

C# für IT-Berufe

```
using System;

namespace CSharp_IT_Berufe
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Informationsteil:");
            Console.WriteLine("- Einführung C#");
            Console.WriteLine("- Windows-Forms");
            Console.WriteLine("- WPF");
            Console.WriteLine("- UWP-Apps");
            Console.WriteLine();
            Console.WriteLine("Aufgabenpool");
            Console.WriteLine();
            Console.WriteLine("Lernsituationen");
            Console.WriteLine();
        }
    }
}
```

4. Auflage

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

Europa-Nr.: 85542

Verfasser:

Dirk Hardy, 46049 Oberhausen

4. Auflage 2018

Druck 5 4 3 2

Alle Drucke derselben Auflage sind parallel einsetzbar, da sie bis auf die Behebung von Druckfehlern untereinander unverändert sind.

ISBN 978-3-8085-8569-6

Alle Rechte vorbehalten. Das Werk ist urheberrechtlich geschützt. Jede Verwertung außerhalb der gesetzlich geregelten Fälle muss vom Verlag schriftlich genehmigt werden.

© 2018 by Verlag Europa-Lehrmittel, Nourney, Vollmer GmbH & Co. KG, 42781 Haan-Gruiten

<http://www.europa-lehrmittel.de>

Satz: Reemers Publishing Services GmbH, Krefeld

Umschlag: braunwerbeagentur, 42477 Radevormwald

Umschlagfotos: Christian Worring-adobestock.com; imageteam-adobestock.com; bilderbox-adobestock.com

Druck: Medienhaus Plump GmbH, 53619 Rheinbreitbach

Vorbemerkung

Die Firma Microsoft suchte in den späten 90er-Jahren eine Antwort auf die enorm erfolgreiche Programmiersprache Java, die zugleich mit einer neuen Technologie verbunden war. Durch die virtuellen Maschinen, in denen der übersetzte Java-Quellcode ausgeführt wurde, war die Grundlage einer Plattformunabhängigkeit und weiterer Vorteile gegeben. Microsoft entwickelte daraufhin die Softwareplattform **.NET**, die die Möglichkeiten der Java-Technologie und zusätzliche Vorzüge haben sollte. Die Sprache **C#** wurde dann speziell für **.NET** entworfen. C# ist eine moderne und vollständig objektorientierte Sprache. Sie ist syntaktisch an die Sprache C++, konzeptionell aber eher an die Sprache Java angelehnt. Das Erlernen der Sprache C# beinhaltet auch die intensive Auseinandersetzung mit der **.NET**-Technologie. Diese Auseinandersetzung ist für die Ausbildung im IT-Bereich ein wichtiger Aspekt.

Aufbau des Buches

Das vorliegende Buch möchte die Sprache C# möglichst anschaulich, praxis- und unterrichtsnah vermitteln. Damit verfolgt dieses Buch einen **praktischen Ansatz**. Es ist die Ansicht des Autors, dass gerade in der schulischen Ausbildung der Zugang zu den komplexen Themen der Programmierung verstärkt durch anschauliche und praktische Umsetzung vorbereitet werden muss. Anschließend können allgemeine und komplexe Aspekte der Programmierung oder auch der Softwareentwicklung besser verstanden und umgesetzt werden.

Das Buch ist in **drei Teile** getrennt. Der **erste Teil** des Buches dient als **Informationsteil** und bietet eine **systematische Einführung in die Sprache C# und in die Grundlagen von .NET**.

Ein ausführlicher Einstieg in die **Windows-Programmierung** rundet den Informationsteil ab. Dabei werden sowohl die klassische GUI-Programmierung mit *Windows-Forms* als auch die Programmierung mit der *Windows Presentation Foundation (WPF)* betrachtet, die mit einem Einstieg in die Programmierung von **Universellen Windows-Plattform-Apps** endet. Die *WPF* ist ein Grafik-Framework, das die Geschäftslogik mithilfe der Auszeichnungssprache *XAML* von der Präsentationslogik trennt.

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

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

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

Als Entwicklungswerkzeug wird in diesem Buch Visual Studio 2017 (Community Edition) von Microsoft genutzt. Diese Entwicklungsumgebung ist kostenfrei als Download im Internet verfügbar.

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

Dirk Hardy
E-Mail: Hardy@DirkHardy.de

Im Herbst 2018

Verlag Europa-Lehrmittel
E-Mail: Info@Europa-Lehrmittel.de

Vorbemerkung	3
Aufbau des Buches	3
Teil 1 Einführung in C#	11
1 Einführung in .NET und C#	13
1.1 Das .NET-Framework.....	13
1.1.1 Entstehung des Frameworks	13
1.1.2 Eigenschaften des .NET-Frameworks	13
1.1.3 Die Komponenten des .NET-Frameworks	14
1.1.4 Kompilierung von .NET-Programmen	14
1.2 Die Sprache C#.....	15
1.2.1 Entwicklung der Sprache C#.....	15
1.2.2 Eigenschaften der Sprache C#	15
1.2.3 Schlüsselworte in C#	16
1.2.4 Prozedurale, strukturierte und objektorientierte Programmierung	16
1.2.5 Bestandteile eines C#-Programms.....	18
2 Das erste C#-Programm.....	19
2.1 Ein C#-Projekt anlegen.....	19
2.2 Das erste C#-Programm	21
2.2.1 Das C#-Grundgerüst.....	21
2.2.2 Namensräume.....	22
2.2.3 Die Klasse Program und die Hauptmethode Main.....	23
2.2.4 Die Ausgabe auf dem Bildschirm	23
2.2.5 Wichtige Regeln eines C#-Programms	24
2.3 Grundlegende Konventionen in C#.....	24
2.3.1 Bezeichner (Namen) in C#.....	24
2.3.2 Trennzeichen.....	25
2.3.3 Kommentare in C#	26
2.4 Datentypen und Variablen.....	27
2.4.1 Variablen in C#	27
2.4.2 Elementare Datentypen	27
2.4.3 Deklaration einer Variable	29
2.4.4 Operationen auf den elementaren Datentypen.....	30
2.4.5 Der flexible Datentyp var.....	32
3 Ein- und Ausgabe unter C#	33
3.1 Ausgabe in C#	33
3.1.1 Ausgabe von Variablen	33
3.1.2 Ausgabe von Sonderzeichen.....	34
3.1.3 Formatanweisungen für WriteLine().....	35
3.2 Eingabe mit ReadLine()	36
3.2.1 Zeichenketten einlesen	36
3.2.2 Konvertierung der Eingabe	37
4 Operatoren in C#	39
4.1 Arithmetische Operatoren	39
4.1.1 Elementare Datentypen und ihre arithmetischen Operatoren	39
4.1.2 Der Modulo-Operator	40
4.1.3 Inkrement- und Dekrementoperatoren	40
4.2 Relationale und logische Operatoren.....	41
4.2.1 Relationale Operatoren.....	41
4.2.2 Logische Operatoren	41
4.3 Bit- und weitere Operatoren	42
4.3.1 Logische Bit-Operatoren.....	42
4.3.2 Bit-Schiebeoperatoren	43

