3 Programmstrukturierung

lernen

C# lernen
ISBN 3-8273-2045-3

In diesem Kapitel werden wir uns mit dem grundsätzlichen Aufbau eines C#-Programms beschäftigen. Sie werden einiges über Klassen und Objekte erfahren, die die Basis eines jeden C#-Programms darstellen, weitere Informationen über Namespaces erhalten und über die Deklaration sowohl von Variablen als auch von Methoden einer Klasse.

Klassen

C# bietet einige Möglichkeiten der Strukturierung eines Programms, allen voran natürlich das Verpacken der Funktionalität in Klassen. So können Sie mehrere verschiedene Klassen erstellen, die jede für sich in ihrem Bereich eine Basisfunktionalität bereitstellt. Zusammengenommen entsteht aus diesen einzelnen Klassen ein Gefüge, das fertige Programm mit erweiterter Funktionalität, indem die einzelnen Klassen miteinander interagieren und jede genau die Funktionalität bereitstellt, für die sie programmiert wurde.

Im Grundsatz sind die hier vorgestellten Konzepte die der objektorientierten Programmierung. Ich habe mich dafür entschlossen, diese nicht in einem eigenen Kapitel zu behandeln, sondern Sie Schritt für Schritt in die Programmierung von C# einzuführen, wobei das Verständnis objektorientierter Konzepte sozusagen ein »Nebenprodukt« ist. Man lernt eben besser, wenn man etwas praktisch anwenden kann. Zusätzliche Informationen zu den einzelnen Konzepten finden Sie immer wieder in diesem Buch.

3.1 Klassen und Objekte

Klassen und Objekte

Bei den Konzepten der objektorientierten Programmierung hat man sich vieles von der Natur abgeschaut. So auch das Konzept der Klassen und Objekte. Wenn von einem Objekt gesprochen wird, handelt es sich immer um etwas, das etwas tun kann. Eine Klasse hingegen stellt eine Art Bauplan für ein Objekt dar, einen Prototypen, der erweitert und verändert werden kann.

Instanzen

Bevor man mit einer Klasse arbeiten kann, muss erst eine Instanz dieser Klasse erzeugt werden. Wir hauchen ihr also in gewissem Sinne Leben ein. Ein Beispiel aus dem Leben macht dies deutlicher. Nehmen wir an, es gäbe eine Klasse mit dem Namen Fahrzeug. Damit kann natürlich keiner etwas anfangen, denn ein Fahrzeug kann alles sein. Es ist ein abstrakter Begriff.

Real wird das Ganze erst, wenn ich z.B. sage, dass ich mir ein Fahrzeug namens Fahrrad kaufe, mit einer bestimmten Reifengröße, einer Farbe und meinetwegen noch einer speziellen Schaltung. In diesem Moment habe ich genau spezifiziert, welche Art von Fahrzeug ich kaufen werde, welche Farbe es hat, welche Reifengröße usw.

Daten und Methoden

Die Farbe, die Reifengröße und die Art der Gangschaltung sind Daten, die zu der Klasse gehören und, wenn ich mir das Fahrrad denn nun wirklich kaufe, festgelegt werden. Die Vorgänge, die das Fahrrad ausführen kann, sind die Methoden. Es kann fahren, bremsen oder umfallen.

Eigenschaften

Zusätzlich zu den Daten hat das Fahrrad aber auch noch Eigenschaften. Im Prinzip handelt es sich dabei auch um Daten, allerdings mit dem Unterschied, dass diese nicht von vornherein festgelegt sein müssen. Eine Eigenschaft eines Fahrrads wäre z.B. die Beschleunigung. Sie kann errechnet werden, nämlich aus der Größe der Räder und der Kraft, mit der ich in die Pedale trete. Damit ist sie zwar nicht unbedingt festgelegt, aber sie ist ein Bestandteil der Klasse.

Ereignisse

Zuletzt beinhaltet eine Klasse auch noch Ereignisse. Wenn ich auf einem Fahrrad sitze und die Klingel betätige, gehen mir normalerweise die Leute aus dem Weg und machen Platz, so dass ich vorbeifahren kann. Das Ereignis »Klingeln« wird also ausgelöst und bewirkt etwas. Sehen wir uns jetzt einmal an, wie eine solche Klasse aussieht.

3.1.1 Deklaration von Klassen

Wir haben gesagt, Klassen bestehen aus Feldern (die die Daten beinhalten), Eigenschaften, Methoden (für die Funktionalität) und Ereignissen. Den Aufbau einer Klasse sehen Sie in Abbildung 3.1.

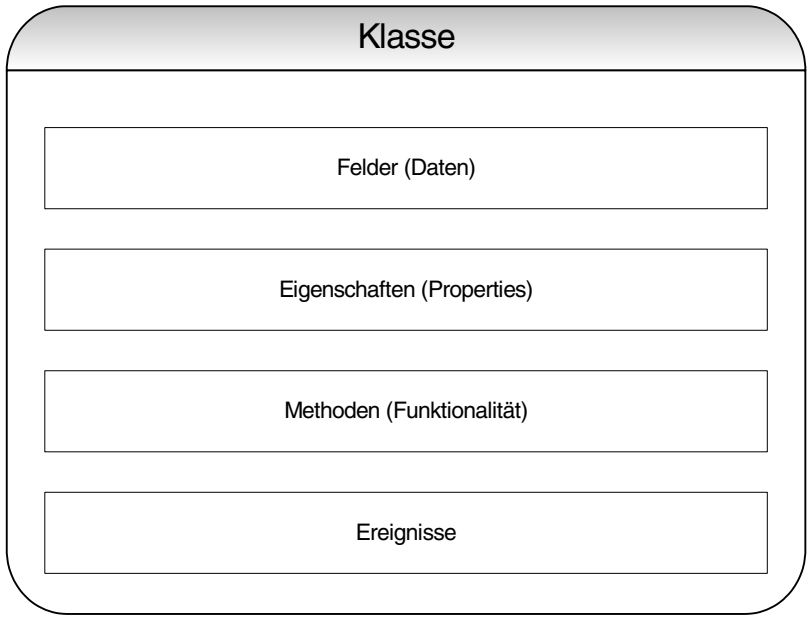

Abbildung 3.1: Der Aufbau einer Klasse

Member oder Attribute

Attribute *Member* *Mitglied*
ein wenig eindeutiger, weil es in der Tat auch noch Bestandteile der Sprache C# gibt, die wirklich Attribute genannt werden. Aus diesem Grund werde ich im weiteren Verlauf den Begriff *Member* verwenden.

Die Deklaration einer Klasse wird mit dem reservierten Wort `class`

ungefähr so aussehen:

```
class Fahrzeug
{

  int    anzahl_Raeder;
         beschleunigung;
  string farbe;

  public void Beschleunigen()
  {
  }
```

```
                Bremsen()
    {
    }

  }
```

```
                                                                anzahl_raeder
beschleunigung      farbe
       Beschleunigen()      Bremsen()
```

aus Feldern und Methoden – natürlich könnten Sie noch weitere Felder hinzufügen und auch noch weitere Methoden. In dieser Form ist die Klasse aber noch nicht benutzbar, wir benötigen zuerst noch eine so genannte Instanz davon.

3.1.2 Erzeugen von Instanzen

existiert, mehrere Objekte daraus zu erzeugen und diesen unterschiedliche Daten zuzuweisen.

new

```
new
```

```
Fahrzeug Fahrrad  = new
Fahrzeug Motorrad =     Fahrzeug();
Fahrzeug Auto     =     Fahrzeug();
```

Wenn der Operator angewendet wird, wird eigentlich eine bestimmte Methode der Klasse aufgerufen, der so genannte Konstruktor. In Kapitel 3.3.10 werden wir noch genauer auf Konstruktoren eingehen.

Abbildung 3.2 zeigt schematisch, wie die Instanzierung funktioniert.

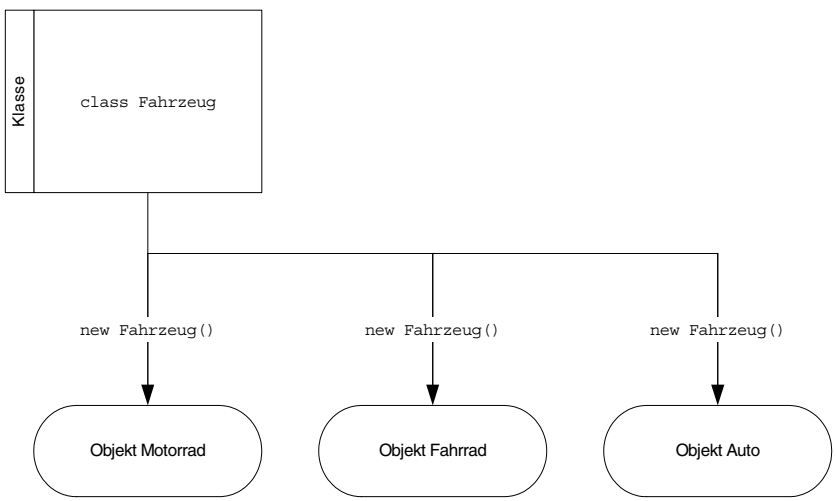

Abbildung 3.2: Erzeugen von Objekten einer Klasse

Die Instanz einer Klasse ist ein Objekt. Ebenso ist ein Objekt eine Instanz einer Klasse. In diesem Buch werden beide Wörter benutzt, auch aus dem Grund, weil es eigentlich keine weiteren Synonyme für diese Begriffe gibt. Es wäre ziemlich schlecht lesbar, wenn in einem Satz dreimal das Wort Instanz auftauchen würde.

3.2 Felder einer Klasse

3.2.1 Deklaration von Feldern

Syntax `[Modifizierer] Datentyp Bezeichner [= Initialwert`

werden. Die eigentliche Zuweisung geschieht aber mit Hilfe der erzeugten Objekte (außer bei statischen Feldern, auf die wir später noch zu sprechen kommen). Der Initialwert ist deshalb wichtig, weil eine Variable in C# vor ihrer ersten Verwendung immer initialisiert werden muss.

Felder bestehen wie Variablen aus einem Bezeichner, durch den sie innerhalb des Programms angesprochen werden können, und aus einem Datentyp, der angibt, welche Art von Information in dem entsprechenden Feld gespeichert werden kann. Einer der einfachsten Datentypen ist der Integer-Datentyp, ein ganzzahliger Typ mit Vorzeichen, den wir bereits kurz angesprochen haben. Die Deklaration eines Felds mit dem Datentyp Integer geschieht in C# über das reservierte Wort :

```
myInteger;
```

Groß-/ Kleinschreibung

Int32 `System`

`Int32`

`string` `System.String`

```
/* Beispiel Initialisierung von Feldern         */
/* Erzeugt einen Fehler bei Zuweisung theNumber */

      TestClass
{
      myNumber;    nicht initialisiertes Feld
      theNumber; //nicht initialisiertes Feld

         static void

    myNumber = 15;                    = Initialisierung
    myNumber = theNumber; // FEHLER: theNumber nicht
                          // initialisiert!!
```

Listing 3.1: Fehler bei der Zuweisung eines nicht initialisierten Felds

3.2.2 Bezeichner und Schreibweisen

sinnvolle Bezeichner

```
x     y

     myString theName theResult
```


, _ . Diese Namen sagen absolut nichts aus, außerdem machen sie eine spätere Wartung schwierig.

Stellen Sie sich vor, Sie sollten ein Programm, das Sie nicht selbst geschrieben haben, mit einigen Funktionen erweitern. Normalerweise kein Problem. Es wird aber ein Problem, wenn der vorherige Programmierer nicht mit eindeutigen Bezeichnern gearbeitet hat. Gleiches gilt für Ihre eigenen Programme, wenn Sie nach längerer Zeit nochmals Änderungen vornehmen müssen. Auch dies kann zu einer schweißtreibenden Arbeit werden, wenn Sie Bezeichner verwendet haben, die keine klare Aussage über ihren Verwendungszweck machen. Glauben Sie mir, wenn ich Ihnen sage, dass es ohnehin schon schwierig genug ist ein größeres Projekt nach langer Zeit zu warten.

Eindeutige Bezeichner sind eine Sache, die nächste Regel ergibt sich aus der Tatsache, dass C# Groß- und Kleinschreibung unterscheidet. Um sicher zu sein, dass Sie Ihre Bezeichner auch über das gesamte Programm hinweg immer gleich schreiben, suchen Sie sich eine Schreibweise aus und bleiben Sie dabei, was immer auch geschieht. Mit C# ist Microsoft von der lange benutzten ungarischen Notation für Variablen und Methoden abgekommen, die ohnehin am Schluss jeden Programmierer unter Windows eher verwirrt hat statt ihm zu helfen (was eigentlich der Sinn einer einheitlichen Notation ist). Grundsätzlich haben sich nur zwei Schreibweisen durchgesetzt, nämlich das so genannte *PascalCasing*
camelCasing

PascalCasing

camelCasing

Schreibweisen des Autors

Punkt für die Qualifizierung mit einem Großbuchstaben begonnen wird.

In diesem Abschnitt wurde bereits von Eigenschaften gesprochen. Eigenschaften sind eine andere Art, auf die Daten eines Objekts zuzugreifen. Während die Daten in den Feldern des Objekts gespeichert sind, sie also direkt auf die Daten zugreifen würden, haben Sie bei Eigenschaften die Möglichkeit, die zurückgelieferten Daten noch zu modifizieren. Wie das genau funktioniert, wird noch in Kapitel 9.1 erklärt. Für den Moment soll genügen, dass derjenige, der von außen auf ein Objekt zugreift, Eigenschaften wie Felder wahrnimmt. In der Verwendung gibt es keinen Unterschied.

Regeln

```
Name5S7
```

```
1stStart
Mein Name
&again
```

Maximallänge

In C++ gibt es die Beschränkung, dass Bezeichner nur anhand der ersten 31 Zeichen unterschieden werden. Diese Beschränkung gilt nicht für C#. Sie könnten also, wenn Sie wollen, durchaus auch wesentlich längere Bezeichner für Ihre Variablen benutzen. Aber ich rate Ihnen davon ab, denn Sie werden sehr schnell feststellen, dass es nichts Schlimmeres gibt, als endlos lange Bezeichner.

reservierte Wörter als Bezeichner

```
          @   (       @       )
{
    Anweisungen
```

nämlich die mangelnde Übersichtlichkeit des Quelltextes. Dass etwas möglich ist, bedeutet noch nicht, dass man es auch unbedingt anwenden sollte. Ich selbst bin bisher in jeder Programmiersprache ganz gut ohne die Verwendung reservierter Wörter als Bezeichner ausgekommen, und ich denke, das wird auch weiterhin so bleiben. Wenn Sie dieses Feature nutzen möchten, steht es Ihnen selbstverständlich zur Verfügung, ich selbst bin der Meinung, dass es unnötig ist.

3.2.3 Modifizierer

Modifizierer	Bedeutung
`public`	Auf die Variable oder Methode kann auch von außerhalb der Klasse zugegriffen werden.
`private`	bzw. des Datentyps zugegriffen werden. Innerhalb von Klassen ist dies Standard.
`internal`	Der Zugriff auf die Variable bzw. Methode ist beschränkt auf das aktuelle Projekt.
`protected`	Klasse bzw. durch Klassen, die von der aktuellen Klasse abgeleitet sind, möglich.
`abstract`	Dieser Modifizierer bezeichnet Klassen, von denen keine Instanz erzeugt werden kann. Von abstrakten Klassen muss immer zunächst eine Klasse abgeleitet werden.
`const`	Der Modifizierer für Konstanten. Der Wert von Feldern, die mit diesem Modifizierer deklariert wurden, ist nicht mehr veränderlich.
	Deklariert ein Ereignis (engl.).