C# für IT-Berufe

Modul 4: GUI-Programmierung mit C#

geeignet für die Lernfelder 8 | 10a | 11a | 12a

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("     GUI-");
       Console.WriteLine("     Programmierung");
       Console.WriteLine("     mit C#");
       Console.WriteLine();
       Console.WriteLine("Aufgabenpool");
       Console.WriteLine();
       Console.WriteLine("Lernsituation");
       Console.WriteLine();
     }
  }
}
```

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

Europa-Nr.: 87454L (4-Jahreslizenz) · Europa-Nr. 87454V (Jahreslizenz)

Verfasser:
Dirk Hardy, 46049 Oberhausen

1. Auflage 2024

978-3-8085-8745-4 (4-Jahreslizenz)
978-3-8085-8779-9 (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 4** möchte die Programmierung von grafischen Benutzeroberflächen (GUI-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 Einführung in die GUI-Programmierung mit Windows-Forms, WPF und den aktuellen MAUI-Apps. 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

1 Einführung in .NET und C#

1.1 Das .NET-Framework und .NET

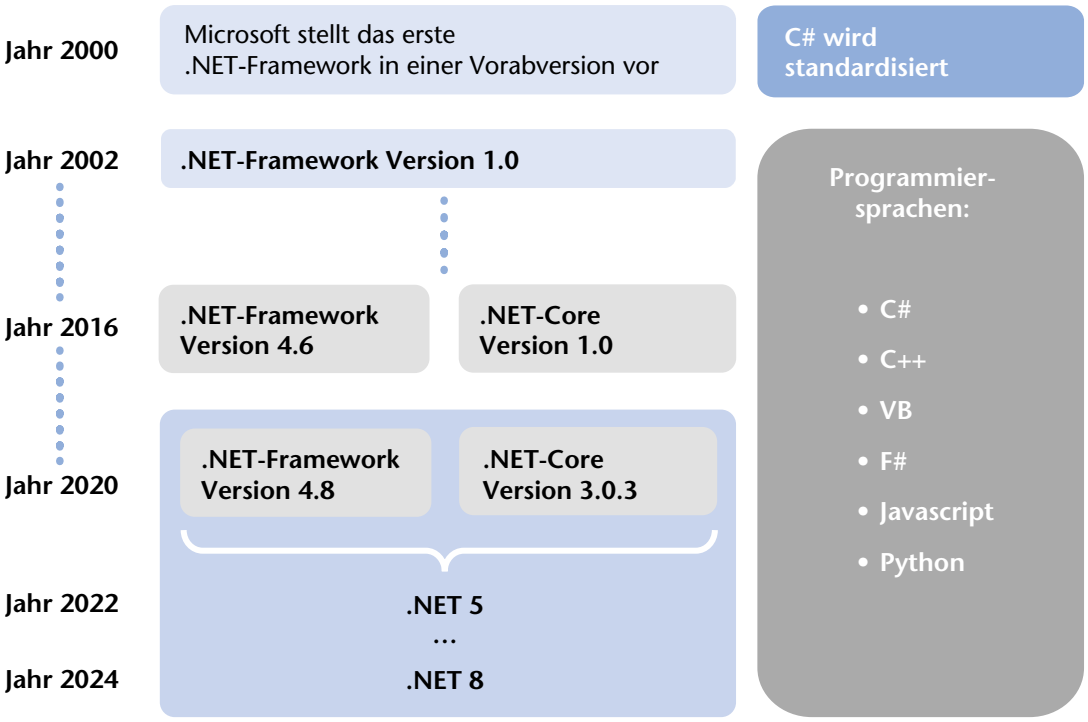
1.1.1 Entstehung des .NET-Frameworks

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:



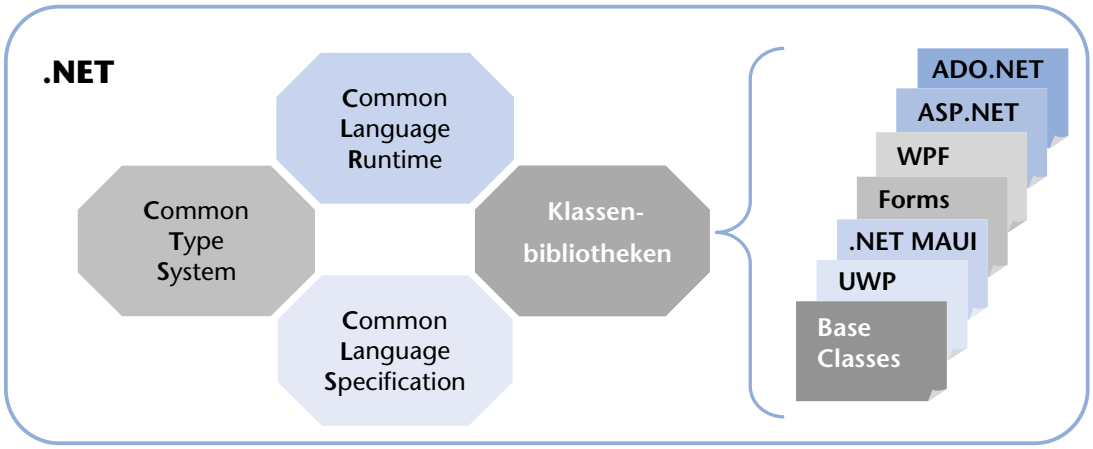
1.1.3 Eigenschaften von .NET

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.



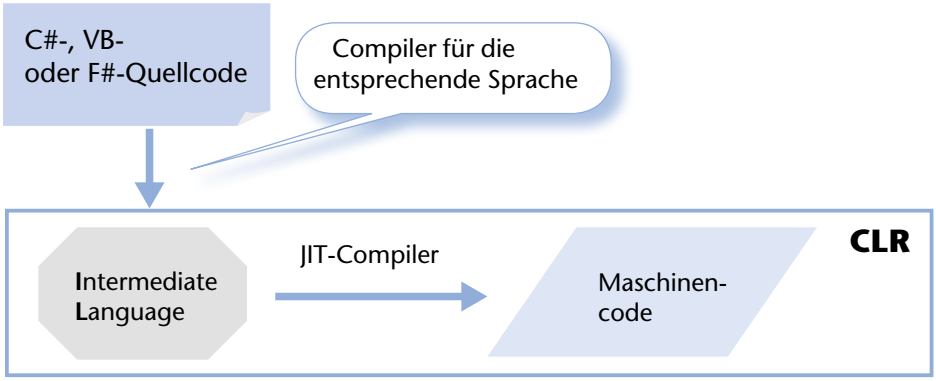
Mit den verschiedenen Klassenbibliotheken können die meisten Anwendungen realisiert werden. Die einzelnen Bibliotheken sind dabei für die folgenden Bereiche verantwortlich:

- **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:

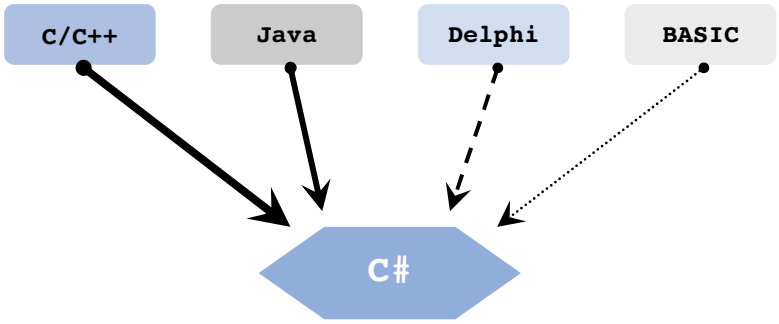


Das Ergebnis der Compilierung in die Intermediate Language nennt man **Assembly.** Eine Assembly kann in Form einer `.exe`- oder einer `.dll`-Datei vorliegen. Ohne die CLR bzw. .NET kann eine solche Datei allerdings nicht gestartet werden, auch wenn die Dateiendung an die bekannten ausführbaren Programme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

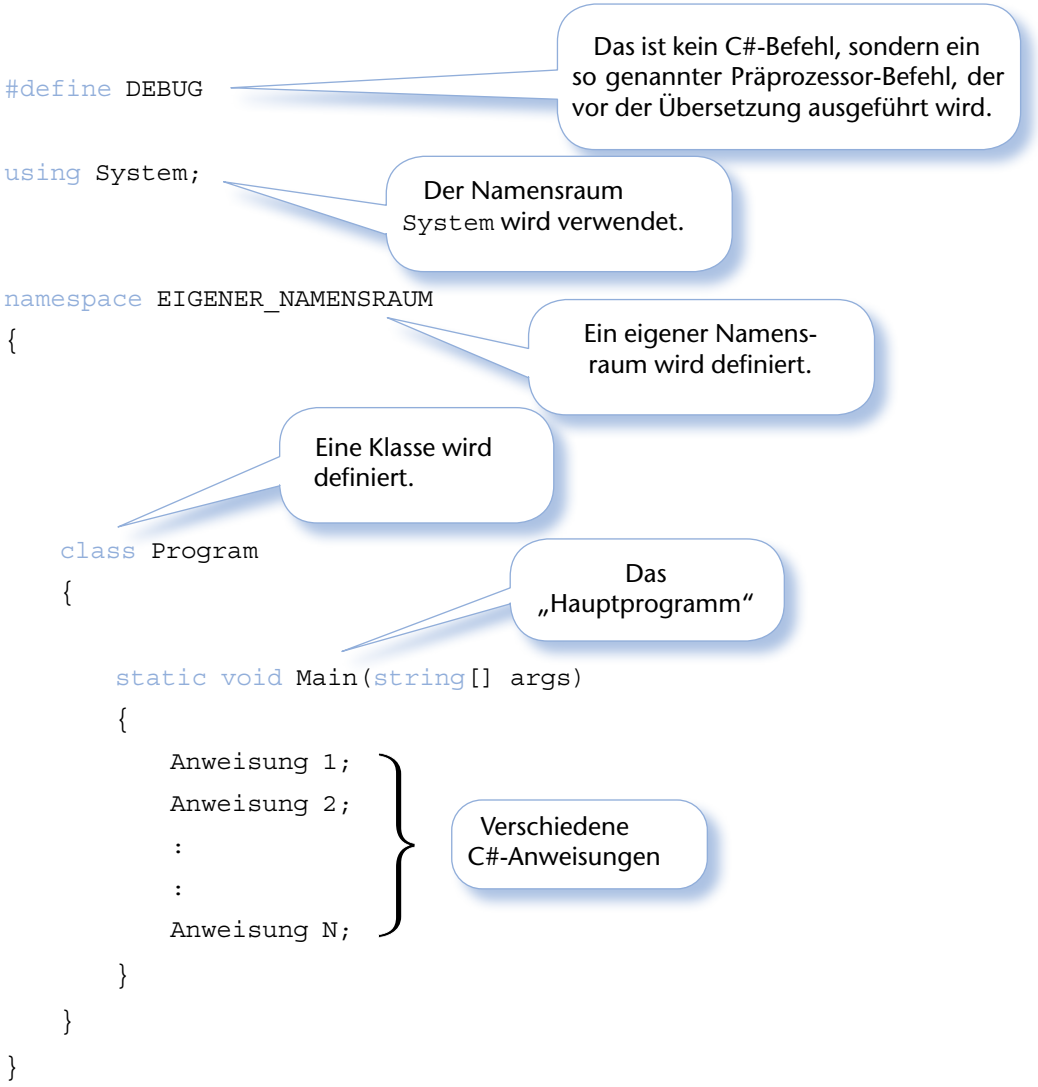
```
Maier
123456
```

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 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;
        }
    }
}
```