4.3.3	Typumwandlung mit cast-Operatoren	44
4.3.4	Zuweisung und gekoppelte Zuweisung	45
4.3.5	Besondere Operatoren	46
4.4	Rang von Operatoren	46
5	Selektion und Iteration	48
5.1	Die Selektion	48
5.1.1	Darstellung der Selektion mit einem Programmablaufplan.....	48
5.1.2	Die einseitige Selektion mit der if-Anweisung.....	49
5.1.3	Die zweiseitige Selektion mit der if-else-Anweisung	49
5.1.4	Verschachtelte Selektionen mit if und if-else.....	52
5.1.5	Mehrfachselektion mit switch	52
5.2	Fuß-, kopf- und zählergesteuerte Iterationen	56
5.2.1	Die do-while-Schleife.....	56
5.2.2	Die while-Schleife	58
5.2.3	Die for-Schleife	59
5.2.4	Abbruch und Sprung in einer Schleife.....	61
6	Das Klassenkonzept in C#	62
6.1	Die erste Klasse in C#	64
6.1.1	Aufbau einer Klasse in C#	64
6.1.2	Werttypen und Verweistypen.....	66
6.2	Methoden in C#	66
6.2.1	Aufbau einer Methode.....	66
6.2.2	Rückgabewert einer Methode.....	68
6.2.3	Lokale Variablen	69
6.2.4	Übergabeparameter einer Methode.....	70
6.2.5	Call by value und call by reference	73
6.2.6	Überladen von Methoden.....	76
6.2.7	Zusammenfassende Hinweise zu Methoden	77
6.3	Weitere Elemente von Klassen.....	78
6.3.1	Konstruktoren und der Destruktor	78
6.3.2	Der this-Verweis.....	82
6.3.3	Statische Klassenelemente	82
6.3.4	Eigenschaften	84
6.4	Strukturen in C#	85
7	Vererbung in C#	87
7.1	Die Vererbung in C#.....	87
7.1.1	Die einfache Vererbung	87
7.1.2	Umsetzung der Vererbung in C#	88
7.1.3	Zugriff auf Attribute.....	89
7.2	Polymorphismus.....	91
7.2.1	Die Klasse object	91
7.2.2	Zuweisungen innerhalb von Vererbungshierarchien.....	92
7.2.3	Virtuelle Methoden.....	93
7.3	Abstrakte Basisklassen.....	96
7.3.1	Eine abstrakte Basisklasse	97
7.3.2	Abstrakte Methoden deklarieren	98
7.4	Interfaces in C#	99
7.4.1	Aufbau eines Interfaces.....	99
7.4.2	Das Interface IDisposable.....	101
8	Überladen von Operatoren	103
8.1	Operator-Methoden in C#	104
8.1.1	Der Aufbau einer statischen Operator-Methode.....	104
8.1.2	Regeln für die Überladung von Operatoren	106
8.2	Konvertierungsoperatoren überladen.....	107
8.2.1	Implizite und explizite Konvertierung	107
8.2.2	Implizite Konvertierungsoperatoren überladen	107
8.2.3	Explizite Konvertierungsoperatoren überladen.....	109

9	Arrays in C#	110
9.1	Ein- und mehrdimensionale Arrays.....	111
9.1.1	Eindimensionale Arrays	111
9.1.2	Die foreach-Schleife.....	113
9.1.3	Mehrdimensionale Arrays	115
9.1.4	Arrays kopieren.....	117
9.1.5	Arrays von Objekten	118
9.2	Sortieren von Arrays	120
9.2.1	Das Sortieren durch Auswahl	120
9.2.2	Statische Sortiermethode Sort	122
9.2.3	Die Interfaces IComparable und IComparer	124
9.3	Besondere Array-Klassen	127
9.3.1	Die Klasse ArrayList.....	127
9.3.2	Die Klasse Hashtable.....	129
10	Dateioperationen in C#	131
10.1	Lesen und Schreiben von Dateien	132
10.1.1	Sequenzielles Lesen und Schreiben	132
10.1.2	Den Dateizeiger positionieren.....	134
10.2	Textdateien lesen und schreiben	135
10.2.1	Textdateien mit dem StreamWriter schreiben	135
10.2.2	Textdateien mit dem StreamReader lesen	136
10.3	Methoden der Klassen File und Directory	137
10.3.1	Methoden der Klasse File	137
10.3.2	Methoden der Klasse Directory	139
11	Fortgeschrittene Themen in C#	141
11.1	Ausnahmen – Exceptions	141
11.1.1	Versuchen und Auffangen (try and catch).....	141
11.1.2	System-Exceptions	143
11.1.3	Der finally-Block:	146
11.1.4	Ausnahmen werfen	147
11.1.5	Eigene Exception-Klassen erstellen	148
11.1.6	Ausnahmen weiterleiten	150
11.2	Delegate und Lambda-Ausdrücke.....	151
11.2.1	Delegate anlegen	151
11.2.2	Lambda-Ausdrücke.....	153
11.3	Der Indexer	156
11.4	Generische Programmierung.....	158
11.4.1	Generische Methoden	158
11.4.2	Generische Klassen	160
12	Windows-Forms-Programmierung – Grundlagen.....	164
12.1	Windows-Programmierung	164
12.1.1	Historische Entwicklung der Windows-Programmierung	164
12.1.2	Ereignisgesteuerte Programmierung	164
12.1.3	Grundbegriffe der Forms-Programmierung.....	165
12.2	Das erste Windows-Forms-Programm	165
12.2.1	Ein Windows-Forms-Projekt anlegen	165
12.2.2	Das erste Windows-Forms-Programm	167
12.2.3	Eine eigene Form-Klasse schreiben.....	168
12.2.4	Das Paint-Ereignis und die erste Textausgabe.....	169
12.2.5	Grafikausgabe mit GDI+	170
12.2.6	Mehrzeilige Textausgabe und Bildlaufleisten	172
13	Der Windows-Forms-Designer und einfache Steuerelemente.....	177
13.1	Windows-Forms-Designer	177
13.1.1	Ein reines Windows-Forms-Projekt anlegen	177
13.1.2	Eigenschaften des Formulars.....	180

