Ingo Patett

C# 4 U

Programmentwicklung mit C#

1. Auflage

Bestellnummer 01671

Bildungsverlag EINS
westermann

Die in diesem Werk aufgeführten Internetadressen sind auf dem Stand zum Zeitpunkt der Drucklegung. Die ständige Aktualität der Adressen kann vonseiten des Verlages nicht gewährleistet werden. Darüber hinaus übernimmt der Verlag keine Verantwortung für die Inhalte dieser Seiten.

service@bv-1.de
www.bildungsverlag1.de

Bildungsverlag EINS GmbH
Ettore-Bugatti-Straße 6-14, 51149 Köln

ISBN 978-3-427-**01671**-7

westermann GRUPPE

Vorwort

Dieses Buch setzt Grundkenntnisse im Umgang mit dem PC voraus. Es ermöglicht dem Leser im Selbststudium das Erlernen der Programmiersprache C#.net. C# ist eine von der Firma Microsoft entwickelte Sprache, die seit dem Jahr 2000 auf dem Markt ist. Sie ist noch eine relativ junge Programmiersprache und wird dadurch noch einigen Wandlungen unterworfen sein. In diesem Buch werden die Versionen C# 7 und .NET 4.7 verwendet. Diese sind seit dem Jahr 2017 verfügbar.

Die einzelnen Kapitel des Buches bauen aufeinander auf. Übungen bzw. Aufgabenstellungen führen oft über mehrere Kapitel hinweg und betrachten Sachverhalte aus verschiedenen Blickwinkeln. Im Kapitel „Arbeiten mit Datenbanken“ werden minimale Kenntnisse der grundlegenden SQL-Befehle vorausgesetzt. Auch sollten für einige Übungsaufgaben Grundkenntnisse zum Erstellen einer HTML-Seite vorhanden sein.

Die Ansprache erfolgt aus Gründen der besseren Lesbarkeit meist in der männlichen Form. Selbstverständlich sind Männer und Frauen immer gleichermaßen gemeint.

Inhaltsverzeichnis

1 Einführung

In diesem Abschnitt wird ein kurzer Überblick über die Programmiersprache C#, das .NET-Framework und die im Buch verwendeten Versionen gegeben. Das erste Programm wird das klassische „Hello World“ sein, mit dem wohl fast jeder eine neue Programmiersprache erlernt. Es werden dabei zwei Varianten vorgestellt, wie man dieses Programm erstellen und ausführen kann: Einmal die klassische Variante mit einem beliebigen Editor und der Kompilierung per Konsolenanweisung und einmal die Variante mithilfe der frei verfügbaren Entwicklungsumgebung Visual Studio von Microsoft. Da die zweite Variante Grundlage für den Rest des Buches ist, wird Visual Studio etwas genauer vorgestellt.

1.1 Die Programmiersprache C# und das .NET-Framework

C# ist eine einfache, sichere, moderne und leistungsfähige Sprache, die rein objektorientiert ist. Sie gehört zu den zehn am meisten verwendeten Programmiersprachen der Welt. Unter anderem wurde sie von Andres Hejlsberg entwickelt, der früher für die Entwicklung von Borland Delphi zuständig war. In die Entwicklung von C# sind Erfahrungen aus anderen Programmiersprachen eingeflossen. So hatten z. B. die Sprachen C++ und Java sehr starken Einfluss, was sich u. a. auch in einer ähnlichen Syntax widerspiegelt. Im August 2000 wurde C# bei ECMA International zur Normung eingereicht und im Jahr 2003 von der ISO genormt. Über die Jahre hinweg wurde die Sprache kontinuierlich weiterentwickelt und immer wieder kamen neue Versionen auf den Markt. 2017 wurde dann C# 7 veröffentlicht. Diese Version wird auch in diesem Buch verwendet.

Die Grundlage für das Programmieren mit C# stellt das .NET-Framework dar, welches ebenfalls von Microsoft entwickelt wurde. Dabei handelt es sich nicht nur um eine besondere Laufzeitumgebung, sondern es wird auch gleichzeitig eine umfangreiche Klassenbibliothek für die Programmierung unter Windows zur Verfügung gestellt. Microsoft hat sich bei der Entwicklung von .NET von vielen schon vorhandenen Technologien leiten lassen. Das Ergebnis war eine Technologie, welche der von Java sehr ähnlich ist. Auch hier wird aus dem Quellcode ein Zwischencode erzeugt und erst zur Laufzeit in nativen Code übersetzt. Dieser Zwischencode wird Common Intermediate Language (CIL) genannt. Zum Ausführungszeitpunkt wird dieser durch einen Just-in-Time-Compiler übersetzt und ausgeführt.