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**.

1.2.6 Zusammenfassung der wesentlichen Aspekte aus Modul 1 und Modul 2

Elementare Datentypen und Variablen:

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

Operatoren:

Operatoren	Beschreibung	Beispiele

Arithmetische Operatoren: `+,-,*,/,%`	Mit diesen Operatoren kann gerechnet werden (wie in der Mathematik).	`int summe = 5 + 3;`
		`int differenz = 7 - 2;`
		`int produkt = 4 * 6;`
		`int quotient = 8 / 2;`
		`int rest = 10 % 3;  // rest == 1`

Inkrement und Dekrement: `++ und --`	Diese Operatoren erhöhen bzw. erniedrigen eine Variable um 1.	`int zahl = 10;`
		`++zahl; // zahl == 11`
		`--zahl; // zahl == 10`

Vergleichsoperatoren: `==, !=, <, >, <=, >=`	Diese Operatoren dienen zum Vergleich von Werten oder Ausdrücken. Der Vergleich ist immer ein Wahrheitswert (true/false).	`int a = 1, b = 2;`
		
		`bool istGleich = (a == b);  // false`
		`bool istUngleich = (a != b);  // true`
		`bool istKleinerAls = (a < b);  // true`
		`bool istGroesserAls = (a > b); // false`

Logische Operatoren: `&& → UND`
`|| → ODER`
`! → NICHT`
	Mit diesen Operatoren werden Ausdrücke (Werte, Vergleiche) verknüpft.	`int a = 1, b = 2;`
		
		`bool b1 = (a > 0) && (b < 10);  // true`
		`bool b2 = (a == b) || (a == 1);  // true`
		`bool negierteBedingung = !(a == b); // true`

Zuweisungsoperator: `=`	Dieser Operator dient zur Zuweisung.	`int a = 5;`

Kontrollstrukturen:

Kontrollstruktur	Beschreibung	Beispiele

if, else if, else	Selektion (einseitig, zweiseitig oder verschachtelt).	`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.`
		`}`

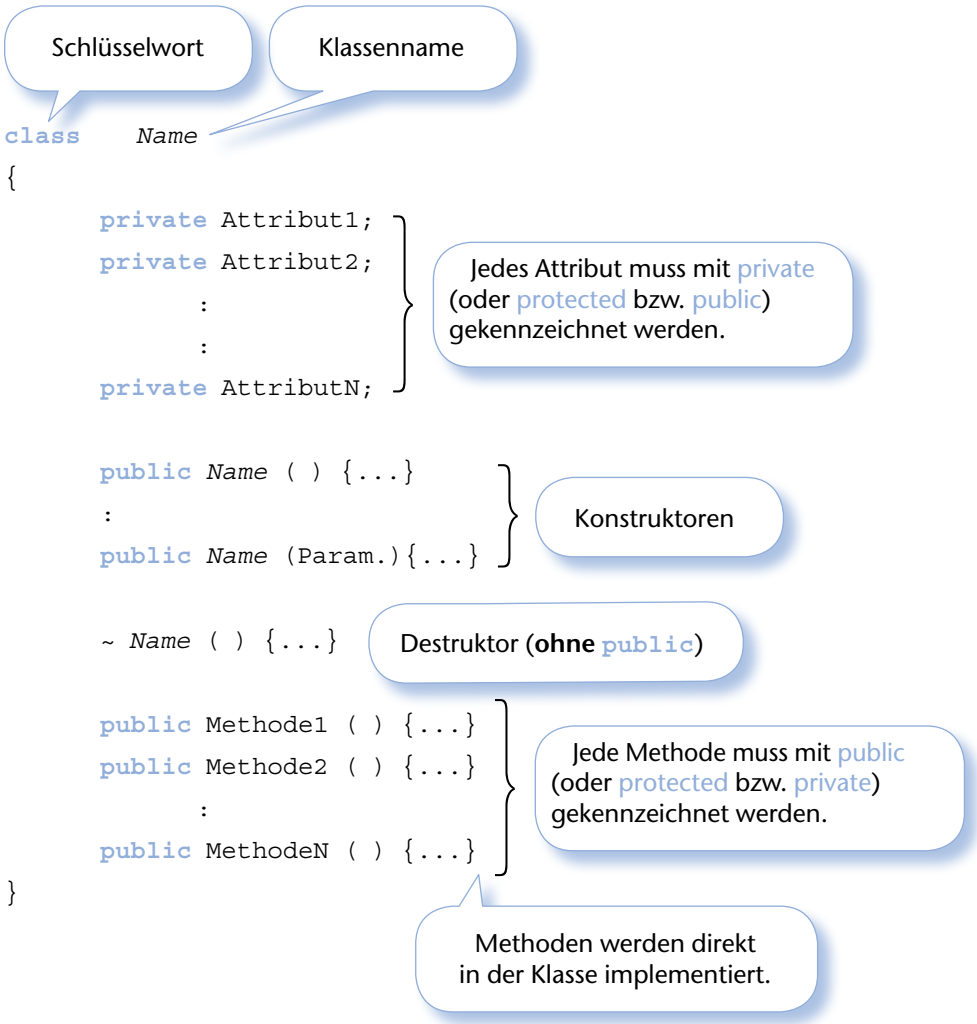
`switch(ausdruck)`	Mehrfachselektion	`switch (ausdruck)`
		`{`
		`case wert1:`
		`// Code, wenn ausdruck == wert1`
		`break;`
		
		`case wert2:`
		`// Code, wenn ausdruck == wert2`
		`break;`
		
		`default:`
		`// Code, wenn keiner der Fälle`
		`// zutrifft.`
		`break;`
		`}`

`while`	Kopfgesteuerte Iteration	`while (Bedingung)`
		`{`
		`// Code wird ausgeführt, solange`
		`// Bedingung wahr ist.`
		`}`

`do-while`	Rumpfgesteuerte Iteration	`do`
		`{`
		`// Code wird mindestens einmal`
		`// ausgeführt,und dann immer weiter,`
		`// solange Bedingung wahr ist.`
		`}`
		`while (Bedingung);`

`for( ; ; )`	Zählergesteuerte Iteration	`for (Initialisierung; Bedingung; Schritt)`
		`{`
		`// Die Initialisierung wird einmalig`
		`// zu Beginn ausgeführt. Code wird`
		`// ausgeführt, solange Bedingung`
		`// wahr ist. Anschließend wird die`
		`// Schrittanweisung ausgeführt.`
		`}`

Aufbau einer Klasse in C#:



Instanziieren 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

        }
        :
}

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

Die Klasse erbt von **CBasis**.

Expliziter Aufruf des Basisklassen-konstruktors

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);
}
```

Arrays in C#:
Anlegen eines einfachen Arrays mit *anzahl*-vielen Elementen vom Typ „Datentyp“

```
Datentyp [] Bezeichner = new Datentyp[anzahl];
```

Anlegen eines mehrdimensionalen Arrays mit entsprechend vielen Elementen vom Typ „Datentyp“. Die Anzahl der Kommata bestimmt die Dimensionen.