13.2	Einfache Steuerelemente verwenden	181
13.2.1	Die Toolbox verwenden.....	181
13.2.2	Label und Textbox.....	183
13.2.3	Auf Ereignisse reagieren.....	183
13.2.4	Buttons, Radiobuttons und Checkboxes.....	185
13.2.5	Listboxen und Kombinationsboxen	186
13.3	Standard-Dialogfelder	188
13.3.1	Datei-öffnen und Datei-speichern-Dialoge	188
13.3.2	Der Verzeichnis-suchen-Dialog.....	190
13.3.3	Dialoge für die Schriftart und Farbe	191
14	Komplexe Steuerelemente und Menüs	193
14.1	Die Baumansicht (TreeView)	193
14.1.1	Anlegen von Knoten (Nodes) in einer TreeView	193
14.1.2	Anlegen von Unterknoten	194
14.1.3	Wichtige Eigenschaften, Methoden und Ereignisse im Überblick.....	195
14.1.4	Bilder für Knoten anzeigen	195
14.2	Die Listenansicht (ListView)	196
14.2.1	Eine Listenansicht vorbereiten.....	196
14.2.2	Die Listenansicht mit Einträgen füllen	197
14.2.3	Einträge der Listenansicht abfragen	198
14.2.4	Wichtige Eigenschaften, Methoden und Ereignisse im Überblick.....	198
14.2.5	Bilder in einer Listenansicht	199
14.3	Menüs erstellen	200
14.3.1	Kontextmenüs erstellen	201
14.4	Neue Formulare hinzufügen	201
14.4.1	Ein neues Formular erstellen	201
14.4.2	Unterformulare aufrufen.....	202
15	Windows Presentation Foundation – Grundlagen	203
15.1	Windows-Programmierung mit WPF	203
15.1.1	Wichtige Aspekte der WPF	203
15.1.2	Eigenschaften der WPF	203
15.2	Das erste WPF-Programm	204
15.2.1	Ein WPF-Projekt anlegen.....	204
15.2.2	Das erste WPF-Programm	204
15.2.3	Eine eigene Fenster-Klasse schreiben.....	205
15.2.4	Eine eigene Applikationen-Klasse schreiben	207
15.3	Grundkonzepte der WPF	209
15.3.1	Das Inhalts-Konzept der WPF und die erste Textausgabe	209
15.3.2	Layout-Container	212
15.3.3	Grafikausgabe mit dem Canvas-Container	215
15.3.4	Mehrzeilige Textausgabe und Bildlaufleisten	219
16	XAML und der WPF-Designer	223
16.1	XAML	223
16.1.1	Extensible Application Markup Language XAML	223
16.1.2	Eigenschaften von XAML im Überblick	224
16.2	Der WPF-Designer	225
16.2.1	Ein WPF-Projekt anlegen.....	225
16.2.2	Den WPF-Designer einsetzen	227
16.3	Einfache und komplexe Steuerelemente	230
16.3.1	Buttons, RadioButtons und Checkboxes	230
16.3.2	Listboxen und Kombinationsboxen	231
16.3.3	Komplexes Steuerelement TreeView.....	233
16.3.4	Menüs einbinden	236
16.3.5	Weitere WPF-Steuerelemente.....	238

16.4	Weitere WPF-Konzepte	240
16.4.1	Abhängigkeitseigenschaften (Dependency Properties)	240
16.4.2	Weitergeleitete Ereignisse (Routed Events)	242
16.4.3	Datenbindungen	244
17	Datenbankzugriff mit ADO.NET	248
17.1	Datenbankzugriff	248
17.1.1	Datenbankanbindung unter dem .NET-Framework	248
17.1.2	Provider nutzen und eine Verbindung aufbauen	249
17.1.3	Beispiel eines Zugriffs auf eine ACCESS-Datenbank	249
17.1.4	Nicht-Select-Befehle absetzen	252
17.1.5	DataAdapter und DataSet	254
17.2	Den Datenbankassistenten von Visual C# nutzen	257
17.2.1	Eine Datenbank einbinden	257
17.2.2	Windows-Forms-Steuerelemente automatisch anbinden	261
17.2.3	WPF-Steuerelemente automatisch anbinden	263
18	Universelle Windows Plattform-Apps	268
18.1	Universelle Windows-Plattform	268
18.2	Die erste UWP-App	269
18.2.1	Ein leeres App-Projekt anlegen	269
18.2.2	Den Werkzeugkasten nutzen	272
18.2.3	Auf Ereignisse reagieren	273
18.3	UWP-Apps gestalten	274
18.3.1	Einfache Steuerelemente	274
18.3.2	Weitere Hinweise zur App-Programmierung	276
Teil 2	Aufgabenpool	278
1	Aufgaben zur Einführung von .NET und C#	279
2	Aufgaben zum ersten C#-Programm	279
3	Aufgaben zur Ein- und Ausgabe unter C#	279
4	Aufgaben zu Operatoren in C#	281
5	Aufgaben zur Selektion und Iteration	282
6	Aufgaben zum Klassenkonzept in C#	286
7	Aufgaben zur Vererbung in C#	290
8	Aufgaben zur Überladung von Operatoren in C#	292
9	Aufgaben zu Arrays in C#	294
10	Aufgaben zu Dateioperationen in C#	299
11	Aufgaben zu fortgeschrittenen Themen in C#	304
12	Aufgaben zu den Grundlagen der Windows-Forms-Programmierung	308
13	Aufgaben zum Designer und einfachen Steuerelementen	311
14	Aufgaben zu komplexen Steuerelementen und Menüs	313
15	Aufgaben zu den WPF-Grundlagen	316
16	Aufgaben zu XAML und dem WPF-Designer	319
17	Aufgaben zur Datenbankanbindung	323
18	Aufgaben zur App-Entwicklung	325
Teil 3	Lernsituationen	328
Lernsituation 1:	Erstellen einer Präsentation mit Hintergrundinformationen zu der Sprache C# (in Deutsch oder Englisch)	329
Lernsituation 2:	Anfertigen einer Kundendokumentation für den Einsatz einer Entwicklungsumgebung in C# (in Deutsch oder Englisch)	330
Lernsituation 3:	Entwicklung eines Verschlüsselungsverfahrens für ein internes Memo-System der Support-Abteilung einer Netzwerk-Firma	332
Lernsituation 4:	Planung, Implementierung und Auswertung eines elektronischen Fragebogens	333

Lernsituation 5: Implementierung einer Klasse zur Simulation der echten Bruchrechnung.....	336
Lernsituation 6: Entwicklung einer Terminverwaltungssoftware mit Datenbankanbindung.....	338

