LERNEN EINFACH GEMACHT

# C#

für dummies®

Grundlagen, Anwendung, Praxiswissen zu C# und .NET

Fortgeschrittene Techniken wie asynchrone Programmierung

Tipps und Tricks für gutes Programmieren

Steffen Steinbrecher

# C# für Dummies

# Schummelseite

## SCHLÜSSELWÖRTER

Die Programmiersprache C# besitzt über 100 Schlüsselwörter. Bei Schlüsselwörtern handelt es sich um vordefinierte reservierte Bezeichner, die eine besondere Bedeutung innerhalb der Sprache haben.

Die folgende Tabelle listet die Schlüsselwörter von C# auf.

| ***Schlüsselwort*** | ***Was es bewirkt*** |
|---|---|
| `abstract` | Das Schlüsselwort gibt an, dass die Klasse beziehungsweise die Klassenelemente keine Implementierung haben. Wird in einer Klassendeklaration verwendet, um anzugeben, dass die Klasse nur die Basisklasse für eine andere Klasse sein soll. Die Implementierung erfolgt dann in der abgeleiteten Klasse, eine abstrakte Klasse darf nicht instanziiert werden. |
| `as` | Der `as`-Operator kann verwendet werden, um bestimmte Typkonvertierungen oder Umwandlungen zwischen kompatiblen Referenztypen durchzuführen. |
| `base` | Das `base`-Schlüsselwort wird verwendet, um aus einer abgeleiteten Klasse auf die Elemente der Basisklasse zuzugreifen (die Elemente einer Klasse werden sehr oft auch als Members bezeichnet). Wird immer dann gebraucht, wenn eine Methode der Basisklasse durch eine andere Methode überschrieben wurde. |
| `bool` | Das Schlüsselwort `bool` ist ein Alias von `System.Boolean`. Es wird zur Deklaration von Variablen zur Speicherung der booleschen Werte `true` und `false` verwendet. |
| `break` | Die `break`-Anweisung beendet die Ausführung der gerade ausgeführten Schleife oder `switch`-Anweisung. |
| `byte` | `byte` kennzeichnet eine 8-Bit-Ganzzahl ohne Vorzeichen und erlaubt das Speichern von Werten von 0 … 255. |
| `case` | Leitet einen von mehreren möglichen Ausführungspfaden in einer `switch`-Anweisung ein. |

| ***Schlüsselwort*** | ***Was es bewirkt*** |
|---|---|
| `catch` | Führt Anweisungen ein, die ausgeführt werden, wenn etwas den Ausführungsfluss in einer `try`-Klausel unterbricht. |
| `char` | Das Schlüsselwort `char` wird zur Deklaration einer Instanz der `System.Char`-Struktur verwendet, die ein Unicode-Zeichen repräsentiert (ein Buchstabe, eine Ziffer, ein Interpunktionszeichen usw.). Der Wert einer `char`-Variable ist ein numerischer 16-Bit-Wert (ordinal). |
| `checked` | Dieser Ausdruck wird verwendet, um eine Überlaufprüfung bei arithmetischen Operationen für ganzzahlige Typen und Konvertierungen explizit zu aktivieren. |
| `class` | Klassen werden mithilfe des Schlüsselworts `class` deklariert. |
| `const` | Das `const`-Schlüsselwort wird verwendet, um ein konstantes Feld oder eine konstante lokale Variable zu deklarieren. Konstante Felder und lokale Felder sind keine Variablen und können daher nicht geändert werden. Konstanten können Nummern, boolesche Werte, Zeichenfolgen oder ein `null`-Verweis sein. |
| `continue` | Die `continue`-Anweisung erzwingt den sofortigen Abbruch der aktuellen Schleifeniteration und beginnt mit der nächsten Iteration. |
| `decimal` | Das `decimal`-Schlüsselwort kennzeichnet einen 128-Bit-Datentyp. Im Vergleich zu anderen Gleitkommatypen verfügt der `decimal`-Typ über höhere Genauigkeit und einen kleineren Wertebereich. Dadurch eignet er sich für Finanz- und Währungskalkulationen. |
| `default` | In einer `switch`-Anweisung leitet das Schlüsselwort einen Ausführungspfad ein, der zu verfolgen ist, wenn kein `case` in der `switch`-Anweisung einen Treffer darstellt. Außerhalb einer `switch`-Anweisung wird das Schlüsselwort `default` als Standardwertausdruck verwendet. Ein Standardwertausdruck `default(T)` erzeugt den Standardwert für einen Typ T. |
| `delegate` | Ein `delegate` ist ein Referenztyp, der verwendet werden kann, um eine benannte oder anonyme Methode zu kapseln. Delegates entsprechen den Funktionszeigern in C++, sind jedoch typsicher und geschützt. |

| ***Schlüsselwort*** | ***Was es bewirkt*** |
|---|---|
| `do` | Die Anweisung `do` führt eine Anweisung oder einen Anweisungsblock aus, solange ein angegebener boolescher Ausdruck `true` ergibt (wird auch als `do`-Schleife bezeichnet). |
| `double` | Das `double`-Schlüsselwort kennzeichnet einen einfachen Typ, der 64-Bit-Gleitkommawerte speichert. |
| `else` | Führt Anweisungen aus, die ausgeführt werden, wenn die Bedingung in einer `if`-Anweisung nicht erfüllt ist. |
| `enum` | Erzeugt einen neuen Typ – eine Gruppe von Werten, die eine Variable annehmen kann. Der neue Typ wird auch *Enumeration* genannt. |
| `event` | Das `event`-Schlüsselwort wird verwendet, um ein Ereignis zu deklarieren. |
| `explicit` | Mit `explicit` wird ein benutzerdefinierter Typkonvertierungsoperator erzeugt, der mit einer Umwandlung aufgerufen werden muss. |
| `extern` | Der `extern`-Modifizierer wird verwendet, um eine extern implementierte Methode zu deklarieren. Der `extern`-Modifizierer wird häufig mit dem `DllImport`-Attribut verwendet, wenn Sie nicht verwalteten Code mit Interop-Diensten aufrufen. |
| `false` | Wird als ein überladener Operator oder als Literal verwendet, als Literal wird der boolesche Wert `false` zurückgeliefert. |
| `finally` | Wird in Verbindung mit dem Schlüsselwort `try` verwendet und wird immer durchlaufen, sogar wenn im `try`-Block eine Ausnahme aufgetreten ist. Im `finally`-Block können alle Ressourcen bereinigt werden, die in einem `try`-Block verwendet wurden. |
| `fixed` | Die `fixed`-Anweisung verhindert, dass der *Garbage Collector* eine Variable verschiebt. Die `fixed`-Anweisung ist nur in einem unsicheren (`unsafe`) Kontext zulässig. |
| `float` | Das `float`-Schlüsselwort kennzeichnet einen einfachen Typ, der 32-Bit-Gleitkommawerte speichert. |
| `for` | Die Anweisung `for` veranlasst den Computer, eine Anweisung oder einen Anweisungsblock mehrfach auszuführen (wird auch als `for`-Schleife bezeichnet). |