```
Datentyp [, ,...,] Bezeichner = new Datentyp[dim1, dim2, ...,dimN];
```

Das folgende Beispiel zeigt ein zweidimensionales Array. Es soll eine Tabelle mit 3 Zeilen und 4 Spalten symbolisieren. Die Tabelle wird sofort mit Werten initialisiert.

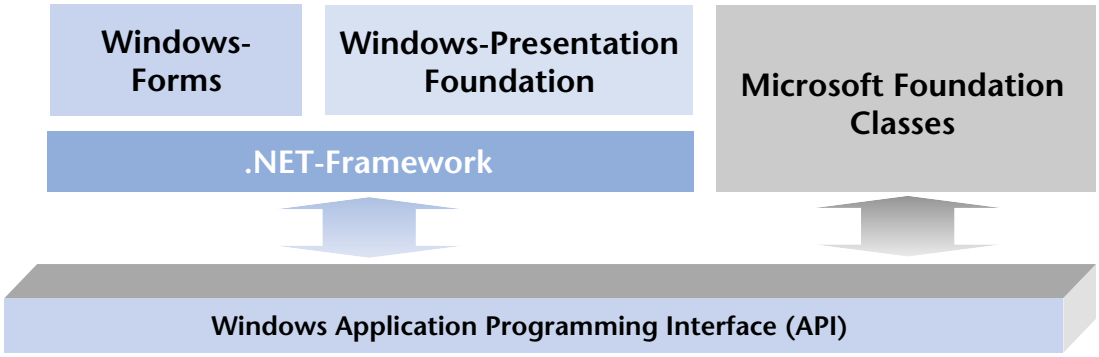
```
int [ , ] tabelle = new int[ 3 , 4 ] {  {5 , 3 , 7 , 2 },
                                        {8 , 6 , 9 , 1 },
                                        {2 , 7 , 3 , 4 } };
```

2 Windows-Forms-Programmierung – Grundlagen

2.1 Windows-Programmierung

2.1.1 Historische Entwicklung der Windows-Programmierung

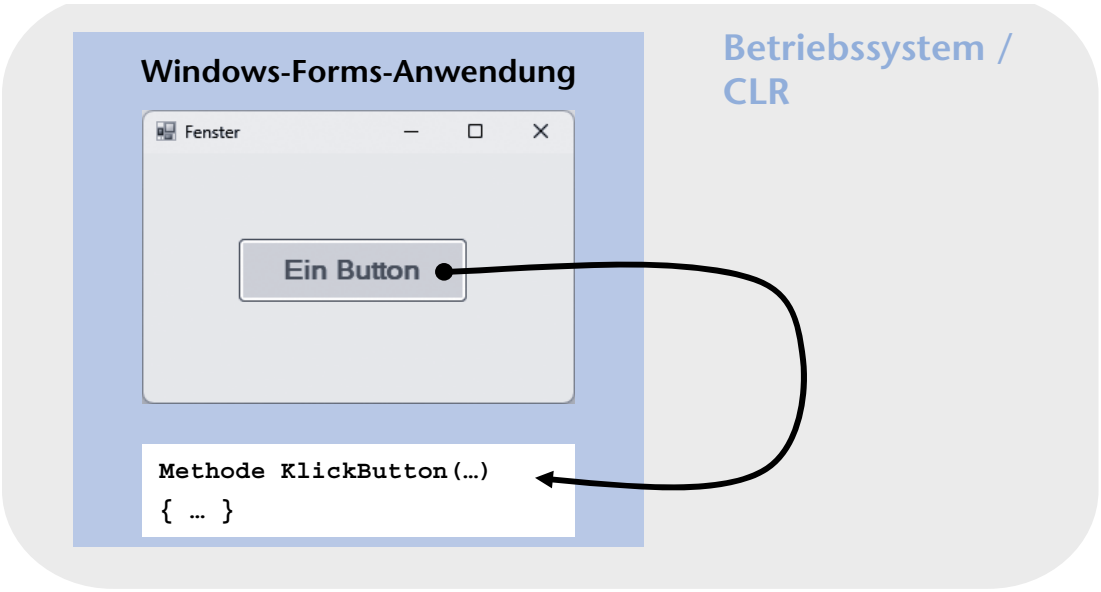
Die Windows-Programmierung gibt es seit der Version Windows 1.0, die im Jahr 1985 vorgestellt wurde. Die Windows-Programme basieren seitdem auf Fenstern. Die Programmierung von Fenstern wird auch **GUI-Programmierung** genannt (für Graphical-User-Interface-Programmierung). Damit ist eine Interaktion zwischen Mensch und Maschine gemeint, die auf einer grafisch gestalteten Oberfläche basiert (im Gegensatz zur Konsolenanwendung). Besonders wichtig ist dabei auch der Einsatz eines Zeigegerätes wie der Maus. Die Windows-Programmierung geschieht über eine Schnittstelle, die so genannte **API** (Application Programming Interface). Diese Schnittstelle bietet alle Funktionalitäten, um ein Windows-Programm zu entwickeln. Die Funktionen der API sind in den Programmiersprachen C und Assembler geschrieben und damit sehr systemnah und schnell. Die direkte Windows-Programmierung nur mit der API ist durchaus möglich, aber relativ komplex. Eine Verbesserung bei der Windows-Programmierung bot eine objektorientierte Klassenbibliothek, die **MFC** (Microsoft Foundation Classes). Diese Bibliothek wurde 1992 mit den ersten Microsoft C/C++-Compilern ausgeliefert. Im Laufe der Jahre wurde die Bibliothek erweitert und ist auch heute noch im Einsatz bei der Windows-Programmierung. Seit der Markteinführung des .NET-Framework gibt es eine weitere Alternative für die Windows-Programmierung – und zwar die Programmierung mit der Klassenbibliothek **Windows-Forms**. Das .NET-Framework kapselt die API und bietet eine objektorientierte Variante der Windows-Programmierung an. Die grundlegende Neuentwicklung der .NET-Technologie bringt auch für die Windows-Programmierung einige Vorteile im Vergleich zur konventionellen Windows-Programmierung. Neben der Windows-Programmierung mit Forms gibt es seit dem Betriebssystem Vista und dem .NET-Framework 3.0 eine weitere Bibliothek, die **Windows Presentation Foundation (WPF)**.



2.1.2 Ereignisgesteuerte Programmierung