Anhang: Index	342
----------------------------	------------

Teil Einführung in C#

1.1	Das .NET-Framework.....	13
1.2	Die Sprache C#	15
2.1	Ein C#-Projekt anlegen	19
2.2	Das erste C#-Programm	21
2.3	Grundlegende Konventionen in C#	24
2.4	Datentypen und Variablen.....	27
3.1	Ausgabe in C#.....	33
3.2	Eingabe mit ReadLine()	36
4.1	Arithmetische Operatoren	39
4.2	Relationale und logische Operatoren.....	41
4.3	Bit- und weitere Operatoren.....	42
4.4	Rang von Operatoren.....	46
5.1	Die Selektion	48
5.2	Fuß-, kopf- und zählergesteuerte Iterationen	56
6.1	Die erste Klasse in C#	64
6.2	Methoden in C#.....	66
6.3	Weitere Elemente von Klassen	78
6.4	Strukturen in C#.....	85
7.1	Die Vererbung in C#.....	87
7.2	Polymorphismus	91
7.3	Abstrakte Basisklassen.....	96
7.4	Interfaces in C#	99
8.1	Operator-Methoden in C#	104
8.2	Konvertierungsoperatoren überladen	107
9.1	Ein- und mehrdimensionale Arrays	111
9.2	Sortieren von Arrays	120
9.3	Besondere Array-Klassen.....	127
10.1	Lesen und Schreiben von Dateien	132
10.2	Textdateien lesen und schreiben	135
10.3	Methoden der Klassen File und Directory	137
11.1	Ausnahmen – Exceptions.....	141
11.2	Delegate und Lambda-Ausdrücke.....	151
11.3	Der Indexer	156
11.4	Generische Programmierung.....	158
12.1	Windows-Programmierung	164
12.2	Das erste Windows-Forms-Programm	165
13.1	Windows-Forms-Designer.....	177

13.2	Einfache Steuerelemente verwenden	181
13.3	Standard-Dialogfelder	188
14.1	Die Baumansicht (TreeView)	193
14.2	Die Listenansicht (ListView)	196
14.3	Menüs erstellen	200
14.4	Neue Formulare hinzufügen	201
15.1	Windows-Programmierung mit WPF	203
15.2	Das erste WPF-Programm	204
15.3	Grundkonzepte der WPF	209
16.1	XAML	223
16.2	Der WPF-Designer	225
16.3	Einfache und komplexe Steuerelemente	230
16.4	Weitere WPF-Konzepte	240
17.1	Datenbankzugriff	248
17.2	Den Datenbankassistenten von Visual C# nutzen	257
18.1	Universelle Windows-Plattform	268
18.2	Die erste UWP-App	269
18.3	UWP-Apps gestalten	274

1 Einführung in .NET und C#

1.1 Das .NET-Framework

1.1.1 Entstehung des Frameworks

In den 90er-Jahren wurde mit Java eine Technik geschaffen, die nicht nur sehr erfolgreich war, sondern auch die Zukunft von Microsoft im Bereich der Programmierung ernsthaft gefährden konnte. Das lag einerseits an der modernen objektorientierten Programmiersprache Java, aber auch an der Plattformunabhängigkeit von Java-Programmen, die mithilfe der Java-Laufzeitumgebung auf den verschiedensten Rechnern und Betriebssystemen ausgeführt werden können. Aus diesen Gründen brauchte Microsoft eine Antwort auf diese neue Technik – und zwar das .NET-Framework. Das Framework kann als eine Weiterentwicklung der Java-Technologie gesehen werden, allerdings zugeschnitten auf die Windows-Betriebssysteme. Inzwischen gibt es auch eine weitere Variante des .NET-Frameworks, das so genannte MONO-Projekt mit der Entwicklungsumgebung Xamarin Studio. Die folgende Grafik zeigt den zeitlichen Verlauf der .NET-Framework-Entwicklung:

Jahr 2000	Microsoft stellt das erste .NET-Framework in einer Vorabversion vor.	Die Sprache C# wird zur Standardisierung eingereicht.
Jahr 2002	.NET-Framework Version 1.0 und das Visual Studio .NET 2002 werden vorgestellt.	Programmiersprachen: • C# • J# [bis 2007] • VB • F# [ab 2010] • C++/CLI [bis 2012] • C++/CX [ab 2012] • Javascript [ab 2012]
Jahr 2005	.NET-Framework Version 2.0 und das Visual Studio .NET 2005 werden herausgebracht.	
Jahr 2008	.NET-Framework Version 3.5 und Visual Studio .NET 2008 sind auf dem Markt.	
Jahr 2010	.NET-Framework Version 4 und Visual Studio .NET 2010 werden herausgebracht.	
Jahr 2012	.NET-Framework Version 4.5 Visual Studio .NET 2012	
Jahr 2015	.NET-Framework Version 4.6 Visual Studio .NET 2015	
Jahr 2018	.NET-Framework Version 4.7 Visual Studio .NET 2017	

**Windows
Store
Apps**

1.1.2 Eigenschaften des .NET-Frameworks

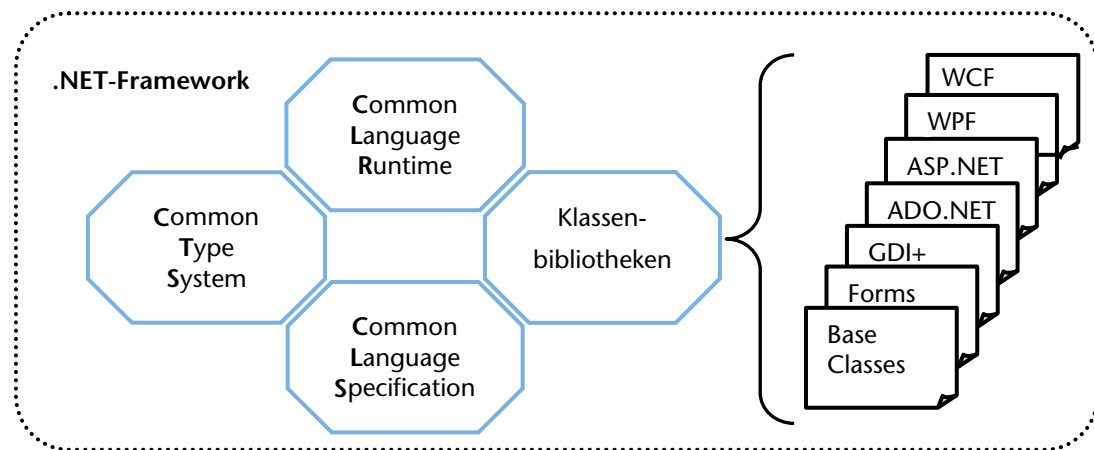
Der große Vorteil des .NET-Frameworks liegt darin, dass es auf den verschiedenen Windows-Betriebssystemen installiert werden kann. Damit sind Programme auf den verschiedenen Windows-Betriebssystemen lauffähig – vorausgesetzt, das entsprechende .NET-Framework ist vorhanden. Die verschiedenen Versionen des Frameworks können alle parallel installiert werden, so dass einem .NET-Programm immer die richtige Version zur Verfügung stehen kann. Die wichtigsten Eigenschaften des Frameworks sind:

- **Sprachunabhängigkeit:** Ein .NET-Programm kann in verschiedenen Sprachen geschrieben werden. Der Zugriff aus einer Sprache wie C# auf Klassen aus anderen Sprachen wie C++/CX ist möglich.
- **Objektorientierung:** So wie in der Java-Technologie ist die Programmierung unter .NET vollständig objektorientiert.