| ***Schlüsselwort*** | ***Was es bewirkt*** |
|---|---|
| `foreach` | Die Anweisung `foreach` führt eine Anweisung oder einen Block von Anweisungen für jedes Element einer Auflistung aus. Die Auflistung muss dafür die Schnittstelle `System.Collections.IEnumerable` oder `System.Collections.Generic.IEnumerable<T>` implementieren. |
| `goto` | `goto` ist eine Sprunganweisung oder ein Sprungbefehl. In C**#**-Programmen dient sie dazu, die Verarbeitung nicht mit dem nachfolgenden Befehl, sondern an einer explizit angegeben Stelle fortzuführen. Hinweis: sollte nicht mehr genutzt werden, ist nur noch aus Kompatibilitätsgründen im Sprachumfang. |
| `if` | Überprüft, ob eine Bedingung `true` ist. Ist sie `true`, führt das Programm bestimmte Anweisungen aus, andernfalls führt es andere Anweisungen aus (siehe `else`-Schlüsselwort). |
| `implicit` | Das `implicit`-Schlüsselwort wird verwendet, um einen impliziten benutzerdefinierten Typkonvertierungsoperator zu deklarieren. |
| `in` | `in` kann in verschiedenen Kontexten verwendet werden, beispielsweise in `foreach`-Schleifen. |
| `int` | `int` kennzeichnet einen ganzzahligen 32-Bit-Datentyp. |
| `interface` | Führt eine Schnittstelle (Interface) ein. Eine Schnittstelle enthält nur die Signaturen von Methoden, Eigenschaften, Ereignissen oder Indexern und keinerlei Implementierung. |
| `internal` | Das Schlüsselwort `internal` ist ein Zugriffsmodifizierer für Typen und Typmembers. Auf interne Typen oder Members kann nur innerhalb derselben Assembly zugegriffen werden. |
| `is` | Überprüft, ob ein Objekt mit einem angegebenen Typ kompatibel ist. |
| `lock` | Mit dem `lock`-Schlüsselwort wird sichergestellt, dass ein Thread nicht auf ein Objekt zugreift, während ein anderer Thread dieses Objekt bearbeitet. `lock` setzt also eine Sperre auf einen entsprechenden Codeabschnitt. |
| `long` | `long` kennzeichnet einen 64-Bit-Ganzzahltyp. |

| ***Schlüsselwort*** | ***Was es bewirkt*** |
|---|---|
| `namespace` | Das `namespace`-Schlüsselwort wird verwendet, um einen Gültigkeitsbereich zu deklarieren, der eine Gruppe von verwandten Objekten enthält. |
| `new` | Wird zum Erstellen von Objekten und zum Aufrufen von Konstruktoren verwendet, erzeugt also ein Objekt aus einer vorhandenen Klasse. |
| `null` | Das Schlüsselwort `null` ist ein Literal, das eine NULL-Referenz darstellt, die auf kein Objekt verweist. `null` ist der Standardwert einer Referenztypvariablen. |
| `object` | Der `object`-Typ ist ein Alias für `Object` in .NET. Im vereinheitlichten Typsystem von C# erben alle Typen, vordefiniert und benutzerdefiniert sowie Verweis- und Werttypen, direkt oder indirekt von `Object`. |
| `operator` | `operator` wird verwendet, um einen integrierten Operator zu überladen oder um eine benutzerdefinierte Konvertierung in einer Klassen- oder Strukturdeklaration bereitzustellen. |
| `out` | `out` ist ein Parametermodifizierer und bewirkt, dass Argumente per Verweis übergeben werden. Das Deklarieren einer Methode mit `out`-Argumenten ist nützlich, wenn eine Methode mehrere Werte zurückgeben soll. |
| `override` | Der `override`-Modifizierer wird benötigt, um die abstrakte oder virtuelle Implementierung einer geerbten Methode, Eigenschaft, eines Indexers oder Ereignisses zu erweitern oder zu überschreiben. |
| `params` | Mithilfe des Schlüsselworts `params` kann ein Methodenparameter angegeben werden, der eine variable Anzahl von Argumenten akzeptiert. |
| `private` | Das `private`-Schlüsselwort ist ein Zugriffsmodifizierer für Members und zeigt an, dass eine Variable oder Methode nur innerhalb der jeweiligen Klasse verwendet werden kann. |
| `protected` | Kennzeichnet Variablen und Methoden einer Klasse als `protected` (geschützt). Auf ein `protected`-Member kann nur innerhalb seiner Klasse und von Instanzen abgeleiteter Klassen zugegriffen werden. |

| ***Schlüsselwort*** | ***Was es bewirkt*** |
|---|---|
| `public` | Mit dem `public`-Schlüsselwort werden Variablen, Klassen oder Methoden als öffentlich gekennzeichnet. Es gibt keine Einschränkungen für den Zugriff auf öffentliche Members. |
| `readonly` | In einer Variablendeklaration gibt `readonly` an, dass die Zuweisung der Variable nur als Teil der Deklaration oder in einem Konstruktor derselben Klasse erfolgen kann. Danach kann die Variable nicht mehr geändert werden. |
| `ref` | Das `ref`-Schlüsselwort gibt einen Wert an, der als Referenz übergeben wird. |
| `return` | Beendet die Ausführung einer Methode und gibt gegebenenfalls einen Wert an den aufrufenden Code zurück. |
| `sbyte` | `sbyte` kennzeichnet einen ganzzahligen 8-Bit-Typ. |
| `sealed` | Der Modifizierer `sealed` verhindert, dass andere Klassen von einer Klasse erben, wenn er auf diese Klasse angewendet wird. |
| `short` | `short` kennzeichnet einen ganzzahligen 16-Bit-Datentyp. |
| `sizeof` | Wird verwendet, um die Größe eines nicht verwalteten Typs in Bytes abzurufen. |
| `stackalloc` | Das `stackalloc`-Schlüsselwort wird in einem unsicheren Codekontext verwendet, um einen Speicherblock auf dem Stapel (Stack) zu belegen. |
| `static` | Zeigt an, dass eine Variable oder Methode zu einer Klasse gehören, statt zu einem aus der Klasse erzeugten Objekt. |
| `string` | Der Typ `string` stellt eine Sequenz von 0 oder mehr Unicode-Zeichen dar. |
| `struct` | Ein `struct`-Typ ist ein ein Werttyp, der in der Regel verwendet wird, um eine kleine Gruppe verwandter Variablen zusammenzufassen, beispielsweise Koordinaten eines Rechtecks oder die Merkmale eines Lagerartikels. |
| `switch` | `switch` ist eine Auswahlanweisung, die vorgibt, abhängig vom Wert eines Ausdrucks einem von mehreren Anweisungspfade zu folgen (einem der möglichen `case`-Fälle). |

| ***Schlüsselwort*** | ***Was es bewirkt*** |
|---|---|
| `this` | Ein Selbstverweis – verweist auf die aktuelle Instanz der Klasse, in der das Schlüsselwort `this` verwendet wird. |
| `throw` | Signalisiert das Auftreten einer Ausnahme während der Programmausführung und zeigt somit an, dass eine außergewöhnliche Situation (in der Regel etwas Unerwünschtes) aufgetreten ist. |
| `true` | Stellt den booleschen Wert `true` für wahr dar. |
| `try` | Der `try`-Block enthält überwachten Code, der möglicherweise eine Ausnahme verursacht. Der Block wird ausgeführt, bis eine Ausnahme ausgelöst wird oder bis er erfolgreich abgeschlossen wird. |
| `typeof` | Kann dazu verwendet werden, um den Typ eines Objekts zur Laufzeit eines Programms zu ermitteln. |
| `uint` | Das Schlüsselwort `uint` kennzeichnet einen ganzzahligen, positiven 32-Bit-Typ. |
| `ulong` | `ulong` kennzeichnet einen ganzzahligen, positiven 64-Bit-Typ. |
| `unchecked` | Das Schlüsselwort `unchecked` wird verwendet, um eine Überlaufüberprüfung bei arithmetischen Operationen für ganzzahlige Typen und Konvertierungen zu unterdrücken. |
| `unsafe` | Das Schlüsselwort `unsafe` erzeugt einen unsicheren Kontext, der für alle Zeigeroperationen erforderlich ist. |
| `ushort` | `ushort` kennzeichnet einen ganzzahligen, positiven 16-Bit-Datentyp. |
| `using` | Hat zwei Hauptverwendungen: Als Direktive zum Importieren von Namensräumen oder als Anweisung, wenn ein Bereich definiert wird, an dessen Ende ein Objekt verworfen wird. |
| `virtual` | Erlaubt, dass eine Methoden-, Eigenschaften-, Indexer- oder Ereignisdeklaration in einer abgeleiteten überschrieben werden darf. |
| `void` | Zeigt an, dass eine Methode keinen Wert zurückgibt. |
| `volatile` | Das Schlüsselwort `volatile` gibt an, dass ein Feld von mehreren Threads geändert werden kann, die zur gleichen Zeit ausgeführt werden. |

| ***Schlüsselwort*** | ***Was es bewirkt*** |
| --- | --- |
| `while` | Die Anweisung `while` führt eine Anweisung oder einen Anweisungsblock immer wieder aus, solange ein angegebener boolescher Ausdruck `true` ergibt (`while`-Schleife). |

Steffen Steinbrecher

# C# für dummies®

Fachkorrektur von Harald Binkle

WILEY

WILEY-VCH Verlag GmbH & Co. KGaA

**C# für Dummies**

**Bibliografische Information der Deutschen Nationalbibliothek**

Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie;detaillierte bibliografische Daten sind im Internet über http://dnb.d-nb.de abrufbar.



**Print ISBN:** 978-3-527-71519-0

**ePub ISBN:** 978-3-527-81709-2

Coverfoto: © bluebay2014 – stock.adobe.com

Korrektur: Matthias Delbrück, Dossenheim/Bergstraße

# Über den Autor

Steffen Steinbrecher beschäftigt sich seit über 15 Jahren mit der Softwareentwicklung. In dieser Zeit lag sein Hauptschwerpunkt auf der Programmiersprache C# und der .NET-Plattform. In seiner Freizeit betreibt er einen Blog zu allem, was die Programmiersprache C# betrifft.

# *Danksagung*

Ich danke meiner Lektorin Andrea Baulig für die nette und verständnisvolle Betreuung während der Manuskripterstellung. Ganz besonders möchte ich mich bei Harald Binkle bedanken, dessen Anregungen diesem Buch den letzten Schliff gaben.

# Inhaltsverzeichnis


- Cover
- Über den Autor
  - Danksagung
- Einleitung
  - Über dieses Buch
  - Wie dieses Buch aufgebaut ist
  - Symbole, die in diesem Buch verwendet werden
  - Wie es weiter geht
- Teil I: Los geht's
  - Kapitel 1: Grundlagen und Einführung
    - Grundlagen der Programmierung
    - .NET-Plattformarchitektur
    - Das Prinzip der objektorientierten Programmierung
  - Kapitel 2: Entwicklungswerkzeuge und Tools
    - Ein Programm erstellen
    - Microsoft Visual Studio 2019 Community Edition
    - Der Marktplatz für fertige Lösungen: NuGet
  - Kapitel 3: Das ABC der Sprache C#
    - Bezeichner
    - Kommentare
    - Datentypen, Variablen und Konstanten
    - Was ist eine Methode?
    - Namensräume
  - Kapitel 4: Operatoren und Programmsteuerung
    - Operatoren

Den Programmablauf steuern: Verzweigungen und Kontrollstrukturen

Vorgänge wiederholen: Schleifen

## Kapitel 5: Zeichenfolgen, Arrays und Datumswerte

Zeichen und Zeichenfolgen

Arrays

Datums- und Zeitberechnungen

## Kapitel 6: Objektorientierte Programmierung mit C#

Klassen, Eigenschaften und Methoden

Vererbung

Abstrakte Klassen

Interfaces

## Kapitel 7: Fehler passieren, macht aber nichts: Ausnahmebehandlung

Mit Ausnahmen umgehen

Wie man einen Fehler »fängt«!

Ausnahmetypen

Fehler auslösen

Eigene Fehlerklassen erstellen

## Kapitel 8: Weitere Sprachfeatures

Delegates

Anonyme Methoden

Lambda-Ausdrücke

Ereignisse – was geht ab?

Generics – generische Datentypen

Nette Helfer: Attribute

Sie wollen mehr? Erweiterungsmethoden

# Teil II: Weitere Sprachfeatures von C#

## Kapitel 9: Zugriffe auf das Dateisystem

Klassen für den Zugriff auf das Dateisystem

Praxisbeispiele

## Kapitel 10: Dateizugriff und Streams

Arbeiten mit Dateien

Arbeiten mit Streams

Komprimieren von Dateien mit .NET-Klassen

Objekte serialisieren

## Kapitel 11: Asynchrone und parallele Programmierung

Grundlagen und Begrifflichkeiten

Asynchrone Programmiermuster

## Kapitel 12: Daten abfragen mit LINQ

LINQ-Grundlagen

Abfragen mit LINQ

Die wichtigsten Abfrageoperatoren

Praxisbeispiele

## Kapitel 13: Grundlagen von ADO.NET

Vorbereitungen

Datenbankzugriffe mit ADO.NET

Das DataSet und der DataAdapter

Asynchrone Datenbankzugriffe

# Teil III: Eine eigene C#-Anwendung schreiben

## Kapitel 14: Fehlersuche und Softwarequalität

Welche Fehlerarten gibt es überhaupt?

Der Debugger

Codequalität verbessern

## Kapitel 15: Benutzeroberfläche mit Windows Forms

Grundgerüst einer Windows-Forms-Anwendung

Benutzeroberfläche erstellen

Windows-Forms-Anwendung: Ja oder Nein?

### Kapitel 16: Moderne Oberflächen mit WPF und XAML

Meine erste WPF-Anwendung

Ereignisse in WPF

Commands

Ressourcen in WPF

Das gewisse Etwas: Styles

Databinding – die Benutzeroberfläche mit Daten versorgen

## Teil IV: Fortgeschrittene Techniken in C# und WPF

### Kapitel 17: Fortgeschrittene Techniken rund um die WPF

Das Beispielprogramm

Das MVVM-Entwurfsmuster

Datenbeschaffung

Eigene Commands implementieren

### Kapitel 18: Modulare Anwendung auf Basis von C# und WPF – ein Beispiel

Die Beispielanwendung

Vorbereitungen und Erstellung der Solution

Styling der Anwendung

Das Dependency-Injection-Entwurfsmuster

Dynamische Oberflächen mit Prism-Regionen

Erstellung eines Prism-Moduls

Austausch- und wiederverwendbare Komponenten

Fazit – Was haben Sie jetzt gewonnen?

## Teil V: Der Top-Ten-Teil

### Kapitel 19: Zehn Dinge in C#, die Sie wirklich lernen und verwenden sollten!

Initialisierer für Objekte und Collections

Automatisch implementierte Eigenschaften initialisieren

null-coalescing Operator ??

String-Interpolation für dynamische Zeichenfolgen
Null-conditional Operator
nameof()-Ausdruck
is- und as-Operatoren
Zeichenfolgen mit String.IsNullOrEmpty() überprüfen
break- und continue-Anweisung
TryParse für die Umwandlung von Zeichenketten

**Kapitel 20: Zehn nützliche Open-Source-Projekte**

MahApps.Metro: Verpassen Sie Ihrer Anwendung den Metro-Style
Modern UI for WPF (MUI)
MaterialDesignInXamlToolkit
Extended WPF Toolkit™
WPFLocalizationExtension
ResXManager
Prism
Sammlung von Erweiterungsmethoden
Modern UI Icons
Material Design Icons

**Stichwortverzeichnis**

**End User License Agreement**

# Tabellenverzeichnis

## Kapitel 3

Tabelle 3.1: Übersicht Datentypen.
Tabelle 3.2: Typsuffixe in Verbindung mit dem `var`-Schlüsselwort.
Tabelle 3.3: Erlaubte implizite numerische Konvertierungen.
Tabelle 3.4: Mögliche explizite numerische Konvertierungen.
Tabelle 3.5: Wichtige Namensräume im .NET-Framework.

## Kapitel 4

Tabelle 4.1: Arithmetische Operatoren.

Tabelle 4.2: Inkrement- und Dekrement-Operatoren.

Tabelle 4.3: Die verschiedenen Zuweisungsoperatoren.

Tabelle 4.4: Die verschiedenen Vergleichsoperatoren.

Tabelle 4.5: Boolesche Operatoren.

## Kapitel 5

Tabelle 5.1: Wichtige Eigenschaften und Methoden zur Analyse von Zeichenfolgen.

Tabelle 5.2: Wichtige Methoden zur Manipulation von Zeichenfolgen.

Tabelle 5.3: Wichtige Escape-Sequenzen in C#.

Tabelle 5.4: Formatbezeichner für `DateTime`-Variablen.

Tabelle 5.5: Formatbezeichner für Zahlenwerte.

Tabelle 5.6: Wichtige Eigenschaften und Methoden von `Array`-Variablen.

Tabelle 5.7: Wichtige Methoden der `Array`-Klasse.

Tabelle 5.8: Wichtige Eigenschaften und Methoden der `DateTime`-Struktur.

Tabelle 5.9: Methoden zum Addieren von Datums- und Uhrzeitangaben.

## Kapitel 6

Tabelle 6.1: Zugriffsmodifizierer zur Einschränkung der Sichtbarkeit.

Tabelle 6.2: Codeausschnitte für Eigenschaften in Microsoft Visual Studio.

## Kapitel 7

Tabelle 7.1: Eigenschaften der `System.Exception`-Klasse.

## Kapitel 9

Tabelle 9.1: Wichtige Klassen für die Arbeit mit Dateien und Verzeichnissen.

Tabelle 9.2: Ausprägungen der `SearchOption`-Enumeration.

Tabelle 9.3: Platzhalterzeichen für das Suchmuster.

Tabelle 9.4: Wichtige Methoden der `Path`-Klasse.

## Kapitel 10

Tabelle 10.1: Methoden der `File`-Klasse, um Datei zu erstellen, zu lesen und zu än...

Tabelle 10.2: Methoden der abstrakten `Stream`-Klasse.

Tabelle 10.3: Wichtige `Stream`-Klassen.

Tabelle 10.4: Klassen für Komprimierung und Dekomprimierung.

## Kapitel 11

Tabelle 11.1: Einige wichtige Werte der `TaskContinuationOptions` Enumeration (nich...

Tabelle 11.2: Mögliche Rückgabewerte einer asynchronen Methode.

## Kapitel 12

Tabelle 12.1: Die wichtigsten LINQ-Aggregat-Operatoren (nicht vollständig).

## Kapitel 13

Tabelle 13.1: Aufbau der Tabelle *categories* in der Northwind-Beispieldatenbank.

Tabelle 13.2: Die Werte der Enumeration `DataRowState`.

Tabelle 13.3: Die Enumeration `DataRowVersion`.

Tabelle 13.4: Aufbau der Tabelle *orders* in der Northwind-Beispieldatenbank.

## Kapitel 14

Tabelle 14.1: Tastenkombinationen im Debug-Modus.

## Kapitel 15

Tabelle 15.1: Verwendete Steuerelemente und ihre Eigenschaften.

# Illustrationsverzeichnis

## Kapitel 1

Abbildung 1.1: Ein Compiler übersetzt Quelltext in ausführbaren Maschinencode (*....

Abbildung 1.2: Ein Interpreter wertet den Quelltext direkt aus und wandelt ihn in...

Abbildung 1.3: Es gibt auch Programmiersprachen, die eine Kombination aus Compile...

Abbildung 1.4: Übersicht .NET-Plattform (High-Level).

Abbildung 1.5: Kompilierung eines .NET-Programms.

Abbildung 1.6: CIL-Code des ersten C#-Programms.

Abbildung 1.7: Wichtige Bestandteile des .NET-Frameworks.

Abbildung 1.8: Klassendiagramm für einen PKW.

Abbildung 1.9: Konkrete Objekte einer Klasse `PKW` zur Laufzeit eines Programms.

Abbildung 1.10: Klassendiagramm Fahrzeug und davon abgeleitete Klassen.

## Kapitel 2

Abbildung 2.1: »HelloWord.cs« im Texteditor Notepad von Microsoft Windows.

Abbildung 2.2: Das Programm mit csc.exe kompilieren.

Abbildung 2.3: Läuft!

Abbildung 2.4: Kompilierfehler wegen eines fehlenden Semikolons.

Abbildung 2.5: Installationswizard der Microsoft Visual Studio Community Edition ...

Abbildung 2.6: Ein neues Projekt in Microsoft Visual Studio anlegen.

Abbildung 2.7: Die unterschiedlichen Projektvorlagen in Microsoft Visual Studio.

Abbildung 2.8: Projekteigenschaften festlegen.

Abbildung 2.9: Die Oberfläche von Microsoft Visual Studio.

Abbildung 2.10: Das Programm wird über die Symbolleiste mit dem Menüpunkt STARTEN

Abbildung 2.11: Ein leeres Fenster zeigt, dass Ihr Programm gestartet ist.

Abbildung 2.12: Die Benutzeroberfläche bekommt Steuerelemente aus der Toolbox.

Abbildung 2.13: Das EIGENSCHAFTEN-Fenster für ein `Button`-Steuerelement.

Abbildung 2.14: Das Programm hat zwei `Button`-Steuerelemente.

Abbildung 2.15: Das lauffähige Programm hat eine Funktionalität: Es zeigt eine Me...

Abbildung 2.16: Ein NuGet-Paket über den NuGet-Paketmanager hinzufügen.

Abbildung 2.17: Der NuGet-Paketmanager: Pakete suchen und installieren.

Abbildung 2.18: Die neu referenzierte Datei und die neue »packages.config«.

## Kapitel 3

Abbildung 3.1: Werttypen in C#.

Abbildung 3.2: Werttypen und der Stack.

Abbildung 3.3: Referenztypen in C#.

Abbildung 3.4: Heap und Stack mit Referenztypen.

Abbildung 3.5: Boxing und Unboxing im Speicher.

Abbildung 3.6: Eine implizite Typkonvertierung ist hier nicht möglich.

Abbildung 3.7: Fehlermeldung bei impliziter Konvertierung mit aktivierter Überlau...

Abbildung 3.8: Unerreichbarer Quellcode in Microsoft Visual Studio.

Abbildung 3.9: Objektkatalog innerhalb von Microsoft Visual Studio.

Abbildung 3.10: Fehler bei Verwendung einer Klasse, deren Namensraum nicht eingeb...

Abbildung 3.11: Aktionen und Hinweise für die Behebung von Fehlern und Warnungen.

## Kapitel 4

Abbildung 4.1: Leider verloren!

Abbildung 4.2: Flussdiagramm des Ratespiels.

Abbildung 4.3: Flowchart der `for`-Schleife.

Abbildung 4.4: Flowchart der `while`-Schleife.

Abbildung 4.5: Raten Sie, bis Sie richtig liegen.

Abbildung 4.6: Flowchart der `do-while`-Schleife.

## Kapitel 5

Abbildung 5.1: Strings zählen zu den Verweistypen!

Abbildung 5.2: Bei jeder Änderung eines Strings wird ein neues Objekt erzeugt und...

Abbildung 5.3: Ein `Array` mit 5 `int`-Werten.

Abbildung 5.4: Fehler beim Zugriff auf ein `Array`.

Abbildung 5.5: Zugriffe auf ein `Array`.

Abbildung 5.6: Ausgabe des Beispielprogramms aus Listing 5.32.

Abbildung 5.7: Zusammenfassung der Quartalsberechnung für ein gegebenes Datum.

## Kapitel 6

Abbildung 6.1: Ein PKW und seine Eigenschaften.

Abbildung 6.2: Codeausschnitt einfügen.

Abbildung 6.3: Eine durch den Codeausschnitt `propfull` eingefügte Eigenschaftendek...

Abbildung 6.4: Die von der Klasse `Fahrzeug` geerbten Members.

Abbildung 6.5: Abstrakte Methoden müssen implementiert werden, sonst gibt es eine...

Abbildung 6.6: Klassendiagramm für `PKW`, `LKW` und `Motorrad` mit `IFahrzeug`-Schnittste...

Abbildung 6.7: Microsoft Visual Studio gibt bei unvollständig implementierter Sch...

Abbildung 6.8: Schnittstelle automatisch implementieren in Microsoft Visual Studi...

## Kapitel 7

Abbildung 7.1: Unbehandelte Ausnahme `DivideByZeroException`.

Abbildung 7.2: Inhalt der Eigenschaft `StackTrace`.

Abbildung 7.3: E-Mail-Client zum Versenden der E-Mail.

## Kapitel 8

Abbildung 8.1: Ereignissender und Ereignisbehandler.

Abbildung 8.2: Beispielhafte Ausgabe des Programms aus Listing 8.14.

Abbildung 8.3: Das `T` wird durch den konkreten Datentyp ersetzt.

Abbildung 8.4: Eine generische Klasse ist typsicher!

Abbildung 8.5: Ausgabe der generischen Klasse.

Abbildung 8.6: Die Verwender erhalten eine Warnung, dass die Methode veraltet ist...

Abbildung 8.7: Die `String`-Klasse ist mit dem `sealed`-Schlüsselwort gegen Vererbung...

Abbildung 8.8: Die Erweiterungsmethode ist direkt (über den Namensraum) verfügbar...

## Kapitel 9

Abbildung 9.1: Ausgabe des Programms aus Listing 9.9.

## Kapitel 10

Abbildung 10.1: Exception beim Serialisieren.

Abbildung 10.2: Inhalt der serialisierten Binärdatei.

Abbildung 10.3: Ausgabedatei des `XmlSerializer`.

Abbildung 10.4: Ausgabedatei des `DataContractJsonSerializer`.

Abbildung 10.5: Das Json.NET-Paket mit dem Nuget-Paketmanager in das Projekt einf...

Abbildung 10.6: Ausgabedatei des Json.NET Serializers.

## Kapitel 11

Abbildung 11.1: Microsoft Windows Task-Manager – Übersicht über laufende Anwendun...

Abbildung 11.2: Windows Task-Manager – Kontextmenü zu einem Prozess.

Abbildung 11.3: Windows Task-Manager – Details zu einem Prozess.

Abbildung 11.4: Parallele Ausführung von zwei Threads.

Abbildung 11.5: Ausgabe des Programms aus Listing 11.1.

Abbildung 11.6: Threads im Debugger von Microsoft Visual Studio.

Abbildung 11.7: Ausgaben des Programms aus Listing 11.10.

## Kapitel 12

Abbildung 12.1: Die LINQ-Architektur im Überblick.

Abbildung 12.2: Datenmodell für LINQ-Abfragen.

Abbildung 12.3: Beispieldaten für den Shop.

Abbildung 12.4: Funktionsweise des `where`-Operators zur Anwendung eines Filters.

Abbildung 12.5: Programmausgabe: Kunden mit PLZ = 12345.

Abbildung 12.6: Funktionsweise des `orderby`-Operators.

Abbildung 12.7: Funktionsweise von `skip`- und `take`-Operator.

Abbildung 12.8: So arbeitet der `distinct`-Operator.

Abbildung 12.9: Funktionsweise von `sum`- und `max`-Aggregratfunktion.

Abbildung 12.10: Eingelesene CSV-Datei mit den Postleitzahlen als anonymer Typ.

## Kapitel 13

Abbildung 13.1: Installationsassistent von EnterpriseDB.

Abbildung 13.2: Umfang der PostgreSQL-Installation.

Abbildung 13.3: Passwort für den Administrationsbenutzer postgres vergeben.

Abbildung 13.4: Die Administrationsoberfläche von *pgAdmin 4*.

Abbildung 13.5: *pgAdmin* – neue Datenbank anlegen.

Abbildung 13.6: Northwind-Beispieldaten importieren.

Abbildung 13.7: Npgsql .NET-Datenprovider über NuGet installieren.

Abbildung 13.8: Eigenschaften einer erfolgreich hergestellten Npgsql-Verbindung.

Abbildung 13.9: Die neu angelegte Produktkategorie.

Abbildung 13.10: Die aktualisierte Produktkategorie.

Abbildung 13.11: Das Zusammenspiel von `DataSet`, `DataAdapter` und Datenbank.

Abbildung 13.12: `DataSet`-Schnellansicht im Microsoft Visual Studio Debugger.

Abbildung 13.13: Originalwert und geänderter Wert einer `DataRow` im Microsoft Visu...

Abbildung 13.14: Ausgabe der asynchron selektierten Bestellungen.

## Kapitel 14

Abbildung 14.1: Die Debug-Symbolleiste von Microsoft Visual Studio.

Abbildung 14.2: Ein gesetzter Haltepunkt (Zeile 28) in Microsoft Visual Studio.

Abbildung 14.3: Deklaration einer Liste, Hinzufügen von Werten und anschließende ...

Abbildung 14.4: Zustand der Variable `myFriends` nach dem Hinzufügen eines Eintrags...

Abbildung 14.5: Das Fenster *Lokal* zur Anzeige von Variablen im aktuellen Gültigke...

Abbildung 14.6: Das Fenster *Haltepunkte* zur Anzeige von Haltepunkten innerhalb de...

Abbildung 14.7: Einstellungen für einen bedingten Haltepunkt.

Abbildung 14.8: Anzeige eines JSON-Strings im Debugger.

Abbildung 14.9: Anzeige eines JSON-Strings in der Schnellansicht.

Abbildung 14.10: Anzeigen von Rückgabewerten einer Methode.

Abbildung 14.11: Anzeige der Klasse `Friend` im Debugger.

Abbildung 14.12: Anzeige der Klasse `Friend` nach Anwendung des `DebuggerDisplay`-Att...

Abbildung 14.13: Wie Sie die `StackTrace`-Eigenschaft für die Fehlersuche verwenden...

Abbildung 14.14: Fehler und Warnungen im Quelltext-Editor von Microsoft Visual St...

Abbildung 14.15: Das Fenster FEHLERLISTE von Microsoft Visual Studio.

Abbildung 14.16: Vorschau der Änderungen einer Schnellaktion.

Abbildung 14.17: Schnellaktion für eine nicht verwendete Variable...

## Kapitel 15

Abbildung 15.1: Grundgerüst einer Windows-Forms-Anwendung.

Abbildung 15.2: Inhalt der Datei *Form1.Designer.cs* mit dem Hinweis, die Datei nic...

Abbildung 15.3: Benutzeroberfläche des Beispielprogramms.

Abbildung 15.4: Die TOOLBOX im Microsoft Visual Studio Designer.

Abbildung 15.5: Das EIGENSCHAFTEN-Fenster in Microsoft Visual Studio.

Abbildung 15.6: Registrierung von Steuerelement-Ereignissen.

Abbildung 15.7: Ereignisse eines Buttons im EIGENSCHAFTEN-Fenster von Microsoft V...

Abbildung 15.8: Ereignisse eines `Button`-Steuerelements im EDITOR-Fenster via Inte...

Abbildung 15.9: Methode für Ereignisbehandlung mit Tastenkombination generieren.

Abbildung 15.10: Ausgabe des Beispielprogramms.

Abbildung 15.11: Ausgabe des Beispielprogramms bei fehlerhafter Eingabe.

## Kapitel 16

Abbildung 16.1: Aufbau der neu erstellten WPF-Anwendung.

Abbildung 16.2: Ergebnis des Listings 16.3 .

Abbildung 16.3: Der visuelle Baum einer WPF-Anwendung in Microsoft Visual Studio.

Abbildung 16.4: Visuelle Darstellung des Grids aus Listing 16.5 .

Abbildung 16.5: `StackPanel` mit vertikaler Ausrichtung.

Abbildung 16.6: `StackPanel` mit horizontaler Ausrichtung.

Abbildung 16.7: Die Elemente werden im `WrapPanel` umbrochen (»gewrapped«), wenn de...

Abbildung 16.8: Vertikal angeordnetes `WrapPanel`.

Abbildung 16.9: Mit dem `DockPanel` können Elemente an den Seiten angedockt werden.

Abbildung 16.10: Auswirkung der `Margin`-Eigenschaft.

Abbildung 16.11: Auswirkung der `Padding`-Eigenschaft.

Abbildung 16.12: Die Eigenschaften `HorizontalAlignment` und `VerticalAlignment` in A...

Abbildung 16.13: Auswirkung der `Visibility`-Eigenschaft auf das Layout.

Abbildung 16.14: Routed Events werden im visuellen Elementbaum von oben nach unte...

Abbildung 16.15: Das Programm mit vordefinierten WPF Commands in Aktion.

Abbildung 16.16: Verwendung einer Ressource in mehreren Steuerelementen (Textbloc...

Abbildung 16.17: Dieser Button verwendet die Ressourcen aus Listing 16.17 .

Abbildung 16.18: Ein separates `ResourceDictionary` in einer separaten XAML-Datei.

Abbildung 16.19: Ausgabe aus Listing 16.20 .

Abbildung 16.20: Darstellung eines Buttons in Abhängigkeit von der `IsMouseOver`-Ei...

Abbildung 16.21: Button mit Standard-Template und mit angepasstem Template für ov...

Abbildung 16.22: Darstellung einer Liste *ohne* `DataTemplate`.

Abbildung 16.23: Darstellung einer Liste *mit* `DataTemplate`.

Abbildung 16.24: Einfache »Zoomfunktion« in WPF.

Abbildung 16.25: Datenfluss beim Databinding.

Abbildung 16.26: Mit dem `UpdateSourceTrigger` legen Sie fest, wann...

## Kapitel 17

Abbildung 17.1: Die Benutzeroberfläche des Beispielprogramms.

Abbildung 17.2: Projektaufbau des Beispielprogramms.

Abbildung 17.3: Die einzelnen Komponenten des MVVM-Entwurfsmusters.

## Kapitel 18

Abbildung 18.1: Die Benutzeroberfläche des Beispielprogramms.

Abbildung 18.2: Aufbau der Beispielanwendung.

Abbildung 18.3: Installation des *Prism Template Pack* Plug-ins.

Abbildung 18.4: Die verschiedenen Projektvorlagen für Prism.

Abbildung 18.5: Die IOC-Container-Auswahl von Prism.

Abbildung 18.6: Initiale WPF-Anwendung mit Prism.

Abbildung 18.7: Vorabversionen von NuGet-Paketen einbinden.

Abbildung 18.8: Das (noch leere) Hauptfenster der Beispielanwendung mit dem Metro...

Abbildung 18.9: Der Dependency-Injection-Container.

Abbildung 18.10: Unterteilung eines Dialogs in verschiedene Regionen.

Abbildung 18.11: Die Beispielanwendung mit den Regionen.

Abbildung 18.12: Vorlagen für `Window`- und `UserControl`-Steuerelemente aus dem *Pris*...

Abbildung 18.13: Aufbau des Prism-Moduls für die Anzeige der Postleitzahlen.

Abbildung 18.14: Aufbau des Projekts `ModularApplicationSample.Inf`...

Abbildung 18.15: Klassendiagramm für den wiederverwendbaren Service zum Verwalten...

# Einleitung

Ich habe schon in verschiedenen Programmiersprachen Anwendungen geschrieben, aber die Programmiersprache C# ist mir persönlich immer noch am liebsten. Alles ist dort sehr gut strukturiert und es gibt eine sehr gute Dokumentation. Die zur Verfügung stehenden Entwicklungswerkzeuge sind sehr ausgereift und es macht einfach Spaß, damit zu arbeiten.

## *Über dieses Buch*

Ich erinnere mich noch recht gut an meine ersten Programmierversuche, diese standen ganz unter dem Motto: Learning by doing! Denn schon Aristoteles sagte:

> *Was man lernen muss, um es zu tun, lernt man, indem man es tut.*

Am besten lernen Sie das Programmieren durch Ausprobieren, daher sollten Sie nach Belieben mit C# experimentieren. Sie können nichts kaputt machen!

Beim Erlernen einer Programmiersprache bringt diese Vorgehensweise jedoch einen gravierenden Nachteil mit sich: Sie legen die Anforderungen fest, die Ihr Programm erledigen soll, und legen los. Wenn Sie aber nicht genau wissen, wie gewisse Themen, beispielsweise das Einlesen von Dateien oder der Zugriff auf eine Datenbank, korrekt umgesetzt werden, wird es schwer, ein gutes Programm zu schreiben. Gerade die Fehlersuche kann bei fehlenden Grundkenntnissen sehr schwierig werden. Dies gilt im übrigen für jede Programmiersprache – also nicht nur für C#. Das lässt sich auch auf andere Lebensbereiche übertragen. Mal

ein Beispiel (das der eine oder andere vielleicht schon kennt): Angenommen, Sie haben sich einen neuen Kleiderschrank gekauft, den Sie selbst aufbauen möchten. Im Grunde wissen Sie, wie das Endresultat aussehen soll, und legen einfach los, ohne vorher die Aufbauanleitung zu lesen. Sie werden den Kleiderschrank sehr wahrscheinlich aufgebaut bekommen, aber am Ende bleiben eventuell ein paar Teile übrig und diese Teile habe möglicherweise entscheidende Auswirkungen auf die Stabilität oder Funktion des Schranks. Bei der Programmierung ist es ganz ähnlich: Es gibt einige Regeln, die es zu beachten gilt, damit die Funktionalität und Stabilität eines Programms gewährleistet ist, gerade bei größeren Projekten sollte man darauf besonders achten.

Sie werden sich jetzt vielleicht fragen: Wie soll ich dieses Buch durchlesen? Die Antwort auf diese Frage richtet sich ein wenig nach Ihrem Kenntnisstand: sowohl bezüglich der Programmierung im Allgemeinen als auch bezüglich der Programmiersprache C# im Besonderen. Für Einsteiger ist es sicherlich am besten, das Buch von vorne bis hinten durchzuarbeiten, insbesondere die ersten beiden Teile. Fortgeschrittene Leser können je nach Kenntnisstand auch einzelne Kapitel überspringen.

Dieses Buch teilt die C#-Programmierung in verschiedene Bereiche. In einigen Kapiteln lernen Sie grundlegende Konzepte der Programmierung, in anderen Kapiteln geht es um die praktische Anwendung von speziellen C#-Features. Im Grunde können Sie an einer beliebigen Stelle einsteigen. Ich habe versucht, die Kapitel und Beispiele so zu gestalten, dass sie nicht aufeinander aufbauen. Wichtige Informationen aus einzelnen Kapiteln werden an geeigneter Stelle wieder aufgegriffen und als Hinweis eingebunden, damit Sie wissen, worum es geht.

Eine weitere kurze Anmerkung zum Buch, bevor es gleich losgeht: Es lässt sich leider nicht vermeiden, dass bei der Erklärung bestimmter Themen gewissen Inhalten vorgegriffen wird, da gewisse Konzepte der Programmiersprache C# andere voraussetzen. An diesen Stellen gebe ich Ihnen aber einen entsprechenden Hinweis auf das Kapitel oder den Abschnitt, in denen das vorgegriffene Thema besprochen wird. In einigen Kapiteln gebe ich Ihnen auch Hinweise auf bereits behandelte Themen, die Sie dann nochmal nachlesen können.

Grundsätzlich können Sie wie folgt vorgehen:

- ✔ Wenn Sie etwas schon kennen, brauchen Sie es nicht erneut zu lesen.
- ✔ Wenn Sie ein bestimmtes Themengebiet besonders interessiert, scheuen Sie sich nicht, an die entsprechende Stelle zu springen. Innerhalb der einzelnen Themengebiete gibt es an geeigneter Stelle Wiederholungen von wichtigen, bereits behandelten Themen. Sie können bei Bedarf jederzeit wieder einen Blick in ein früheres Kapitel werfen.

Wenn Sie sich mit der Programmierung beschäftigten – egal um welche Programmiersprache es sich handelt –, müssen einigen Voraussetzungen erfüllt sein, daher treffe ich hier ein paar einfache Annahmen:

- ✔ Der in diesem Buch gezeigte Code kann auf fast jedem beliebigen Rechner ausgeführt werden. Ideale Voraussetzung ist ein recht aktueller Computer mit Microsoft Windows als Betriebssystem.
- ✔ Sie sind mit der grundlegenden Bedienung eines Computers vertraut. Sie müssen kein Microsoft-Windows-Profi sein, um mit der Programmierung in C#