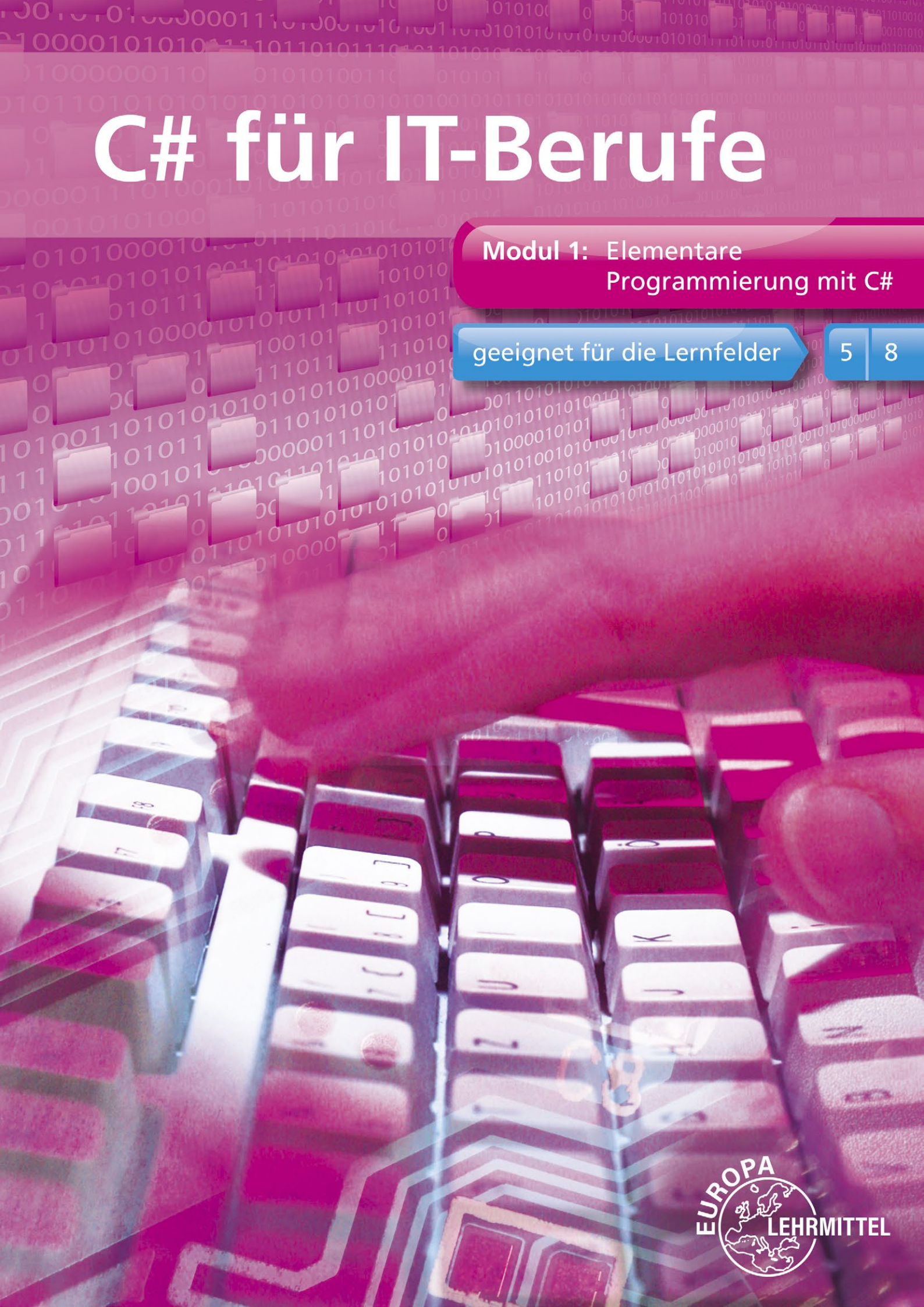
C# für IT-Berufe

Modul 1: Elementare Programmierung mit C#

geeignet für die Lernfelder 5 8



EUROPA LEHRMITTEL

EUROPA-FACHMODULREIHE
Programmierung für die
IT-Ausbildung

C# für IT-Berufe

```
using System;

namespace CSharp_IT_Berufe
{
  class Program
  {
     static void Main(string[] args)
     {
       Console.WriteLine("Informationsteil:");
       Console.WriteLine("     Elementare");
       Console.WriteLine("     Programmierung");
       Console.WriteLine("     mit C#");
       Console.WriteLine();
       Console.WriteLine("Aufgabenpool");
       Console.WriteLine();
       Console.WriteLine("Lernsituationen");
       Console.WriteLine();
     }
  }
}
```

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

Europa-Nr.: 87423L (4-Jahreslizenz) · Europa-Nr. 87423V (Jahreslizenz)

Verfasser:
Dirk Hardy, 46049 Oberhausen

1. Auflage 2024

978-3-8085-8742-3 (4-Jahreslizenz)
978-3-8085-8776-8 (Jahreslizenz)

www.europa-lehrmittel.de
Satz: Typework Layoutsatz & Grafik GmbH, 86153 Augsburg
Titel: braunwerbeagentur, 42477 Radevormwald
Titelfotos: Christian Worring-adobestock.com; imageteam-adobestock.com; bilderbox-adobestock.com

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 Software-Plattform **.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 Moduls

Das vorliegende **Modul 1** möchte die elementare Programmierung mit der Sprache C# möglichst anschaulich, praxis- und unterrichtsnah vermitteln. Damit verfolgt dieses Modul 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 Modul ist in **drei Teile** getrennt. Der **erste Teil** des Moduls dient als **Informationsteil** und bietet eine **systematische Einführung in die elementare Programmierung (strukturierte Programmierung) mit der Sprache C# und in die Grundlagen von .NET.**

Der **zweite Teil** des Moduls 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 Moduls beinhaltet **Lernsituationen** basierend auf den Lernfeldern „Benutzerschnittstellen gestalten und entwickeln", „Funktionalität in Anwendungen realisieren" und „Kundenspezifische Anwendungsentwicklung durchführen" aus dem aktuellen 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 Modul 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 Modul **Visual Studio 2022** (Community Edition) von Microsoft genutzt. Diese Entwicklungsumgebung ist kostenfrei als Download im Internet verfügbar.