- **Verwalteter Code** (managed code): Ein .NET-Programm läuft in einer eigenen Laufzeitumgebung und kann besser kontrolliert werden. Beispielsweise werden die Speicherverwaltung und die automatische Speicherbereinigung durch die Laufzeitumgebung geregelt.
- **Plattformunabhängigkeit**: Diese Eigenschaft ist nur beschränkt gültig, da das Framework eigentlich für Windows-Betriebssysteme konzipiert ist. Andere Projekte (wie das MONO-Projekt) schaffen die Möglichkeit, .NET-Programme auch auf andere Betriebssysteme zu portieren.

1.1.3 Die Komponenten des .NET-Frameworks

Das .NET-Framework besteht aus verschiedenen Komponenten, die dafür sorgen, dass die oben beschriebenen Eigenschaften umgesetzt werden können. Neben der Laufzeitumgebung (Common Language Runtime, **CLR**), in der die .NET-Programme ausgeführt werden, definiert eine Sprachspezifizierung (Common Language Specification, **CLS**) die Anforderungen, die eine .NET-Programmiersprache haben muss. Beispielsweise muss der Index eines Arrays immer mit Null beginnen. In einer Typspezifizierung (Common Type System, **CTS**) wird zusätzlich ein sprachunabhängiges Datentypensystem festgelegt, mit dem eine .NET-Programmiersprache arbeiten können muss. Daneben verfügt das .NET-Framework über eine sehr mächtige Klassenbibliothek, mit der nicht nur viele grundlegende Funktionalitäten bereitgestellt sondern auch die Windows-Programmierung, die Internet-Programmierung oder auch Datenbankanbindungen realisiert werden. Die nächste Abbildung zeigt diese Komponenten noch einmal im Überblick.



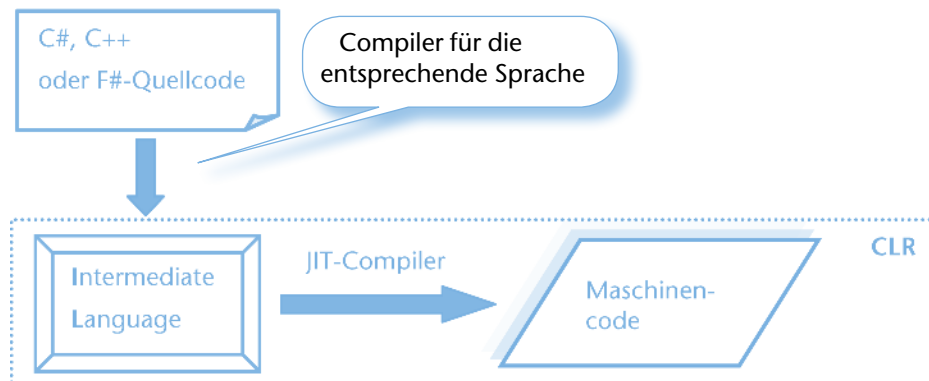
Mit den verschiedenen Klassenbibliotheken können die meisten Anwendungen realisiert werden. Die einzelnen Bibliotheken sind dabei für die folgenden Bereiche verantwortlich:

- ▶ **Base Classes (Base Class Library)**: eine Sammlung von Klassen für elementare Funktionalitäten wie Dateioperationen, mathematische Funktionen oder auch reguläre Ausdrücke.
- ▶ **Windows-Forms**: Die Klassen sind die Grundlage für die Entwicklung von Windows-Applikationen mit klassischen Elementen wie Fenstern, Menüs, Dialogen oder Buttons.
- ▶ **GDI+ (Graphic Device Interface +)**: Mit dieser Schnittstelle werden grafische Ausgaben in einem Fenster oder auf einem Drucker realisiert. Linien, Rechtecke, Bitmaps und weitere Elemente können angezeigt werden.
- ▶ **ADO.NET (ActiveX Data Objects .NET)**: Mit diesen Klassen werden Datenbanken angesprochen. Sie sind auch die Grundlage für verteilte Anwendungen.
- ▶ **ASP.NET (Active Server Pages .NET)**: Die Entwicklung von Webanwendungen wird mithilfe dieser Bibliothek realisiert.
- ▶ **WPF (Windows Presentation Foundation)**: Die Entwicklung von modernen GUI-Applikationen wird mit dieser Bibliothek realisiert.
- ▶ **WCF (Windows Communication Foundation)**: Erlaubt die Entwicklung von verteilten Anwendungen.

1.1.4 Kompilierung von .NET-Programmen

Ein .NET-Programm wird nicht mehr direkt in eine ausführbare Datei, sondern in eine Art Zwischencode (Intermediate Language, **IL**) übersetzt. Dieser Zwischencode wird dann von der .NET-Laufzeitumgebung ausgeführt. Dabei übersetzt der so genannte **Just in time-Compiler (JIT-Compiler)** den Zwischencode in nativen Code, der dann auf der jeweiligen Plattform ausführbar ist. Während der Ausführung überwacht die CLR dabei sicherheitsrelevante Aspekte und sorgt mit einem speziellen Dienst (dem **garbage-collector**) dafür, dass nicht mehr benötigter Speicher freigegeben wird. Das ganze System ist vergleichbar mit dem Java-Bytecode und den virtuellen Maschinen der Java-Plattformen.

Die folgende Abbildung zeigt den schematischen Ablauf einer Kompilierung:

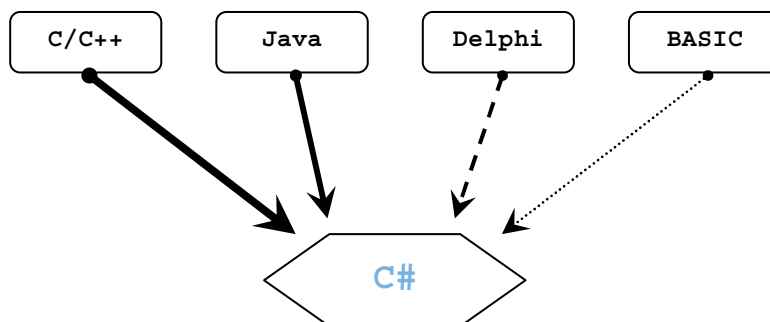


Das Ergebnis der Kompilierung in die Intermediate Language nennt man **Assembly**. Eine Assembly kann in Form einer .exe- oder einer .dll-Datei vorliegen. Ohne die CLR bzw. das .NET-Framework kann eine solche Datei allerdings nicht gestartet werden, auch wenn die Dateierendung an die bekannten ausführbaren Programme unter Windows erinnert. In einer Assembly stehen neben dem Programmcode weitere Informationen wie beispielsweise die benötigten Framework-Klassen (in bestimmten Versionen) und weitere Abhängigkeiten.