Die bisherigen Konsolenprogramme haben mit dem Benutzer über Tastatureingaben kommuniziert. Dabei wartet ein Konsolenprogramm so lange, bis der Benutzer die Eingabe getätigt hat. Erst dann werden die nächsten Anweisungen ausgeführt. Bei der Windows-Programmierung wird ein anderes Konzept verwendet, um mit dem Benutzer zu interagieren – und zwar mithilfe der ereignisgesteuerten Programmierung. Der Benutzer kann dabei verschiedene Aktionen ausführen (beispielsweise auf einen Button klicken) und mit einem solchen Ereignis ist dann eine Methode verbunden, die ausgeführt wird (eine so genannte Ereignisbehandlungsmethode). Im Prinzip wartet ein Windows-Forms-Programm in einer Art Schleife darauf, dass ein Ereignis eintritt, welches behandelt werden muss. Dabei gibt es nicht nur Ereignisse, die ein Benutzer auslöst, sondern auch Ereignisse, die vom

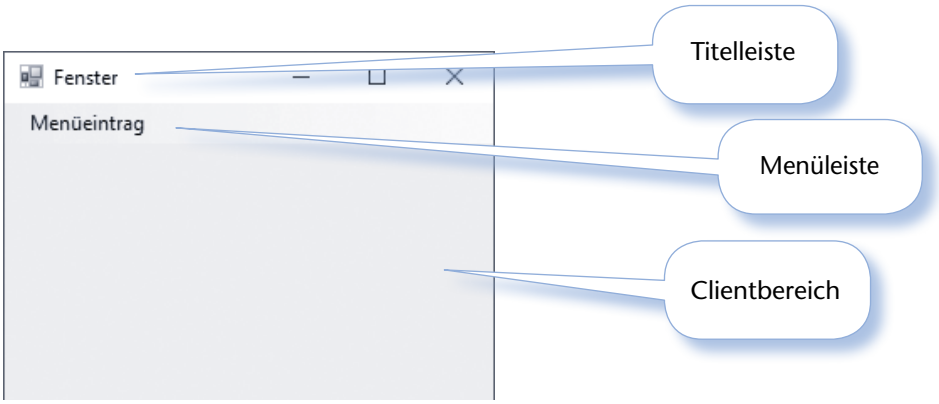
Betriebssystem ausgelöst werden können. Ein besonders wichtiges Ereignis ist dabei das `Paint`-Ereignis, welches immer dann ausgelöst wird, wenn der Inhalt des Fensters neu angezeigt werden muss. Auf dieses Ereignis und die damit verbundene Methode wird später noch detailliert eingegangen. Die folgende schematische Darstellung soll den Zusammenhang noch einmal verdeutlichen:



Die Windows-Forms-Anwendung läuft innerhalb der Common Language Runtime (CLR), die natürlich innerhalb des Betriebssystems läuft und über die API mit dem Betriebssystem kommuniziert. Das Klicken auf einen Button wird vom Betriebssystem registriert und die Anwendung erhält eine entsprechende Nachricht. Diese Nachricht führt dazu, dass eine bestimmte Methode aufgerufen wird, die natürlich genau für diesen Fall implementiert wurde.

2.1.3 Grundbegriffe der Forms-Programmierung

Die Basis einer Windows-Forms-Anwendung ist ein Fenster, auch **Formular** oder **Form** genannt. Neben einer Titelleiste kann ein Formular auch über eine Statuszeile und eine Menüleiste verfügen. Das Formular wird von einem Rahmen umgeben. Innerhalb des Formulars befindet sich der Bereich, in dem der Inhalt angezeigt wird. Dieser Bereich heißt **Clientbereich**.



2.2 Das erste Windows-Forms-Programm

2.2.1 Ein Windows-Forms-Projekt anlegen

Ein Windows-Forms-Projekt kann auf zwei Arten angelegt werden:

- Als leeres Projekt
- Als Windows-Forms-App

In einem ersten Schritt wird ein leeres Projekt angelegt. Das hat den Vorteil, dass die Entwicklungsumgebung keine Quellcode-Dateien generiert, sondern der Aufbau Schritt für Schritt vollzogen werden kann. Bei den fortgeschrittenen Themen im nächsten Kapitel wird dann die Windows-Forms-Projektform gewählt.