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

Dirk Hardy

E-Mail: Hardy@DirkHardy.de

Im Frühjahr 2024

Verlag Europa-Lehrmittel

E-Mail: Info@Europa-Lehrmittel.de

1 Einführung in .NET und C#

1.1 Das .NET-Framework und .NET

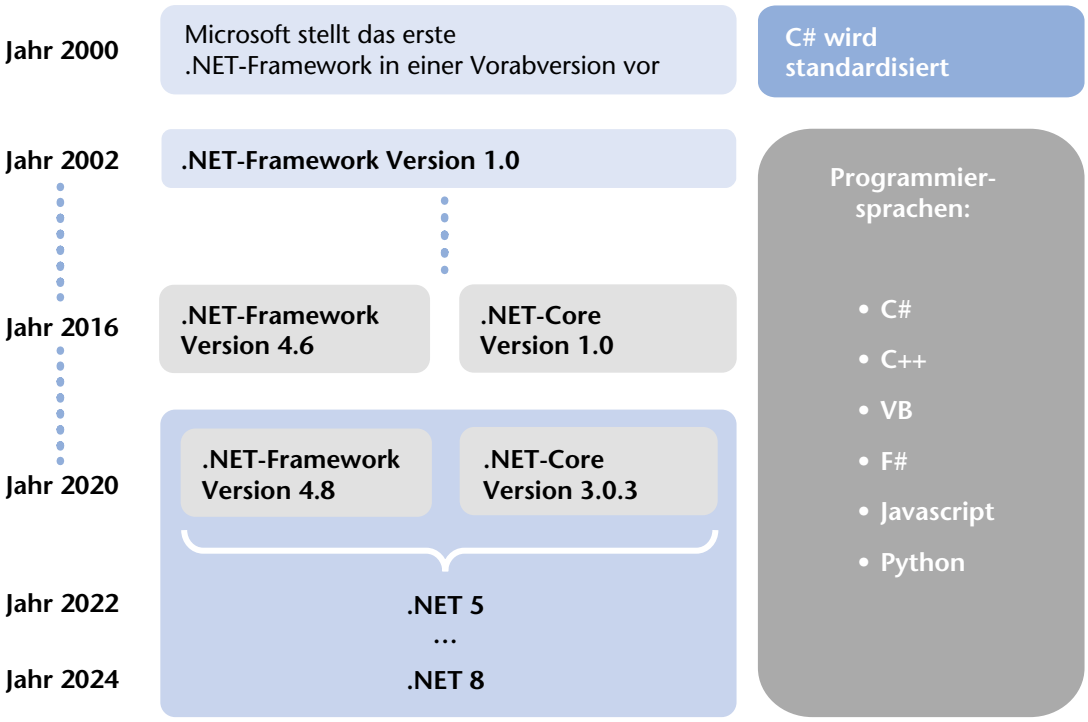
1.1.1 Entstehung des .NET-Frameworks

In den 90-er 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. Mit dem MONO-Projekt wurde aber auch eine Variante geschaffen, die auf Linux-Betriebssystemen läuft.

1.1.2 Entstehung von .NET Core und .NET 5+

Das .NET-Framework war sehr erfolgreich in den ersten Jahren, hatte aber immer die Problematik, dass es nur auf Windows-Systemen einsetzbar war. Mit der Einführung von **.NET Core** im Jahr 2016 reagierte Microsoft auf diesen Umstand und brachte ein plattformunabhängiges und quelloffenes Framework auf den Markt. Bis zum Jahr 2020 wurde **.NET Core** parallel zum **.NET-Framework** entwickelt. Ab dem Jahr 2020 wurden beide Systeme zu **.NET 5** zusammengeführt und in den Jahren danach folgten **.NET 6** und **.NET 7.** Das ursprüngliche **.NET-Framework** wird hingegen nicht weiterentwickelt und bleibt auf dem letzten Stand als Version 4.8 bestehen – es wird aber weiterhin von Microsoft unterstützt, das es immer noch unzählige Anwendungen gibt, die auf diesem Framework aufbauen. Die neuen Frameworks sollen aber die Zukunft der Softwareentwicklung sein.

Die folgende Grafik zeigt den zeitlichen Verlauf der .NET-Entwicklung:



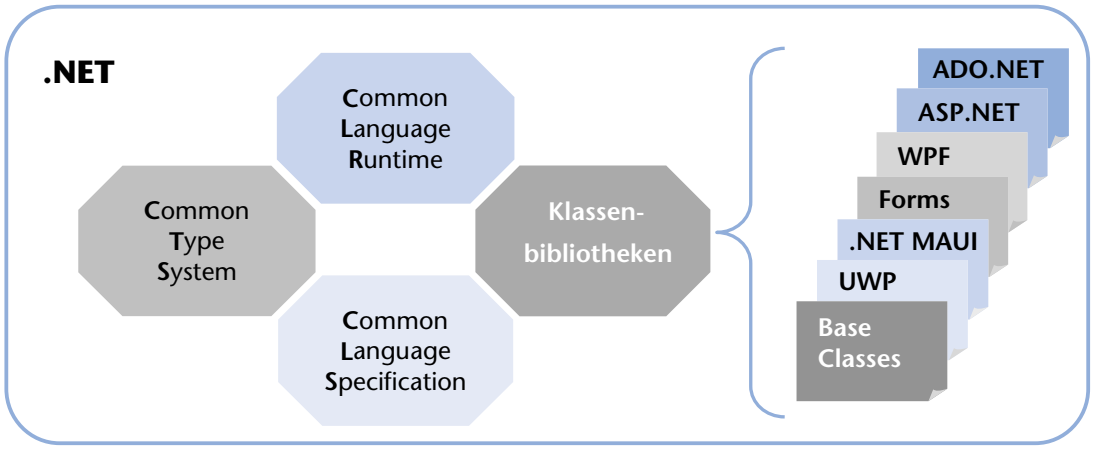
1.1.3 Eigenschaften von .NET

Der große Vorteil von .NET liegt inzwischen in der Plattformunabhängigkeit – bis 2016 war das ursprüngliche .NET-Framework auf verschiedenen Windows-Betriebssystemen lauffähig und nur das MONO-Projekt brachte eine gewisse Plattformunabhängigkeit. Das hat sich mit .NET Core und den Nachfolgern .NET 5+ geändert. Die wichtigsten Eigenschaften von .NET sind:

- **Sprachunabhängigkeit:** Ein .NET-Programm kann in verschiedenen Sprachen geschrieben werden.
- **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:** Programme können auf verschiedene Plattformen portiert werden.

1.1.4 Die Komponenten von .NET

.NET 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 Datentypen-System festgelegt, mit dem eine .NET-Programmiersprache arbeiten können muss. Daneben verfügt .NET über eine sehr mächtige Klassenbibliothek, mit der nicht nur viele grundlegende Funktionalitäten bereitgestellt werden, sondern auch die Windows-Programmierung, die Internet-Programmierung oder auch Datenbankanbindungen realisiert werden. Die nächste Abbildung zeigt diese Komponenten noch einmal im Überblick.



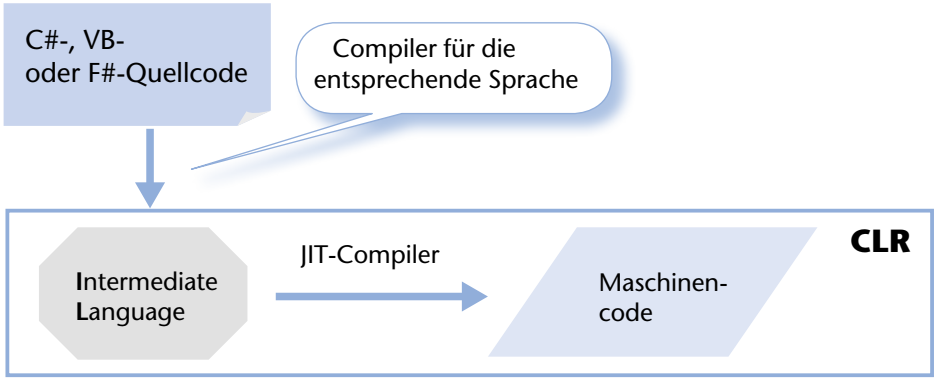
Mit den verschiedenen Klassenbibliotheken können die meisten Anwendungen realisiert werden. Die einzelnen Bibliotheken sind dabei für die folgenden Bereiche verantwortlich:

- **Base Classes (Base Class Library):** eine Sammlung von Klassen für elementare Funktionalitäten wie Dateioperationen, mathematische Funktionen oder auch reguläre Ausdrücke.
- **UWP Apps (Universal Windows Platform Apps):** Mit diesem Framework können universelle Apps für Windows-Plattformen entwickelt werden (Windows 10 oder Windows 11).
- **.NET MAUI (Multi-Platform App UI):** Das ist ein plattformübergreifendes Framework zur Erstellung von mobilen Apps und Desktop-Anwendungen mit C# und XAML.
- **Windows-Forms:** Die Klassen sind die Grundlage für die Entwicklung von Windows-Applikationen mit klassischen Elementen wie Fenstern, Menüs, Dialoge oder Buttons.
- **WPF (Windows Presentation Foundation):** Die Entwicklung von modernen GUI-Applikationen wird mit dieser Bibliothek realisiert.
- **ADO.NET (ActiveX Data Objekts .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.

1.1.5 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 sogenannte **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:

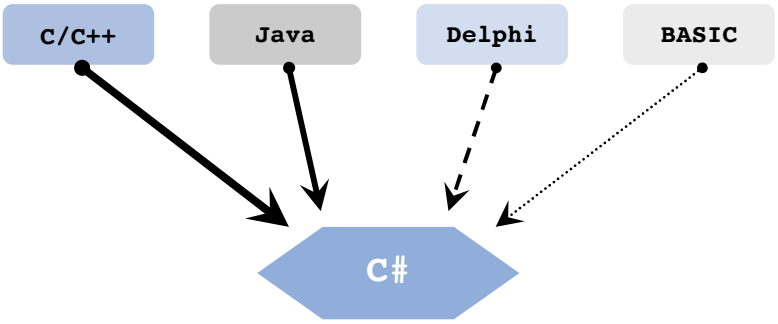


Das Ergebnis der Compilierung in die Intermediate Language nennt man **Assembly.** Eine Assembly kann in Form einer `.exe`- oder einer `.dll`-Datei vorliegen. Ohne die CLR bzw. .NET kann eine solche Datei allerdings nicht gestartet werden, auch wenn die Dateiendung an die bekannten ausführbaren Programme 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 **ECMA**[1] zur Standardisierung eingereicht und im Jahre 2003 auch von der ISO[2] 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 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.

1.2.2 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
- Schnelle und effektive Softwareentwicklung (Windows-Anwendungen, Web-Anwendungen) mit Unterstützung durch mächtige .NET-Klassenbibliotheken
- Komfortable Anbindung von beliebigen Datenbanken

1.2.3 Schlüsselworte in C#

- Die Sprache C# hat einen Wortschatz[3] 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#:

abstract	as	base	bool
break	byte	case	catch
char	checked	class	const
continue	decimal	default	delegate
do	double	else	enum
event	explicit	extern	false
finally	fixed	float	for
foreach	goto	if	implicit
in	int	interface	internal
is	lock	long	namespace
new	null	object	operator
out	override	params	private
protected	public	readonly	ref
return	sbyte	sealed	short
sizeof	stackalloc	static	string
struct	switch	this	throw
true	try	typeof	uint
ulong	unchecked	unsafe	ushort
using	virtual	volatile	void
while	where (kontextabhängig)	var (kontextabhängig)	partial (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 Paradigmen[4] 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.