1.2 Die Sprache C#

1.2.1 Entwicklung der Sprache C#

Parallel zur ersten Vorabversion von .NET stellte Microsoft im Jahr 2000 die Sprache C# vor. Sie wurde im gleichen Jahr bei der ECMA¹ zur Standardisierung eingereicht und im Jahre 2003 auch von der ISO² genormt. C# wurde im Rahmen der .NET-Technologie entwickelt und ist deshalb auch perfekt auf die Besonderheiten von .NET abgestimmt. Die Federführung bei der Entwicklung von C# hatte *Anders Hejlsberg*, der als Chefentwickler der Programmiersprache Delphi große Erfahrung in der Entwicklung einer objektorientierten Programmiersprache hatte. Die Sprache C# vereinigt viele Vorteile anderer Programmiersprachen – vor allem der Sprachen C++ und Java. Die Nähe zu C++ wird vor allem durch die Syntax deutlich, denn die meisten elementaren Anweisungen sehen fast identisch aus. Von Java wurde beispielsweise das Konzept der Verweistypen übernommen, so dass C# keine Zeiger verwenden muss. Ihren Namen verdankt die Sprache einerseits der Programmiersprache C/C++ und andererseits einem Symbol aus der Musik. Die Raute „#“ soll dabei für das Kreuz stehen, das einen Ton um einen Halbton erhöht. Damit soll deutlich werden, dass C# eine Weiterentwicklung der Sprache C/C++ ist.



1.2.2 Eigenschaften der Sprache C#

Die folgenden Eigenschaften zeichnen die Sprache C# aus:

- ▶ Moderne, objektorientierte Sprache
- ▶ „Etwas“ einfacher zu erlernen als C++ (Zeiger müssen nicht verwendet werden.)
- ▶ Plattformunabhängig konzipiert (beispielsweise auch für Linux mit dem MONO-Projekt verwendbar)
- ▶ Schnelle und effektive Softwareentwicklung (Windows-Anwendungen, Web-Anwendungen) mit Unterstützung durch mächtige .NET-Klassenbibliotheken
- ▶ Komfortable Anbindung von beliebigen Datenbanken

¹ ECMA ist eine private Organisation zur Normung von Informations- und Telekommunikationssystemen. ECMA hat das Ziel, Standards zu entwickeln und dabei mit anderen Normungsorganisationen zusammenzuarbeiten.

² ISO ist die Internationale Organisation für Normung. Sie vereinigt die unterschiedlichen Normungsorganisationen der Länder wie beispielsweise das Deutsche Institut für Normung DIN.

1.2.3 Schlüsselworte in C#

- Die Sprache C# hat einen Wortschatz³ von ungefähr 80 reservierten Worten – den so genannten **Schlüsselworten**. Die Schlüsselworte sind die Grundlage der Programme in C#. Die folgende Tabelle zeigt die Schlüsselworte von C#:

<code>abstract</code>	<code>as</code>	<code>base</code>	<code>bool</code>
<code>break</code>	<code>byte</code>	<code>case</code>	<code>catch</code>
<code>char</code>	<code>checked</code>	<code>class</code>	<code>const</code>
<code>continue</code>	<code>decimal</code>	<code>default</code>	<code>delegate</code>
<code>do</code>	<code>double</code>	<code>else</code>	<code>enum</code>
<code>event</code>	<code>explicit</code>	<code>extern</code>	<code>false</code>
<code>finally</code>	<code>fixed</code>	<code>float</code>	<code>for</code>
<code>foreach</code>	<code>goto</code>	<code>if</code>	<code>implicit</code>
<code>in</code>	<code>int</code>	<code>interface</code>	<code>internal</code>
<code>is</code>	<code>lock</code>	<code>long</code>	<code>namespace</code>
<code>new</code>	<code>null</code>	<code>object</code>	<code>operator</code>
<code>out</code>	<code>override</code>	<code>params</code>	<code>private</code>
<code>protected</code>	<code>public</code>	<code>readonly</code>	<code>ref</code>
<code>return</code>	<code>sbyte</code>	<code>sealed</code>	<code>short</code>
<code>sizeof</code>	<code>stackalloc</code>	<code>static</code>	<code>string</code>
<code>struct</code>	<code>switch</code>	<code>this</code>	<code>throw</code>
<code>true</code>	<code>try</code>	<code>typeof</code>	<code>uint</code>
<code>ulong</code>	<code>unchecked</code>	<code>unsafe</code>	<code>ushort</code>
<code>using</code>	<code>virtual</code>	<code>volatile</code>	<code>void</code>
<code>while</code>	<code>where</code> (kontextabhängig)	<code>var</code> (kontextabhängig)	<code>partial</code> (kontextabhängig)

Die Bedeutungen der einzelnen Schlüsselworte werden Schritt für Schritt im Laufe dieses Informationsteils erklärt.

1.2.4 Prozedurale, strukturierte und objektorientierte Programmierung unter C#

In der Programmierung können verschiedene Paradigmen⁴ unterschieden werden. Es gibt Sprachen wie C, mit denen beispielsweise nur strukturiert (und auch prozedural) programmiert werden kann. Andere Sprachen wie C++ können sowohl strukturiert (und prozedural) als auch objektorientiert programmiert werden. Die Sprache C# ist hingegen eine rein objektorientierte Sprache. Trotzdem spielt die strukturierte Programmierung auch bei C# eine Rolle, denn innerhalb des objektorientierten Rahmens muss auch strukturiert programmiert werden.

Zum besseren Verständnis werden diese Begriffe kurz erläutert:

Strukturierte Programmierung

Die strukturierte Programmierung zeichnet sich durch Kontrollstrukturen wie die **Auswahl** (IF-ELSE) oder die **Wiederholungen** (FOR, WHILE usw.) aus. Damit erhält ein Programm eine nachvollziehbare Struktur. In den Anfängen der Programmierung war es üblich, Sprunganweisungen (GOTO) in einem Programm zu benutzen. Dadurch wird ein Programm sehr unübersichtlich und fehleranfällig. Strukturierte Programme sind hingegen übersichtlicher und besser wartbar.

Beispiel:

```
FÜR Var := 1 BIS 5 MIT SCHRITTWEITE 1
    SCHREIBE AUF BILDSCHIRM Var
```

```
1
2
3
4
5
```

³ Die Anzahl der Schlüsselworte ist abhängig von der jeweiligen Version. Spätere Versionen haben in der Regel mehr Schlüsselworte. Die obige Angabe bezieht sich auf die Version 7.

⁴ Paradigma kommt aus dem Griechischen und heißt so viel wie Muster oder Vorbild.

