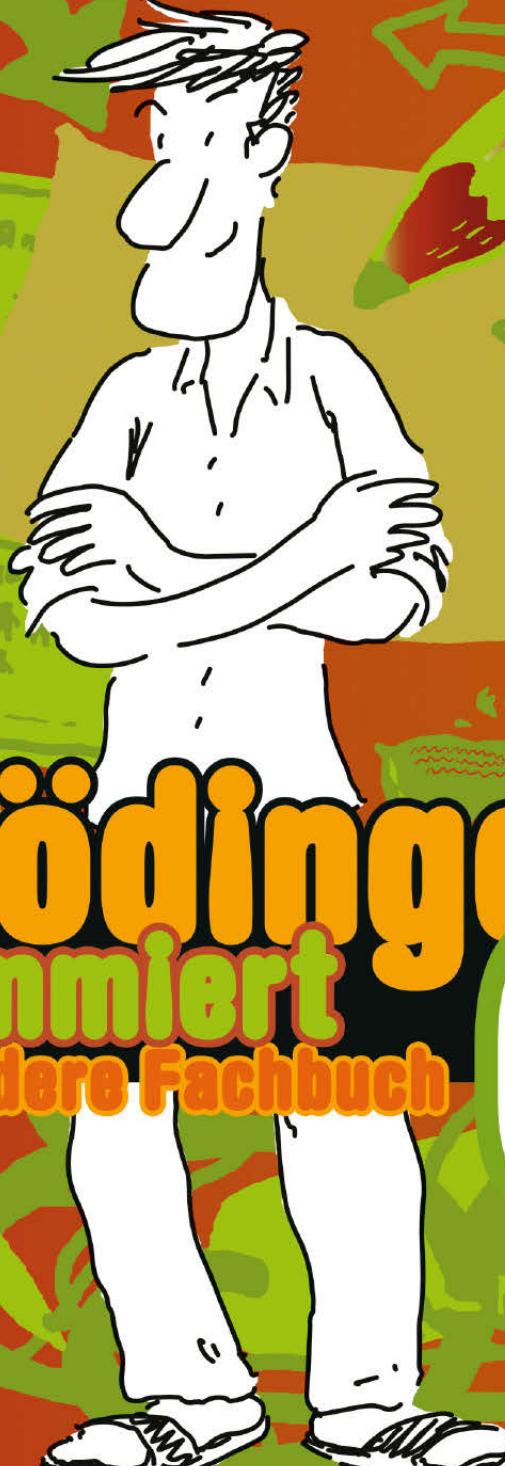


Bernhard Wurm



Schrödinger programmiert

Das etwas andere Fachbuch

Inkl.
Downloads

Mit Syntax-
Highlighting!

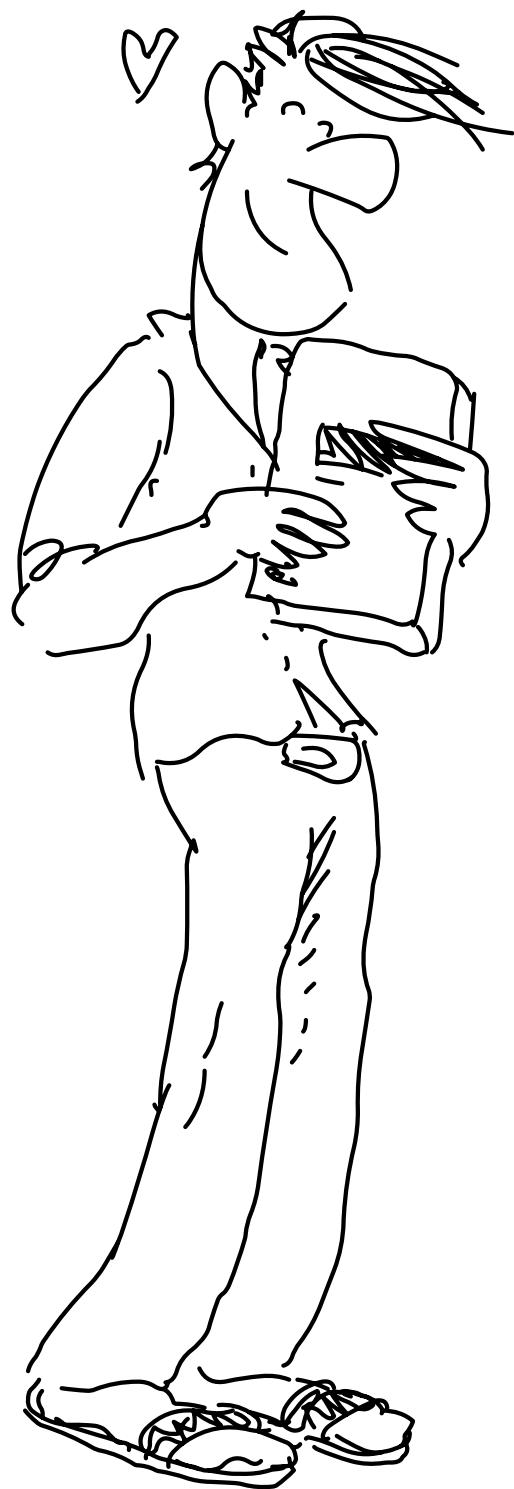
► Von den Sprachgrundlagen über XAML
bis zur komplexen Anwendung

► Ob LINQ, Generics oder C# 13:
Hol dir die Juwelen aller Versionen!

► Durchblicken, mitmachen und
genießen!

VIERTE AUFLAGE

 Rheinwerk
Computing



MIT TALENT, KATZEN-
PHOBIE UND LÄSSIGEM
SCHÜTTWERK BESTACH
SCHRÖDINGER DIE
RHEINWERK-JURY.
FÜR IHN GEHT JETZT
EIN TRAUM IN ERFÜLLUNG.

Liebe Leserin, lieber Leser,

DU HAST DIR WAS VORGENOMMEN:

C# □

Da sind wir dabei.

Lernen musst du zwar selbst, aber wir geben unser Bestes, um dich zu unterstützen:

Gut,
dass ihr das gleich
klarstellt.

Wir haben einen
hervorragenden Autor

engagiert, der sich für dich ins Zeug legt, dir Sprachfeatures und Konzepte anschaulich erklärt und deine Entwicklung zum Profi von Anfang an im Blick hat: Best Practices gehören immer dazu, und auch bei etwas anspruchsvoller Themen lässt er dich nicht im Stich. Von »Hallo Welt« bis zur eigenen mobilen App.

Versprochen.

Wir
bringen dich mit
Schrödinger

zusammen. Nein, auch der nimmt dir das Lernen nicht ab. Ehrlich gesagt, denkt er nicht einmal daran. Das wäre auch sowieso zu schade, denn du würdest

fantastische Übungen

verpassen und womöglich gar die Illustration. Aber Spaß macht es mit Schrödinger bestimmt, und ein paar **schlaue Fragen** hat er obendrein parat.

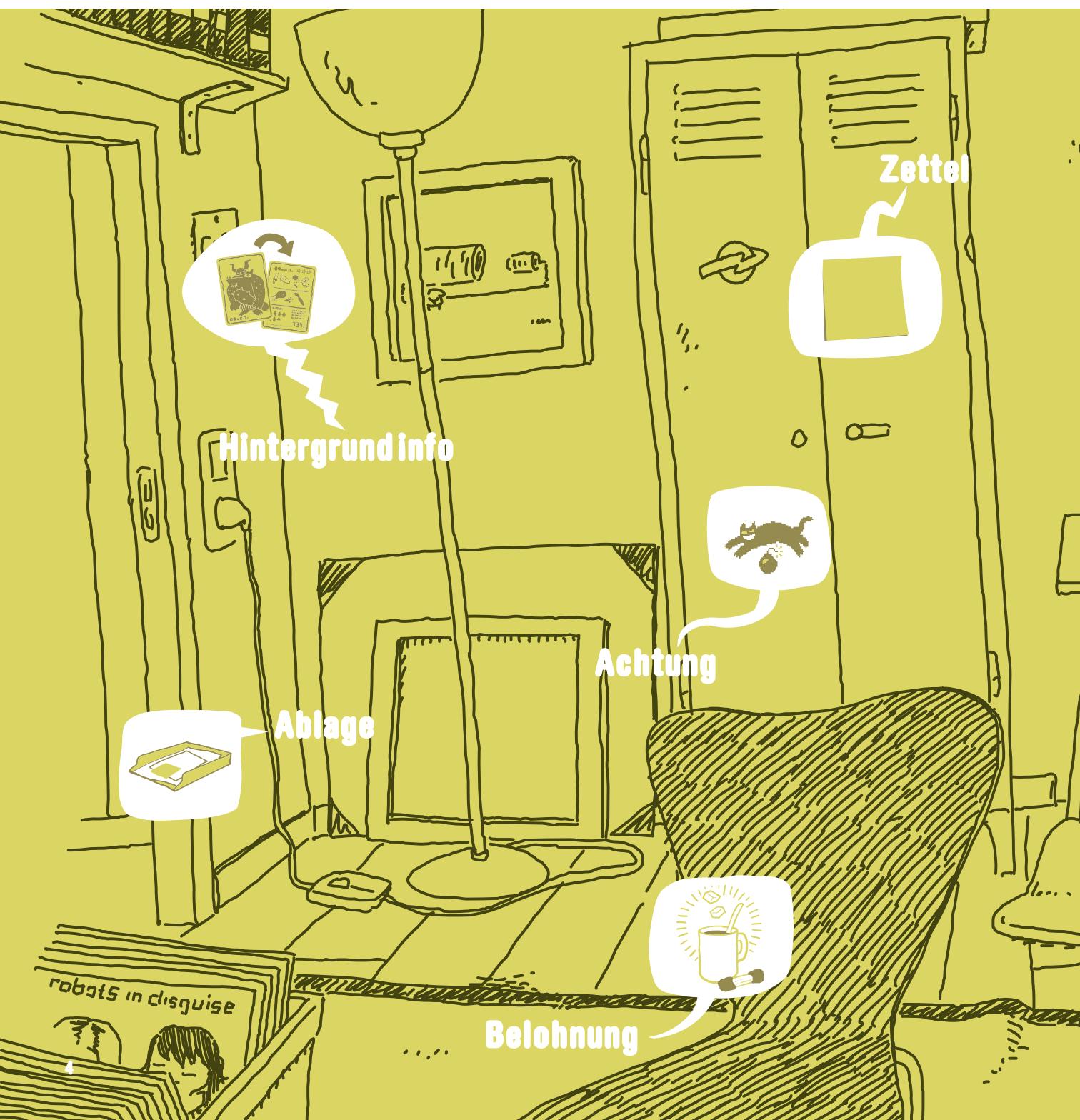
Wir haben ein
Expertenteam

herbeigeholt, das den Code einfärbt, Pfeile und Hinweise anbringt, Spuren legt und gelegentlich die Lösungen auf den Kopf stellt, damit du nicht zu früh spinxt.

Können wir jetzt loslegen?
Sonst bin ich schon mal in der Werkstatt und setze die Entwicklungsumgebung auf.

Na dann: Viel Erfolg!

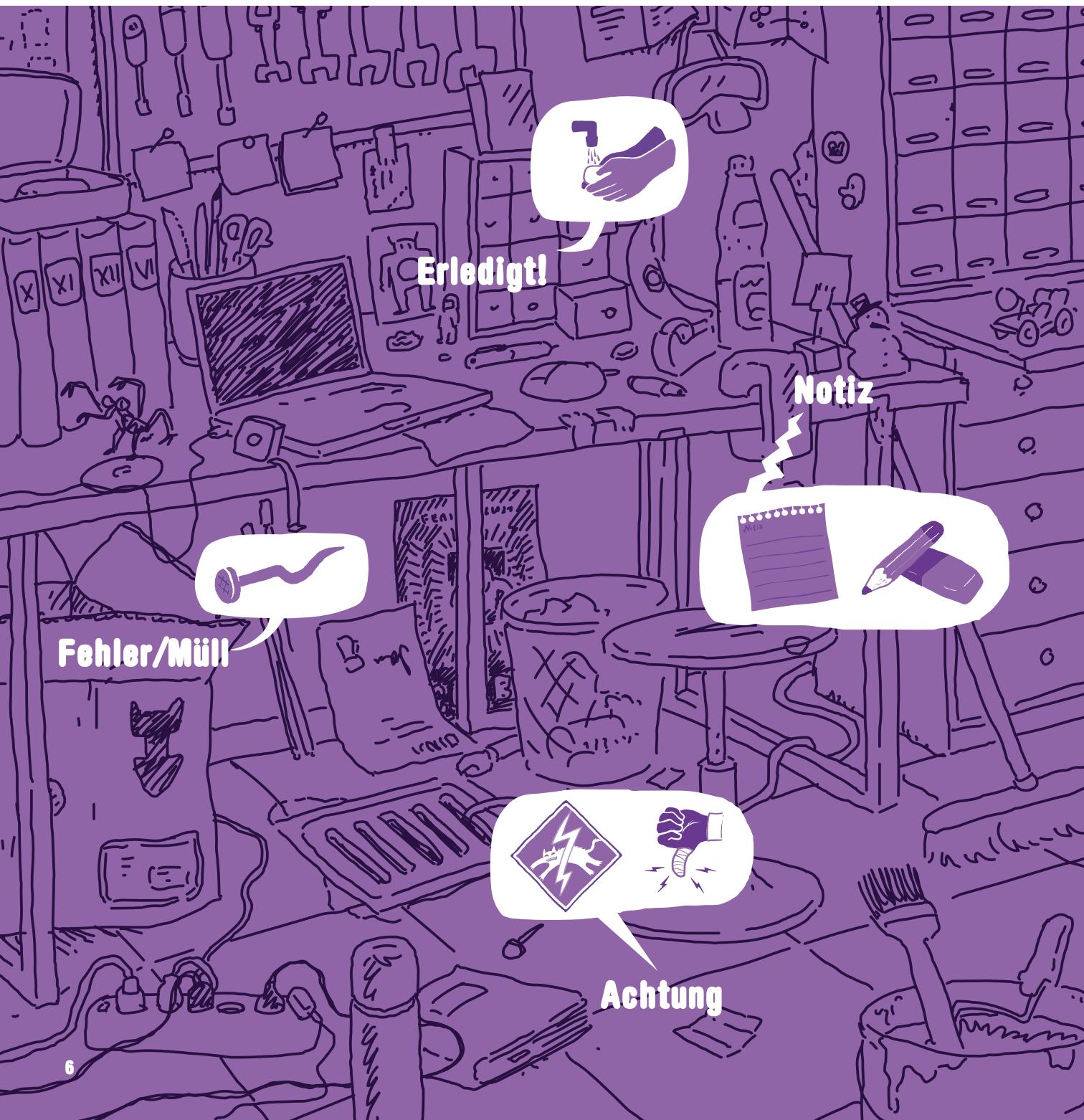
Schrödingers Büro



Die nötige Theorie, viele Hinweise und Tipps



Schrödingers Werkstatt



Unmengen von Code , der ergänzt , verbessert und repariert werden will



Schrödingers Wohnzimmer



Viel Kaffee, Übungen und die verdienten Pausen



—EINS—

Compiler und
Entwicklungs-
umgebungen

Ein guter Start ist der halbe Sieg

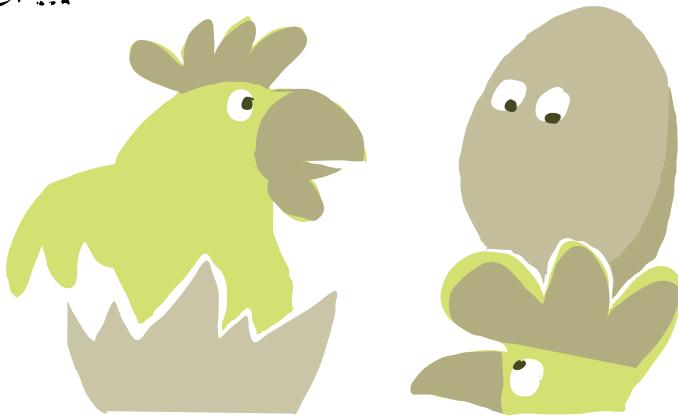
Schrödinger steht am Anfang seines neuen Jobs als C#-Entwickler. Sein Problem: Er kann noch gar kein C#. Sein erster Schritt zur Lösung: Er hat sich Hilfe geholt. Und er hat richtig Lust auf die Sache. Beste Voraussetzungen also. Jetzt ist der zweite Schritt an der Reihe: Installieren! Aber was?

Compiler und Compiler

Hallo Schrödinger, es ist schön, dass es dir schon unter den Nägeln brennt. Und wir werden auch gleich loslegen mit dem großen Spaß. Doch wie du weißt, braucht ein guter Handwerker auch ein **gutes Handwerkszeug**. Also besorgen wir dir jetzt gleich mal ein paar Dinge, die du als angehender C#-Programmierer brauchst.

Das wichtigste Programm, um mit einem C#-Programm richtig loszulegen, ist der **Compiler**. Dieser übersetzt deinen verhältnismäßig gut lesbaren Programmcode in eine andere, für den Computer einfacher verständliche Sprache.

Ich brauche also ein Programm,
um mein Programm für den
Computer verständlich zu machen?
Das ist ja wie mit der Henne und
dem Ei...



Fast noch schlimmer:

Es ist erstmal noch kein Maschinencode. Dein Programmcode wird in die **Intermediate Language** (IL-Code genannt) übersetzt. Erst **beim Ausführen des Programms** wird nun der IL-Code in **Maschinencode** übersetzt. Dies übernimmt ein weiterer Compiler – der sogenannte **Just-in-time-Compiler** (JIT-Compiler genannt).

Warte mal. Ich brauche einen Übersetzer von C# in den IL-Code und dann wieder einen von IL-Code in Maschinencode? Das ist ja umständlich!

Das wirkt vielleicht so, aber du musst wissen, **dass C# eine von mehreren Programmiersprachen ist, die auf der Entwicklungsplattform .NET laufen**. Es gibt auch andere Programmiersprachen, wie zum Beispiel Visual Basic.NET, F# usw., die ebenfalls nicht direkt in den Maschinencode übersetzt werden, und somit kannst du – sofern du so etwas willst – einfach zwischen verschiedenen Programmiersprachen wechseln und auch Bausteine von anderen Sprachen verwenden.

*C#, F#, Visual Basic, dann gibt es auch noch C, C++, Java. Glaubt denn jetzt jeder, dass er eine **eigene** Programmiersprache entwickeln muss?*



Nachdem du also mithilfe des C#-Compilers dein Programm kompiliert hast, erhältst du eine Datei, die IL-Code beinhaltet. Diese Datei nennt man **Assembly**. Ein Assembly kann, muss aber keine ausführbare EXE-Datei sein. Es kann auch eine Bibliothek – also eine DLL-Datei – sein, die verschiedene Funktionen beinhaltet, die wiederum von anderen Dateien verwendet werden können.

Aber ich dachte immer, in einer EXE-Datei steht Maschinencode? In echt steht aber IL-Code drin, und trotzdem kann sie ausgeführt werden?

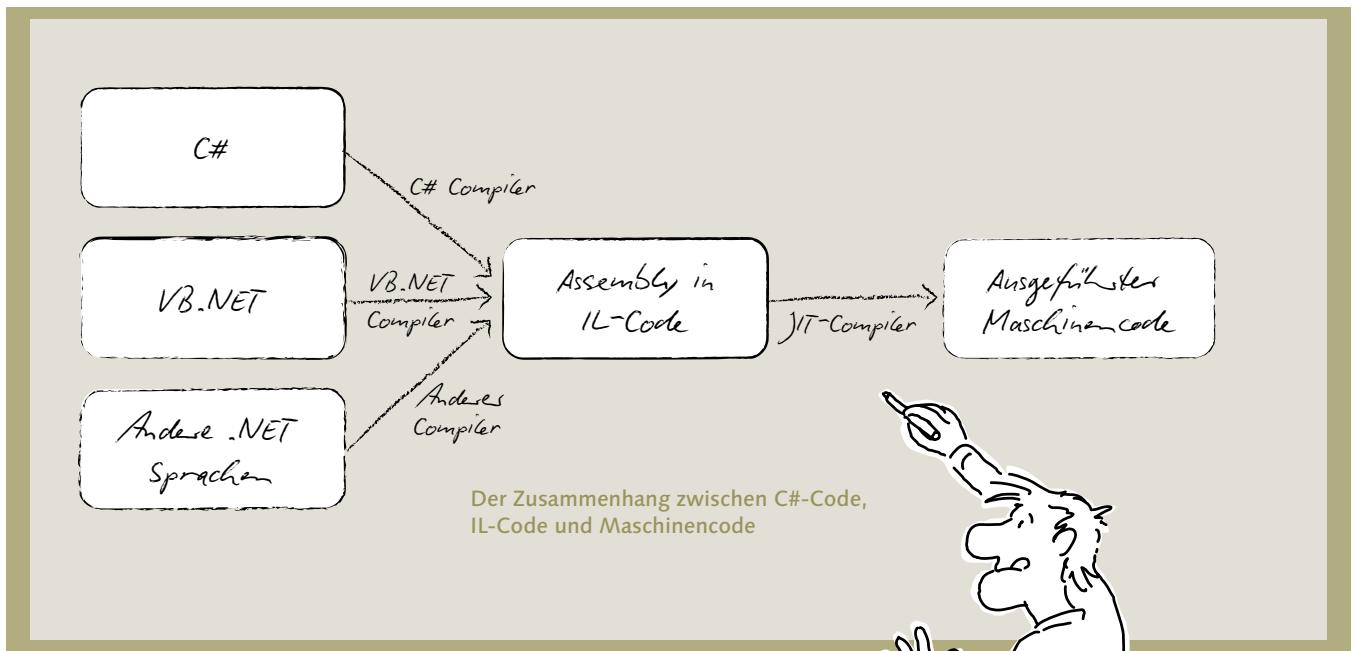
Du kannst dir das so vorstellen, dass der erste Befehl in dieser EXE-Datei ein Verweis auf die Common Language Runtime (kurz CLR) ist. Diese startet den JIT-Compiler, der den Code direkt, noch bevor dieser ausgeführt wird, in richtigen Maschinencode übersetzt und ihn dann ausführt.



[Hintergrundinfo]

Immer, wenn Code eines Assemblies ausgeführt werden soll, tritt der JIT-Compiler in Aktion. Der ist so optimiert, dass immer nur der aktuelle Codeblock kompiliert wird und nicht mehr. Das kompilierte Ergebnis wird zwischengespeichert und nur noch dieser schnelle Code ausgeführt.

Außerdem hat der IL-Code den Vorteil, dass er nicht nur in Windows funktioniert, sondern beispielsweise auch unter Linux und unterschiedlichen Rechnerarchitekturen (x86 oder ARM) laufen kann, da er **plattformunabhängig** ist und **erst bei der Ausführung in den Maschinencode übersetzt** wird.



Du hast übrigens bereits einen alten Compiler auf deinem Windows-Rechner installiert.

Der Compiler ist die **csc.exe** (csc steht für C-Sharp-Compiler) und in dem Verzeichnis **C:\Windows\Microsoft.NET\Framework64\v4.0.30319** zu finden.

Je nach Windows-Version unterscheidet sich auch die standardmäßig installierte .NET-Framework-Version etwas.

Na, dann lass uns loslegen!

Schrödinger, du willst doch das Aktuellste lernen, das es gibt, oder?

Nat\"urlich, ich bin doch nicht von 2023!



[Hintergrundinfo]

Die erste Version des .NET Frameworks und C# wurde 2002 veröffentlicht. 2016 veröffentlichte Microsoft eine Neuentwicklung mit dem Namen .NET Core. Dieses legte den Fokus auf die plattformübergreifende Entwicklung, sollte effizienter sein und Altlas-ten entfernen. Alle Neuentwicklungen werden seither dort gemacht und nur mehr teilweise im .NET Framework nachgezogen.

Wir werden uns also auf die aktuelle .NET-Version stützen und alles, was .NET Framework heißt, ignorieren.

Core wurde inzwischen wieder aus dem Namen entfernt.

[Ablage]

Die aktuelle .NET-Version erhältst du kostenlos unter <https://dotnet.microsoft.com> oder mit der Installation der Entwicklungsumgebung Visual Studio unter <https://visualstudio.microsoft.com>.

Du brauchst eine IDE!

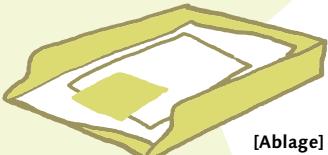
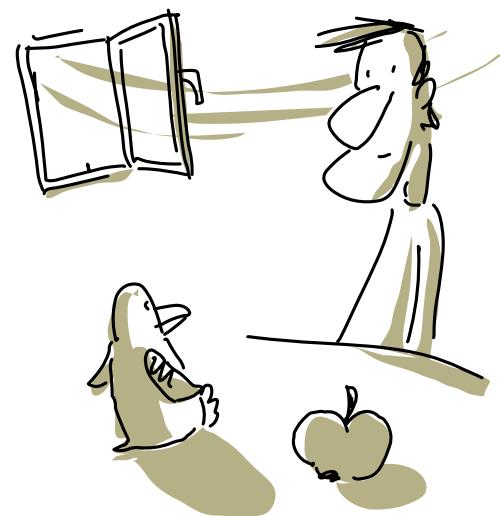
Obligatorisch ist zwar nur der Compiler, aber in einem x-beliebigen Texteditor macht programmieren keinen Spaß. Auch das Kompilieren kann komfortabler sein als auf der Kommandozeile. Die Entwicklungsumgebung – auch gerne IDE genannt – unterstützt dich bei deinen Projekten.



[Begriffsdefinition]

IDE steht für Integrated Development Environment, also eine integrierte Entwicklungsumgebung. Dieses Programm unterstützt dich bei der Entwicklung und ermöglicht auch das Kompilieren deiner Programme.

Es gibt selbstverständlich verschiedene Entwicklungsumgebungen, mit denen du C# programmieren kannst. Und du hast natürlich die Auswahl. **Selbst unter Linux** kannst du, wenn du willst, mit MonoDevelop C#-Programme entwickeln. Eine ebenfalls gute Alternative für die .NET-Entwicklung unter Windows, Mac und Linux ist **Visual Studio Code** von Microsoft. Um Anwendungen für den Windows 11-Store entwickeln zu können, benötigst du selbst Windows 11, und am besten eignet sich hierbei natürlich die Entwicklungsumgebung **direkt von Microsoft**. Daher schlage ich dir vor, du konzentrierst dich auf **Visual Studio**. Auch hier gibt es eine kostenlose Edition – die sogenannte **Community Edition**.



[Ablage]

Egal, welche Entwicklungsumgebung du verwendest, C# ist C# und bleibt C# – die Syntax ist also überall ident. Die IDEs unterscheiden sich im Allgemeinen bei Komfortfunktionen für den Entwickler.

Visual Studio Community Edition

Visual Studio von Microsoft gibt es in verschiedenen Versionen. Die großen, kostenpflichtigen Enterprise-Versionen integrieren sämtliche Funktionalitäten, die das Entwicklerherz begeht und auch einige, die du nie im Leben brauchen wirst. Es steht aber auch eine sehr gute kostenlose Versionen zur Verfügung: die **Visual Studio Community Edition**. Auch sie bietet alles was du brauchst, um **Webanwendungen, Desktopanwendungen oder Universal Windows Platform Apps** zu erstellen. Sogar wenn du für Android oder für den Apfel entwickeln möchtest, ist dies mit der Community Edition möglich. Dies ist in Wahrheit eine kostenlose Professional Edition, die für **kleine Teams, Open Source Projekte und die Wissenschaft** gedacht ist.



Noch bin ich eine
Ein-Mann-Armee,
das ist klein genug.

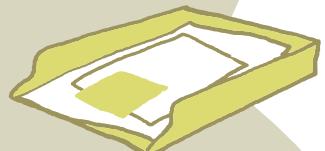
[Hintergrundinfo]

Der lange Weg der Namensfindung zu **Universal Windows Platform Apps** startete mit dem Begriff Metro Apps, der nach einem Rechtsstreit fix zu **Windows Store Apps** wurde. Mit Windows 8.1 kamen die sogenannten **Universal Apps**, die eine teilweise gemeinsame Codebasis für Windows Store und Windows Phone erlaubten. **Windows 10** und die Windows Universal Platform perfektionieren dieses Ziel: Es läuft die absolut gleiche App auf dem Raspberry Pi wie auch auf einem Tablet oder Desktop – und heißt zu Recht **Universal Windows Platform App**.



[Ablage]

Die Entwicklungsumgebungen kannst du unter www.visualstudio.com herunterladen.



Installiere dir bitte die **Visual Studio Community Edition**, und dann geht es los.

Der Spaß geht los!



Die Software ist wie gesagt kostenlos. Du musst dich jedoch mit deinem Microsoft-Account an Visual Studio anmelden, damit es länger als 30 Tage funktioniert. Das kannst du gefahrlos machen. Einfach anmelden bzw. kostenlos registrieren, falls du noch keinen Account hast, und der Spaß kann losgehen.

Ist ja klar, dass die wieder alle meine Daten haben wollen. Aber gut - kostet ja nichts.

Nachdem du die Installation abgeschlossen hast, kannst du dein erstes Projekt starten.

Anmelden mit dem Microsoft-Account



Visual Studio 2022

Open recent



Get started



Clone a repository

Get code from an online repository like GitHub or Azure DevOps



Open a project or solution

Open a local Visual Studio project or .sln file



Open a local folder

Navigate and edit code within any folder



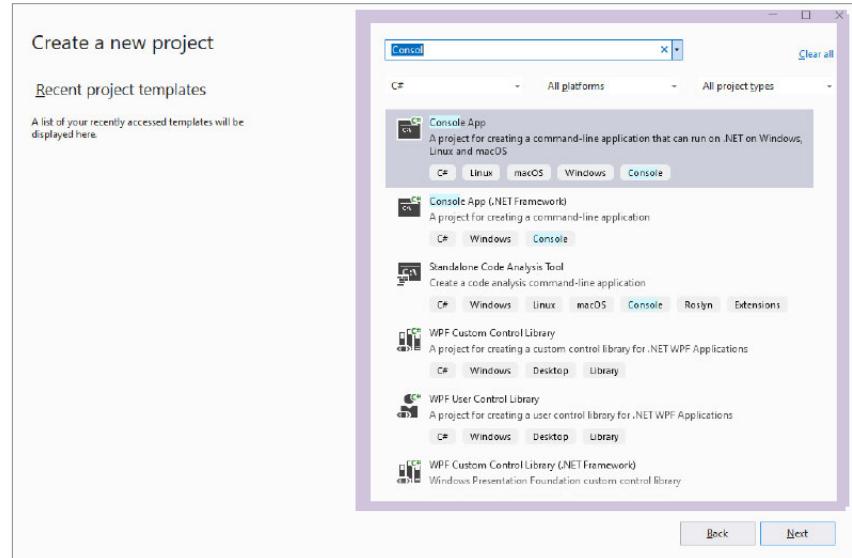
Create a new project

Choose a project template with code scaffolding to get started

[Continue without code →](#)

Visual Studio – Startbildschirm

Durch einen Klick auf **Neues Projekt** erhältst du eine Auswahl mit den möglichen Projekttypen für diese Visual-Studio-Version. Diese Version beinhaltet alles, was du für klassische Desktopanwendungen benötigst, also ältere Windows-Forms-Anwendungen, moderne WPF-Anwendungen, Konsolenprogramme und auch Klassenbibliotheken, die dir ermöglichen, Funktionen in DLL-Dateien zu packen und projektübergreifend zu verwenden.



Auswahl der Projekttypen von Visual Studio

Dein erstes Projekt

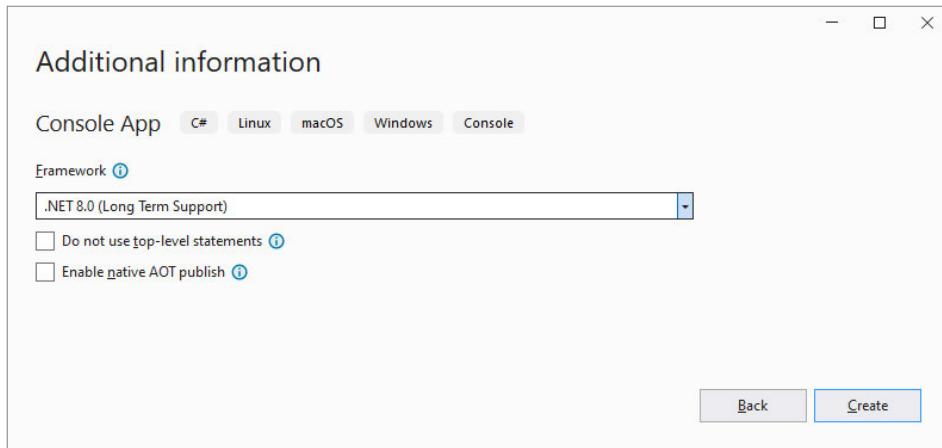
Am besten beginnst du zunächst mit einer **Konsolenanwendung**. Die eignet sich gut für die ersten Schritte, da du nicht von schönen, bunten Oberflächen abgelenkt wirst.

»Console App« gibt's da zweimal.

Viele Projekttypen gibt es einmal mit und einmal ohne den Zusatz ».NET Framework«. Wir verwenden ausschließlich die **ohne Zusatz**. Dafür sind wir auch mit Linux etc. kompatibel und modern unterwegs. Dein Projekttyp ist also einfach **Console App**.

Genau, lassen wir das alte Zeug einfach weg.
Find ich gut.

Nachdem du den Projektnamen, zum Beispiel **HalloSchrödinger**, und den Speicherort ausgewählt hast, wirst du aufgefordert, eine .NET-Version zu wählen. Denn auch von der neuen Implementierung gibt es natürlich unterschiedliche Versionen.



Framework-Auswahl

[Achtung]

Long Term Support vs. Standard Term Support

Etwa alle zwei oder drei Versionen wird eine Version mit **Long Term Support (LTS)** veröffentlicht. Diese hat den Vorteil, dass es offiziell eine längere Unterstützung gibt, nämlich 3 Jahre. Bei Versionen mit Standard Term Support (STS) endet diese bereits nach 18 Monaten. Das bedeutet auch, dass du diese Version früher upgraden musst und beispielsweise in der Azure Cloud etc. ohne Upgrade relativ frühzeitig nicht mehr betreiben kannst.



Ich werde weise wählen!



Im großen Hauptfenster siehst du deinen Programmcode. Das ist dein Spielplatz, wo du an deinem Code arbeitest, ihn verfeinerst, Fehler suchst und möglicherweise manchmal fast verzweifelst. Im rechten Bereich findest du den Projektmappen-Explorer oder den sogenannten Solution-Explorer, wenn du Visual Studio auf Englisch installiert hast. Dieser zeigt dir alle Dateien in deiner Projektmappe an.



```
// See https://aka.ms/new-console-template for more information
Console.WriteLine("Hello, World!");
```

Grundgerüst einer Konsolenanwendung

Eine Software ist wie ein **großer Schrank**. Er sieht zunächst wie ein großes kompaktes Etwas aus, aber wenn du ihn öffnest, hast du viele verschiedene Schubladen und Fächer mit Dingen darin.



Eine Software besteht ebenfalls aus vielen einzelnen Teilen und oftmals aus **verschiedenen Projekten**. Die Projektmappe besteht also aus Projekten und ein Projekt wieder aus vielen verschiedenen Dateien und Programmcodes.

Am Ende ergibt das alles zusammen deine Software.



[Notiz]

Jedes Programm startet automatisch an einem bestimmten Einstiegspunkt. Dieser ist im Allgemeinen als **Main**-Funktion bekannt. Inzwischen wird dir dieses Codegerüst rundherum nicht mehr angezeigt und du startest direkt in der ersten Zeile mit deinem Code. Wenn du die **Main**-Funktion sehen möchtest, wähle bei der Framework-Auswahl das Häkchen **Do not use Top-Level statements**.

A screenshot of the Microsoft Visual Studio IDE. The main window shows the code editor with a C# file named "Program.cs" containing the following code:

```
namespace HalloSchroedinger
{
    internal class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Hello, World!");
        }
    }
}
```

The Solution Explorer on the right side shows a single project named "HalloSchroedinger" with one item: "Program.cs". The status bar at the bottom indicates "Restored D:\Bernhard\42Entwickler Drive\Bücher\Schrödinger Programmiert C#\4. A...".

Gleicher Programmcode ohne Top-Level-Statements

Was bedeutet das Ganze hier?

Lass uns einfach starten
und das Feature Top-Level-Statements verwenden!

Ja, einfach ist gut!

1 // See <https://aka.ms/new-console-template> for more information

*1 Das ist nur ein Kommentar, der wird vom Compiler ignoriert. Einzelne Kommentare starten mit //.

Console.WriteLine("Hello, World!"); 2

*2 Der Text "Hello World" wird mithilfe der Funktion Console.WriteLine in der Konsole angezeigt.

[Notiz]

Texte (sogenannte Strings) sind in C# immer innerhalb von Anführungszeichen zu setzen, ansonsten weiß der Compiler nicht, dass es sich um einen Text und nicht um irgendetwas anderes handelt.



```
Console.WriteLine("Bitte Namen eingeben:");
var name = Console.ReadLine(); 3
Console.WriteLine("Hallo " + name);
```

*3 Natürlich kannst du mit den Elementen, die der Benutzer eingeibt, auch arbeiten. Dazu musst du eine Variable definieren. Die Variable ermöglicht dir, Werte im Arbeitsspeicher des Computers zu halten und damit zu arbeiten. Damit der Compiler weiß, dass es sich um eine Variable handelt, schreib einfach var vor den Variablennamen, den du frei wählen kannst.

Später werde ich dir die verschiedenen Variablentypen zeigen, aber dazu gleich mehr.

Der **Variablen** kannst du nun mittels **Console.ReadLine** einen Wert zuweisen, und zwar genau den Wert, den der Benutzer eingeibt.

Wie du schon gesehen hast, kannst du mittels **Console.WriteLine** Text ausgeben. Es gibt aber auch ein **Console.ReadLine**, womit du Text einlesen kannst, den der Benutzer eingibt. Durch **Console** kannst du auf das Konsolenfenster zugreifen und entsprechend Eigenschaften setzen, wie zum Beispiel den Fenstertitel, oder auch Funktionen aufrufen, etwa zur Ein- oder Ausgabe.

*Das scheint ja einfach zu sein, aber warne kurz,
das will ich gleich mal ausprobieren.*

Gerne. Drück einfach auf **Start**, um das Programm zu kompilieren und auszuführen.

Bravo, Schrödinger. Du hast dein erstes Programm geschrieben.
Und dein Programm hat dich gleich begrüßt.

Toll!

Wenn du in Visual Studio auf den **Start**-Knopf drückst, wird automatisch der Compiler angeworfen. Außerdem wird das Programm gleich ausgeführt, und der JIT-Compiler springt zu deiner **Main**-Funktion, kompiliert diese und führt deinen Code nativ als Maschinencode aus.

Nativ?

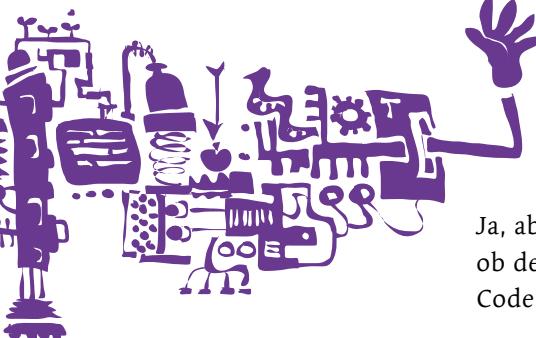
[Notiz]

Nativ bedeutet, der Code wird direkt in Maschinencode von der CPU ausgeführt und nicht über weitere Umwege oder interpretierte Zwischensprachen.



*Puh, der Computer treibt
ganz schön viel Aufwand für
eine einfache Begrüßung.*

Ja, aber der Ablauf ist immer der gleiche, unabhängig davon, ob dein Programm aus 1.000.000 Zeilen oder lediglich drei Zeilen Code besteht.



Theorie und Praxis



[Einfache Aufgabe]

Zeig mir doch mal, ob du alles verstanden hast. Verbinde die Satzzeile mit einem Bleistift.



Der C#-Compiler erzeugt ...

... eine IDE.

Der JIT-Compiler erzeugt ...

... gibt etwas auf der Konsole aus.

Visual Studio ist ...

... liest eine Zeile von der Konsole.

Console.ReadLine ...

... Maschinencode.

Console.WriteLine ...

... IL-Code.

Lösung:
Der C#-Compiler erzeugt IL-Code.
Der JIT-Compiler erzeugt Maschinencode.
Visual Studio ist eine IDE.
Console.ReadLine liest eine Zeile von der Konsole.

Console.WriteLine gibt etwas auf der Konsole aus.

Der C#-Compiler erzeugt IL-Code.
Der JIT-Compiler erzeugt Maschinencode.

Lösung:

Wunderbar, Schrödinger. Die Theorie hast du verstanden. Und dass du die Praxis auch verstanden hast, kannst du anhand der nächsten Aufgabe beweisen:



[Schwierige Aufgabe]

Schreibe doch ein kleines Programm, das Vor- und Nachnamen nacheinander abfragt und anschließend den Namen vollständig ausgibt.

Wenn ich `Console.ReadLine()` schreibe, startet das Programm nicht und es ist rot unterkriegt.

Was läuft da falsch?



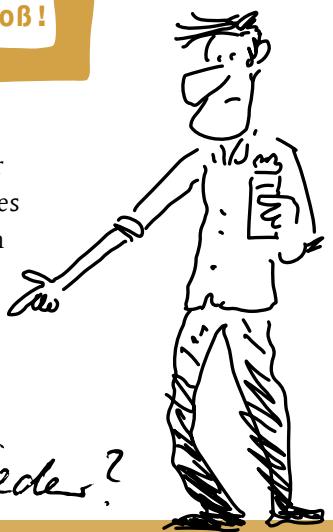
[Achtung/Vorsicht]

C# ist Case-Sensitive. Achte also auf die Groß- und Kleinschreibung der Befehle und Funktionsaufrufe.

`Console.ReadLine()` X

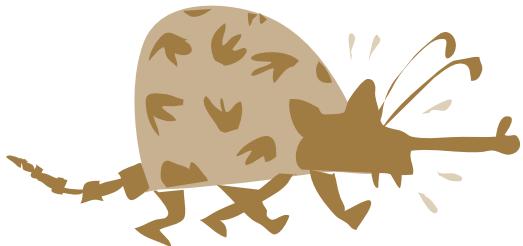
Das L in `ReadLine()` gehört groß!

Das Programm startet nur, wenn der Compiler keinen Fehler findet, nur dann kann er es übersetzen und starten. Dennoch kann es sein, dass dein Programm nicht tut, was es soll, weil du zwar keinen Fehler in der Syntax (im Satzbau) hast, doch einen logischen Fehler. In dem Fall hilft dir Debugging weiter.



Was ist jetzt Debugging wieder?

Debuggen hilft dir bei der **Fehlersuche**. Du kannst Punkte setzen, an denen das Programm einfach stehen bleibt, damit du dir Variablenwerte ansehen kannst, usw.



Die Lösung:

```
Console.WriteLine("Vorname eingeben:");
var vorname = Console.ReadLine();
Console.WriteLine("Nachname eingeben:");
var nachname = Console.ReadLine();
Console.WriteLine(vorname + " " + nachname);
```



[Belohnung]

Wenn dein Programm funktioniert, dann kannst du jetzt gerne ein bisschen von deiner Karriere als Spieleentwickler bei Blizzard träumen.

Was gelernt!

Denkst du jetzt, du hättest noch nichts gelernt?
Dann pass mal auf, ich habe dir das Wichtigste aufgeschrieben:

- C# ist eine der vielen Programmiersprachen, die auf der .NET-Plattform basieren. Auch Visual Basic, F#, Python.NET etc. basieren auf .NET, wobei C# nach wie vor das Flaggschiff dieser Plattform ist.
- Die aktuellen .NET-Versionen sind Open Source und laufen auch auf macOS oder Linux.
- Die Intermediate Language ist die Zwischensprache, in die jede .NET-Sprache übersetzt wird. Egal, ob du C# oder Visual Basic programmierst, nach dem Kompilieren kommt IL-Code heraus.
- Visual Studio ist eine von mehreren möglichen Entwicklungsumgebungen, mit denen du C# programmieren kannst. Wenn du willst, kannst du sogar in der Kommandozeile kompilieren.
- Die LTS-Versionen von .NET haben mindestens 3 Jahre Support, während die STS-Versionen lediglich 18 Monate Support genießen.
- Mit **F5** startest du den Debugger. Der hilft dir bei der Fehlersuche und wird noch dein Freund werden!

—ZWEI—

Datentypen
und deren
Behandlung

Ein netter Typ

Wie heißt es so schön, guten Freunden gibt man doch einen Kaffee – oder? Ich würde sagen, ich stelle dir ein paar freundliche Typen vor, und am Ende trinken wir eine schöne heiße Tasse Kaffee.

Das klingt gemütlich, findet Schrödinger und entspannt sich. Gemütlich? Sieben Typen allein für Zahlen? Konvertieren, kompatibel, Kommentare? Doch Schrödinger bleibt locker und lernt dabei sogar noch, mit Kamelen umzugehen.

INHALTSVERZEICHNIS

Kapitel 1: Ein guter Start ist der halbe Sieg

Compiler und Entwicklungsumgebungen

Seite 25

Compiler und Compiler	26	Dein erstes Projekt	33
Du brauchst eine IDE!	30	Theorie und Praxis	39
Visual Studio Community Edition	31	Was gelernt!	42
Der Spaß geht los!	32		

Kapitel 2: Ein netter Typ

Datentypen und deren Behandlung

Seite 43

Dieses Glas für diesen Wein	44	Das ständige Hin und Her zwischen ja und nein	60
Grundlagen im Kamelreiten	48	Gut kommentieren!	61
Übungen für den Barkeeper	50	Kommentare im Einsatz	62
Rechnen mit Transvestiten	51	Andere für sich denken lassen	62
Alles nur Klone!	56	Compiler-Spiele	63
Ja oder nein?	57	Viele neue Freunde	64
Was gibt's zu essen?	58		

Kapitel 3: Alles unter Kontrolle

Bedingungen, Schleifen und Arrays

Seite 65

Bedingungen	66	Durch Variationen bleibt es interessant	70
In der Kürze liegt die Würze	68	Der Herr der Fernbedienung	72

Ist noch Bier da?	74	Ich habe es mir anders überlegt	88
Einer von vielen	75	Oder mach doch weiter...	89
Es geht auch ohne case	76	Zurück zu den Schuhschränken	90
Switch-Expressions	77	Wenn aus einem Schuhschrank	
Pattern-Matching	78	eine Lagerhalle wird	91
Zwillinge	79	Wiederholung, Wiederholung!	93
Ein Schuhschrank muss her	82	Code muss man auch lesen können	94
Arbeiten in den Tiefen des Schuhschranks – von Kopf bis Fuß	83	Jetzt kommt das coole Zeug!	96
Die ganze Welt ist Mathematik und aller guten Dinge sind drei vier	85	Arrays verbinden ab C# 12	100
Schau's dir an mit dem Debugger	86	Der Blick durchs Fenster	102
Solange du nicht fertig bist, weitermachen...	87	... oder einmal alles	103
		Nochmals durchgekaut, damit es sicher sitzt!	103

Kapitel 4: Sexy Unterwäsche – von kleinen Teilen bis gar nichts

Strings, Characters und Nullable Types

Seite 105

Zeichenketten – Strings	106	Verdrehte Welt	114
Kleine Teile – einzelne Zeichen	107	Sein oder nicht sein?	118
Kleine und große Teile	108	Verweise auf nichts	120
Einfacher und schneller	109	Nichts im Einsatz	122
Noch einfacher: Variablen im Text verwenden	112	Damit bei so viel null nichts verloren geht	123
Etwas Besonderes sollte es sein	113		

Kapitel 5: Eine endliche Geschichte

Enumerationen

Seite 125

Rot – Gelb – Grün	126	Eine Mischung aus Zahlen und Arrays	134
Tageweise	128	Stimmungsbilder in der Praxis	137
Tell me why I don't like mondays...	131	Haltet das Banner hoch!	141
WoW-Völker	132	Auf wenige Sätze heruntergebrochen	144

Kapitel 6: Teile und herrsche

Methoden

Seite 145

Teilen statt Kopieren	146	Zum Schluss noch ganz kurz	161
Originale und überteuerte Kopien	149	Ich will das ganz anders oder auch gar nicht –	
Eins ist nicht genug	153	Methoden überladen	163
Ich rechne mit dir	154	Das Ganze noch einmal umgerührt	166
Wenn sich nichts bewegt und alles statisch ist	155	Ein knurrender Magen spornt bestimmt	
Ich hätte gerne das Original!	155	zu Höchstleistungen an	168
Sommerschlussverkauf – alles muss raus	156	Originale zurücksenden	169
Tauschgeschäfte, die nicht funktionieren	158	Maximale Originale	171
TryParse und Enums	159	Eine kleine Zusammenfassung für dich	172

Kapitel 7: Klassengesellschaft

Objekte, Eigenschaften und Sichtbarkeiten

Seite 173

Mein Alter, meine Augenfarbe,		Geburtenkontrolle	194
mein Geburtsdatum	174	Geburtenkontrolle und detektivisches Talent	197
Eine Aufgabe für den Accessor	178	Mehrlingsgeburt	202
Ich sehe was, was du nicht siehst	179	Partielle Klassen	204
Eigenschaften aufpoliert und bereit		Meine partiellen Daten	206
für die Bühne	180	Gemeinsame Werte von dicken Freunden	207
Tanzen mit Elvis – wenn keiner da ist,		Eigene Wertetypen	208
ist keiner da	182	Strukturen überall	210
Geheimniskrämerei und Kontrollfreak	183	Strukturen ohne Namen	214
Darf ich jetzt oder nicht?	184	Ich habe viele Namen	216
Zusammen, was zusammengehört!	188	Eigene Typen nochmals vom Sofa aus betrachtet	218
Zusammen und doch getrennt	190	Die Nachteile der Wertetypen ausgetrickst	221
Laufen, kämpfen, sterben	192	Gelernt ist gelernt!	223
Vom Leben und Sterben	193		

Kapitel 8: Es wird Zeit für Übersicht!

Namespaces

Seite 225

Eine Ordnung für die Klassen	226	Visual Studio findet die Namespaces für dich	234
Was ist denn nur in diesem Namespace vorhanden?	229	Statische Klassen einbinden	234
Vorhandene Systembausteine	232	Mathematik für Einsteiger	235
		Zum Mitnehmen	236

Kapitel 9: Erben ohne Sterben

Objektorientierte Programmierung

Seite 237

Geisterstunde	238	Geister haben viele Gestalten	249
Schleimgeister sind spezielle Geister	240	Geister, die sich nicht an die Regeln halten	252
Fünf vor zwölf	242	Gestaltwandler unter der Lupe	253
Geister fressen, Schleimgeister fressen, Kannibalen fressen – alles muss man einzeln machen	248	Nochmals drüber nachgedacht	254
Enterben	249	Hier noch ein Merkzettel	258

Kapitel 10: Abstrakte Kunst

Abstrakte Klassen und Interfaces

Seite 259

Abstrakte Klassen	260	Eine Cola bitte	274
Unverstandene Künstler	262	Freundin vs. Chef – Runde 1	276
Das Meisterwerk nochmals betrachtet	264	Bei perfekter Verwendung	277
Abstrakte Kunst am Prüftisch	265	Freundin vs. Chef – Runde 2	278
Allgemein ist konkret genug	267	Freundin vs. Chef – Runde 3	280
Fabrikarbeit	268	Interfaces außer Rand und Band	281
Alles unter einem Dach	269	In der Praxis: Mehr als nur Beschreibungen	283
Kaffee oder Tee? Oder doch lieber eine Cola?	270	Abstraktion und Interfaces auf einen Blick	287
Kaffeemaschine im Einsatz	272		

Kapitel 11: Gleich und doch ganz anders

Records, der Star unter den eigenen Datentypen

Seite 289

Immutability – die Würfel sind gefallen	290	Orcs, Trolle und Elfen als Klassen, Strukturen und Records	293
---	------------	--	------------

Kapitel 12: Airbags können Leben retten

Exceptionhandling

Seite 297

Mach's stabil!	298	Sträucher im Sägewerk – ArgumentException	310
Einen Versuch war es wert	300	Bezahlung ohne Ware – ArgumentNullException	310
Nur unter bestimmten Umständen	303	Bewusste Fehler	311
Fehler über Fehler	304	Selbst definierte Fehler	312
Über das Klettern auf Bäume	308	Fehler in freier Wildbahn	313
Klettern auf nicht vorhandene Bäume – NullReferenceException	308	Das Matruschka-Prinzip	314
Auf Sträucher klettern – FormatException	309	Alles noch einmal aufgerollt	316
		Dein Fehler-Cheat-Sheet	320

Kapitel 13: Ein ordentliches Ablagesystem muss her

Collections und Laufzeitkomplexität

Seite 321

Je größer der Schuhschrank, desto länger die Suche	322	Selbstwachsende Schuhschränke	334
Komplizierte Laufschuhe	323	Eine Array-Liste	335
Geschwindigkeitsprognosen	326	Ringboxen	336
Es muss nicht immer gleich quadratisch sein	328	Listige Arrays und ihre Eigenheiten	337
Geschwindigkeitseinschätzung und Buchstabensuppe	331	Listige Arrays und ihre Verwendung	337
		The Need for Speed	338
		Es wird konkreter	339

Sortieren bringt Geschwindigkeit – SortedList	342	Alles eindeutig – das HashSet	357
Listenreiche Arbeit	344	Schnelles Arbeiten mit Sets	358
Es geht noch schneller!	346	Das große Bild	360
Im Rausch der Geschwindigkeit	348	Der große Test, das Geheimnis und die Verwunderung	363
Alternative Initialisierungen	350	Noch einmal durchleuchtet	368
Wörterbücher in der Anwendung		Dein Merkzettel rund um die Collections aus Laufzeiten	373
... oder was im Regelfall schiefgeht	351		
Von Bäumen und Ästen	355		
Ein Verwendungsbeispiel	356		

Kapitel 14: Allgemein konkrete Implementierungen

Generizität

Seite 375

Konkrete Typen müssen nicht sein	376	Aus allgemein wird konkret	388
Das große Ganze	377	Hier kommt nicht jeder Typ rein.	389
Mülltrennung leicht gemacht	378	Ähnlich, aber nicht gleich!	390
Der Nächste bitte	381	Varianzen hin oder her	392
Allgemein, aber nicht für jeden!	383	Varianzen in der Praxis	395
Immer das Gleiche und doch etwas anderes	385	WoW im Simulator	398
Fabrikarbeit	387	Damit's auch hängen bleibt	400

Kapitel 15: Linke Typen, auf die man sich verlassen kann

LINQ

401

Linke Typen, auf die man sich verlassen kann	402	Listen zusammenführen	409
Shoppen in WoW	405	Fix geLINQt statt handverlesen	417
Gesund oder gut essen?	408	Merkzettel	420

Kapitel 16: Blumen für die Dame

Delegaten und Ereignisse

Seite 421

Ein Butler Delegat übernimmt die Arbeit	422	Eine Runde für alle	435
Im Strudel der Methoden	425	Auf in die Bar!	436
Die Butlerschule	428	Wiederholung, Wiederholung	439
Die Wahl des Butlers	431	Die delegierte Zusammenfassung	442
Ereignisreiche Tage	432		

Kapitel 17: Der Standard ist nicht genug

Extension-Methoden und Lambda-Expressions

Seite 443

Extension-Methoden	444	Gruppieren	458
Auf die Größe kommt es an	448	Verknüpfen	459
Erweiterungen nochmals durchschaut	450	Gruppieren und Verknüpfen kombiniert	460
Softwareentwicklung mit Lambdas	452	Left Join	461
Lambda-Expressions auf Collections loslassen	455	VerLINQte LAMbdAS	463
Ein Ausritt auf Lamas	456	Lamas im Schnelldurchlauf	466
Filtern	456		

Kapitel 18: Die Magie der Attribute

Arbeiten mit Attributen

Seite 467

Die Welt der Attribute	468	Der Attribut-Meister erstellt eigene Attribute!	480
Die Magie erleben	470	Meine Klasse, meine Zeichen	482
Das Ablaufdatum-Attribut	472	Selbstreflexion	484
Die Magie selbst erleben	473	Die Psychologie lehrt uns:	
Eine magische Reise in dein Selbst	474	Wiederholung ist wichtig!	488
In den Tiefen des Kaninchenbaus	477		

Kapitel 19: Ich muss mal raus

Dateizugriff und das Internet

Seite 489

Daten speichern	490	Dem Fließband vorgeschalteter Fleischwolf	519
Rundherum oder direkt rein	491	Nutze die Attributmagie!	521
Rein in die Dose, Deckel drauf und fertig	493	X(M)L entspricht XXL	522
Deine Geheimnisse sind bei mir nicht sicher	494	Der Größenvergleich	523
Das Mysterium der Dateiendungen	497	Die kleinste Größe – JSON	524
Das Gleiche und doch etwas anders	500	Wir sind viele	525
Das Lexikon vom Erstellen, Lesen, Schreiben, Umbenennen	501	Schr\u00fcldinger	529
Ran an die Tastatur, rein in die Dateien	506	Das World Wide Web. Unendliche Weiten	532
Von der Sandburg zum Wolkenkratzer	508	Deine Seite, meine Seite	534
Fließbandarbeit	512	Probe, Probe, Leseprobe	536
Wenn das Fließband nicht ganz richtig läuft	515	Punkt für Punkt fürs Hirn	538

Kapitel 20: Komm zurück, wenn du fertig bist

Asynchrone und parallele Programmierung

Seite 539

Zum Beispiel ein Download-Programm	540	async/await/cancel	560
Was so alles im Hintergrund laufen kann	547	Unkoordinierte Koordination	562
Gemeinsam geht es schneller	549	Anders und doch gleich	567
Jetzt wird es etwas magisch	553	Gemeinsam Kuchen backen	568
Wenn jeder mit anpackt, dann geht alles schneller	555	Wenn das Klo besetzt ist	573
Rückzug bei Kriegsspielen	558	Das Producer-Consumer-Problem	573
		Dein Spickzettel	579

Kapitel 21: Nimm doch, was andere schon gemacht haben

Die Paketverwaltung NuGet

Seite 581

Bibliotheken für Code	582	Pakete statt Projekte	591
Fremden Code aufspüren	585	Die Welt ist schon fertig	592
Eigene NuGet-Pakete erstellen	588		

Kapitel 22: Die schönen Seiten des Lebens

Einführung in XAML

Seite 593

Unendliche Weiten	594	Layouthelferlein	613
Hinzufügen der Komponenten für MAUI-Apps in Visual Studio	597	Tabellen über Tabellen	614
Die MAUI-Architektur	598	Schrödingers Notizen	617
Diese X-Technologien	600	Das ist alles eine Stilfrage	631
Grundstruktur des Projekts	602	Von der Seite in die Anwendung	633
Ruf deine Freundin an	608	Sonne, Strand und XAML	636

Kapitel 23: Models sind doch schön anzusehen

Das Model-View-ViewModel-Entwurfsmuster

Seite 639

Einführung in MVVM	640	Alleine oder zu zweit?	668
Mein erstes eigenes Model	644	Aus Klein mach Groß und zurück	669
Eine Technik, sie alle zu binden!	649	Klein aber fein	670
Eine Eigenschaft für alle Infos	650	Notizen über Models	673
Wenn nur jeder wüsste, was er zu tun hätte	651	Auf mein Kommando	683
Los geht's! Notify-Everybody	654	Kommandierende Butler	685
Ein Laufsteg muss es sein!	657	Dem Zufall das Kommando überlassen	689
Über Transvestiten und Bindungsprobleme	666	MVVM Punkt für Punkt	694
Über Bindungsprobleme und deren Lösungen	667		

Kapitel 24: Weniger ist mehr

MVVM Community Toolkit

Seite 695

Programmcode generieren lassen	696	Das solltest du trotz Automatik noch wissen	710
Lass uns mal rechnen, aber möglichst ohne Aufwand	701		

Kapitel 25: Funktioniert das wirklich?

Unit-Testing

Seite 715

Das Problem: Testen kann lästig werden	716	Unit-Tests sind nicht alles	723
Die Lösung: Unit-Tests – Klassen, die Klassen testen	717	Testgetriebene Softwareentwicklung – oder wie du Autofahren lernst	724
Ja, ich teste!	719	Darfst du schon fahren?	725
Das Testprojekt erstellen	720	Let's do it!	730
Die Ausführung ist das A und O!	722	Dein Test-Merkzettel	731
Spezielle Attribute	723		

Kapitel 26: Schrödingers Zukunft

Vorschau auf C# 13 und .NET9

Seite 733

Schrittweise verfeinert	734	Lange Eigenschaften gekürzt	739
Params für alle	734	Neuerungen in LINQ	740
Alles erweitern!	735	Warten auf Godot	742

Index	743
-------------	-----

Schrödinger programmiert C#

Design & Elektronik

„HÄTTE ES DOCH SOLCHE BÜCHER VOR 20 JAHREN SCHON GEGEBEN!“

Christian Mantey, IT-Dozent

„Überraschend gut! Sehr zu empfehlen!“

Leser-Feedback:

„Das Layout der Seiten ist genial.“

DAS ALLES und noch viel mehr:

- 👉 Operatoren, Schleifen, Datentypen
- 👉 Klassen und Vererbung
- 👉 Wichtige Entwurfsmuster
- 👉 GUI-Entwicklung mit XAML
- 👉 LINQ einsetzen
- 👉 async/await verwenden
- 👉 Parallel Programmierung
- 👉 Dateizugriffe und Streams
- 👉 Mobile Apps mit MAUI

Schrödinger ist unser Mann fürs Programmieren. Er kann schon was, aber noch nicht C#. Zum Glück hat er einen Kumpel, der ihm alles in Ruhe zeigt. Erst einmal die Grundlagen von C#, dann nach und nach immer mehr Magie, und am Ende sogar richtig schicke GUIs.



 Rheinwerk
Computing

Leser-Feedback:

„Jetzt bin ich platt. Ich kann nicht aufhören zu lesen.“

c't zur Schrödinger-Reihe

„Ein neuer Weg bei der Vermittlung von Entwickler-Fachwissen.“

Vom Feinsten! Die volle Packung C#.

Die nötige Theorie, viele Hinweise und Tipps [im Büro], Unmengen von gutem, aber auch schlechtem Code, der verbessert und repariert werden will [in der Werkstatt], viele Übungen und wohlverdiente Pausen [zuhause im Wohnzimmer]. Mittendrin: Schrödinger. Und natürlich du.

SCHRÖDINGER GARANTIERT:

- Gründlicher Einstieg
- Profi-Konzepte von Anfang an
- Beispiele und Aufgaben mit Lösungen
- Für Einsteiger und Umsteiger perfekt

Ein echtes Fachbuch, nur eben ganz anders!



Unser Autor: Bernhard Wurm kann sich noch gut an seine Lehrjahre erinnern und bringt selbst mit, was Schrödinger jetzt braucht: Nerven, Ausdauer, Neugierde und Spaß am logischen Denken. Chapeau! So wird aus seinem Kumpel ein richtiger C#-Entwickler, den er am Ende glatt einstellen würde.



Gedruckt in Deutschland
Mineralölfreie Druckfarben
Zertifiziertes Papier

Für Windows
Programmierung/C#
ISBN 978-3-367-10623-3

€ 49,90 [D] € 51,30 [A]