Microsoft bot seine Laufzeitumgebung lange Zeit nur für Windows-Betriebssysteme an. Um .NET auch für andere Betriebssysteme verfügbar zu machen, wurde mit dem Mono-Projekt eine quelloffene Variante der Laufzeitumgebung geschaffen, welche heutzutage auch von Microsoft unterstützt und gefördert wird. Mit Mono ist es möglich, Programme, die für die Microsoft-.NET-Umgebung geschrieben wurden, auch unter Linux oder MacOS auszuführen.

In diesem Buch wird Microsofts .NET-Version 4.7 unter Windows 10 verwendet. Neuere Versionen können aber ohne Probleme installiert werden. Diese werden dann parallel zu

4 Anweisungsfolgen und Methoden

In diesem Kapitel werden wir kennenlernen, wie man einfache Algorithmen mithilfe eines Struktogramms plant und diese dann in ein C#-Programm umsetzt. Dabei beschränken wir uns erst einmal auf eine lineare Abfolge von Anweisungen, die sogenannten Anweisungsfolgen. Trotzdem können auch solche Programme komplex und unübersichtlich werden, wenn es sich um eine große Anzahl von Anweisungen handelt. Wenn die Komplexität zunimmt, ist es notwendig, das Programm zu strukturieren, um es wartbar zu halten. Hier werden wir die Methoden als ein Mittel dazu kennenlernen. Zunächst einmal beschäftigen wir uns mit der Anweisungsfolge, auch Sequenz genannt.

4.1 Anweisungsfolge (Sequenz)

Anweisungen beschreiben einzelne Arbeitsschritte, die zur Lösung einer Problemstellung erforderlich sind. Anweisungen werden nacheinander (sequenziell) von oben nach unten und genau einmal abgearbeitet. Einzelne Anweisungen in Programmen können z. B. Zuweisungen, Berechnungen oder Ein- und Ausgaben sein und werden immer mit einem Semikolon abgeschlossen. Sollen für die Lösung einer Aufgabe mehrere Anweisungen hintereinander ausgeführt werden, spricht man von einer Anweisungsfolge oder auch Sequenz.

In einem Struktogramm würde eine Anweisung oder Anweisungsfolge wie folgt dargestellt:

Auftrag

Es soll ein Programm entwickelt werden, in das der Benutzer die getankte Benzinmenge in Litern und den Preis pro Liter in Euro eingeben kann. Danach sollen die Kosten für das Tanken berechnet und ausgegeben werden. Die Ausgabe im Konsolenfenster soll wie folgt aussehen:

Titel des Konsolenfensters: Berechnung der Tankkosten

Ausgabe im Fenster:

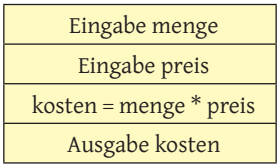
Geben Sie bitte die Tankmenge in Liter ein: 44,3

Geben Sie bitte den Preis pro Liter in Euro ein: 1,30

Die Tankkosten betragen 57,59 Euro.

Zum Beenden drücken Sie eine beliebige Taste ...

Als erstes wird ein Struktogramm entwickelt, um sich die grundlegende Struktur des Algorithmus klar zu machen. Für die Aufgabe sieht das Struktogramm wie folgt aus:



Hierbei wird meistens nur der grundlegende Algorithmus abgebildet. Auf Ausgaben, welche einer benutzerfreundlichen Gestaltung des Programms dienen, wird dabei verzichtet. Durch diese Verkürzungen lenken die benutzerfreundlichen Teile im Struktogramm nicht vom eigentlichen Algorithmus ab. Sie werden erst beim Schreiben des Programms hinzugefügt.