Das Beispiel zeigt eine Wiederholung in so genanntem Pseudocode⁵. Dieser Code beschreibt den Ablauf des Programmes, ohne allerdings auf eine spezielle Programmiersprache einzugehen. In dem Beispiel wird eine Variable `Var` so lange um 1 erhöht, bis der Wert 5 erreicht ist. Jeder Wert der Variablen wird dann auf dem Bildschirm ausgegeben.

Prozedurale Programmierung

Die prozedurale Programmierung teilt Programme in kleine Einheiten (Prozeduren oder Funktionen), die für bestimmte Aufgaben verantwortlich sind. Sind diese Prozeduren einmal geschrieben und getestet, dann können sie immer wieder benutzt werden – das spart Entwicklungszeit und führt auch zu einer besseren Lesbarkeit des Programms.

Beispiel:

```
PROZEDUR Ausgabe
    SCHREIBE AUF BILDSCHIRM "Hallo"
ENDE

FÜR Var := 1 BIS 5 MIT SCHRITTWEITE 1
    AUFRUF Ausgabe
```

```
Hallo
Hallo
Hallo
Hallo
Hallo
```

Das Beispiel in Pseudocode zeigt eine Prozedur mit dem Namen `Ausgabe`. Diese Prozedur hat eine Anweisung, die das Wort „Hallo“ auf den Bildschirm schreibt. Die bereits bekannte Wiederholung aus dem Beispiel vorher läuft dann 5-mal und ruft jedes Mal die Prozedur `Ausgabe` auf. Damit steht 5-mal das Wort „Hallo“ auf dem Bildschirm.

Objektorientierte Programmierung

Die objektorientierte Programmierung möchte Objekte der realen Welt in einem Programm abbilden. Damit sollen Problemstellungen aus beliebigen Bereichen (Geschäftsprozesse, wissenschaftliche Untersuchungen usw.) geeigneter als mit den anderen Programmierparadigmen in Programme umgesetzt werden können.

Im Mittelpunkt der objektorientierten Programmierung steht die **Klasse**, aus der dann konkrete Objekte gebildet werden. Diese Objekte haben bestimmte Eigenschaften (Attribute) und so genannte Methoden, mit denen diese Eigenschaften beispielsweise verändert werden können.

Beispiel:

```
KLASSE Kunde
    Name
    Telefon
ENDE

BILDE OBJEKT K1 VON Kunde
K1.Name    := "Maier"
K1.Telefon := "123456"

SCHREIBE AUF BILDSCHIRM K1.Name und K1.Telefon
```

```
Maier
123456
```

In dem Beispiel wird eine Klasse `Kunde` definiert. Von dieser Klasse können dann konkrete Objekte wie `K1` (für Kunde 1) gebildet werden. Die Eigenschaften des Objektes (`Name`, `Telefon`) können dann mit Werten belegt werden. In diesem Beispiel erhält das Objekt `K1` den Namen „Maier“ und die Telefonnummer „123456“. Anschließend werden `Name` und `Telefon` des Objektes auf den Bildschirm geschrieben.

⁵ Pseudocode ist eine Art Sprache, mit der der Ablauf eines Programmes beschrieben wird. Pseudocode zeichnet sich dadurch aus, dass er näher an der natürlichen Sprache als an einer Programmiersprache ist. Ein Programm, das in Pseudocode geschrieben ist, kann problemlos in jede Programmiersprache übersetzt werden.

1.2.5 Bestandteile eines C#-Programms

Ein C#-Programm besteht aus einer Folge von endlich vielen und eindeutigen Anweisungen⁶, die mithilfe der Schlüsselworte und selbst gewählter Namen für bestimmte Elemente wie Klassen oder Objekte gebildet werden. Zusätzlich kann ein C#-Programm auch Anweisungen enthalten, die nicht zum eigentlichen Programm gehören, aber die Erstellung des Programms steuern. Das folgende Beispiel zeigt ein einfaches C#-Programm:

```
#define DEBUG
```

Das ist kein C#-Befehl, sondern ein so genannter Präprozessor-Befehl, der vor der Übersetzung ausgeführt wird.

```
using System;
```

Der Namensraum System wird verwendet.

```
namespace EIGENER_NAMENSRaum  
{
```

Ein eigener Namensraum wird definiert.

Eine Klasse wird definiert.

```
class Program  
{
```

Das „Hauptprogramm“

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

```
{
```

```
    Anweisung 1;
```

```
    Anweisung 2;
```

```
    :
```

```
    :
```

```
    Anweisung N;
```

```
}
```

```
}
```

```
}
```

Verschiedene C#-Anweisungen

In dem obigen Beispiel wird deutlich, dass auch ein einfaches C#-Programm schon einen relativ komplizierten Aufbau hat. Das liegt daran, dass C# eine vollständig objektorientierte Sprache ist und deshalb immer auch eine Klasse definiert werden muss. Dieser Aufbau wird nun in den folgenden Kapiteln Schritt für Schritt erläutert.

⁶ Eine endliche Folge von eindeutigen Anweisungen an den Computer nennt man **Algorithmus**.

2 Das erste C#-Programm

2.1 Ein C#-Projekt anlegen

Die integrierte Entwicklungsumgebung Visual C# ist eine komfortable Umgebung, um C#-Programme zu entwickeln. Besonders erfreulich ist der Umstand, dass die Umgebung als Community Edition kostenfrei im Internet bereit steht. Ein C#-Programm besteht aus einer oder mehreren Quellcodedateien. Diese Dateien werden in einem Projekt organisiert. Visual C# unterscheidet im Prinzip folgende Projektarten:

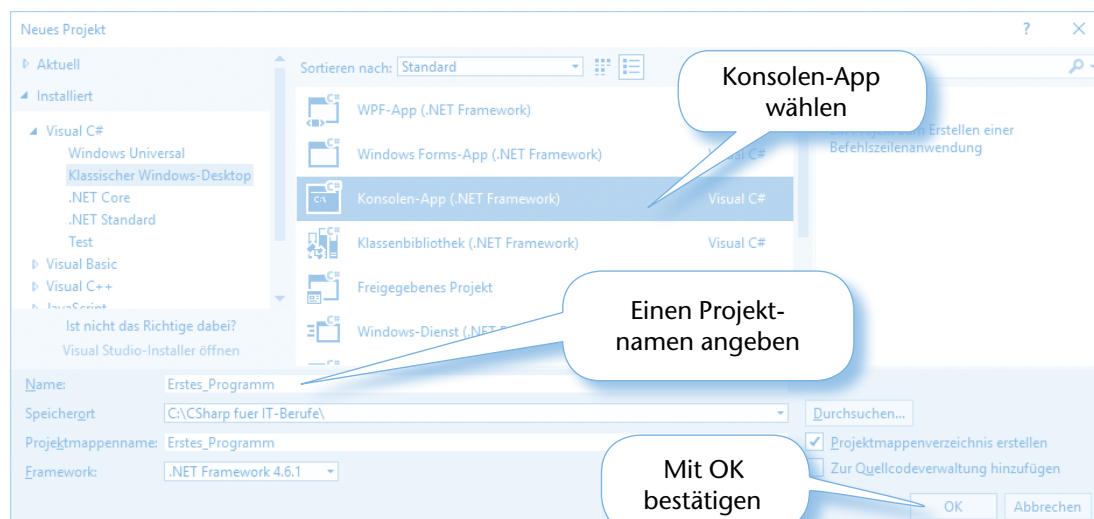
- Desktop-Apps:
 - Windows-Forms-App (Windows-Applikation unter .NET)
 - WPF-App (Moderne GUI-Applikationen unter .NET)
 - Konsolen-App (ähnlich einem DOS-Programm)
 - Bibliotheken (Sammlung von Funktionalitäten bzw. Klassen)
- Universelle Windows-Apps:
 - Leere App (Einfache App ab Windows 10)
 - Bibliotheken / Komponenten (ab Windows 10)
- .NET-Core Apps:
 - Konsolen-App (.NET-Core)
 - Bibliotheken (.NET-Core)

