PEARSON Education

ADDISON-WESLEY

MASTER CLASS

Dirk Frischalowski

Visual C# 2005

Einstieg für Anspruchsvolle

Auf CD: Visual C# 2005 Express Edition

ADDISON-WESLEY [in Kooperation mit] PEARSON Studium

6

Namespaces

6.1 Namespaces deklarieren

6.1.1 Einführung

Wenn Sie umfangreichere Anwendungen mit .NET erstellen, fällt einer gut gewählten Namensgebung sowie der geschickten Verwaltung von Klassen und anderen Typen eine größere Rolle zu. In einfachsten Anwendungen verwenden Sie vielleicht nur eine einzige oder wenige Klassen, deren Deklaration sich sofort an das Einbinden eines Namespace anschließt, ohne dass selbst ein eigener Namespace deklariert wird.

Beispiel
Klassendeklaration ohne Namespace
Listing 6.1
OhneNamespace.cs (Projekt MitUndOhneNamespaceProj)

```
using System;
class TestAusgabe
{
  public void Ausgabe()
  {
    Console.WriteLine("Ohne Namespace...");
  }
}
```

Wenn Sie eine solche Klasse ansprechen möchten, verwenden Sie direkt den Klassennamen, z.B. `TestAusgabe`. Jetzt kann es beispielsweise passieren, dass ein anderer Entwickler auf die gleiche Idee kommt und seine Klasse genauso benennt und diese ebenfalls in keinen Namespace einbettet. Sie haben aber keine Möglichkeit mehr, beide Klassen gleichzeitig zu verwenden. Des Weiteren steht Ihnen nur der Klassenname als Gedankenstütze zur Verfügung, um das Einsatzgebiet der Klasse zu bestimmen.

Aus den genannten Gründen ist es von Vorteil, Klassen und andere Typen zusätzlich in einem Namespace (Namensraum) unterzubringen. Der Klassenname muss nur noch innerhalb des Namespace eindeutig sein. Außerdem kann der Name des Namespace dazu dienen, die Funktion der Klasse genauer zu spezifizieren. Eine Klasse zum Umrechnen von Währungen kann sich z.B. im Namespace `Waehrungen` (oder `Currencies`) befinden.

Beispiel
Eine Klasse in einen Namespace einbetten

Die Klasse `TestAusgabeNS` wird im Namespace `MitNamespace` untergebracht. Der vollständige Name der Klasse lautet nun `MitNamespace.TestAusgabeNS`.

Listing 6.2
MitNamespace.cs
(Projekt MitUndOhneNamespaceProj)

```
using System;
namespace MitNamespace
{
  class TestAusgabeNS
  {
    public void Ausgabe()
    {
      Console.WriteLine("Mit Namespace...");
    }
  }
}
```

Um die Klasse `TestAusgabe` zu verwenden, ist keine weitere Aktion notwendig. Entweder Sie binden die Source-Datei oder die Assembly in das Projekt ein. Anders sieht es nun beim Zugriff auf die Klasse `TestAusgabeNS` aus. Entweder Sie binden den Namespace `MitNamespace` über die `using`-Anweisung ein (jetzt können Sie direkt den Klassennamen verwenden) oder Sie verwenden die Klasse mit ihrem voll qualifizierten Namen `MitNamespace.TestAusgabeNS`.

Listing 6.3
itUndOhneNamespace.cs
(Projekt MitUndOhneNamespaceProj)

```
using System;
using MitNamespace;
namespace CSharpBuch.Kap06
{
  class MitUndOhneNamespace
  {
    static void Main(string[] args)
    {
      new TestAusgabe().Ausgabe();
      new TestAusgabeNS().Ausgabe();
      new MitNamespace.TestAusgabeNS().Ausgabe();
      Console.ReadLine();
    }
  }
}
```

Namespaces deklarieren also Gültigkeitsbereiche. In einem Namespace können Sie folgende Typen deklarieren:

- Weitere Unter-Namespaces (dadurch entstehen verschachtelte Namespaces wie z.B. `System.Text`)
- Klassen (`class`)
- Schnittstellen (`interface`)
- Strukturen (`struct`)
- Aufzählungen (`enum`)
- Delegates (`delegate`)

Wird kein Namespace verwendet, wird dieser als globaler, unbenannter oder Standard-Namespace bezeichnet. Die darin deklarierten Bezeichner können überall verwendet werden.

Hinweis

Natürlich können Sie statt eines Namespace den Klassennamen aus dem Namen des Namespace und dem Klassennamen zusammensetzen. Der Quellcode würde allerdings auf diese Weise nicht wesentlich besser lesbar sein, wenn statt des Klassennamens `Mathe` der Name `CSharpBuchKap06Mathe` verwendet werden würde.

6.1.2 Verschachtelte Namespaces

Namespaces können auch verschachtelt werden. Dadurch lassen sich Hierarchien von Gültigkeitsbereichen erzeugen. Außerdem können Sie Klassen und andere Typen noch besser nach ihrem Einsatzgebiet strukturieren. Damit sich Ihre Namespace-Deklarationen nicht mit denen anderer Entwickler oder anderer Firmen überschneiden, können Sie in der ersten Ebene z.B. Ihren Firmennamen verwenden.

Tabelle 6.1 Beispiele für verschachtelte Namespace-Deklarationen

Namespace	Einsatzgebiet der darin enthaltenen Typen
`DFSoftware.Utils.Mathe`	Hilfsmethoden für mathematische Operationen
`DFSoftware.Taschenrechner`	Hauptanwendung – ein Taschenrechner
`DFSoftware.Web.Update`	Methoden, um die Anwendung über das Web zu aktualisieren

Beispiel Namespaces zur besseren Strukturierung verschachteln

Innerhalb des Namespace `CSharpBuch` werden sämtliche anderen Typen und Namespaces untergebracht, die irgendetwas mit diesem Buch zu tun haben. Zum einfacheren Auffinden der Beispiele wird pro Kapitel ein eigener Unter-Namespace mit dem Kapitelnamen bereitgestellt. Nun wäre es auch möglich gewesen, nochmals pro Abschnitt einen eigenen Namespace `Unterkapitel` mit einer fortlaufenden Nummerierung zu deklarieren. Über den Projektnamen sind die Anwendungen aber bereits ausreichend eindeutig gekennzeichnet.

Listing 6.4 Verschachtelte Namespaces.cs

```
using System;
namespace CSharpBuch
{
  namespace Kap06
  {
    namespace Unterkapitel
    {
      class Test
      {
        static void Main(string[] args)
```

Listing 6.5
Verschachtelte Namespaces.cs

```
        {
          new CSharpBuch.Kap06.Unterkapitel.Test().Ausgabe();
        }
        public void Ausgabe()
        {
          Console.WriteLine("Verschachtelung von Namespaces");
          Console.ReadLine();
        }
      }
    }
  }
}
```

6.1.3 Syntax von Namespaces

- Ein Namespace wird durch das Schlüsselwort `namespace` eingeleitet. Zugriffsmodifizierer kommen bei Namespaces nicht zum Einsatz. Ihr Inhalt ist immer `public`.
- Danach folgt der Name des Namespace, bei dem es sich um einen gültigen Bezeichner handeln muss.
- Der Inhalt eines Namespace wird in geschweifte Klammern eingeschlossen.
- Verschachtelte Namespaces entstehen entweder durch die Verschachtelung mehrerer Namespace-Deklarationen oder durch die Verwendung eines Punkts im Namen des Namespace. Der Punkt trennt dabei jeweils einen Bestandteil des Namespace.

Statt

```
namespace CSharpBuch
{
  namespace Kap06
  {
    namespace Unterkapitel
    {
```

lässt sich auch verkürzt schreiben:

```
namespace CSharpBuch.Kap06.Unterkapitel
{
```

Innerhalb einer C#-Datei können Sie mehrere Namespace-Deklarationen verwenden.

```
namespace Namespace1
{}
namespace Namespace2
{}
```

Ein Namespace kann sich durchaus über mehrere Dateien und Assemblies erstrecken. So können Sie beispielsweise eine Assembly erstellen, welche die wichtigsten Klassen eines Namespace umfasst, und eine weitere Assembly, die spezielle Hilfsklassen enthält. Der Inhalt des Namespace ist die Summe aller Typdefinitionen in allen diesen Dateien. In diesem Fall

müssen Sie darauf achten, dass jeder Bezeichner nur einmal innerhalb des Namespace vorkommt. Ansonsten erhalten Sie einen Compiler-Fehler, da der Typname nicht mehr eindeutig ist.

Hinweis

Der Name eines Namespace ist völlig unabhängig vom Namen oder dem Speicherort einer Datei. So kann sich der Inhalt eines Namespace aus mehreren Dateien zusammensetzen, die an völlig unterschiedlichen Orten gespeichert sind. Namespaces dienen nur dazu, die darin enthaltenen Typen zu strukturieren.

6.1.4 Namensgebung

Der Name eines Namespace sollte den Einsatzzweck der enthaltenen Typen kennzeichnen. In diesem Fall sollte der Name im Plural geschrieben werden. Enthält Ihr Namespace Klassen zur Matrizenrechnung, könnte er `DFSoftware.Matrizen` heißen.

Namespaces werden wie Typnamen in der Pascal-Schreibweise bezeichnet. Darin beginnt jedes Hauptwort mit einem Großbuchstaben. Verwenden Sie möglichst einen mehrstufigen Namespace für Ihre Typen, der z.B. aus dem Firmen-, Familien- oder Produktnamen und mindestens einer Unterkategorie besteht. Hilfsklassen können Sie z.B. in einem Namespace `Util` unterbringen. Die Klassen einer Anwendung gelangen dann in einen Namespace, der dem Namen der Anwendung entspricht, z.B.

```
DFSoftware.DFWord
DFSoftware.Taschenrechner
```

6.2 Namespaces einbinden

Damit man auf den Inhalt eines Namespace zugreifen kann, müssen Sie die Source-Datei einbinden oder eine Referenz auf die implementierende Assembly hinzufügen. Jetzt können Sie schon einmal auf den Inhalt des Namespace zugreifen.

Damit Sie nicht jedes Mal den voll qualifizierten Namen einer Klasse oder eines anderen Typs verwenden müssen, der sich in einem Namespace befindet, binden Sie den Namespace mittels der `using`-Direktive ein.

```
CSharpBuch.Kap06.Test t = new CSharpBuch.Kap06.Test();
// oder
using CSharpBuch.Kap06;
...
Test t = new Test();
```

Der Direktive `using` folgt der vollständige Name des Namespace. Hierbei ist zu beachten, dass exakt nur der angegebene Namespace eingebunden wird. Die Reihenfolge bei Verwendung mehrerer `using`-Direktiven ist nicht signi-

fikant. Mit anderen Worten: Eine using-Direktive hat keinen Einfluss auf eine andere using-Direktive. Besitzt der Namespace weitere verschachtelte Unter-Namespaces, müssen diese separat eingebunden werden, z.B.

```
using System;
using System.Text;
```

Die using-Direktive kann sich außerhalb oder innerhalb einer Namespace-Deklaration befinden. Befindet sie sich darin, ist sie auch nur im Gültigkeitsbereich des Namespace aktiv. Die using-Direktive muss sich aber in jedem Fall vor allen anderen Typdefinitionen befinden.

Beispiel
Einsatz der using-Direktive

- Globale Gültigkeit der using-Direktive:

```
using System;
namespace CSharpBuch
{
```

- Gültigkeit von using nur innerhalb des Namespace und Unter-Namespace

```
namespace CSharpBuch
{
  using System;
```

- Fehler, using muss vor allen Typdefinitionen stehen

```
namespace CSharpBuch
{
  class Test ...
  ...
  using System; // Fehler
```

Hinweis

Der Zugriff auf einen Typ kann nur über den voll qualifizierten Namen oder den reinen Typnamen (unqualifizierter Name) erfolgen. Es erfolgt keine Zusammensetzung von unvollständigen Bezeichnern

```
using System;
...
// Der Compiler erzeugt einen Fehler, da es keinen Namespace
// Text gibt. Eine automatische Zusammensetzung von
// System.Text.StringBuilder ist nicht möglich
Text.StringBuilder sb = new Text.StringBuilder();
```

6.2.1 Aliase

Beim Einbinden mehrerer Namespaces kann es nun vorkommen, dass derselbe Typname in mehreren Namespaces definiert ist. Der Zugriff auf den Typ kann in diesem Fall nicht ohne Angabe des vollständigen Namespace vor dem Typnamen erfolgen. Um dennoch Schreibarbeit zu vermeiden, können Sie Aliase für Namespaces und deren Typen erzeugen. Der using-Direktive folgt zur Definition eines Aliases ein Bezeichner und mit einem Gleichheitszeichen getrennt der Name eines Namespace oder eines voll qualifizierten Typs. Die Definition von Aliases muss vor allen Typdeklarationen erfolgen, also wie beim üblichen Gebrauch der using-Direktive.

```
using K6Test = CSharpBuch.Kap06.Test;
using Kap6 = CSharpBuch.Kap06;
...
K6Test t = new K6Test();
Kap6.Test t = new Kap6.Test();
```

Hinweis

Befinden Sie sich in einem bestimmten Namespace, können Sie implizit auf alle Bezeichner über die unqualifizierten Namen zugreifen, die in diesem Namespace definiert werden.

Achtung

Das Einbinden zweier Namespaces, die den gleichen Bezeichner enthalten, erzeugt noch keinen Compiler-Fehler. Erst die unqualifizierte Verwendung des Bezeichners wird vom Compiler bemängelt, da er nicht feststellen kann, welcher gemeint ist. Ein Ausweg besteht in der Verwendung des voll qualifizierten Namens bei der Erzeugung eines Objekts.

```
using NameSpaceA; // enthält den Typ Test
using NameSpaceB; // enthält auch den Typ Test
...
Test t = new NameSpaceB.Test();
```

Beispiel
Verwendung von Aliases für Namespaces und Typen

In der Anwendung wird für den Namespace `MitNamespace` und den Typ `MitNamespace.TestAusgabe` jeweils ein Alias definiert. Außerdem befindet sich der Typ `TextAusgabe` noch einmal ohne Namespace in der Anwendung. Über `new` wird dann dreimal ein Objekt vom betreffenden Typ erzeugt und sofort im Anschluss über den Punkt die Methode `Ausgabe()` aufgerufen. Dies ist also auch ein Beispiel, welches die Objekterzeugung mit einem sofortigen Methodenaufruf verbindet. Allerdings haben Sie keinen Zugriff auf das Objekt, da es keiner Referenzvariablen zugewiesen wird. Zuerst wird die Methode `Ausgabe()` der Klasse `TestAusgabe` aufgerufen, die sich in keinem Namespace befindet. Dann wird zweimal die Methode `Ausgabe()` der Klasse verwendet, die im Namespace `MitNamespace` untergebracht ist.

Listing 6.6
NamespaceAliase.cs

```
using System;
using MN = MitNamespace;
using MNC = MitNamespace.TestAusgabe;
namespace NamespaceAliase
{
  class NamespaceAliase
  {
    static void Main(string[] args)
    {
      new TestAusgabe().Ausgabe();
      new MN.TestAusgabe().Ausgabe();
      new MNC().Ausgabe();
      Console.ReadLine();
    }
```

Listing 6.6 (Forts.)
NamespaceAliase.cs

```
  }
}
namespace MitNamespace
{
  class TestAusgabe
  {
    public void Ausgabe()
    {
      Console.WriteLine("Mit Namespace...");
    }
  }
}
class TestAusgabe
{
  public void Ausgabe()
  {
    Console.WriteLine("Ohne Namespace...");
  }
}
```

Der Namespace global und der ::-Operator

Bei der Suche nach einem Bezeichner beginnt der Compiler in der aktuellen Klasse, dann in den anderen Klassen des aktuellen Namespace. Ist er darin nicht enthalten, werden die eingebundenen Namespaces durchsucht. Probleme treten auf, wenn Sie beispielsweise für die Member einer Klasse Bezeichner vergeben, die denen aus anderen eingebundenen Namespaces entsprechen. Diese Member der anderen Namespaces sind dann verborgen und können nur mit einem Zugriff über den globalen Namespace angesprochen werden. Diese Vorgehensweise steht ab der Version 2.0 des .NET Framework zur Verfügung.

Der Bezeichner `global` steht für die Wurzel aller Typen und Namespaces der globalen Umgebung. Bei der Angabe von `global` links vom Namespace beginnt die Suche nach einem Typ im globalen (unbenannten) Namespace. Das .NET-Framework referenziert `global::System` z.B. als den `System`-Namespace. Die Angabe von `global` ist nicht notwendig, wenn es keine Überlagerungen von Namen gibt. Wenn Sie jedoch auf gleichnamige Member eines anderen Namespace zugreifen möchten, müssen Sie den voll qualifizierten Namen von `global` aus angeben.

Der Namespace-Aliasqualifizierer `::` (ein weiterer Operator) wird verwendet, um einen Alias (für Namespaces) oder den globalen Namensraum zu referenzieren.

Beispiel
Verwendung des Namespace-Aliasqualifizierers

Sie definieren in Ihrer Klasse zwei Variablen mit den Namen `System` und `Console`. Dann führt der Aufruf der Methode `WriteLine()` über `Console.WriteLine()` bereits bei der Kompilierung zu einem Fehler. Sie müssen dann den vollen Namen, angefangen von der Wurzel `global` angeben. `global` wird durch den `::`-Operator mit dem folgenden Namespace bzw. Member verbunden.

```
global::System.Console.WriteLine("der ::-Operator");
```

Tipp

Sie sollten in der Regel Bezeichner so wählen, dass sie nicht mit den Namen des Namespace `System` oder anderen Namespaces übereinstimmen. Andernfalls wird ihr Code schwerer verständlich und schlechter zu warten. Auch Fehler lassen sich dann mitunter schwerer finden.
Der Zugriff über `global` wird in der Regel nur dann benötigt, wenn in einer neuen Version des .NET Framework neue Namespaces hinzugekommen sind, die dann eventuell zu Überschneidungen bei den Bezeichnern führen.

Auch auf Aliase kann der `::`-Operator angewandt werden. Für obiges Beispiel aus dem Abschnitt Aliase wäre auch der folgende Aufruf möglich:

```
new MN::TestAusgabe().Ausgabe();
```

Hinweis

Der `::`-Operator kann nicht für Aliase verwendet werden, die für einen Typ stehen, z.B.

```
using MNC = MitNamespace.TestAusgabe;
```

6.3 Die using-Anweisung

Neben der `using`-Direktive zum Einbinden von Namespaces existiert außerdem die `using`-Anweisung, die dazu dient, für ein Objekt einen begrenzten Gültigkeitsbereich zu schaffen. Dazu wird die `using`-Anweisung direkt im Code eingesetzt. In runden Klammern wird eine Instanz einer Klasse erzeugt. Diese ist nur innerhalb des folgenden Anweisungsblocks gültig. Der Typ des erzeugten Objekts muss die Schnittstelle `IDisposable` implementieren, da beim Verlassen des Gültigkeitsbereichs automatisch die Methode `Dispose()` aufgerufen wird (mehr dazu im Kapitel zu Interfaces). Implementiert der Typ diese Schnittstelle nicht, meldet bereits der Compiler einen Fehler.

Beispiel
Begrenzte Gültigkeitsbereiche mit der using-Anweisung

Direkt hinter der `using`-Anweisung wird in runden Klammern ein neues Objekt erzeugt, dessen Gültigkeitsbereich sich nur auf den folgenden Anweisungsblock der `using`-Anweisung beschränkt. Die Klasse `TestObjekt` implementiert das Interface `IDisposable`, was im konkreten Fall heißt, dass sie eine Methode `Dispose()` mit der angegebenen Signatur besitzt. Beim Verlassen des Gültigkeitsbereichs der `using`-Anweisung wird die Methode `Dispose()` aufgerufen, die noch vor der Methode `ReadLine()`, also unmittelbar nach Verlassen des Blocks, ausgeführt wird.

Listing 6.7
BeschraenkteGueltigkeit.cs

```
using System;
namespace CSharpBuch.Kap06
{
  class BeschraenkteGueltigkeit
  {
    static void Main(string[] args)
    {
      using(TestObjekt to = new TestObjekt())
      {
        to.Ausgabe();
      }
      Console.ReadLine();
    }
  }
}
class TestObjekt: IDisposable
{
  public void Ausgabe()
  {
    Console.WriteLine("Halli Hallo");
  }
  public void Dispose()
  {
    Console.WriteLine("Aufräumen mit Dispose");
  }
}
```

6.4 Übungsaufgaben

Aufgabe 1

Erstellen Sie eine neue Konsolenanwendung. Ändern Sie den Namespace in einen dreiteiligen Namen, der den Namen Ihrer Firma (oder den Familiennamen) und zwei weitere sinnvolle Untergliederungen enthält. Fügen Sie in die gleiche Datei einen weiteren Namespace mit einer beliebigen Klasse ein. Fügen Sie dieser Klasse eine Methode hinzu.

Erzeugen Sie ein Objekt dieser Klasse. Verwenden Sie dazu den voll qualifizierten Namen der Klasse und rufen Sie die Methode auf.

Aufgabe 2

Erstellen Sie für die Klasse aus Aufgabe 1 einen Aliasnamen und verwenden Sie diesen, um ein Objekt dieser Klasse zu erzeugen.

Aufgabe 3

Verwenden Sie zum Erstellen des Klassenobjekts aus Aufgabe 1 die `using`-Anweisung. Rufen Sie im Anweisungsblock dieser Anweisung die Methode der Klasse auf.