Als zweiter Schritt wird nun der Algorithmus in C# umgesetzt. Eine Ein-zu-eins-Umsetzung des Algorithmus würde so aussehen:

Quellcode: Tankkosten (nur der reine Algorithmus)

```
using System;
namespace Tankkosten {
   class Program {
      static void Main(string[] args) {
         // Eingabe menge
         double menge = Convert.ToDouble(Console.ReadLine());
         // Eingabe preis
         double preis = Convert.ToDouble(Console.ReadLine());
         double kosten = menge * preis;     // kosten = menge * preis
         Console.WriteLine(kosten);         // Ausgabe kosten
      }
   }
}
```

In den ersten beiden Zeilen erhält der Benutzer die Möglichkeit, die Tankmenge und den Preis pro Liter einzugeben. Da alle Eingaben in die Konsole erst einmal als Text interpretiert werden, auch wenn es Zahlen sind, müssen diese noch in einen Gleitkommawert umgewandelt werden. Dies wird durch den Befehl `Convert.ToDouble(Console.ReadLine());` realisiert. Wir gehen erst einmal davon aus, dass der Benutzer nur korrekte Eingaben macht. Alle anderen Eingaben, z. B. die Eingabe von Buchstaben oder Wörtern, würden

zu einer Exception (Exceptionen werden in Kapitel 9 behandelt) führen, da diese nicht in double-Werte umgewandelt werden können. Danach werden die Kosten berechnet und ausgegeben.

Obwohl das Programm syntaktisch korrekt ist und auch die Tankkosten richtig berechnet und ausgibt, ist es in der Praxis kaum zu benutzen. Es ist schlicht und einfach nicht benutzerfreundlich genug. Die Benutzerfreundlichkeit des Programms muss nun noch gestaltet werden. Dieses kann z. B. durch die Ausgabe von entsprechenden Textzeilen geschehen. Dabei sind der Phantasie und dem Geschmack des Entwicklers keine Grenzen gesetzt. Da es in diesem Buch aber vor allem um das Erlernen von C# geht und weniger um Fragen des Oberflächendesigns von Programmen, werden wir als Gestaltungselemente nur jene einfügen, welche zur Bedienung des Programms unbedingt notwendig sind. Durch die spätere Entwicklung von Programmen mithilfe von Windows-Forms oder WPF erhalten diese automatisch eine benutzerfreundlichere Oberfläche.

Hier nun ein Beispiel, wie das obere Programm entsprechend erweitert werden kann:

Quellcode: Tankkosten (benutzerfreundlicher)

```
using System;
namespace Tankkosten {
  class Program {
     static void Main(string[] args) {
       Console.Title = "Berechnung der Tankkosten";
       Console.Write("Geben Sie bitte die Tankmenge in Liter ein: ");
       double menge = Convert.ToDouble(Console.ReadLine());
       Console.Write("Geben Sie bitte den Preis pro Liter in Euro
                      ein: ");
       double preis = Convert.ToDouble(Console.ReadLine());
       double kosten = menge * preis;
       Console.WriteLine("Die Tankkosten betragen {0:F2} Euro. \n",
                         kosten);
       Console.WriteLine("Zum Beenden drücken Sie eine beliebige
                         Taste …");
       Console.ReadKey();
     }
  }
}
```

Die neu hinzugekommenen Zeilen werden nun kurz erläutert: In Zeile 5 wird durch `Console.Title` festgelegt, was in der Titelzeile des Ausgabefensters stehen soll. In den Zeilen 6 und 8 wird dem Benutzer gesagt, was er eingeben soll. Die Ausgabe (Zeile 11) wird durch einen kurzen Text erweitert und formatiert. Durch „\n“ wird eine Leerzeile erzeugt. In Zeile 12 wird der Benutzer darauf hingewiesen, was er zum Beenden des Programms tun muss. Der Befehl `Console.ReadKey();` wartet auf eine Tasteneingabe vom Benutzer. So lange bleibt das Ausgabefenster offen.

Das Starten des Programms ergibt folgende Ausgabe:

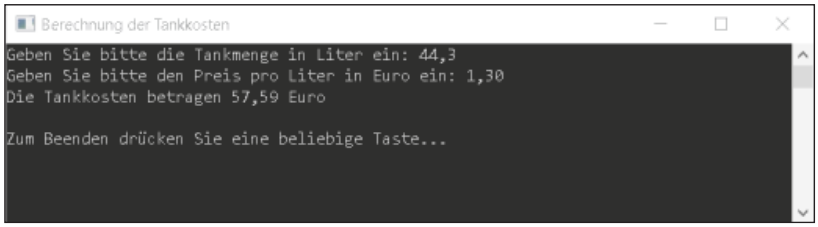


Abb. 4.1: Ausgabe des Programms „Berechnung der Tankkosten“

Obwohl das Programm jetzt theoretisch fertig ist, gibt es noch einiges zu tun, um den Quellcode lesbarer zu gestalten. Damit erhöht sich auch die Wartbarkeit des Programms. Deswegen werden wir uns im nächsten Abschnitt mit Methoden beschäftigen, mit deren Hilfe wir die jetzige Lösung weiter strukturieren können.

4.2 Methoden

In C# werden alle Anweisungen innerhalb einer Methode ausgeführt. Unter einer Methode kann man einen Codeblock verstehen, welcher eine Reihe von Anweisungen enthält. Ein Programm führt diese Anweisungen aus, indem die entsprechende Methode im Programm aufgerufen wird. Die Main-Methode ist die erste Methode, die wir schon kennengelernt haben. Sie ist der Einstiegspunkt in jede C#-Anwendung und wird als erstes aufgerufen, wenn das Programm gestartet wird. Da die Main-Methode eine zwingende Notwendigkeit darstellt, könnte nun der Rest des Programms ebenfalls in diese hineingeschrieben werden. Diese Vorgehensweise würde den Quelltext aber sehr schnell unübersichtlich machen. Daher werden einzelne Programmteile in Methoden zusammengefasst und diese dann an den entsprechenden Stellen aufgerufen. Bei der Verwendung von Methoden ergeben sich folgende Vorteile:

- **Übersichtlichkeit:** Komplexe Programme werden in kleine Teilprogramme zerlegt, damit die Komplexität heruntergebrochen wird. Damit ist der Kontrollfluss leichter zu erkennen. In klassischen Programmen heißen die Funktionen daher auch Unterprogramme.
- **Vermeidung von Redundanzen im Quellcode:** Wiederkehrende Programmteile sollen nicht immer wieder neu programmiert, sondern an einer Stelle angeboten werden. Damit werden Redundanzen im Quelltext vermieden. Änderungen an der Funktionalität oder die Suche nach Fehlern lassen sich leichter durchführen.
- **Hohe Wiederverwendbarkeit von Quellcode**: Programmteile, die einmal geschrieben wurden, können auch in andere Programme eingebunden werden. So spart man Entwicklungsaufwand.

Methoden bestehen generell aus einer Methodensignatur und einem Methodenrumpf. Die Methodensignatur setzt sich aus dem Namen der Methode (Methodenname), der Anzahl und Reihenfolge der Übergabeparameter (Parameterliste) und dem Typ des Rückgabewerts (Rückgabetyp) zusammen.

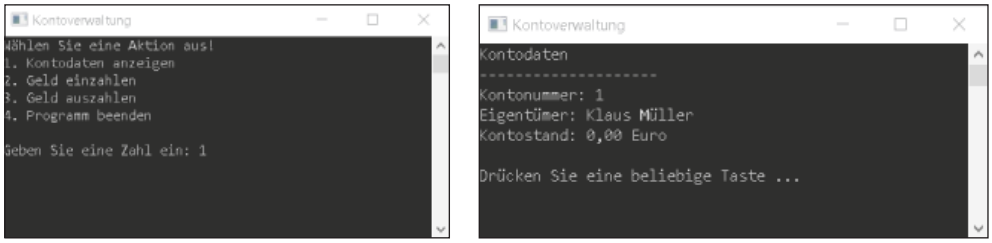
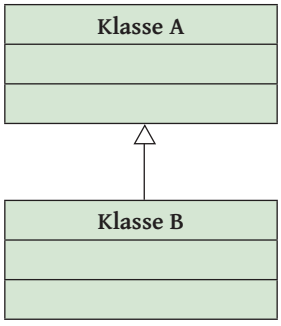


Abb. 7.3: Beispiele für Ausgaben des Programms „Kontoverwaltung“

Das Programm stellt nur exemplarisch die Arbeit mit einem Objekt, sprich mit einem Konto, dar. Um es praxistauglicher zu gestalten, muss es noch erweitert werden. Es sollte möglich sein, mehrere Konten anzulegen, zu bearbeiten und zu löschen. Dies könnte z. B. mithilfe einer ArrayList (siehe Kapitel 6.4.1) geschehen. Auch das Menü müsste dementsprechend angepasst werden. Das würde aber den hier gewählten Rahmen sprengen und kann vom Leser bei Bedarf selbst programmiert werden.

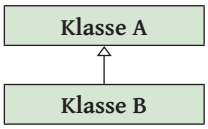
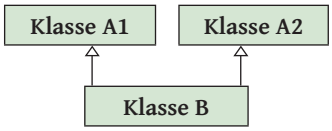
7.4 Vererbung

Neben der Kapselung ist die Vererbung ein weiteres Basisprinzip der OOP. Über die Vererbung kann man eine neue Klasse auf Basis einer bereits vorhandenen Klasse erzeugen, die als **Elternklasse** bezeichnet wird. Die abgeleitete Klasse übernimmt dabei sämtliche Eigenschaften und Methoden der Elternklasse und es werden neue Eigenschaften und Methoden hinzufügt. Die abgeleitete Klasse erweitert damit die Elternklasse. Im folgenden Beispiel wird die Klasse B von der Klasse A abgeleitet. Der Pfeil mit ungefüllter Spitze zwischen den Klassen visualisiert die Vererbung und zeigt immer auf die Ober- oder auch Basisklasse. In UML wird dies wie folgt dargestellt:



Man unterscheidet zwei Arten von Vererbungen:

- Einfachvererbung
- Mehrfachvererbung

Art	Erläuterung	Darstellung im UML-Klassendiagramm
Einfachvererbung	Bei der Einfachvererbung hat eine abgeleitete Klasse genau eine Basisklasse, von der sie erbt.	
Mehrfachvererbung	Bei der Mehrfachvererbung hat eine abgeleitete Klasse mindestens zwei Basisklassen, von der sie erbt.	

C# erlaubt aus verschiedenen Gründen auf direktem Weg nur die Einfachvererbung. Eine Mehrfachvererbung kann nur indirekt mithilfe von **Interfaces** realisiert werden. Vererbung wird in C# mit einem Doppelpunkt dargestellt.

Syntax:
Vererbung

```
class Klassenname : Basisklasse {
…
}
```

Beispiele

```
class Katze : Säugetier {
        ...
}
class Eigenheim : Haus {
        ...
}
```

Die Vererbung kann durch Zugriffsrechte für Attribute und Methoden eingeschränkt werden. Eine abgeleitet Klasse erbt dann alle Attribute und Methoden von einer Elternklasse, die nicht private sind. Attribute oder Methoden, welche vererbt, aber nach außen nicht sichtbar sein sollen, werden mit dem Schlüsselwort protected gekennzeichnet.

Beispiele

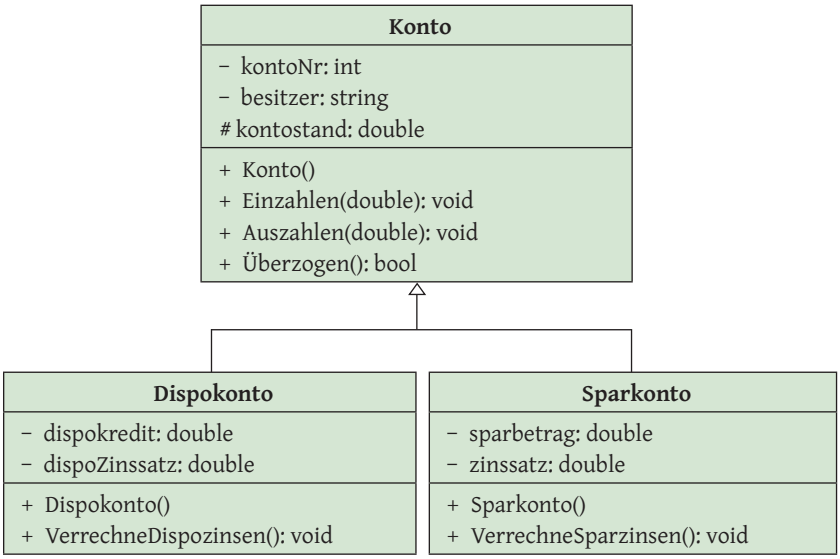
```
class A {
   private int x;
   protected int y;
   public int z;
}
class B : A {
   ...
}
```

In diesem Beispiel besitzt die Klasse A drei Attribute. Das Attribut x ist private, y ist protected und z ist public. Die abgeleitete Klasse B erbt alles, was nicht private ist, und kann darauf direkt zugreifen, also in diesem Fall die Attribute y und z.

Auftrag

Von der Klasse Konto aus dem letzten Auftrag sollen ein Sparkonto und ein Konto mit einem Dispokredit (Dispokonto) abgeleitet werden. Das Sparkonto wird ab einem festgelegten Betrag höher verzinst. Das Dispokonto stellt einen Dispokredit zur Verfügung, der einen besseren Zinssatz hat. Es soll jeweils eine Methode zur Verfügung gestellt werden, die die Zinsen bzw. Dispozinsen ermittelt und mit dem aktuellen Kontostand verrechnet.

Zunächst erweitern wir unser UML-Klassendiagramm um die beiden neuen Klassen, die mittels Einfachvererbung aus der Klasse Konto abgeleitet werden. Da wir auf den Kontostand für die Berechnungen direkt zugreifen wollen, wird das Zugriffsrecht auf protected geändert.



Nun müssen noch die beiden Klassen implementiert und die Klasse Konto angepasst werden. In der Klasse Konto ändern wir nur das Zugriffsrecht des Attributs kontostand von private auf protected.

```
namespace Kontoverwaltung {
    class Konto {
        private int kontoNr;
        private string besitzer;
        protected double kontostand;
…
  }
}
```

Kann nun vererbt werden.

Auftrag

Erstellen Sie ein WPF-Projekt und gestalten Sie die Oberfläche wie in Bild 11.5 dargestellt. Beim Klick auf den Button soll die Anwendung geschlossen werden. Dazu ist das entsprechende Ereignis zu implementieren.

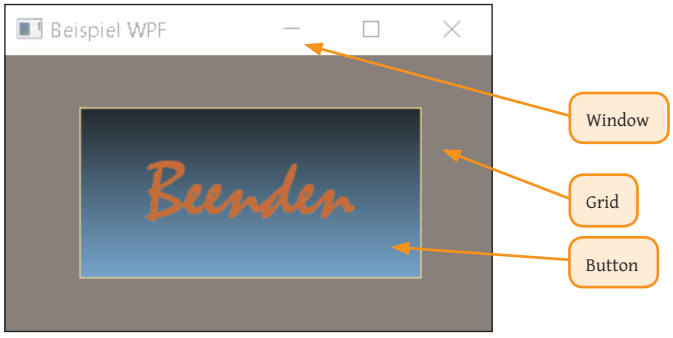


Abb. 11.5: WPF-Projekt mit Button „Beenden“

Zunächst wird ein WPF-Projekt angelegt, erzeugt und gespeichert. Danach erscheint wieder das Hauptfenster, auf dem die Steuerelemete platziert werden können. Zuvor werden aber einige Eigenschaften im Eigenschaftsfenster festgelegt.

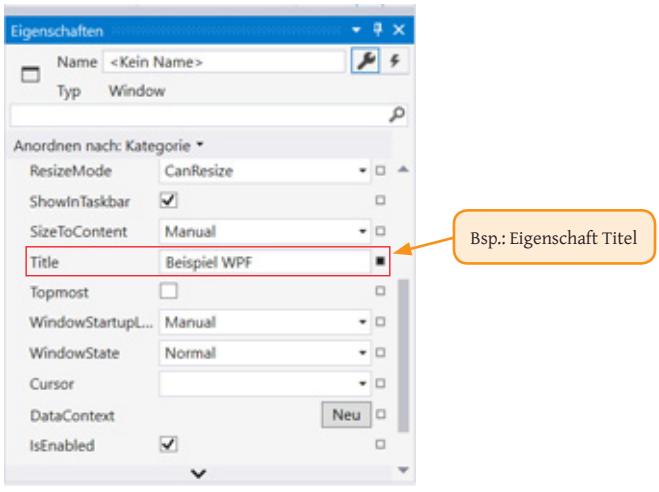


Abb. 11.6: Eigenschaften im Eigenschaftsfenster festlegen

Es werden nun noch die Eigenschaften „Name“ in „frmMain“, „Width“ und „Height“ laut der unten stehenden Tabelle geändert. Alle Eigenschaften, die nicht aufgeführt sind, behalten ihre Standardwerte und müssen nicht geändert werden.

Steuerelement	Eigenschaft
Window	Name: frmMain Title: Beispiel WPF Height: 200 Width: 300

Als Nächstes wird ein Grid aus der Toolbox „Oberfläche“ gezogen und die Farbe des Hintergrunds im Eigenschaftsfenster auf den Wert „#FF6B5E5B“ gesetzt. Dazu wird zunächst der Menüpunkt „Pinsel mit Volltonfarbe“ ausgewählt und dann der Farbwert gesetzt.

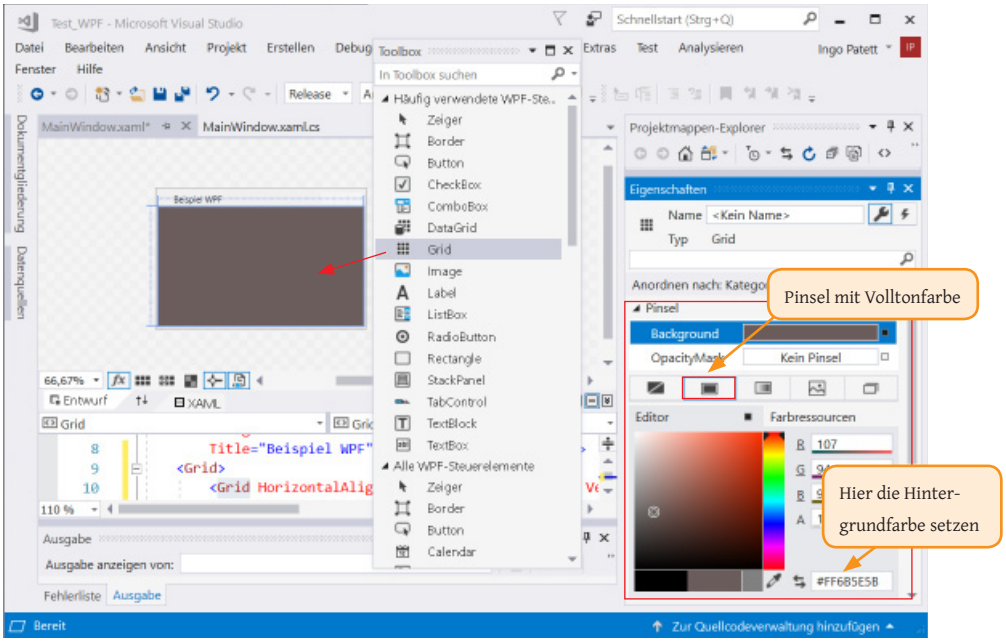


Abb. 11.7: Pinsel und Farbe auswählen

Generell gehört ein Grid zu den sogenannten Panel-Elementen in WPF. Panel-Elemente sind Komponten, die andere Elemente aufnehmen und deren Rendern steuern. Sie bestimmen z. B. deren Größe und Position auf dem Window. WPF bietet eine Reihe von vordefinierten Panel-Elementen. Die folgende Tabelle enthält eine Übersicht (Auszug):

Element	Beschreibung
Canvas	Definiert einen Bereich, in dem weitere Elemente mithilfe von Koordinaten positioniert werden können. Wird u. a. dazu verwendet, um 2D- oder 3D-Grafiken darzustellen.
DockPanel	Definiert einen Bereich, in dem weitere Elemente entweder horizontal oder vertikal zueinander angeordnet werden können.
Grid	Definiert einen flexiblen Rasterbereich, der aus Spalten und Zeilen besteht. Weitere Elemente können hierauf durch die Margin-Eigenschaft genau positioniert werden.

Bildquellenverzeichnis

fotolia.com, New York: Cake78 (3D & photo) Titel
iStockphoto.com, Calgary: Lyudinka 94; Soloviivka 94
stock.adobe.com, Dublin: yopinco 94

Wir arbeiten sehr sorgfältig daran, für alle verwendeten Abbildungen die Rechteinhaberinnen und Rechteinhaber zu ermitteln. Sollte uns dies im Einzelfall nicht vollständig gelungen sein, werden berechtigte Ansprüche selbstverständlich im Rahmen der üblichen Vereinbarungen abgegolten.