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.

Beispiel:

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

```
1
2
3
4
5
```

Das Beispiel zeigt eine Wiederholung in so genanntem Pseudocode[5]. 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 (Geschäftprozesse, 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
```

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.

In dem Beispiel wird ein 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.

1.2.5 Bestandteile eines C#-Programms

Ein C#-Programm besteht aus einer Folge von endlich vielen und eindeutigen Anweisungen[6], 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

using System;

namespace EIGENER_NAMENSRAUM
{
    class Program
    {
        static void Main(string[] args)
        {
            Anweisung 1;
            Anweisung 2;
            :
            :
            Anweisung N;
        }
    }
}
```

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

Der Namensraum `System` wird verwendet.

Ein eigener Namensraum wird definiert.

Eine Klasse wird definiert.

Das „Hauptprogramm“

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:

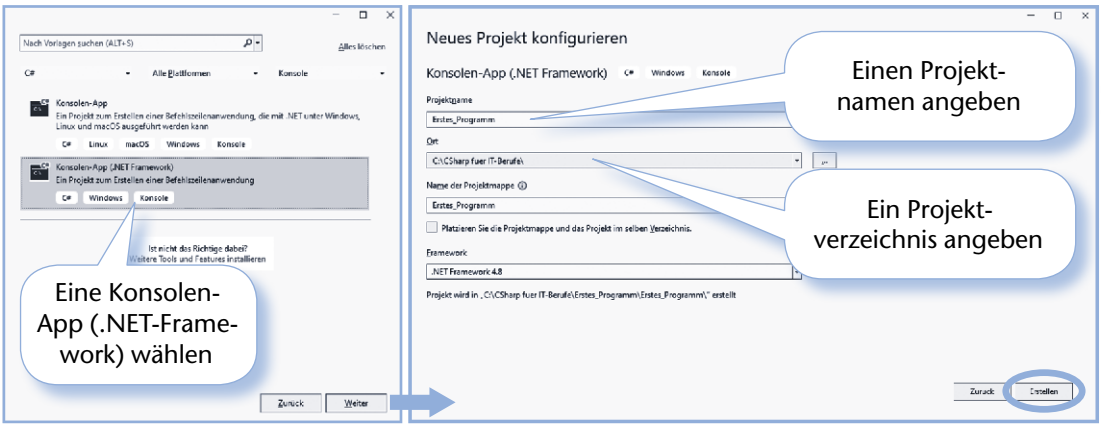
➤ Desktop-Apps:
- Windows-Forms-App (Windows-Applikation unter .NET)
- WPF-App (Moderne GUI-Applikationen unter .NET)
- Konsolen-App (ähnlich einem DOS-Programm)
- Bibliotheken (Sammlung von Funktionalitäten bzw. Klassen)

➤ .NET-MAUI-Apps:
- .NET-MAUI-App (plattformunabhängige App)
- .NET MAUI class library (Klassenbibliothek unter .NET MAUI)

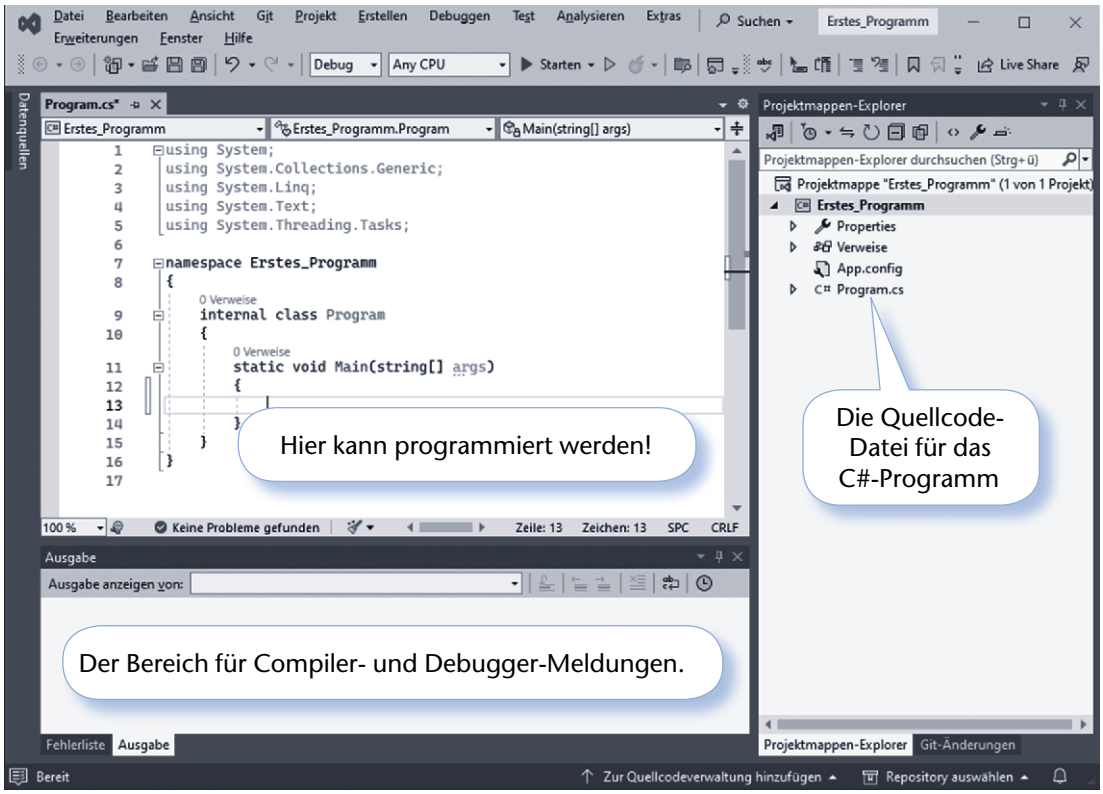
In diesem Buch sind hauptsächlich zwei Projektarten von Bedeutung: die Desktop-Apps und später die .NET-MAUI-App. Bei den Desktop-Anwendungen werden zuerst die Konsolenanwendung und dann die Windows-Forms-Anwendung sowie die WPF-Anwendung behandelt. Die Konsolenanwendung ist ausreichend, um eine einfache 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.

Anlegen eines neuen Projektes:

- Starten Sie Visual Studio 2022
- Wählen Sie den Menüpunkt Datei → Neu → Projekt.



Nach dem Bestätigen mit „Erstellen“ 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.

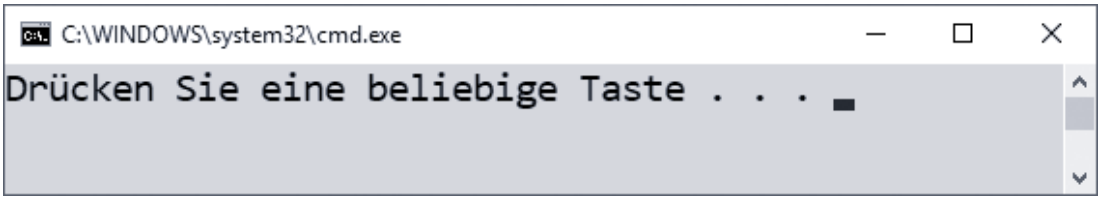
Ausführen eines C#-Programms

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

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

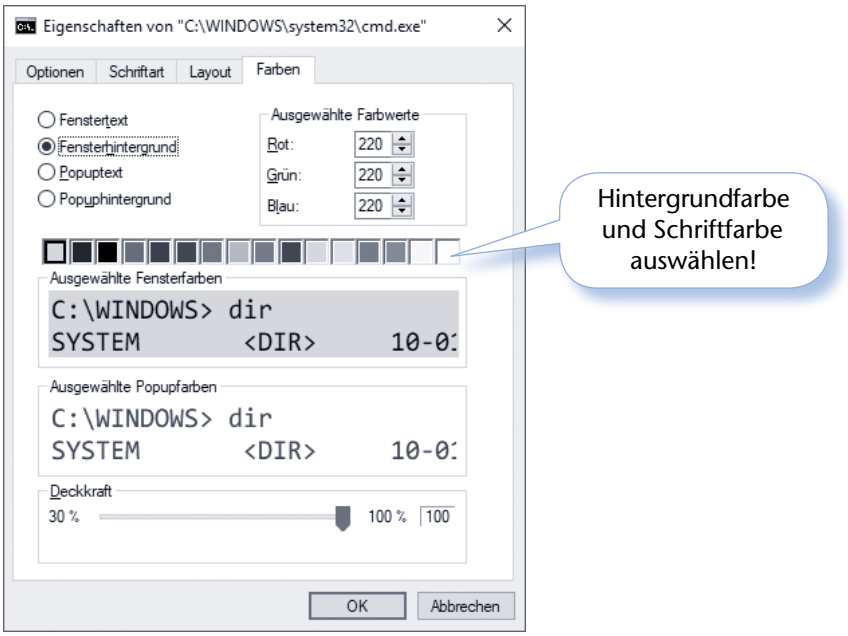
Fortgeschrittene werden später das Starten mit Debugging (Menüpunkt: Debuggen → Debugging starten oder F5 drücken) verwenden, wenn kompliziertere Programme analysiert werden müssen. Für den Anfang ist jedoch die oben beschriebene Vorgehensweise völlig ausreichend.

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



Hinweis:

➤ Das Konsolenfenster erscheint normalerweise als Fenster mit schwarzem Hintergrund und weißer Schriftfarbe. Über die Eigenschaften des Fensters können Hintergrundfarbe und Schriftart eingestellt werden.



2.2 Das erste C#-Programm

2.2.1 Das C#-Grundgerüst

Das erste einfache Beispiel eines C#-Programms wurde bereits im ersten Kapitel dargestellt. Dieses Grundgerüst soll nun genauer betrachtet werden, da es die Ausgangsbasis für alle weiteren C#-Programme ist.

Das C#-Grundgerüst

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
namespace IT_BERUFE_CSHARP
{
    class Program
    {
        static void Main(string[] args)
        {
            //Hier findet die erste Programmierung statt!
        }
    }
}
```

Mit der using-Anweisung werden Namensräume angesprochen.

Ein eigener Namensraum wird angelegt.

Ein Klasse wird definiert.

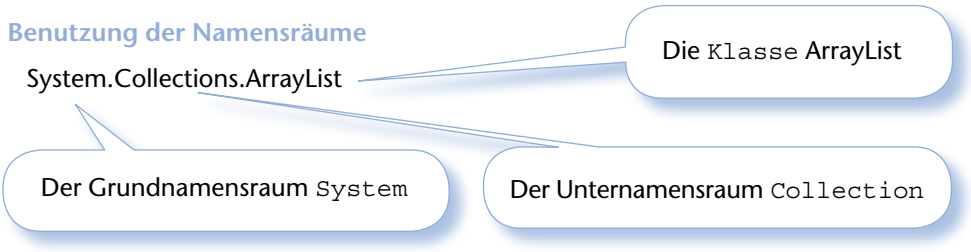
Eine statische Methode mit Namen `Main` wird angelegt. Diese Methode entspricht dem Hauptprogramm.

Im Prinzip würde es ausreichen, sich zu Beginn auf die Hauptmethode `Main` (bzw. deren Inhalt) zu beschränken. Denn bis zu den Kapiteln über Methoden und das Klassenkonzept in C# ist nur diese Hauptmethode interessant, weil bis dahin alle C#-Programme komplett in dieser Methode geschrieben werden. Trotzdem werden die wichtigen Komponenten in den folgenden Unterkapiteln kurz erläutert. Im Laufe des Informationsteils werden diese Komponenten dann weiter beleuchtet.

2.2.2 Namensräume

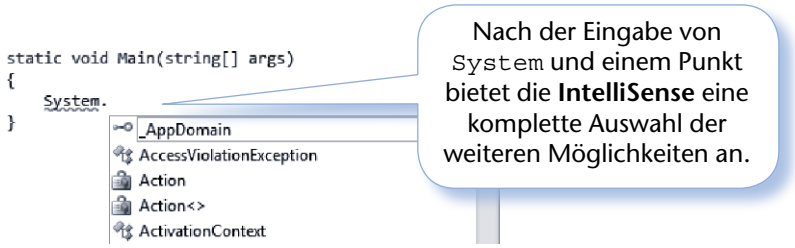
Bei der Entwicklung von C#-Programmen werden viele verschiedene Komponenten benötigt, um beispielsweise auf den Bildschirm zu schreiben, von der Tastatur einzulesen oder eine Datenbank anzusprechen. Diese Komponenten stehen in Form einer Klassenbibliothek zur Verfügung. Damit diese Bibliothek eine Struktur erhält, werden beispielsweise alle Klassen, die zur Bildschirmausgabe und zur Tastatureingabe nötig sind, in einem eigenen Namensraum zusammengefasst. Andere Klassen, die beispielsweise für das Lesen und Schreiben nötig sind, werden ebenfalls in einem eigenen Namensraum zusammengefasst. Da Namensräume wiederum weitere Namensräume (Unternamensräume) enthalten können, entsteht eine gute Struktur. Der grundlegende Namensraum heißt `System`. Darin befinden sich die wichtigsten Klassen und natürlich weitere Namensräume. Die einzelnen Unternamensräume oder Klassen eines Namensraumes werden mithilfe des so genannten Punktoperators (also eines Punktes) angesprochen, wie das folgende Beispiel zeigt:

Benutzung der Namensräume



Hinweis

Die Entwicklungsumgebung Visual C# verfügt über eine benutzerfreundliche Hilfe – die **IntelliSense**. Diese Hilfe vereinfacht die Eingabe von C#-Befehlen und die Suche nach Namensräumen und Klassen. Sobald ein erstes Zeichen eingetippt wird, versucht die IntelliSense einen Vorschlag für eine angemessene Ergänzung zu machen. In der Regel hilft dieses Werkzeug sehr bei der Entwicklung.



Der using-Befehl

Dieser Befehl vereinfacht die Nutzung der Namensräume. Wird ein Namensraum mit dem `using`-Befehl angegeben, so können alle Komponenten des Namensraumes direkt angesprochen werden, ohne vorher den Namensraum angeben zu müssen.

Beispiel: ohne using-Befehl

Auswahl der Klasse ArrayList aus dem Namensraum `System.Collection`

```
System.Collections.ArrayList eineListe;
```

Beispiel: mit using-Befehl

Auswahl der Klasse `ArrayList` aus dem Namensraum `System.Collection`

```
using System.Collections;

ArrayList eineListe;
```

Die Klasse kann nun direkt angesprochen werden, ohne den Namensraum anzugeben.

2.2.3 Die Klasse `Program` und die Hauptmethode `Main`

Die Sprache C# ist eine vollständig objektorientierte Sprache. Deshalb besteht jedes C#-Programm aus mindestens einer Klasse. Bei einem neuen Konsolenprojekt legt die Enwicklungsumgebung automatisch die Klasse `Program` an. Diese Klasse könnte auch durchaus einen anderen Namen haben; allerdings ist es sinnvoll, dass die Klasse so heißt, denn sie soll das Programm verkörpern. Innerhalb der Klasse gibt es eine so genannte statische Methode `Main`. Diese Methode wird beim Starten eines C#-Programms ausgeführt, sie ist also das Hauptprogramm. Deshalb darf diese Methode auch nicht umbenannt werden, denn sonst würde der Compiler das „Hauptprogramm" nicht finden. Bis zu den Kapiteln über Methoden und Klassen werden alle C#-Anweisungen innerhalb dieser statischen Main-Methode geschrieben.

Hinweis für den Leser

Zu diesem Zeitpunkt ist es nicht wichtig, wenn die Begriffe Klasse und statische Methode nicht wirklich verstanden wurden. Das kann erst im Laufe der nächsten Kapitel erfolgen. Es reicht völlig aus, wenn eine grobe Orientierung über die Komponenten des C#-Programms vorhanden ist. Durch die Konzentration auf den Inhalt der `Main`-Methode sind die Grundlagen der C#-Programmierung in den nächsten Kapiteln recht einfach zu erlernen.

2.2.4 Die Ausgabe auf dem Bildschirm

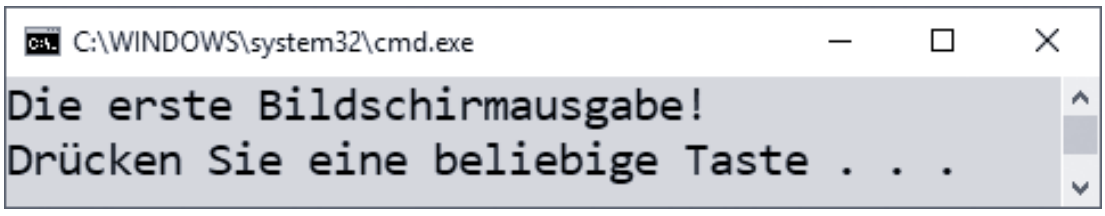
Nun soll das erste Programm um eine Möglichkeit erweitert werden, um auf den Bildschirm zu schreiben. Damit wäre eine erste Kommunikationsmöglichkeit zwischen Benutzer und Programm geschaffen. Die Ausgangsbasis ist wieder das Grundgerüst, welches von der Entwicklungsumgebung automatisch bei einem neuen Projekt angelegt wird:

```
using System;

namespace IT_BERUFE_CSHARP
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Die erste Bildschirmausgabe!");
        }
    }
}
```

Die statische Methode `WriteLine` der Klasse `Console` schreibt eine **Zeichenkette** auf den Bildschirm und macht danach automatisch einen Zeilenumbruch.

Nach dem Starten erscheint dann folgende Bildschirmausgabe:



Was ist eine Zeichenkette?

Eine Zeichenkette ist eine Reihe von endlich vielen Zeichen, also Buchstaben, Ziffern und Sonderzeichen, die in doppelten Anführungsstrichen eingerahmt sind.

2.2.5 Wichtige Regeln eines C#-Programms

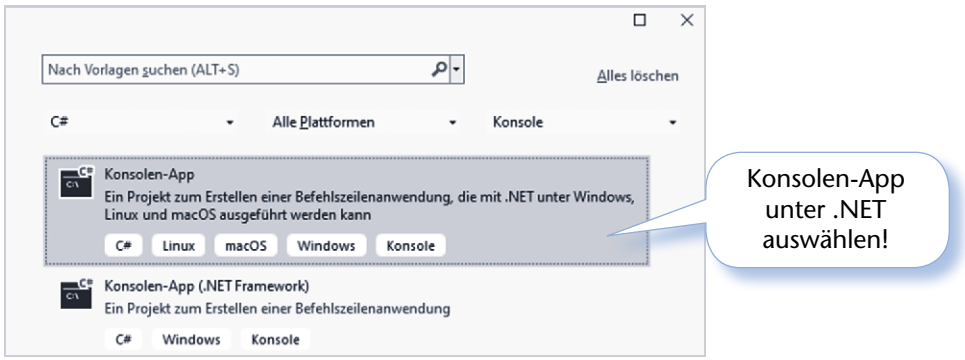
Nach den ersten Beispielen von einfachen C#-Programmen können folgende wichtige Grundregeln festgehalten werden:

- Jedes C#-Programm hat mindestens eine Klasse und eine Hauptmethode `Main`.
- Die Anweisungen in der Hauptmethode werden in geschweifte Klammern { } eingeschlossen – ebenso die Definition der Klasse und des eigenen Namensraumes.

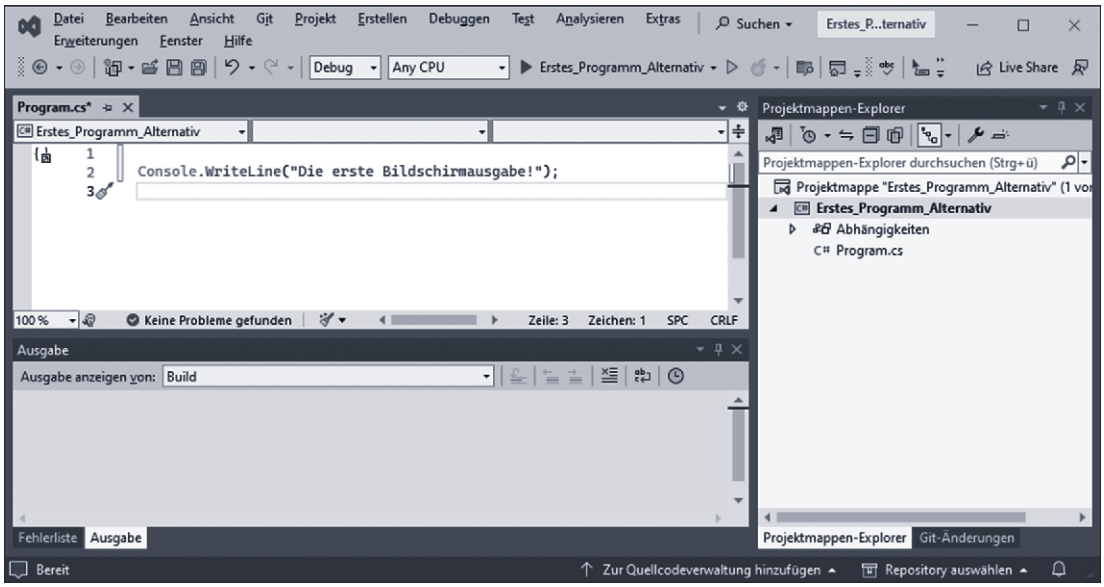
- Jede Anweisung in C# (wie beispielsweise `Console.WriteLine(…);`) wird mit einem Semikolon beendet.
- Mit dem `using`-Befehl können Komponenten von Namensräumen einfacher angesprochen werden.
- **C# unterscheidet zwischen Groß- und Kleinschreibung!**

2.2.6 Alternative zum Hauptprogramm ab C#-Version 9.0

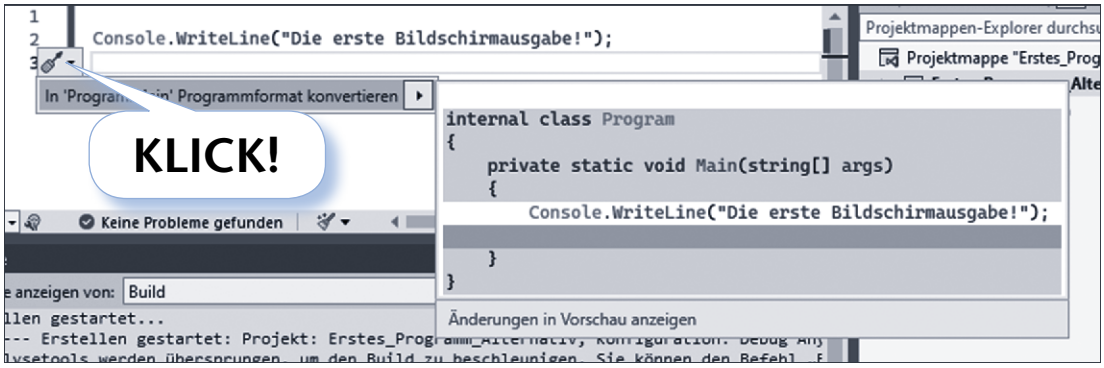
Mit der Version C# 9.0 gibt es eine Alternative zum Hauptprogramm (Hauptmethode) „Main“ – die ausführbaren Quellcodezeilen werden direkt zu Beginn des Programms angegeben und müssen nicht mehr in die statische Hauptmethode einer Klasse eingebettet werden. Das erinnert dann an Sprachen wie Python, die ebenfalls ohne Programmgerüst auskommen. Allerdings ist es dazu nötig, dass als Projektart eine Konsolen-App unter .NET und nicht unter .NET-Framework gewählt wird.



Nach dem Anlegen dieser Projektart erscheint das bereits bekannte Fenster, aber ohne Programmgerüst.



Die Entwicklungsumgebung bietet aber auch an, auf das herkömmliche Format zu wechseln.



Hinweis:

Weitere Elemente der Programmierung wie **Namespaces** oder eigene **Klassen** und **Methoden** werden dann bei dieser Variante einfach unter dem Quellcode des „Hauptprogramms" angelegt, so dass kein Unterschied zu der herkömmlichen Variante entsteht – dazu aber später mehr in den entsprechenden Kapiteln.

2.3 Grundlegende Konventionen in C#

Nachdem die ersten C#-Programme kennen gelernt wurde, müssen nun noch weitere Aspekte erarbeitet werden, die bei der Erstellung der Programme wichtig sind.

Dazu stellen sich folgende Fragen:

- Wie werden Namen in C# gebildet (für Variablen usw.)?
- Wie werden Leerzeilen, Zeilenumbrüche oder Leerzeichen im Quellcode interpretiert?
- Wie können Kommentare im Quellcode geschrieben werden?

2.3.1 Bezeichner (Namen) in C#

Wie in allen Programmiersprachen gibt es in C# Namen für Variablen, Konstanten, Methoden, Strukturen und Klassen. Diese selbst gewählten bzw. definierten Namen unterliegen einer gewissen Konvention, die unbedingt eingehalten werden muss:

- Das erste Zeichen muss ein Buchstabe sein (Unterstrich ist auch erlaubt „_").
- Der Rest kann aus Ziffern und Buchstaben gestaltet werden.
- Der Bezeichner darf nicht mit einem Schlüsselwort von C# übereinstimmen.

Beispiele:

`zahl`	gültiger Bezeichner
`_zahl`	gültiger Bezeichner
`2mal7`	kein gültiger Bezeichner (erstes Zeichen ist Ziffer)
`break`	kein gültiger Bezeichner (Schlüsselwort in C#)
`zahl 1`	kein gültiger Bezeichner (Leerzeichen nach `zahl`)

Kamel-Notation[1] und Pascal-Notation

Die Benennung von Bezeichnern ist dem Programmierer freigestellt. Allerdings hat es sich als sinnvoll erwiesen, dass in einem Programm von Anfang an eine bestimmte Konvention eingehalten wird. Deshalb werden hier die so genannte *Kamel-Notation* und die *Pascal-Notation* vorgeschlagen. Die Kamel-Notation sollte für Variablen verwendet werden. Sie sieht vor, dass der Bezeichner immer mit einem Kleinbuchstaben anfängt. Sollte der Variablenname aus mehreren Komponenten bestehen, so fängt die nächste Komponente dann mit einem Großbuchstaben an. Die Pascal-Notation sollte für Methodennamen verwendet werden. Der einzige Unterschied ist, dass die Bezeichner sofort mit einem Großbuchstaben beginnen. Die folgenden Beispiele verdeutlichen diese Konventionen:

Variablennamen mit der Kamel-Notation

- eineVariable
- vieleAutos
- neuesObjekt

Methodennamen mit der Pascal-Notation

- SetzeWert()
- OeffneDatei()
- SchliesseFenster()

1 Die *Kamel-Notation* hat ihren Namen wegen der „Höcker" im Wort, die sich durch die eingestreuten Großbuchstaben ergeben.

2.3.2 Trennzeichen

Zwischen den verschiedenen Anweisungen und Ausdrücken eines C#-Programms muss als Trennzeichen ein Semikolon stehen. Mehrere zusammengehörende Anweisungen werden (wie bei der `Main`-Methode) innerhalb geschweifter Klammern zusammengefasst.

Zwischen Schlüsselworten in C# und eigenen Bezeichnern muss zur Unterscheidung entweder ein Leerzeichen oder ein Operator (z.B. das Additionssymbol „+“) stehen.

Dabei werden das Zeilenende, der Tabulator und mehrere Leerzeichen als ein Leerzeichen bzw. als ein Trennzeichen interpretiert.

Beispiel:

Alle drei Programme sind identisch in ihrer Funktionalität. Sie unterscheiden sich nur in der Anordnung des Quelltextes:

Programm 1:

```
using System;
namespace IT_BERUFE_CSHARP
{
    class Program
    {
      static void Main(string[] args)
      {
        System.Console.WriteLine ("Leerzeichen und Co.!");
      }
    }
}
```

Zwischen Schlüsselwort (hier `void`) und Bezeichner (hier Main) muss ein Trennzeichen stehen (hier ein Leerzeichen).

Die Anweisung, um auf den Bildschirm zu schreiben, wird in einer Zeile geschrieben und mit einem Semikolon beendet.

Programm 2:

```
using System;
namespace IT_BERUFE_CSHARP
{
    class Program
    {
      static void Main(string[] args)
      {
        System.Console.WriteLine
         ("Leerzeichen und Co.!")
        ;
      }
    }
}
```

Die Anweisung, um auf den Bildschirm zu schreiben, wird auf drei Zeilen verteilt und mit einem Semikolon beendet.

Programm 3:

```
using System;namespace IT_BERUFE_CSHARP{class Program{
static void Main(string[] args){System.Console.WriteLine
("Leerzeichen und Co.!");}}}
```

Das ganze Programm wird einfach hintereinander geschrieben.

Es ist ersichtlich, dass die erste Variante deutlich leserlicher als die zweite Variante ist. Die dritte Variante dient natürlich nur der Anschauung – so sollte kein C#-Programm aussehen. In allen drei Fällen erscheint nach dem Starten allerdings dieselbe Bildschirmausgabe: