

C# für IT-Berufe

Modul 3: Fortgeschrittene
Programmierung mit C#

geeignet für die Lernfelder

8

11a

12a

C# für IT-Berufe

```
using System;

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

Verfasser:

Dirk Hardy, 46049 Oberhausen

Die in diesem Lehr- und Übungsmodul genannten Software-, Hardware- und Handelsnamen sind in der Mehrzahl auch eingetragene Warenzeichen.

Unter Verwendung von Screenshots aus:

Visual Studio Community Edition 2022, © Microsoft

1. Auflage 2024

978-3-8085-8744-7 (4-Jahreslizenz)

978-3-8085-8778-2 (Jahreslizenz)

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

© 2024 by Verlag Europa-Lehrmittel, Nourney, Vollmer GmbH & Co. KG, 42781 Haan-Gruiten

www.europa-lehrmittel.de

Satz: Typework Layoutsatz & Grafik GmbH, 86153 Augsburg

Titel: braunwerbeagentur, 42477 Radevormwald

Titelfotos: Christian Worryng-adobestock.com; imagetteam-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 3** möchte die fortgeschrittene 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 **verständliche Darstellung von fortgeschrittenen Themen in der Programmierung mit C#**. Als Grundlage dienen die Inhalte von **Modul 1 (elementare Programmierung in C#)** und **Modul 2 (objektorientierte Programmierung mit C#)** oder **gleichwertige Kenntnisse**. Zusätzlich wird am Ende des ersten Kapitels eine Zusammenfassung der wesentlichen Inhalte aus Modul 1 und Modul 2 geboten.

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 Lernsituation kann aber auch als **Projektidee** 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

Vorbemerkung	3
Aufbau des Moduls.....	3
1 Einführung in .NET und C#	6
1.1 Das .NET-Framework und .NET	6
1.1.1 Entstehung des .NET-Frameworks	6
1.1.2 Entstehung von .NET Core und .NET 5+	6
1.1.3 Eigenschaften von .NET	7
1.1.4 Die Komponenten von .NET	7
1.1.5 Kompilierung von .NET-Programmen	8
1.2 Die Sprache C#	8
1.2.1 Entwicklung der Sprache C#	8
1.2.2 Eigenschaften der Sprache C#	9
1.2.3 Schlüsselworte in C#	9
1.2.4 Prozedurale, strukturierte und objektorientierte Programmierung unter C# ...	9
1.2.5 Bestandteile eines C#-Programms	11
1.2.6 Ein C#-Projekt anlegen	11
1.2.7 Zusammenfassung der wesentlichen Aspekte aus Modul 1 und Modul 2	14
2 Dateioperationen in C#	19
2.1 Lesen und Schreiben von Dateien	20
2.1.1 Sequenzielles Lesen und Schreiben	20
2.1.2 Den Dateizeiger positionieren	22
2.2 Textdateien lesen und schreiben	23
2.2.1 Textdateien mit dem StreamWriter schreiben	23
2.2.2 Textdateien mit dem StreamReader lesen	24
2.3 Methoden der Klassen File und Directory	25
2.3.1 Methoden der Klasse File	25
2.3.2 Methoden der Klasse Directory	27
3 Ausnahmen – Exceptions	29
3.1 Versuchen und Auffangen (try and catch)	29
3.2 System-Exceptions	31
3.3 Der finally-Block	35
3.4 Ausnahmen werfen	36
3.5 Eigene Exception-Klassen erstellen	37
3.6 Ausnahmen weiterleiten	38
4 Elementarer Datenbankzugriff	40
4.1 Datenbankbindung unter dem .NET-Framework	40
4.2 Provider nutzen und eine Verbindung aufbauen	41
4.3 Beispiel eines Zugriffs auf eine ACCESS-Datenbank	41
4.4 Nicht-Select-Befehle absetzen	44
4.5 DataAdapter und DataSet	46
5 Themen für Experten.....	50
5.1 Delegate und Lambda-Ausdrücke.....	50
5.1.1 Delegate anlegen	50
5.1.2 Lambda-Ausdrücke.....	52
5.2 Der Indexer	55
5.3 Generische Programmierung.....	57
5.3.1 Generische Methoden	57
5.3.2 Generische Klassen	59

Aufgaben	63
1 Aufgaben zur Einführung von .NET und C#.....	63
2 Aufgaben zu Dateioperationen in C#.....	63
3 Aufgabe zu Ausnahmen – Exceptions.....	66
4 Aufgabe zum elementarer Datenbankzugriff	67
5 Aufgaben zu Themen für Experten	68
Lernsituation	71
Entwicklung eines Vokabeltrainer-Prototypen für einen Kunden im Bereich Fremdsprachen	71
Index	73

1 Einführung in .NET und C#

1.1 Das .NET-Framework und .NET

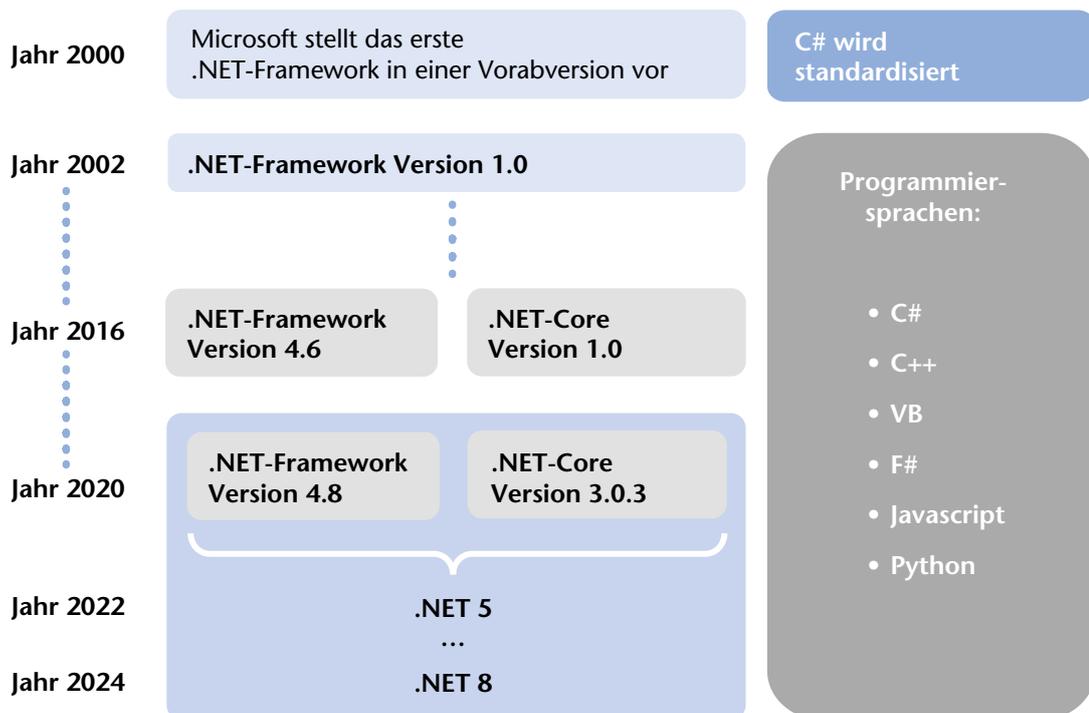
1.1.1 Entstehung des .NET-Frameworks

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:



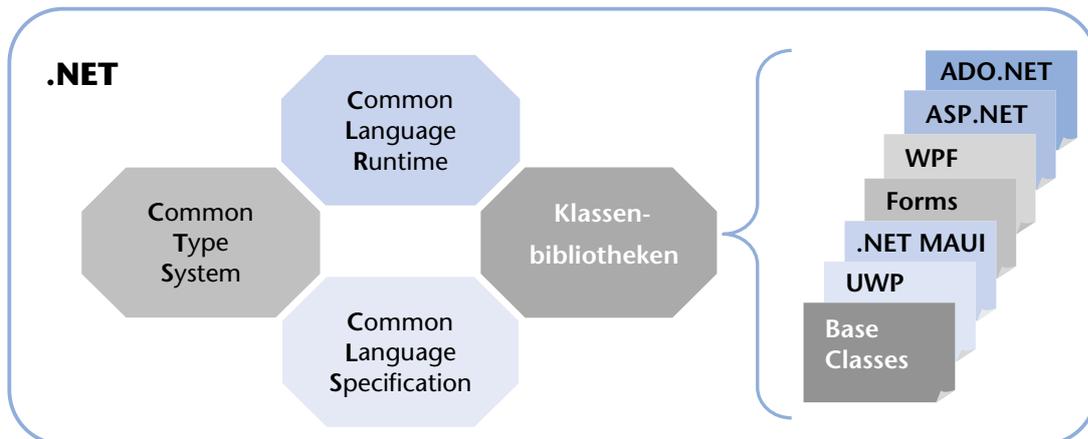
1.1.3 Eigenschaften von .NET

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 Datenbankverbindungen realisiert werden. Die nächste Abbildung zeigt diese Komponenten noch einmal im Überblick.



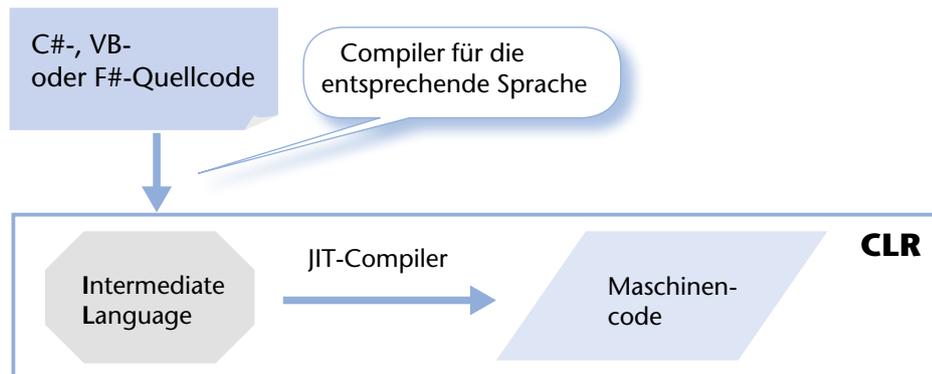
Mit den verschiedenen Klassenbibliotheken können die meisten Anwendungen realisiert werden. Die einzelnen Bibliotheken sind dabei für die folgenden Bereiche verantwortlich:

- **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 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.

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:

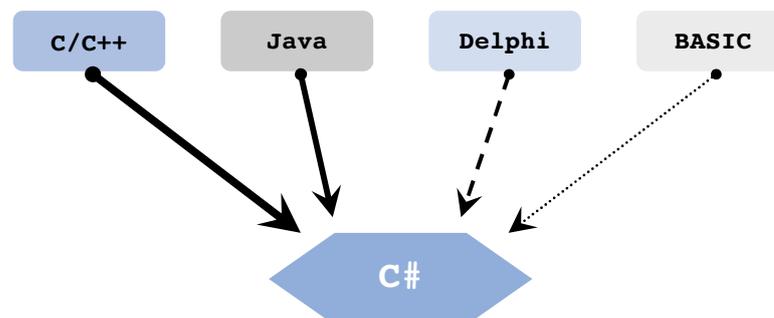


Das Ergebnis der Kompilierung in die Intermediate Language nennt man **Assembly**. Eine Assembly kann in Form einer `.exe`- oder einer `.dll`-Datei vorliegen. Ohne die CLR bzw. .NET kann eine solche Datei allerdings nicht gestartet werden, auch wenn die Dateiendung an die bekannten ausführbaren Programme 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 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.

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 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
```

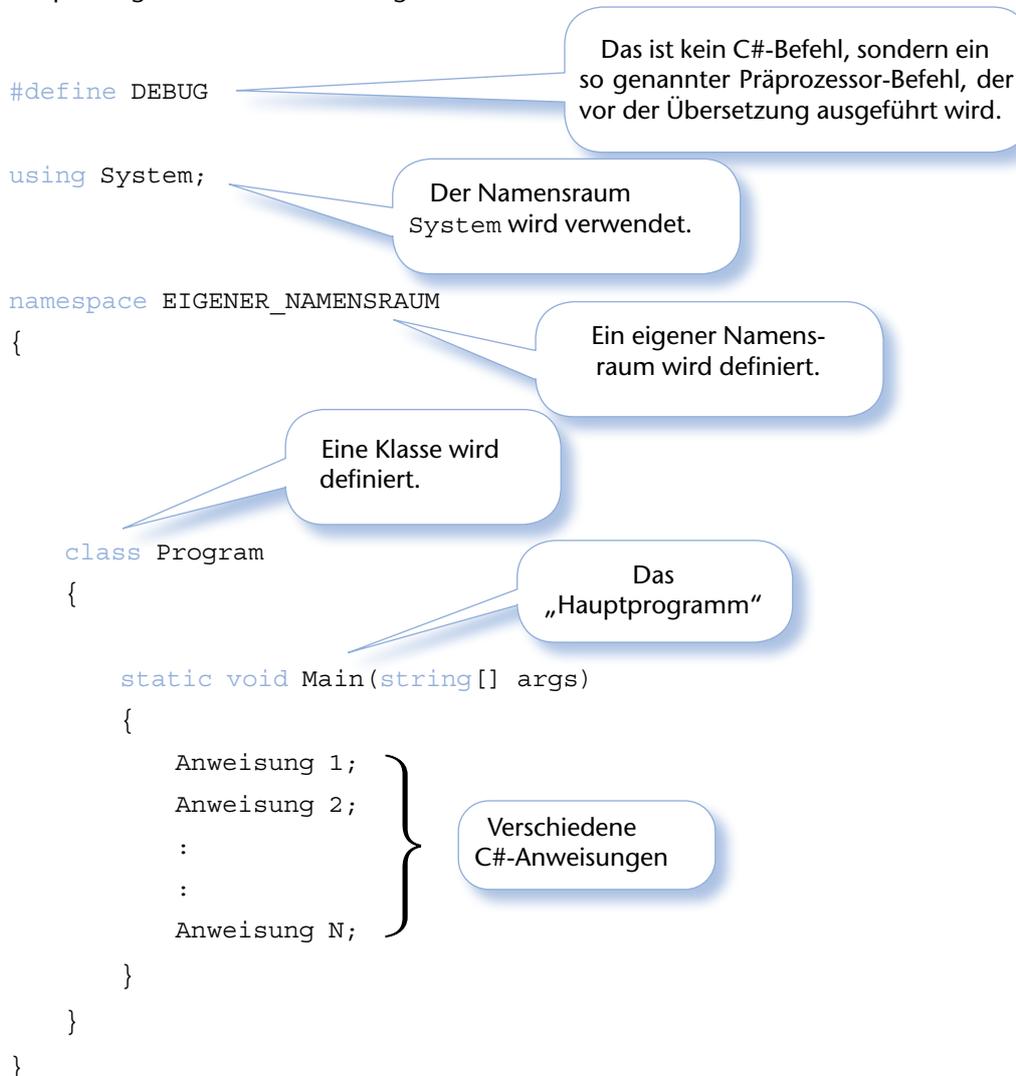
```
Maier
123456
```

In dem Beispiel wird 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.

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:



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.

1.2.6 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)

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

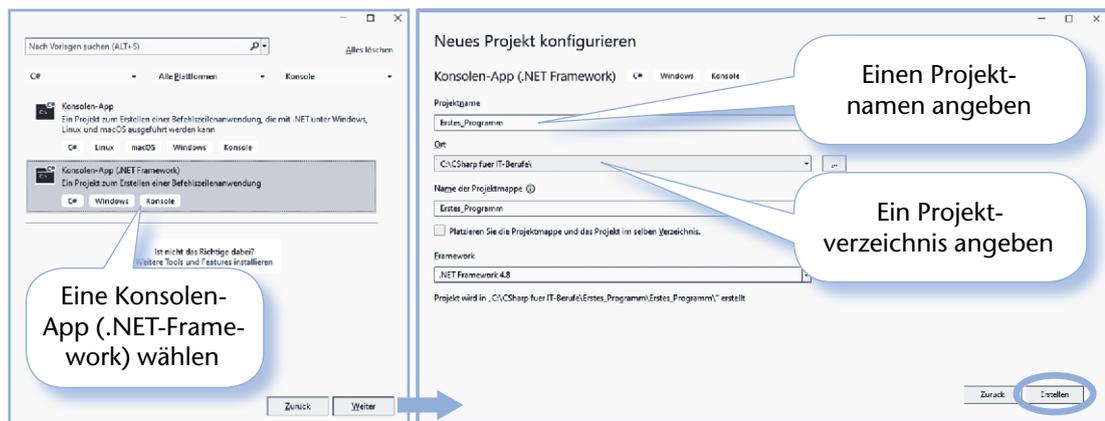
► .NET-MAUI-Apps:

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

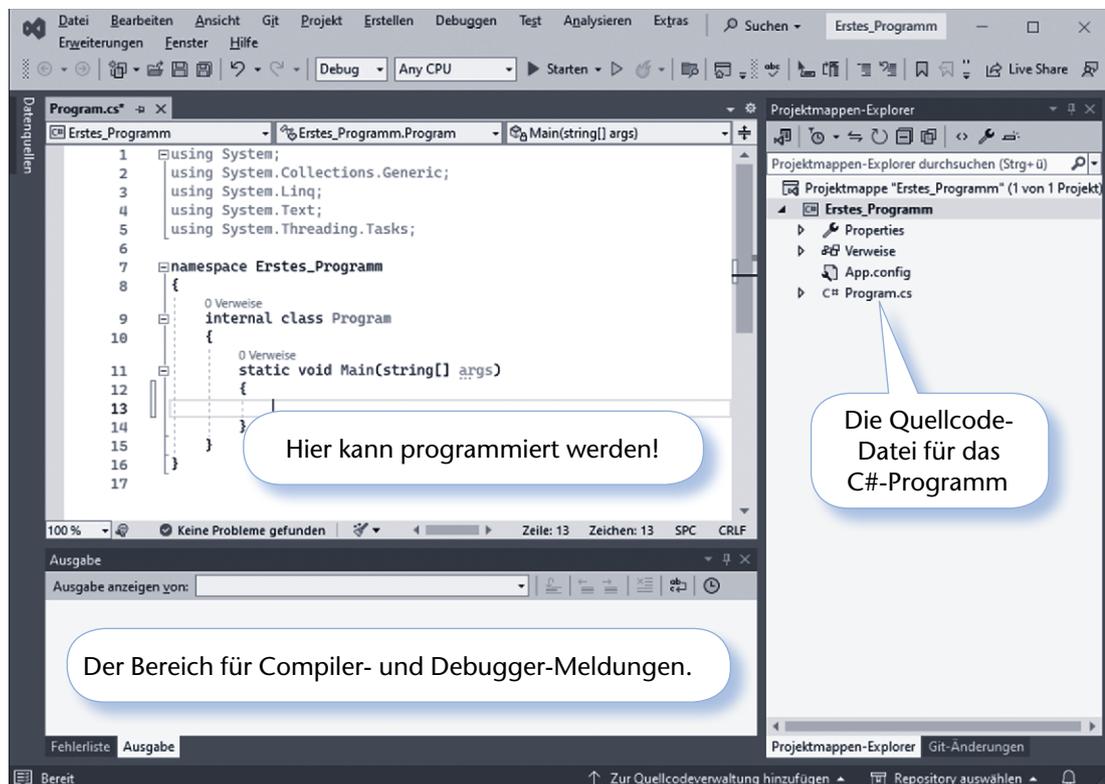
In diesem Buch sind hauptsächlich zwei Projektarten von Bedeutung: die Desktop-Apps und später die .NET-MAUI-App. Bei den Desktop-Anwendungen werden zuerst die Konsolenanwendung und dann die Windows-Forms-Anwendung sowie die WPF-Anwendung behandelt. Die Konsolenanwendung ist ausreichend, um eine einfache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

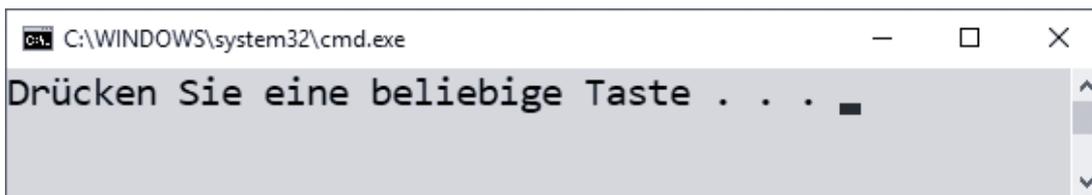
Ausführen eines C#-Programms

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

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

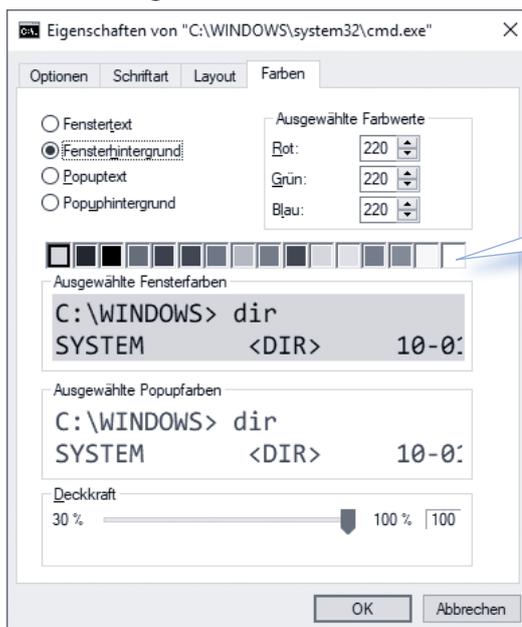
Fortgeschrittene werden später das Starten mit Debugging (Menüpunkt: Debuggen → Debugging starten oder F5 drücken) verwenden, wenn kompliziertere Programme analysiert werden müssen. Für den Anfang ist jedoch die oben beschriebene Vorgehensweise völlig ausreichend.

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.



Hintergrundfarbe
und Schriftfarbe
auswählen!

1.2.7 Zusammenfassung der wesentlichen Aspekte aus Modul 1 und Modul 2

Elementare Datentypen und Variablen:

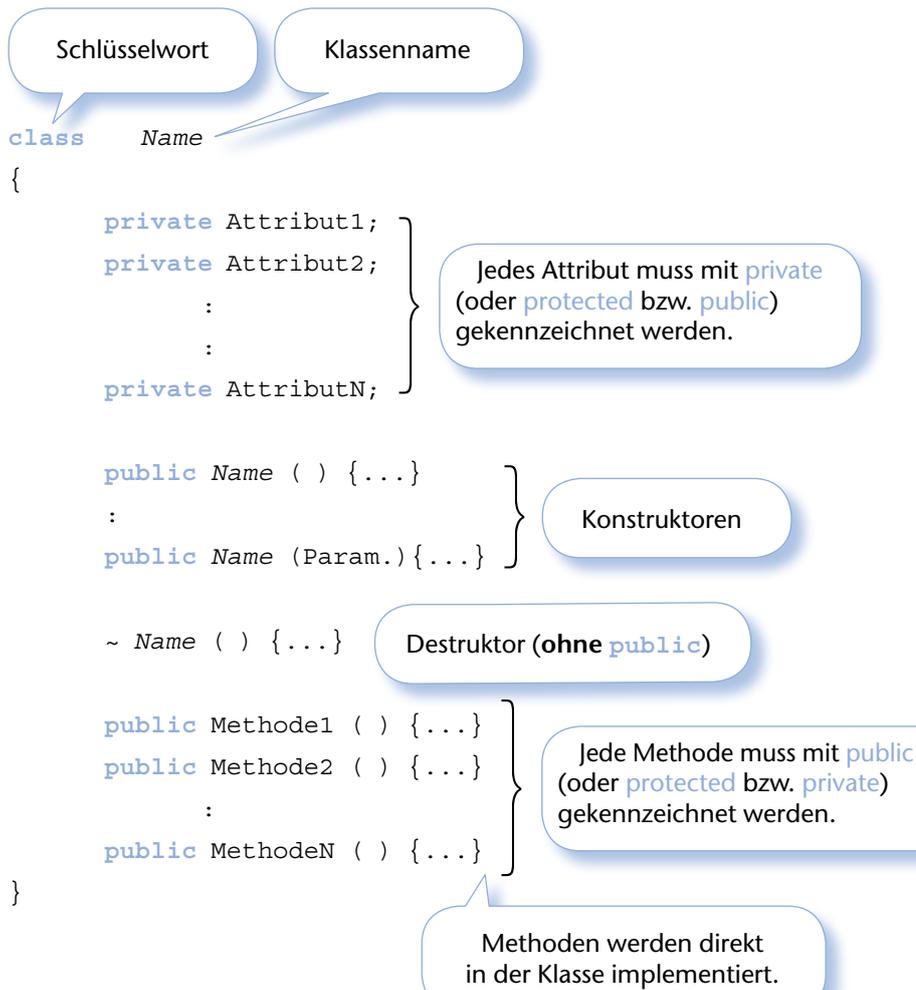
Datentyp	Beschreibung	Beispiel
<code>int</code>	Speichert ganze Zahlen	<code>int ganzeZahl = 42;</code>
<code>double</code>	Speichert Gleitkommazahlen	<code>double gleitkommaZahl = 3.14;</code>
<code>char</code>	Speichert einzelne Zeichen	<code>char zeichen = 'A'</code>
<code>bool</code>	Speichert Wahrheitswerte (true/false)	<code>bool istWahr = true;</code>
<code>string</code>	Speichert Zeichenketten	<code>string text = "Hallo, Welt!";</code>

Operatoren:

Operatoren	Beschreibung	Beispiele
Arithmetische Operatoren: <code>+, -, *, /, %</code>	Mit diesen Operatoren kann gerechnet werden (wie in der Mathematik).	<code>int summe = 5 + 3;</code> <code>int differenz = 7 - 2;</code> <code>int produkt = 4 * 6;</code> <code>int quotient = 8 / 2;</code> <code>int rest = 10 % 3; // rest == 1</code>
Inkrement und Dekrement: <code>++</code> und <code>--</code>	Diese Operatoren erhöhen bzw. erniedrigen eine Variable um 1.	<code>int zahl = 10;</code> <code>++zahl; // zahl == 11</code> <code>--zahl; // zahl == 10</code>
Vergleichsoperatoren: <code>==, !=, <, >, <=, >=</code>	Diese Operatoren dienen zum Vergleich von Werten oder Ausdrücken. Der Vergleich ist immer ein Wahrheitswert (true/false).	<code>int a = 1, b = 2;</code> <code>bool istGleich = (a == b); // false</code> <code>bool istUngleich = (a != b); // true</code> <code>bool istKleinerAls = (a < b); // true</code> <code>bool istGroesserAls = (a > b); // false</code>
Logische Operatoren: <code>&&</code> → UND <code> </code> → ODER <code>!</code> → NICHT	Mit diesen Operatoren werden Ausdrücke (Werte, Vergleiche) verknüpft.	<code>int a = 1, b = 2;</code> <code>bool b1 = (a > 0) && (b < 10); // true</code> <code>bool b2 = (a == b) (a == 1); // true</code> <code>bool negierteBedingung = !(a == b); // true</code>
Zuweisungsoperator: <code>=</code>	Dieser Operator dient zur Zuweisung.	<code>int a = 5;</code>

Kontrollstrukturen:

Kontrollstruktur	Beschreibung	Beispiele
<code>if</code> , <code>else if</code> , <code>else</code>	Selektion (einseitig, zweiseitig oder verschachtelt).	<pre> if (Bedingung1) { // Code wird ausgeführt, // wenn Bedingung1 wahr ist } else if (Bedingung2) { // Code wird ausgeführt, // wenn Bedingung2 wahr ist. } else { // Code wird ausgeführt, // wenn keine der Bedingungen // wahr ist. } </pre>
<code>switch(ausdruck)</code>	Mehrfachselektion	<pre> switch (ausdruck) { case wert1: // Code, wenn ausdruck == wert1 break; case wert2: // Code, wenn ausdruck == wert2 break; default: // Code, wenn keiner der Fälle // zutrifft. break; } </pre>
<code>while</code>	Kopfgesteuerte Iteration	<pre> while (Bedingung) { // Code wird ausgeführt, solange // Bedingung wahr ist. } </pre>
<code>do-while</code>	Rumpfgesteuerte Iteration	<pre> do { // Code wird mindestens einmal // ausgeführt, und dann immer weiter, // solange Bedingung wahr ist. } while (Bedingung); </pre>
<code>for(; ;)</code>	Zählergesteuerte Iteration	<pre> for (Initialisierung; Bedingung; Schritt) { // Die Initialisierung wird einmalig // zu Beginn ausgeführt. Code wird // ausgeführt, solange Bedingung // wahr ist. Anschließend wird die // Schrittanweisung ausgeführt. } </pre>

Aufbau einer Klasse in C#:**Instanzieren von Objekten in C#:**

```
CEineKlasse objektVerweis;
```

Einen Verweis der Klasse anlegen

Anschließend kann diesem Verweis dann ein konkretes Objekt im Speicher zugeordnet werden. Mit dem `new`-Operator wird ein solches Objekt im Speicher angelegt und dem Verweis zugewiesen:

```
objektVerweis = new CEineKlasse();
```

Ein Objekt im Speicher anlegen

Löschen von Objekten in C#:

Wird ein Objekt „verweislos“, so wird es automatisch vom *garbage-collector* aus dem Speicher entfernt.

```
CEineKlasse objektVerweis;
```

```
objektVerweis = new CEineKlasse();
```

```
objektVerweis = null;
```

Dem Verweis wird der `null`-Verweis zugewiesen, damit ist das Objekt ohne Verweis und wird gelöscht.

Vererbung in C#:

```
class CBasis
{
    public CBasis ()
    {
        // Standardkonstruktor
    }
    :
}
```

Die Klasse erbt von CBasis.

```
class CErbe : CBasis
{
    public CErbe () : base ()
    {
        // Standardkonstruktor
    }
    :
}
```

Expliziter Aufruf des Basisklassenkonstruktors

Überladen von Operatoren in C#:**Regel 1:**

Eine Operator-Methode muss immer öffentlich (**public**) und statisch (**static**) sein.

Regel 2:

Die Grundfunktionalität des Operators kann nicht geändert werden. Der +-Operator braucht beispielsweise zwei Operanden (Operand_1 + Operand_2). Daran ändert auch eine Überladung nichts.

Regel 3:

Es können nur existierende Operatoren überladen werden. Neue Operatoren können nicht entworfen werden. Beispielsweise kann man keinen „§-Operator“ definieren, denn einen solchen Operator stellt C# nicht zu Verfügung.

Regel 4:

In C# ist es zwingend, dass zusammengehörende Operatoren (beispielsweise == und != oder < und >) auch immer zusammen überladen werden.

Regel 5:

Folgende Operatoren können nicht überladen werden:

```
?:    new  is  typeof  sizeof  =
```

Ebenso können die gekoppelten Zuweisungsoperatoren (wie += oder *=) nicht überladen werden.

Die gekoppelten Zuweisungen werden implizit in die richtige Form übersetzt (beispielsweise `x += 5` in `x = x + 5`).

Das folgende Beispiel zeigt die Überladung des Plus-Operators in einer Klasse `CBruch`, die einen mathematischen Bruch verkörpern soll:

```
public static CBruch operator +(CBruch op_1, CBruch op_2)
{
    int z = op_1.zaehler * op_2.nenner + op_2.zaehler * op_1.nenner;
    int n = op_1.nenner * op_2.nenner;
    return new CBruch(z , n);
}
```


2 Dateioperationen in C#

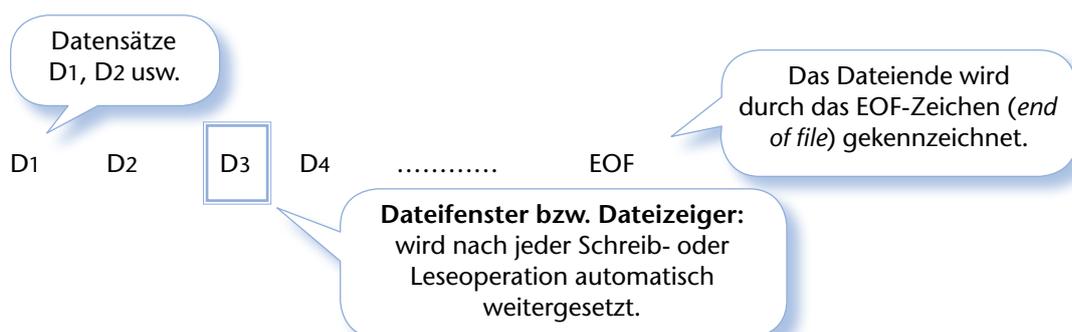
In den Anfängen der Computergeschichte erfolgte die Datensicherung auf Lochkarten. Später wurden dann Magnetbänder eingesetzt, um die Daten zu sichern. Auch bei den ersten Homecomputern in den 80er-Jahren war das Standard-Datensicherungsgerät eine Art Kassettenrekorder, auf den die Computerdaten in akustischer Form übertragen wurden. Diese Art der Datensicherung war nicht nur unkomfortabel, sondern auch relativ fehleranfällig. Die problemlose Datensicherung heutzutage auf CD, DVD oder externe Festplatten und Memory-Sticks hat nicht mehr viel gemeinsam mit diesen überholten Techniken. Trotzdem ist das Grundprinzip der Datensicherung gleich geblieben. Mit einer Programmiersprache wie C# können Daten so wie früher auf einem Magnetband sequenziell in eine Datei geschrieben werden. Das kann man sich so vorstellen, dass mithilfe der so genannten Stream-Objekte in C# die Daten hintereinander in die Datei geschrieben und auch so ausgelesen werden können.

Hinweis:

Eine Datei (engl. *file*) ist eine Sammlung von Daten bzw. Datensätzen. Jeder Datensatz kann aus mehreren Komponenten zusammengesetzt sein. Das Sichern von Daten auf externen Speichern (Festplatte usw.) erfolgt in Form von solchen Dateien. Die Verwaltung der Dateien übernimmt dabei das Betriebssystem.

Grundsätzlich können zwei Organisationen von Dateien unterschieden werden: die **sequenzielle** und die **direkte Organisation**. Bei der sequenziellen Organisation ist die Vorstellung eines Magnetbandes treffend. Die Daten werden hintereinander in die Datei geschrieben und können auch nur in dieser Reihenfolge gelesen werden. Diese Form des Dateizugriffs ist sehr einfach zu programmieren. Der entscheidende Nachteil ist die Schnelligkeit. Bei einer großen Datei müssen beispielsweise erst enorm viele Datensätze eingelesen werden, um den gesuchten Datensatz zu erhalten.

Die folgende Grafik zeigt den sequenziellen Zugriff auf eine Datei. Dabei bewegt sich ein so genanntes Dateifenster (Dateizeiger) nach jedem Lesezugriff auf den nächsten Datensatz.



Bei der direkten Organisation wird hingegen das Dateifenster genau positioniert und es können Datensätze an einer bestimmten Stelle aus der Datei gelesen werden. Sowohl der sequenzielle als auch der direkte Zugriff sind mit den Stream-Klassen in C# möglich.

Im Namensraum `System.IO` stellt das .NET-Framework eine Vielzahl von Klassen zu Verfügung, mit denen die Dateioperationen umgesetzt werden können. Dabei gehen die Funktionalitäten aber weit über das Lesen und Schreiben von Dateien hinaus. Es gibt Klassen, die das Kopieren oder Verschieben von Dateien ermöglichen, aber auch Klassen, die Verzeichnisse auslesen oder erstellen. Auch die Behandlung von komprimierten Dateien und der Datenaustausch über das Netzwerk sind möglich – diese komplexeren Themen können aber nicht Gegenstand dieses Buches sein.