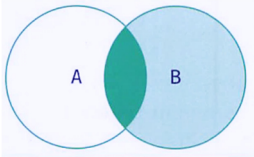
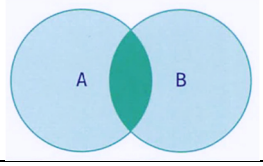
Aggregatfunktion	Bedeutung	Beispiel
COUNT (Spalte)	Liefert die Anzahl der Datensätze, die in der Spalte einen definierten Wert enthalten. Werte, die NULL sind, werden nicht gezählt.	`SELECT COUNT (P.PersonID) FROM Person P`
SUM (Spalte)	Liefert die Summe der Werte der Spalte.	`SELECT SUM (P.Gewicht) FROM Person P`
MIN (Spalte)	Liefert den kleinsten Wert der Spalte.	`SELECT MIN (P.Gewicht) FROM Person P`
MAX (Spalte)	Liefert den größten Wert der Spalte.	`SELECT MAX (P.Groesse) FROM Person P`
AVG (Spalte)	Liefert den Durchschnittswert über alle Werte der Spalte.	`SELECT AVG (P.Groesse) FROM Person P`

In den aufgeführten Beispielen werden die Aggregatfunktionen immer nur auf eine ganze Tabelle angewendet. Daher besteht das Abfrageergebnis immer nur aus einem Datensatz. In SQL ist es aber auch möglich, eine Tabelle in Gruppen „aufzuteilen" d. h., die Datensätze einer Tabelle zu gruppieren und die Aggregatfunktionen jeweils auf die Gruppen anzuwenden. Dazu dient die Anweisung „GROUP BY". Im Beispiel für die SQL-Abfrage mit GROUP BY soll die Durchschnittsgröße von allen Personen gruppiert nach Wohnort ermittelt werden.

Gruppieren von Ergebnismengen	
Syntax	`SELECT .... FROM ... GROUP BY Spalte HAVING Bedingungen;`
Erläuterung	`Spalte = Spalte, nach der gruppiert werden soll` `Bedingungen = Bedingungen, nach denen die Gruppen gefiltert werden`
Beispiel	`SELECT AVG (P-Groesse) FROM Person P GROUP BY P.OrtID;`

7.5.9 Abfragen über mehrere Tabellen

Alle Abfragen bezogen sich bisher nur auf eine Tabelle. In vielen Fällen werden aber Informationen aus mehreren Tabellen benötigt. Dazu müssen die entsprechenden Tabellen in der FROM-Klausel angegeben und der Primärschüssel der einen Tabelle mit dem dazugehörigen Fremdschlüssel der anderen Tabelle gleichgesetzt werden. Das geschieht mithilfe der WHERE-Klausel oder mithilfe des Schlüsselworts „INNER JOIN" in der FROM-Klausel Außerdem gibt es die Schlüsselwörter „LEFT JOIN", „RIGHT JOIN" und „FULL JOIN", die wie INNER JOIN verwendet werden, aber eine unterschiedliche Ergebnismenge liefern.

Schlüsselwort	Beschreibung	Beispiel
INNER JOIN	*Liefert die Werte der linken Tabelle und die Werte der rechten Tabelle, welche miteinander verknüpft sind.* 	`SELECT ...` `FROM TabelleA A INNER` `JOIN TabelleB B` `ON A.TabelleBID =` `B.TabelleBID`
LEFT JOIN	*Liefert alle Werte der linken Tabelle und alle Werte der rechten Tabelle, die eine Verknüp fung mit der linken Tabelle besitzen.* 	`SELECT ...` `FROM TabelleA A LEFT` `JOIN TabelleB B` `ON A.TabelleBID =` `B.TabelleBID`
RIGHT JOIN	*Liefert alle Werte der rechten Tabelle und alle Werte der linken Tabelle, die eine Verknüpfung mit der rechten Tabelle besitzen.* 	`SELECT ...` `FROM TabelleA A RIGHT` `JOIN TabelleB B` `ON A.TabelleBID` `= B.TabelleBID`
FULL JOIN	*Liefert alle Werte der rechten und der linken Tabelle, unabhängig davon, ob eine Verknüpfung besteht oder nicht.* 	`SELECT ....` `FROM TabelleA A FULL` `JOIN TabelleB B` `ON A.TabelleBID =` `B.TabelleBID`

Im Beispiel einer SQL-Abfrage über zwei Tabellen sollen alle Personen ausgegeben werden, die in Hamburg wohnen. Die Verknüpfung der Tabellen erfolgt in der WHERE-Klausel

Beispiel ohne INNER JOIN:

```
SELECT P.Name, P.Vorname, O.Name FROM Person P, Wohnort O WHERE O.OrtID = P.OrtID AND
O.Name = 'Hamburg';
```

Die gleiche Ergebnismenge erhält man auch mithilfe von INNER JOIN.

Beispiel mit INNER JOIN

```
SELECT P.Name, P.Vorname, O.Name FROM Person P INNER JOIN Wohnort O ON P.OrtID = O.OrtID
WHERE O.Name = 'Hamburg';
```

7.5.10 Datums- und Zeitfunktionen

Es gibt viele nützliche Funktionen zur Berechnung von Zeitangaben. Welche Funktionen zur Verfügung stehen, hängt sehr stark vom benutzten Datenbanksystem ab. Exemplarisch werden hier nur kurz die SQLite-Funktionen *„date(...)“* *„time(...)"* und *„datetime(...)“* vorgestellt.

Funktion	Beschreibung
date(timestring, modifier,...)	Gibt ein Datum im Format YYYY-MM-DD zurück.
time(timestring, modifier,...)	Gibt eine Zeit im Format HH:MM:SS zurück.
datetime(timestring, *modifier,...)*	Gibt ein Datum und eine Zeit im Format YYYY-MM-DD HH:MM:SS zurück.

```
select date('now');  Gibt das aktuelle Datumzurück.
SELECT date('now','start of month', '+1 month','-1 day'); Gibt den letzten Tag des
aktuellen Monats zurück.
```

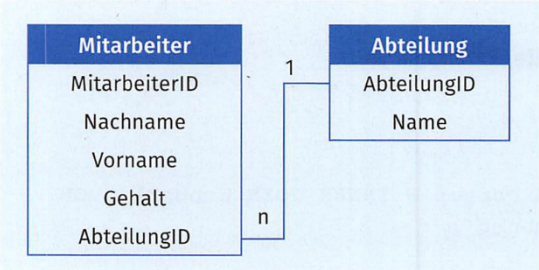
Kompetenzcheck

1. 1 Welche Aussagen sind richtig?

   a. DKL ist ein Bestandteil von SQL.
   b. Mithilfe der Befehle der TCL kann die Datenintegrität sichergestellt werden.
   c. Texte werden mit den Klammern {} gekennzeichnet.
   d. VARCHAR ist der Datentyp für variable Texte.
   e. In SQL-Abfragen kann nicht gerechnet werden.
   f. Abfragen beginnen mit dem Schlüsselwort „GET".
   g. Platzhalter in SQL sind "%" und "_"
   h. COUNT(...) liefert die Anzahl der Datensätze der Spalte, welche einen Wert enthalten.
   i. LEFT JOIN liefert nur die Werte der linken Tabelle.
   j. Mit DELETE FROM... werden immer komplette Datensätze gelöscht.

2. Erstellen Sie zu den folgenden Aufgaben die nötigen SQL-Anweisungen. Die gesamte Aufgabenstellung bezieht sich auf das abgebildete Datenmodell

Es kann eine SQLite-Datenbank „mitarbeiterverwaltung.db" im Downloadbereich heruntergeladen werden. Dort sind die Tabellen mit einigen Datensätzen zur Kontrolle vorhanden. Die Datenbank kann z. B. mit kostenlosen Open-Source-Tool „DB-Browser für SQLite" geöffnet und bearbeitet werden. Damit ist es auch möglich, die SQL-Anweisungen zu testen. Download-Link für diesen DB-Browser: https://sqlitebrowser.org/blog/version-3-12-0-released/



- *Es sollen beide Tabellen angelegt werden. Dabei sind sinnvolle Datentypen zu wählen.*
- *Es soll in die Tabelle „Mitarbeiter" eine Spalte „Wohnort" mit einer Textlänge von 10 eingefügt werden.*
- *Es soll der Datentyp der Spalte „Wohnort" auf eine Textlänge von 50 geändert werden.*
- *Es soll ein neuer Mitarbeiter mit folgenden Daten in die Datenbank aufgenommen werden. (Klaus Müller, Gehalt: 2 500,00 €, Abteilung: noch unbekannt)*
- *Es soll die Abteilung „Einkauf" eingefügt werden.*
- *Es sollen von allen Mitarbeitern der Nachname, der Vorname und das Gehalt ausgegeben werden. Die Liste soll alphabetisch nach Nachnamen aufsteigend sortiert sein.*
- *Es sollen von den Mitarbeitern der Abteilung Buchhaltung der Nachname der Vorname ausgegeben werden.*
- *Es soll eine Namenliste aller Mitarbeiter mit Gehältern zwischen 2 000,00 € und 2 500,00 € ausgegeben werden.*
- *Es soll eine Namenliste aller Mitarbeiter ausgegeben werden, deren Nachname mit „Sch" beginnt.*
- *Es soll eine Namenliste aller Mitarbeiter ausgegeben werden, in deren Nachname die Zeichenkette „mann" vorkommt.*

- *Es soll das durchschnittliche Gehalt aller Mitarbeiter angezeigt werden.*
- *Alle Gehälter werden um 1,5% erhöht.*
- *Es sind alle Mitarbeiter zu löschen, für die in der Datenbank kein Nachname angegeben ist.*
- *Es ist die Spalte „Wohnort" in der Tabelle „Mitarbeiter" zu löschen.*
- *Es ist die Tabelle „Abteilung" zu löschen.*

3. Bearbeiten Sie im Handout die Aufgaben 9, 10 und 11 und werten Sie u.a. einen englischen Text zu SQL aus.

7.6 Datenbankabfrage mit C#

8 Software testen und dokumentieren



Das Testen von Software und die Erstellung von Dokumentationen gehören genauso zum Alltag der Softwareentwicklung wie das Programmieren selbst. Auch wenn das Testen und die Dokumentation von Software oftmals nicht die gebührende Aufmerksamkeit erfahren, kann ihre Bedeutung für die Entwicklung und Erweiterung von Software nicht hoch genug ein geschätzt werden. Dabei sollte mit dem Testen und Dokumentieren so früh wie möglich im Softwareprojekt begonnen werden, denn die Korrektur von Fehlern verursacht umso weniger Kosten, je früher diese entdeckt werden. Fehler, welche sich in der Entwicklungsphase in der Regel leicht korrigieren lassen, sind im Live-Betrieb einer Software nur noch mit erheblichem Mehraufwand zu beseitigen. Der Grund, warum Fehler nicht bereits während der Entwicklung entdeckt werden, ist meist ein mangelhafter oder nicht vorhandener Testprozess. Das Gleiche gilt für die Dokumentation von Software im Allgemeinen und für die Erstellung der Entwicklerdokumentation im Speziellen. Je früher damit begonnen wird, desto präziser fällt diese aus. Zum Beispiel kann bei einem nicht vorherzusehenden Ausfall eines Teammitglieds dessen Arbeit schneller und besser nachvollzogen werden, sodass eventuelle Termine nicht in Gefahr geraten. Wie eine Software getestet werden kann und welche Dokumentationen erforderlich sind, damit beschäftigt sich dieses Kapitel.

8.1 Den Testprozess beschreiben

Das Testen von Software verfolgt das Ziel, mit vertretbarem Aufwand an Zeit und Personal möglichst verlässliche Aussagen über das Verhalten von Software zu machen und darüber, ob die gestellten Anforderungen erfüllt werden.

In vielen Fällen wird das Testen auf die funktionalen Anforderungen reduziert, dabei sind die nicht funktionalen Anforderungen genauso wichtig. Allerdings sind diese schwerer zu testen, weil sie stärker subjektiven Einflüssen unterliegen. Die gewonnenen Erkenntnisse aus den einzelnen Tests werden zur Behebung von Fehlern genutzt oder um nicht funktionale Qualitätsanforderungen (siehe Kapitel 5.4.1) zu optimieren. Trotz aller Maßnahmen sind Tests nur als Stichproben anzusehen, welche nicht den vollständigen Nachweis der Korrektheit einer Software erbringen. Das heißt, es kann nicht mit Bestimmtheit gesagt werden, ob die Software fehlerfrei ist oder nicht. Ein Test kann lediglich feststellen, ob bestimmte Testfälle erfolgreich oder nicht erfolgreich waren. Dem begegnet man in der Softwareentwicklung mit einer Reihe von Maßnahmen. Das Testen wird beispielsweise nicht mehr als isolierte Phase im Softwareentwicklungsprozess angesehen, welche erst nach der Implementierung einsetzt, sondern als eine wichtige Maßnahme zur Qualitätssicherung, welche sich durch alle Phasen der Softwareentwicklung zieht. In größeren Softwareprojekten werden zusätzlich manuelle Prüfverfahren eingesetzt, bei denen die Software von unabhängigen Sachverständigen auf die Erfüllung vorgegebener Prüfkriterien hin untersucht wird.

8.1.1 Prozessschritte

Egal, ob eine Software nach einem agilen oder klassischen Vorgehensmodell entwickelt wird: Die Grundlagen für das Testen von Software bleiben dieselben. Folgende Punkte müssen beim Testen von Software geplant und beachtet werden:

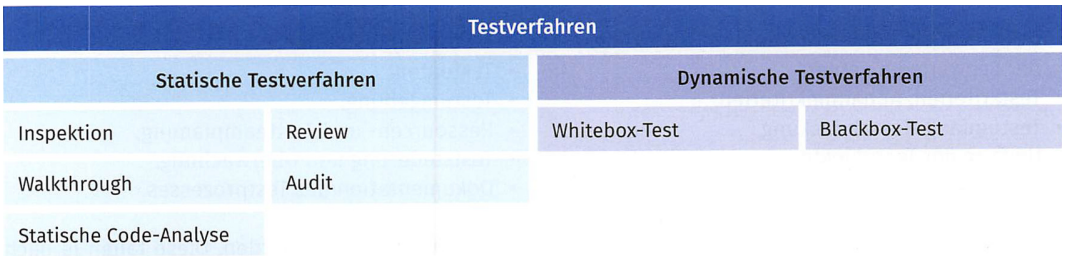
- *Definition der Fehlerklassen und Bewertung der Schwere der Fehler*
- *Testkriterien, Abnahmekriterien*
- *Testumfang, Testabdeckung*
- *Umfang der Testobjekte*
- *Prioritäten im Test, Risikobewertung*
- *Testarten, Teststufen, Testdaten*
- *Testumgebung*
- *Ressourcen-und Testteamplanung*
- *Teststeuerung und Überwachung*
- *Dokumentation des Testprozesses*

Das Testen kann in verschiedene grundlegende Prozessschritte eingeteilt werden. Diese fallen je nach Vorgehensmodell unterschiedlich stark aus, werden anders durchlaufen und werden auch teilweise anders bezeichnet, aber bleiben im Kern identisch.

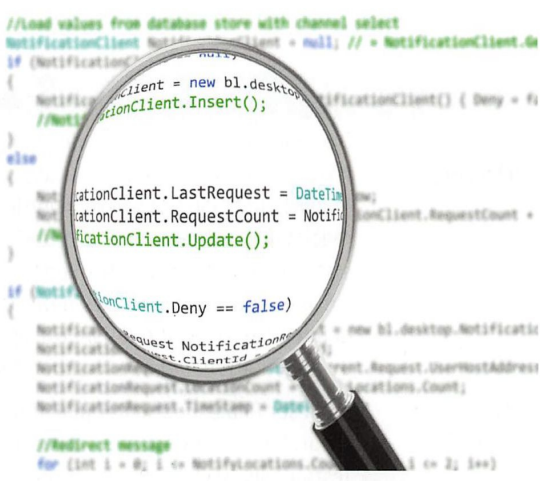
Prozessschritt	Beschreibung
Testplanung	*Die Testplanung beschreibt den vollständigen Testprozess des jeweiligen Softwareprojektes. Der Testplan sollte so früh wie möglich im Prozess der Softwareentwicklung erstellt werden. Allerdings muss er dann während des Projektablaufes regelmäßig angepasst werden, sodass die Einzeltests im Umfang zweckmäßig und effizient ausgeführt werden können. Inhalte des Testplans sind u.a. die Teststrategie, die Testziele, die Kriterien für den Testbeginn und das Testende, die zu verwendenden Testarten, die Hilfsmittel und Werkzeuge zum Testen, die Richtlinien zur Dokumentation und alle Angaben zur Testorganisation.*
Testentwurf	*In dieser Phase betrachtet man die Details der einzelnen Testobjekte und entwirft dafür die einzelnen Tests. Dazu werden die Testziele der Testplanung in konkrete Testfälle übersetzt und eine Testspezifikation angefertigt. Aktivitäten beim Testentwurf sind u.a. die Testfallfindung für die einzelnen Testobjekte, das Festlegen der Testabläufe und deren Testreihenfolge, das Bestimmen der Testdaten und ob es sich um manuelle oder automatisierte Tests handeln soll.*
Testvorbereitung	*Aufbauend auf der Testspezifikation werden die dort festgelegten Testszenarien vorbereitet und die benötigten Ressourcen zur Verfügung gestellt. Dazu zählen z. B. die Auswahl der durchzuführenden Testfälle, das Aufbauen der Testumgebung, das Vorbereiten und Bereitstellen der Testdaten, das Bereitstellen aller notwendigen Dokumente.*
Testdurchführung	*Alle Tests werden auf Grundlage der Testspezifikation durchgeführt und dokumentiert. Bei dynamischen Tests wird dazu das zu testende Programm ausgeführt und bei statischen Tests einer analytischen Prüfung unterzogen. Dabei ist eine Empfehlung, dass ein Testobjekt nicht vom Entwickler selbst, sondern von anderen unabhängigen Personen getestet wird.*
Testauswertung	*In diesem Schritt wird jedes Testobjekt einzeln bewertet, die Ergebnisse je Testfall überprüft und eine Liste aller gefundenen Abweichungen erstellt. Dazu wird das Testergebnis mit dem erwarteten Ergebnis verglichen und anschließend eine Entscheidung über das Testergebnis gefällt. Nach dem Okay gilt der Testfall als erledigt. Ansonsten wird der Fehler beschrieben und der Testfall bleibt offen. Am Ende bekommt jedes einzelne Testobjekt eine Empfehlung über die Freigabe.*
Testabschluss	*Nach Freigabe der gesamten Software kann der Testprozess abgeschlossen werden. Alle Testdokumente werden vervollständigt und im Anschluss archiviert.*

8.1.2 Testverfahren

Testverfahren geben an, wie etwas getestet wird. Bei den Testverfahren wird grundsätzlich zwischen statischen und dynamischen Verfahren unterschieden.



Statische Testverfahren dienen hauptsächlich dem Testen von Software, ohne diese auf einem Rechner auszuführen. Bei den Prüfobjekten handelt es sich meistens um Entwicklungs dokumente oder um den Quellcode selbst. Prinzipiell kann aber jedes Arbeitsergebnis der Softwareentwicklung durch statische Test techniken überprüft werden, z. B. die Anforde rungsspezifikation, die Testspezifikation oder das Benutzerhandbuch für den Kunden.



Zu den statischen Testverfahren gehören Inspektion, Review, Walkthrough und Audit. Diese Verfahren sind häufig Bestandteil formalisierter Qualitätsmanagementansätze im Projektalltag. Sie werden in fast allen Phasen einer Entwicklung angewendet. Teammitglieder, externe Spezialisten oder auch Benutzer überprüfen dabei anhand einer aufgestellten Checkliste oder spontan die Korrektheit des erstellten Programms.

Testverfahren	Beschreibung
Inspektion	Stark formalisiertes Verfahren, bei dem der Programmcode nach genau festgelegter Vorgehensweise durch ein Gutachterteam untersucht wird. Nach empirischen Untersuchungen können durch Inspektionen 50% bis 75% aller Entwurfsfehler gefunden werden.
Review	Ist im Vergleich zur Inspektion weniger formal und durch weniger Aufwand gekennzeichnet. Das Verfahren bewirkt einen ähnlichen Nutzen, d.h., 60% bis 70% aller Fehler werden gefunden.
Walkthrough	Unstrukturierte Vorgehensweise; der Entwickler präsentiert sein Programm und ein oder mehrere Gutachter stellen spontan Fragen.
Audit	Strukturiertere Vorgehensweise als bei Walkthrough; der Entwickler präsentiert sein Programm und der oder die Gutachter stellen Fragen nach einem vorher festgelegten Prüfkatalog.
Statische Code-Analyse	Eine statische Code-Analyse kann beispielsweise bezüglich der Architektur, dem Design oder dem einheitlichen Einsatz der Programmiersprache und dem Ein halten von Namenskonventionen durchgeführt werden. Hier werden mehr die nicht funktionalen Aspekte des Programms beachtet. In aller Regel erfolgt die Überprüfung auch durch Werkzeugunterstützung.

Die statischen Testverfahren haben sowohl Vor- als auch Nachteile:

Statische Testverfahren	
Vorteile	**Nachteile**
➢ *Frühe Fehlererkennung* ➢ *Fehlerzustände werden direkt aufgedeckt* ➢ *Verringert die Anzahl aufwendiger und teurer dynamischer Testtechniken* ➢ *Prüft auch auf Programmierkonventionen, Standards, Design* ➢ *Prüfung erfasst auch nicht funktionale Qualitätskriterien wie die Wartbarkeit*	➢ *Weniger geeignet für Integrations- und Systemtests* ➢ *Fehler, welche nur bei der Programmausführung entstehen, z. B. Laufzeitfehler, werden nicht gefunden*

Dynamische Testverfahren dienen dem Auffinden von Fehlern durch Ausführen des Programms in einer Testumgebung. Die wichtigsten dynamischen Testverfahren sind der Whitebox- und der Blackbox-Test.

Testverfahren	Beschreibung

Whitebox-Test	Dieses Testverfahren zählt zu den am meisten angewendeten dynamischen Test verfahren. Die Tests werden hierbei unter Kenntnis des Quellcodes und der Softwarestruktur entwickelt und durchgeführt. Diese Testtechnik wird deshalb auch als strukturbasiertes oder strukturelles Testverfahren bezeichnet. Die Verfahren für das strukturbasierte Testen beruhen entweder auf der Überdeckung des Kontrollflusses oder des Datenflusses. Beim kontrollflussorientierten Ansatz werden Kontrollstrukturen wie Anweisungen, Verzweigungen oder Bedingungen genutzt, um Testziele zu definieren. Zu den bekanntesten Verfahren aus diesem Bereich gehören: ➢ *Anweisungsüberdeckung,* ➢ *Zweigüberdeckung,* ➢ *Pfadüberdeckung,* ➢ *einfache und mehrfache Bedingungsüberdeckung.* Allen diesen Verfahren ist gemeinsam, dass mit einer bestimmten Anzahl von Testfällen alle vorhandenen Kontrollstrukturen und deren Pfade ausgeführt werden sollen. Der jeweilige Überdeckungsgrad muss dann unter Zuhilfenahme von Werkzeugen gemessen werden. Whitebox-Tests finden vorwiegend bei Unit- und Komponententests Anwendung.
Blackbox-Test	Der Blackbox-Test gilt als Standardtest in der Softwareentwicklung. Bei diesem Verfahren wird das Verhalten des Testobjektes überprüft, ohne dass Softwarestruktur oder Quellcode bekannt sind. Die Testfälle werden auf Basis der Softwarespezifikation entworfen und entsprechende Testdaten abgeleitet. Beim eigentlichen Testen werden die realen Testergebnisse mit den erwarteten Ergebnissen verglichen und ausgewertet. Das Blackbox-Verfahren kann in allen Teststufen zum Einsatz kommen. Beim Komponenten- und Integrationstest ist aber eine entsprechende Testumgebung notwendig. Es ist relativ aufwendig, alle Testfälle für den Blackbox-Test aus der Softwarespezifikation abzuleiten und teilweise sogar unmöglich, weshalb komplette Blackbox-Tests manchmal wirtschaftlich nicht vertretbar sind. Um die Anzahl der Testfälle gezielt zu verkleinern, gibt es verschiedene Methoden, z. B. ➢ *Äquivalenzklassenbildung,* ➢ *Grenzwertanalyse.*

Auch bei den dynamischen Testverfahren gibt es Vor- und Nachteile.

Vorteile	**Nachteile**
➢ *Programm wird tatsächlich ausgeführt; so können u.a. auch Laufzeitfehler gefunden werden* ➢ *Zusammenspiel von einzelnen Systemteilen kann gut getestet werden* ➢ *Testen nicht funktionaler Qualitätsanforderungen wie das Zeitverhalten oder der Ressourcen verbrauch möglich*	➢ *Lauffähige Testumgebung ist Voraussetzung; muss manchmal erst aufwendig geschaffen werden* ➢ *Testen nur jener Programmteile, welche auch ausgeführt werden* ➢ *Nur das Fehlverhalten als solches wird aufgedeckt; der Grund dafür muss später genau spezifiziert werden, z. B. durch Debugging*

8.1.3 Testarten

Testarten geben an, was getestet werden soll. Dabei wird zwischen funktionalen und nicht funktionalen Tests unterschieden. Diese Tests beziehen sich in der Regel auf die Überprüfung der funktionalen und nicht funktionalen Anforderungen an die Software. Die Grundlage dafür sind die Qualitätsmerkmale der ISO/IEC 9126.

Zu testendes Qualitätsmerkmal	**Beschreibung von Testarten**

Funktionalität	Funktionstests beziehen sich darauf, ob die Software die in der Anforderungsspezifikation gestellten funktionalen Anforderungen erfüllt. **Testbereiche** ➢ *Richtigkeit: Fähigkeit, vereinbarte Ergebnisse oder Wirkungen zu liefern* ➢ *Angemessenheit: Eignung der Funktionen für die festgelegten Aufgaben* ➢ *Interoperabilität: Fähigkeit, mit den vorgegebenen Systemen zusammenzuwirken* ➢ *Ordnungsmäßigkeit: Erfüllung anwendungsspezifischer Normen, Vereinbarungen und gesetzlicher Bestimmungen* ➢ *Sicherheit: Fähigkeit, einen unberechtigten Zugriff auf das Programm und Daten zu verhindern*
Zuverlässigkeit	Zuverlässigkeitstests prüfen, ob das Testobjekt ein bestimmtes Leistungsniveau unter bestimmten Bedingungen über einen bestimmten Zeitraum aufrechterhalten kann. Hierzu gehören Robustheitstests sowie Tests, die prüfen, wie schnell das System im Fehlerfall wieder einsatzfähig ist (Recovery-Tests). Auch Stresstests fallen in diesen Bereich: Sie beobachten das Systemverhalten bei Überlastung. **Testbereiche** ➢ *Reife: Häufigkeit des Versagens durch Fehlerzustände* ➢ *Fehlertoleranz: Fähigkeit, ein spezifiziertes Leistungsniveau bei Softwarefehlern oder Nichteinhaltung der spezifizierten Schnittstellen zu bewahren* ➢ *Wiederherstellbarkeit: Fähigkeit, bei einem Versagen das Leistungsniveau wiederherzustellen und die direkt betroffenen Daten wiederzugewinnen*
Benutzbarkeit	Benutzbarkeitstests (Usability-Tests) prüfen typischerweise, wie gut und wie einfach ein System für die Benutzer zu bedienen ist. **Testbereiche** ➢ *Verständlichkeit: Aufwand für den Benutzer, das Konzept und die Anwendung zu verstehen* ➢ *Erlernbarkeit: Aufwand für den Benutzer, die Anwendung zu erlernen (z. B. Bedienung, Ein- und Ausgabe)* ➢ *Bedienbarkeit: Aufwand für den Benutzer, die Anwendung zu bedienen*

Zu testendes Qualitätsmerkmal	Beschreibung von Testarten
Effizienz	Effizienz wird durch Performance-Tests (z. B. Testen von Antwortzeiten oder der Verarbeitungsgeschwindigkeit) und Lasttests (z. B. Systemauslastung bei vielen gleichzeitigen Zugriffen) überprüft. **Testbereiche** ➢ *Zeitverhalten: Antwort- und Verarbeitungszeiten sowie Durchsatz bei der Funktionsausführung* ➢ *Verbrauchsverhalten: Anzahl und Umfang der benötigten Betriebsmittel für die Erfüllung der Funktionen*
Änderbarkeit/ Wartbarkeit	Dieses Qualitätsmerkmal wird typischerweise durch Reviews und werkzeuggestützte statische Code-Analyse geprüft (Wartbarkeitstests). **Testbereiche** ➢ *Analysierbarkeit: Aufwand, um Mängel oder Ursachen von einem Versagen zu diagnostizieren oder um änderungsbedürftige Teile zu bestimmen* ➢ *Modifizierbarkeit: Aufwand zur Ausführung von Verbesserungen, zur Fehlerbeseitigung oder Anpassung an Veränderungen in der Einsatzumgebung* ➢ *Stabilität: Wahrscheinlichkeit des Auftretens unerwarteter Wirkungen von Änderungen* ➢ *Prüfbarkeit: Aufwand, der zur Prüfung der geänderten Software notwendig ist*
Übertragbarkeit	Durch Portabilitätstests wird geprüft, wie leicht sich die Software in eine andere Umgebung übertragen lässt. **Testbereiche** ➢ *Anpassbarkeit: Besteht die Möglichkeit, Software an verschiedene, festgelegte Umgebungen anzupassen?* ➢ *Installierbarkeit: Aufwand, der zum Installieren der Software in einer fest gelegten Umgebung notwendig ist* ➢ *Austauschbarkeit: Möglichkeit, diese Software anstelle einer anderen Software zu verwenden, sowie der dafür notwendige Aufwand*

8.1.4 Teststufen

Die geplanten Tests werden in einem Softwareprojekt in der Regel nicht zur gleichen Zeit durchgeführt, sondern sie erfolgen in Abhängigkeit vom Entwicklungsstand der zu entwickelnden Software. Dabei werden je nach Entwicklungsstand verschiedene Aspekte der Software getestet.

Test	Beschreibung
Modul-, Unit-, Komponententest	Test einer einzelnen Einheit oder Teile dieser Einheit auf Funktionalität
Integrationstest	Überprüfung des fehlerfreien Zusammenwirkens voneinander abhängiger Systemkomponenten; Testschwerpunkt sind die Schnittstellen zwischen den Komponenten
Systemtest	Abschließender Test des Gesamtsystems hinsichtlich der funktionellen und nicht funktionalen Anforderungen, welche zu erfüllen sind
Abnahmetest	Test des Gesamtsystems in der Kundenumgebung; wird in der Regel durch den Kunden selbst oder mit dessen Beteiligung durchgeführt; dieser Test bildet die Grundlage für die Entscheidung zur Abnahme des Projektes



Eine weitere Einteilung von Softwaretests richtet sich nach den Softwareversionen, welche den Entwicklungstand der Software widerspiegeln. Solche Versionen sind beispielsweise die Alpha- und die Betaversion einer Software oder der Release Candidate. Dementsprechend gibt es auch entsprechend benannte Tests wie den Alpha- oder den Betatest.

Der Alphatest hängt somit mit der gleichnamigen Alphaphase der Software zusammen, ebenso wie es eine Betaphase und Betatests gibt. Beide Phasen sind wichtige Abschnitte in der Entwicklung eines Softwareprojekts, unterscheiden sich jedoch in einem wichtigen Aspekt. Alphatests werden meistens noch „intern" von den jeweiligen Entwicklern und einem kleinen Kreis ausgewählter externer Tester durchgeführt, während die verschiedenen Betatests unter realitätsnahen Bedingungen durch mögliche spätere Benutzer durchgeführt werden. Diese Art von Tests betrifft insbesondere Standardsoftware, z. B. Spiele, die einem bestimmten Personenkreis zum Test zur Verfügung gestellt werden.

Test	Beschreibung
Alphatest	Der Alphatest ist der erste Test, welcher nicht nur durch die Entwickler allein durchgeführt wird. Getestet wird die sog. Alphaversion der Software. Obwohl dieser Begriff nicht genau bestimmt ist, enthält die Alphaversion in der Regel bereits die wesentlichen Bestandteile des Softwareprodukts, muss aber noch um viele Funktionen erweitert werden. Durch den frühen Beginn des Testens können schon rechtzeitig grundlegende Fehler gefunden und beseitigt werden. Die Alphaversion wird zum Testen, neben den Entwicklern, meist einem kleinen exklusiven Kreis von externen Testern zur Verfügung gestellt.
Betatest	Ein Betatest ist normalerweise die zweite große Testphase bei der Entwicklung von Software. Während eines Betatests wird das neue Produkt einem Teil der Zielgruppe vorgeführt und zum Testen überlassen. In der Betaversion sind schon alle Funktionen des Programms implementiert. Diese wurden aber noch nicht ausreichend genug getestet. Das Programm kann daher noch viele Fehler enthalten, die einen produktiven Einsatz nicht empfehlenswert machen.
Release-Candidate	Ein Release Candidate ist die abschließende Testversion der Software. Darin sind alle Funktionen schon verfügbar und die meisten Fehler sind behoben. Aus dem Release Candidate wird die endgültige Version erstellt, um einen abschließenden Systemtest durchzuführen.

8.1.5 Testdokumentation

Die einzelnen Testschritte und deren Ergebnisse müssen protokolliert und dokumentiert werden. Dazu hat das IEEE einen Standard veröffentlicht, nach dem man sich richten kann. Im IEEE 829 (Standard for Software Test Documentation) wurden der Inhalt und die Form von acht Basisdokumenten beschrieben. Dieser Standard wurde dann durch die ISO/IEC/IEEE 29119 ersetzt. Es müssen nicht alle acht Dokumente für die Dokumentation des Testprozesses verwendet werden. Welche Dokumente Anwendung finden, hängt auch ganz entscheidend von Art und Größe des Softwareprojektes ab. Die Dokumente lassen sich in drei große Kategorien einteilen, welche hier nur kurz überblicksmäßig dargestellt werden.

Kategorie	Dokumente
Übersicht	➢ • *Testkonzept*
Testspezifikation	➢ • *Testentwurfsspezifikation* ➢ • *Testfallspezifikation* ➢ • *Testablaufspezifikation*
Testbericht	➢ • *Testobjektübergabebericht* ➢ • *Testprotokoll* ➢ • *Testabweichungsbericht* ➢ • *Testabschlussbericht*

Das erste Dokument, welches Auszubildende erstellen oder bearbeiten müssen, ist das Testprotokoll, welches zu den Testberichten gehört. In einem Testprotokoll werden alle Ergebnisse eines Tests festgehalten, die während des Testens auftraten. Das heißt, der Tester führt die in den Testfällen definierten Aufgaben durch und dokumentiert die Ergebnisse. Das kann im einfachsten Fall schlicht ein „OK" oder „Fehler" sein. In der Praxis wird es oftmals aber auch eine Beschreibung sein, was im Fehlerfall die genaue Fehlermeldung bzw. das exakte Fehlverhalten war. Im Folgenden ist ausschnittsweise ein Testprotokoll für das Testen der Korrektheit eines Datums dargestellt.

Testprotokoll Nr. 1

Datum: 03.05.20.. Tester: Karin Mäuser

Uhrzeit: 11:00 Uhr Aufsicht: Felix Schmidt

Verwendetes Testszenario: Testszenario 1

Eingabe eines Datums. Wenn das Datum korrekt ist, dann soll dieses angezeigt werden. Ansonsten wird auf die fehlerhafte Eingabe hingewiesen.

Beim Test bitte ausfüllen:

Nr.	Testdaten	Erwartete Ausgabe	Ergebnis	Bemerkungen
1	10.03.2020	10.03.2020	Ok	
2	29.02.1900	Datum inkorrekt	Fehler	Datum wird angezeigt
3	29.02.2000	29.02.2000	Ok	
4	abc	fehlerhafte Eingabe	Ok	
5	0.13.1000	Datum inkorrekt	Ok	
6	...	...	...	
...				

Anzahl der positiv verlaufenden Tests: 4

Anzahl der negativ verlaufenden Tests: 1

Notwendige Korrektur: Schaltjahre werden nicht richtig erkannt.

Karin Mäuser — Unterschrift des Testers

Felix Schmidt — Unterschrift der Aufsicht

Kompetenzcheck

1. Welche Aussagen sind richtig?

a. Mit Softwaretests sollen Aussagen darüber getroffen werden, ob die spezifizierten Anforderungen erfüllt wurden oder nicht.
b. Durch Umfangreiche Tests kann sichergestellt werden, dass die Software keine Fehler mehr enthält.
c. Zum Testen muss die Software in jedem Fall ausgeführt werden.
d. Durch Testen wird die Qualität von Software erhöht.
e. Bei manchen Vorgehensmodellen müssen keine Testdaten bestimmt werden.
f. Der Testprozess lässt sich grob in die Schritte Testplanung, Testentwurf, Testvorbereitung, Testdurchführung, Testauswertung und Testabschluss einteilen.
g. Am Ende des Testentwurfes wird eine Testspezifikation angefertigt.
h. Die Testverfahren lassen sich in dynamische und in statische Testverfahren einteilen.
i. Beim Review wird der Quellcode von einem Gutachterteam durchsucht, der Programmierer ist aber nicht dabei.
j. Beim Audit stellt der Programmierer sein Programm vor und die Gutachter stellen ohne weitere Vorbereitung Fragen.
k. Beim Whitebox-Test sind die Struktur des Programms und der Quellcode bekannt.
l. Beim Blackbox-Test werden sämtliche Programmpfade getestet.

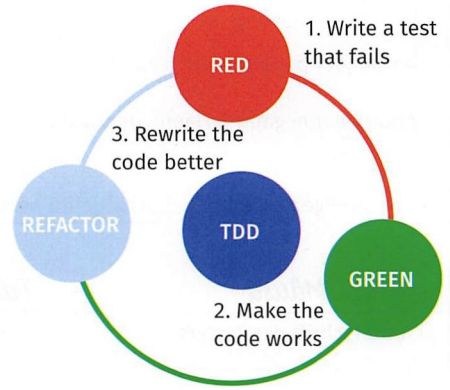
m. Die Effizienz eines Programms kann durch Performancetests überprüft werden.
n. Die Analysierbarkeit und die Modifizierbarkeit eines Programms werden im Rahmen der nicht funktionalen Anforderungen getestet.
o. Beim Integrationstest wird überprüft, inwieweit sich fremde Bibliotheken in eine Klasse integrieren lassen.
p. Der Abnahmetest findet am Ende des Projektes zusammen mit dem Auftraggeber statt.
q. Zum Alphatest werden sehr viele externe Tester hinzugezogen.
r. Das fertige Testprotokoll ist fester Bestandteil der Testspezifikation.
s. Der Testbericht enthält auch Angaben zum Abschluss des Testprozesses.

2. Überprüfen Sie u. a. Ihr Wissen über das Testen von Software und führen Sie Softwaretests durch, indem Sie die Aufgaben 13, 14 und 15 im Handout bearbeiten.

3. Bearbeiten Sie die Aufgabe 16 im Handout und führen Sie einen Walkthrough durch. Die Grundlage dafür bietet die Datenbankanwendung aus dem letzten Kapitel.

8.2 Testgetriebene Entwicklung erläutern

Bei der testgetriebenen Entwicklung (Test Driven Development - TDD) wird das Testen der Software dazu verwendet, den gesamten Prozess der Softwareentwicklung zu steuern. TDD ist also mehr eine Designstrategie als eine Teststrategie, welche das Testen vor dem Erstellen des eigentlichen Quellcodes ansiedelt.



Das Ziel besteht darin, die Qualität der Software zu erhöhen und den Wartungsaufwand zu verringern. TDD wird meistens im Rahmen agiler Vorgehensmodelle verwendet, kommt aber in abgewandelter Form auch immer häufiger bei klassischen Vorgehensmodellen zum Einsatz. Durch das frühe Erstellen der Testfälle erhalten Entwickler schon sehr zeitig ein Feedback zur implementierten Lösung. So können Probleme rechtzeitig erkannt werden. Im Gegensatz zum nachgeschalteten Testen, welches noch häufig bei den klassischen Vorgehensmodellen wie dem Wasserfallmodell eingesetzt wird, sind die Testfälle im TDD der Ausgangspunkt für folgendes Vorgehen:

Schritte der Testgetriebenen Softwareentwicklung	
Schritt	**Beschreibung**
Schreiben eines Tests	Zuerst wird ein Test geschrieben, welcher natürlich meistens fehlschlägt. Es ist ein Grundprinzip der TDD, die einzelnen Tests noch vor der eigentlichen Programmierung der Komponenten zu schreiben (Test First).
Implementierung des Programmcodes	Danach wird genauso viel an Programmcode geschrieben, dass der Test erfolgreich durchlaufen wird. Dabei wird erst einmal die einfachste Lösung implementiert, durch die der Test erfüllt werden kann.
Refaktorisierung der Lösung	Zum Schluss wird der Programmcode nochmals überarbeitet und aufgeräumt, um eventuelle Redundanzen zu vermeiden und die Struktur zu verbessern. Dabei wird das Prinzip des Programmes als solches aber nicht mehr verändert.

Dieses Vorgehen wird solange wiederholt, bis alle Anforderungen an die Software erfüllt werden und der Code in das Produktivsystem übertragen werden kann. Die testgetriebene Entwicklung besitzt sowohl Vor- als auch Nachteile, welche in der folgenden Tabelle aufgeführt sind.

Testgetriebene Softwareentwicklung	
Vorteile	**Nachteile**
➢ *Software auf qualitativ hochwertigem Niveau* ➢ *Software weniger wartungsintensiv und fehleranfällig* ➢ *Fehleranalyse und eventuelle Wartungsarbeiten sind einfacher und schneller durchzuführen* ➢ *Sowohl Code als auch Systemarchitektur sind sauber strukturiert und klar verständlich* ➢ *Effektive Vermeidung von Redundanzen und nicht benötigten Codebestandteile*	➢ *Setzt nur auf der Ebene der Komponenten und Units an; Integrations- und Systemtest nicht berücksichtigt* ➢ *Viele kleine Einzeltests, welche eventuell im Entwicklungsprozess der Software ihren Sinn verlieren, aber trotzdem nicht entfernt werden* ➢ *Kein ganzheitlicher Blick, da immer nur kleine Teilausschnitte betrachtet werden* ➢ *Entwicklung einer guten Systemarchitektur schwierig*

Kompetenzcheck

1. Welche Aussagen sind richtig?

a. Bei der TDD werden die Tests vor der eigentlichen Implementation des Programmcodes geschrieben.
b. Bei der Programmierung versucht man dann, den Test so einfach wie möglich zu erfüllen.
c. In der Refaktorisierungsphase wird der Test an das Programm angepasst.
d. Ein Vorteil der TDD ist, dass man viele kleine Einzeltests erhält.
e. Ein weiterer Vorteil ist, dass Code-Redundanzen vermieden werden.
f. Die Entwicklung einer guten Systemarchitektur ist mit dieser Vorgehensweise schwierig.

8.3 Softwaredokumentationen erstellen

Die Softwaredokumentation hat die Aufgabe, digitale Prozesse verständlich zu erläutern. Anwendungen sollen möglichst umfassend und einfach er klärt werden. Da es verschiedene Zielgruppen gibt, fällt die Dokumentation in der Regel sehr umfang reich aus. Besonders bei der Wartung von Software erweist sich eine gute Programmdokumentation als besonders wertvoll. Oft sind die ursprünglichen Entwickler nicht mehr erreichbar oder die Hauptentwicklung liegt schon zu lange zurück. In diesen Fällen erleichtert eine gute Dokumentation das Verständnis der Software und verkürzt wesentlich die notwendige Einarbeitungszeit. Des Weiteren ist ohne eine ausreichende und gute Benutzerdokumentation eine Software meistens schwer nutzbar.



Dokumentation

Unter einer Dokumentation ist die Nutzbarmachung von Informationen zum Zwecke der weiteren Verwendung zu verstehen. Sie dient dazu, diese gezielt auffindbar zu machen. Die enthaltenen Informationen sollten vollständig, übersichtlich, verständlich und inhaltlich korrekt sein. Dokumentationen erfolgen meistens in schriftlicher Form. Jedoch können auch Bilder, Videos oder Tonaufzeichnungen als Dokumentation dienen.

In diesem Zusammenhang muss auch betont werden, dass das Erstellen der Dokumentation nicht eine einzelne, in sich abgeschlossene Phase ist, welche erst nach der eigentlichen Fertigstellung der Software erfolgt, sondern ein kontinuierlicher Prozess, welcher sich über die gesamte Entwicklungszeit erstreckt. Deswegen stehen die nachfolgenden Ausführungen unter der Prämisse, dass jeder Softwareentwicklungs- und Wartungsprozess zugleich auch ein Dokumentationsprozess ist.

Leider ist es in der Praxis oft so, dass der größte Teil der Dokumentation (z. B. Projekt-, System- bzw. Benutzerdokumentation) nicht aktuell, unvollständig und oft schwer zu lesen und zu verstehen ist. Dieses liegt u.a. daran, dass viele Entwickler der Meinung sind, dass eine ausreichende Dokumentation der Arbeitsergebnisse den Arbeitsfortschritt nur verzögert, welches teilweise auch vom Management unterstützt wird, um möglichst schnell zu laufenden Applikationen zu gelangen. Außerdem wird das eigentliche Schreiben der Dokumentation von den

Entwicklern im Vergleich zur Programmierung als monotoner Prozess empfunden. Ein weiterer Punkt ist, dass Dokumentationen von Zeit zu Zeit an neue Entwicklungsstände angepasst werden müssen, wofür meistens die Zeit fehlt.

Für das Schreiben von guten Dokumentationen gibt es einige grundlegende Anforderungen, welche allgemeingültig sind und die Qualität von diesen entscheidend verbessern. Außerdem sollte darüber nachgedacht werden, wer welche Dokumentation verfasst.

8.3.1 Grundlegende Anforderungen

Eine der wichtigsten Anforderungen an eine gute Dokumentation ist die Möglichkeit für die Benutzer, die für sie relevanten Informationen schnell und einfach zu finden. Dazu muss die Dokumentation gut strukturiert und verständlich geschrieben sein. Der schwierigste Teil dabei ist, die Dokumentation so zu verfassen, dass sie auch der Zielgruppe entspricht. Dies gilt insbesondere für die Systemdokumentation. Diese Texte werden gleichermaßen von Systemadministratoren, Programmierern, aber auch von Projekt-Leitern oder dem Auftraggeber gelesen. Wegen unterschiedlicher Vorkenntnisse dieser Gruppen kann es sinnvoll sein, im Rahmen der einzelnen Systemdokumente nochmals zu differenzieren.

Kriterien einer guten Dokumentation

- *Die Informationen werden schnell und einfach gefunden.*
- *Es sind nur die relevanten Informationen enthalten.*
- *Die Informationen sind verständlich dargestellt.*
- *Die Informationen sind zielgruppengerecht aufbereitet.*
- *Die Informationen können praxisnah nachvollzogen werden.*
- *Das Dokument ist schnell zu lesen.*
- *Die Inhalte sind aktuell.*
- *Die Inhalte sind widerspruchsfrei.*
- *Die Inhalte sind vollständig.*

Eine Benutzerdokumentation hingegen muss anderen Ansprüchen genügen. Sie muss die Fragen der Benutzer präzise beantworten und sie befähigen, das Produkt vollständig und effizient zu nutzen. Technische Details, welche die Benutzer nicht zwangsläufig kennen, haben in einer Benutzerdokumentation ebenso wenig verloren wie hochtrabende Phrasen und Schlagwörter aus der Marketingabteilung. Eventuelle Warnhinweise und kritische Einstellungen sollten auf jeden Fall so gestaltet sein, dass sie beim Lesen ins Auge fallen und nicht überlesen werden können. Bei sehr großen Software Projekten kann eine extra Qualitätsprüfung der Dokumente erfolgen. Dabei wird festgestellt, in welchem Ausmaß die Qualitätsanforderungen an die Dokumentation erfüllt sind. Diese Qualitätsprüfung der Dokumentation erfolgt in der Regel in Form von speziellen Dokumentationsaudits (siehe Kapitel 5.8.1, Abschnitt 2, S. 610).

8.3.2 Ersteller

Das Erstellen von Softwaredokumentation wird von Entwicklern oft als lästiges Übel empfunden und zählt nicht zu den Aufgaben, die gerne gemacht werden. Trotzdem wird ein großer Teil der Softwaredokumentation von diesen geschrieben. Dass Entwickler die Dokumentationen schreiben, hat Vor- und Nach teile. Ein Vorteil ist die unmittelbare Beteiligung an der Entwicklung. Damit können Entwickler Dinge beschreiben, die einem Unbeteiligten möglicherweise entgehen. Dieser Vorteil stellt aber in vielen Fällen auch einen Nachteil dar, weil die Entwickler tief in der Materie stecken und einem Außenstehenden die benötigten Informationen nicht in allgemeinverständlicher Form vermitteln können.

Dieses gilt ganz besonders für die Benutzerdokumentation. Um eine gute Benutzerdokumentation zu schreiben, muss man die Sicht des Benutzers einnehmen können und sich von der rein technischen Sichtweise lösen. Dieses fällt nicht allen Entwicklern leicht. Um dieses Problem zu lösen, wird in vielen Fällen die Arbeit zur Erstellung einer Dokumentation zwischen den Entwicklern und technischen Redakteuren/-innen aufgeteilt. Die Entwickler stellen die wichtigen Fakten zusammen und schreiben eine erste Fassung der Dokumentation. Sie konzentrieren sich dabei ausschließlich auf die Inhalte und liefern alle benötigten Informationen. In einem zweiten Schritt bereiten technische Redakteure/-innen die Inhalte so auf, dass daraus eine für den Benutzer verständliche Dokumentation entsteht.

Kompetenzcheck

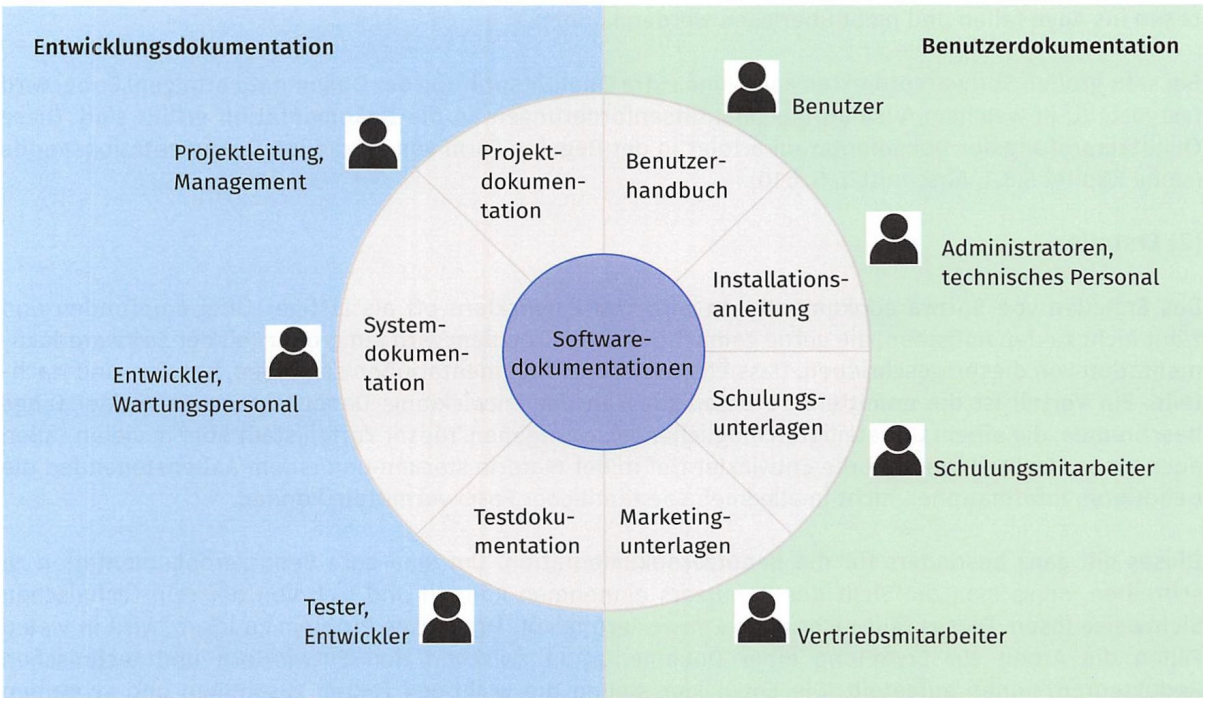
1. Welche Aussagen sind richtig?

   a. Durch eine gute Dokumentation des Quellcodes und des Softwareentwurfes ist es möglich, dass sich neue Entwickler schnell in das Softwareprojekt und den Quellcode einarbeiten.
   b. Jede Dokumentation muss alle verfügbaren Informationen enthalten.
   c. Dokumentationen müssen auf ihre Zielgruppe zugeschnitten sein.
   d. Gute Dokumentationen sind schnell zu lesen.
   e. Dokumentationen müssen nicht aktuell und vollständig sein.
   f. Eine Software sollte immer erst nach ihrer Erstellung dokumentiert werden.
   g. Dokumentationen werden immer von den Entwicklern der Software geschrieben.

2. Bearbeiten Sie im Handout die Aufgaben 17, 18 und 19 und werten u.a. einen englischen Text zur Dokumentation von Software aus.

8.4 Verschiedene Dokumentationsunterlagen unterscheiden

Der Inhalt und damit die Art der Dokumentation richten sich zuerst nach den Zielgruppen. Diese können grob in die Gruppen „Entwickler“ und „Benutzer" eingeteilt werden. Die Entwickler benötigen die Dokumente zur Planung, Durchführung und Absicherung des Entwicklungsprozesses sowie im späteren Lebenszyklus zur Wartung der Software. Die Benutzer müssen die Software installieren und betreiben können. Weitere Benutzerdokumente dienen zum Erlernen und Bedienen der Software. Auch Marketingunterlagen können indirekt zur Benutzerdokumentation



einer Software gezählt werden.

8.4.1 Entwicklungsdokumentation

Nicht nur die Benutzer erwarten gut dokumentierte Software. Auch Softwareentwickler und die Projektleitung benötigen sie, um ihre Aufgaben und Ziele zu realisieren. Wird ein Softwaresystem entwickelt, wird es in der Regel auch dokumentiert. Die Entwicklungsdokumentation dient im Wesentlichen dazu, den Entwicklungsprozess nachvollziehbar zu dokumentieren und Systemdetails zu beschreiben, um die Software ohne Probleme auch nach Jahren beispielsweise warten zu können.

Dokument	Beschreibung
Projektdokumentation	Hier wird die Planung und Leitung des Projektes festgehalten, z. B. der Projektauftrag, der Projektplan, die Projektstatusberichte und der Projektabschlussbericht.
Systemdokumentation	Hierzu zählen alle Dokumente, welche für die Konstruktion und die Wartung der Software benötigt werden. Inhalte der Dokumente sind z.B.: ➢ *Anforderungsspezifikation (Lasten- und Pflichtenheft),* ➢ *Abnahmespezifikation* ➢ *Vorgehensmodell,* ➢ *Entwurf der Software (Systemarchitektur, spezielle Algorithmen, Entwurfsmuster, Schnittstellen USW.),* ➢ *Fragen der Verwendung und der Verwaltung von Daten (Datenmodelle, verwendete Dateien, Zugriffsrechte usw.),* ➢ *Fragen des Datenschutzes und der Datensicherheit,* ➢ *kommentierter Quellcode,* ➢ *Programmversionen, Versionsverwaltung.* Zur Dokumentation werden bevorzugt Beschreibungs- und Modellierungssprachen wie UML-Diagramme, Struktogramme, ER-Modelle usw. eingesetzt (siehe Kapitel 5.4.4).
Testdokumentation	Hier werden alle Maßnahmen der Qualitätssicherung dokumentiert. Dieses betrifft die Testplanung, die Testspezifikation sowie die Testberichte. Inhalte können u.a. sein: Audit-Protokolle, Test- und Review-Berichte, der Abnahmebericht, Testfälle, Verfahren und Szenarien, mit denen die Software überprüft wurde.

8.4.2 Benutzerdokumentation

Dieser Teil der Dokumentation richtet sich an die Kunden, die die Software kaufen oder lizensieren. Nach dem sie als potenzielle Kunden von der Leistung und Qualität des Produktes überzeugt sind, müssen sie die Software meistens in Eigenregie installieren und nutzbar machen, um sie benutzen zu können. Dazu werden Installationsanweisungen benötigt, bei größeren Softwaresystemen sogar speziell für Administratoren erstellte Administrationshandbücher. Die eigentlichen Benutzer benötigen in der Regel zur leichteren Einarbeitung in die Software ein Benutzerhandbuch. Bei Software mit größerem Funktionsumfang werden sehr oft Schulungen angeboten und auch von den Kunden eingefordert. Dazu sind neben dem Benutzerhandbuch auch speziell aufbereitete Schulungsunterlagen erforderlich. Die wirtschaftlichen Entscheider bzw. das Management bei potenziellen Kunden müssen auf der anderen Seite mit überzeugenden Marketingunterlagen versorgt werden.

Dokument	Beschreibung
Benutzerhandbuch	Ist eine Sammlung von Informationen, welche dem Benutzer einen sicheren und ordnungsgemäßen Umgang mit der Software ermöglichen, wobei das Benutzerhandbuch in vielen Fällen nicht mehr in gedruckter Form zur Verfügung gestellt wird, sondern in Form von Online-Hilfen. **Inhalte sind zum Beispiel:** ➢ *Funktionsumfang der Software,* ➢ *konkrete Beschreibung der Funktion der einzelnen Oberflächenelemente (Menü, Buttons usw.),* ➢ *praxisnahe Beispiele,* ➢ *Verhalten bei eventuell auftretenden Fehlern der Software, mögliche Fehlermeldungen,*
Installationsanleitung	Liefert eine Beschreibung der erforderlichen Hardware und Software, um das Softwareprodukt auszuführen und eine Anleitung für die Installation von diesem auf dem System. **Inhalte könnten sein:** ➢ *benötigte Betriebssysteme und deren Versionen,* ➢ *vorausgesetzte Software-Umgebungen, z. B. Standardbibliotheken oder Laufzeitsysteme,* ➢ *Erläuterung der Verfahrensweisen zur Installation und Deinstallation der Software,* ➢ *Verfahrensweisen, um die Software zu aktualisieren (Updates).*
Schulungsunterlagen	Speziell für die Schulung von Kunden erstellte Unterlagen; ergänzen die Inhalte des Benutzerhandbuches um didaktische und methodische Ansätze zur Vermittlung dieser Inhalte und entsprechende Übungsaufgaben.
Marketingunterlagen	Hier werden alle Dokumente zusammengefasst, welche im Zusammenhang mit der Vermarktung des Produktes stehen. Marketingunterlagen müssen optisch ansprechend, kurz und aussagekräftig zu inhaltlichen Aspekten informieren. Wichtig sind auch Referenzen auf andere erfolgreiche Anwender der Software. Praktisch wird das beispielsweise realisiert durch Flyer, Präsentati onen im Internet oder Whitepapers mit Managementinformationen.

Kompetenzcheck

1. Welche Aussagen sind richtig?

   a. Die Projektdokumentation enthält in der Regel dieselben Informationen wie die Systemdokumentation, welche nur für eine andere Zielgruppe aufbereitet wurde.
   b. Systemdokumentationen können UML-Diagramme oder Datenmodelle enthalten.
   c. Auf die Erstellung einer Testdokumentation kann in aller Regel verzichtet werden.
   d. Ein Benutzerhandbuch muss den sicheren Umgang mit der Software ermöglichen.
   e. In Installationsanleitungen sind auch Informationen zum Deinstallieren der Software zu finden.
   f. Marketingunterlagen werden u.a. von Vertriebsmitarbeitern des Softwareunternehmens genutzt.

2. Bearbeiten Sie im Handout die Aufgabe 20 der und erstellen Sie ein Benutzerhandbuch für Ihre Datenbanklösung.

9 Den Prozess der Softwareentwicklung evaluieren

Unter einer Evaluation wird die Bewertung oder Begutachtung von Projekten, Prozessen usw. verstanden.

Eine Evaluation dient der rückblickenden Wirkungskontrolle oder der Reflexion und dem Verständnis von didaktischen Situationen und Problemen. Dabei wird untersucht, ob und inwieweit die Vorgehensweise und die eingesetzten Ressourcen geeignet waren, den angestrebten Zweck zu erfüllen. Durch eine Evaluation werden nützliche und gesicherte Daten ermittelt, welche zur Verbesserung des Softwareentwicklungsprozesses beitragen können. Standardverfahren zur Datenerfassung sind Befragung, Beobachtung, Test und Datenanalyse. Nachfolgend sind einige Fragen, welche bei einer Evaluierung des Softwareprozesses gestellt werden können, aufgeführt.

Befragung im Rahmen der Evaluierung des Softwareprozesses

1. War das Vorgehensmodell für die Aufgabenstellung geeignet?
2. Konnte mit der gewählten Programmiersprache die Aufgabe gelöst werden?
3. Hat die gewählte IDE die Programmierung und Projektverwaltung optimal unterstützt?
4. Hat die Kommunikation im Team funktioniert?
5. Welche Probleme sind bei der Teamarbeit aufgetreten und warum?
6. War die Projektplanung ausreichend?
7. An welchen Stellen und warum musste von der Projektplanung abgewichen werden?
8. Wurde die Software ausreichend getestet?
9. Welche Schwierigkeiten sind beim Testprozess aufgetreten?
10. Konnte eine vollständige Dokumentation erstellt werden?

Kompetenzcheck

1. Führen eine Evaluation über den Entwicklungsprozess Ihrer Software durch. Bearbeiten Sie dabei folgende Schritte:
   a. Finden Sie oder Ihr Projektteam für die Evaluation relevante Fragen und beantworten Sie diese.
   b. Leiten Sie aus den Antworten Schlussfolgerungen über den Softwareentwicklungsprozess ab.
   c. Stellen Sie Ihr Projekt und die Ergebnisse der Evaluation den Mitschülerinnen und Mitschülern vor.