In diesem Buch sind hauptsächlich zwei Projektarten von Bedeutung: die Desktop-Apps und später die leere Universelle Windows-App. Bei den Desktop-Anwendungen werden zuerst die Konsolenanwendung und dann die Windows-Forms-Anwendung sowie die WPF-Anwendung behandelt. Die Konsolenanwendung ist ausreichend, um eine einfache Ein- und Ausgabemöglichkeit für die ersten C#-Programme zu haben. Die Konsolenanwendung ist natürlich nicht so ansprechend wie ein Windows-Programm, aber, um die Grundlagen der Sprache C# zu lernen, völlig ausreichend.

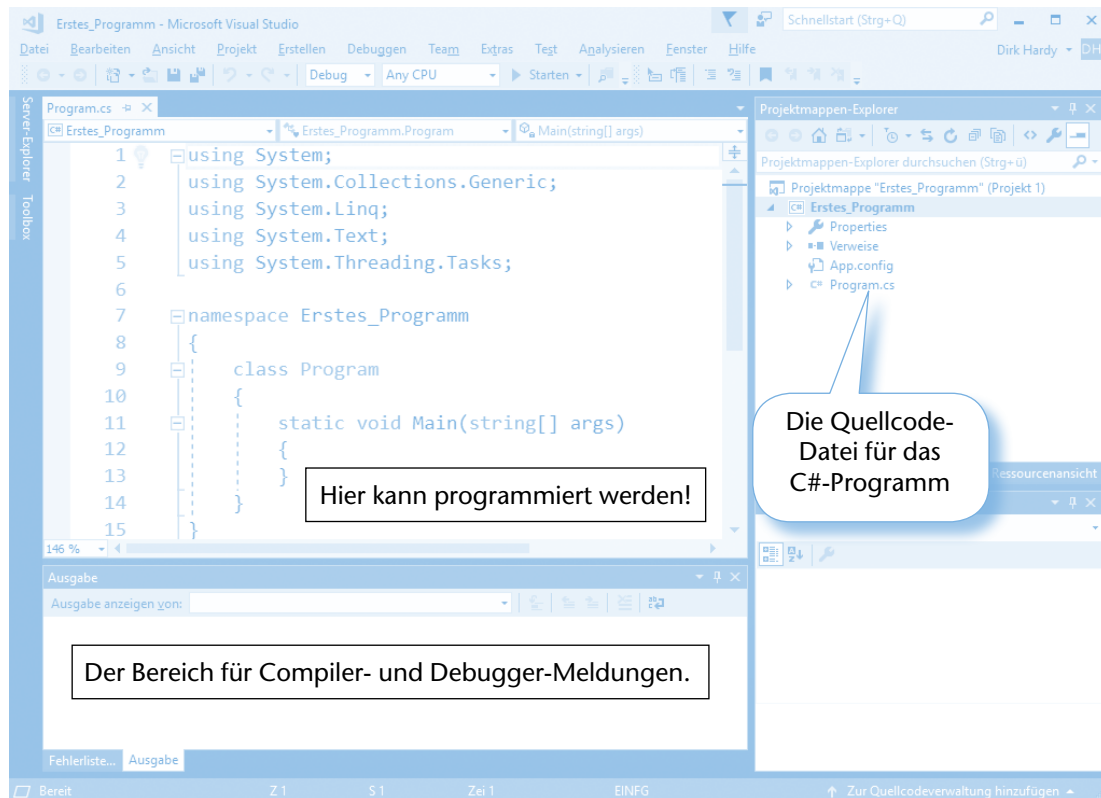
Die Konsolenanwendung ist natürlich nicht so ansprechend wie ein Windows-Programm, aber, um die Grundlagen der Sprache C# zu lernen, völlig ausreichend.

Anlegen eines neuen Projektes:

- Starten Sie Visual C# 2017
- Wählen Sie den Menüpunkt Datei → Neu → Projekt.



Nach dem Bestätigen mit „OK“ wird ein neues Projekt angelegt und in der Entwicklungsumgebung angezeigt.



Die Entwicklungsumgebung hat ein Projekt mit dem gewählten Namen (hier „Erstes_Programm“) angelegt. Zusätzlich zum Projekt wurde eine Projektmappe mit demselben Namen angelegt. Innerhalb dieser Projektmappe können beliebig viele weitere Projekte angelegt werden. Der Projektmappenname kann auch anders benannt werden (auf die rechte Maustaste über dem Namen klicken und „Umbenennen“ wählen). Innerhalb des Projektes sind Properties (Eigenschaften), Verweise und die Quellcode-Datei „Programm.cs“ angelegt. Unter den Eigenschaften können Informationen zu dem Projekt abgerufen werden (Assembly-Informationen) und über die Verweise können weitere Bibliotheken eingebunden werden, die dann für das aktuelle Programm zur Verfügung stehen. Beispielsweise wird bei einer Konsolenanwendung immer die System-Assembly eingebunden, in der alle grundlegenden Funktionalitäten für ein C#-Programm vorhanden sind. In der Quellcode-Datei „Programm.cs“ ist bereits ein Grundgerüst vorhanden, welches ein lauffähiges C#-Programm darstellt – allerdings ohne Funktionalitäten.

Ausführen eines C#-Programms

Um das Programm zu compilieren und anschließend auszuführen gibt es verschiedene Möglichkeiten unter Visual C#:

- Menüpunkt: Debuggen → Starten ohne Debugging
- Tastenkombination: STRG + F5

Fortgeschrittene werden später das Starten mit Debugging (Menüpunkt: Debuggen → Debugging starten oder F5 drücken) verwenden, wenn kompliziertere Programme analysiert werden müssen. Für den Anfang ist jedoch die oben beschriebene Vorgehensweise völlig ausreichend.

Nach dem Starten des obigen ersten Programms erscheint dann folgendes Fenster:

