3.10.4 Vermeiden von Mehrdeutigkeiten

3.10.4 Vermeiden von Mehrdeutigkeiten▼▲

Namespaces dienen zur Strukturierung und Gruppierung von Klassen mit ähnlichen Merkmalen, aber auch zur Vermeidung von Mehrdeutigkeiten. Konflikte aufgrund gleicher Typbezeichner werden durch Namespaces vermieden. Allerdings kann die Bekanntgabe mehrerer Namespaces mit `using` Probleme bereiten, sollten in zwei verschiedenen Namespaces jeweils gleichnamige Typen existieren. Dann hilft `using` auch nicht weiter. Angenommen, in den beiden fiktiven Namespaces `MyApplication` und `YourApplication` wäre jeweils eine Klasse `Person` definiert, dann würde der folgende Code wegen der Uneindeutigkeit des Klassenbezeichners einen Fehler verursachen:

```
using MyApplication;
using YourApplication;

class Demo {
  static void Main(string[] arr) {
    Person obj = new Person();
    [...]
  }
}
```

Die Problematik lässt sich vermeiden, wenn der Namespace der Klasse `Person` näher spezifiziert wird, beispielsweise mit:

```
MyApplication.Person person = new MyApplication.Person();
```

Es gibt auch noch eine weitere Möglichkeit, um den Eindeutigkeitskonflikt oder eine überlange Namespace-Angabe zu vermeiden: die Definition eines Alias. Während die einfache Angabe ohne Alias hinter `using` nur einen Namespace erlaubt, ersetzt ein Alias den vollständig qualifizierenden Typbezeichner. Damit könnte die Klasse `Person` in den beiden Namespaces auch wie folgt genutzt werden:

```
using FirstPerson = MyApplication.Person;
using SecondPerson = YourApplication.Person;
[...]
FirstPerson person = new FirstPerson();
```

Genauso können Sie, falls Sie Spaß daran haben, die Klasse `Console` »umbenennen«, z. B. in `Ausgabe`:

```
using Ausgabe = System.Console;
[...]
Ausgabe.WriteLine("Hallo Welt");
```

3.10.5 Namespaces festlegen

3.10.5 Namespaces festlegen▼▲

Jedem neuen C#-Projekt wird von der Entwicklungsumgebung automatisch ein Namespace zugeordnet. Standardmäßig sind Namespace- und Projektbezeichner identisch.

Solange sich Typen innerhalb desselben Namespaces befinden, können sie sich gegenseitig direkt mit ihrem Namen ansprechen. Die Klassen `DemoA`, `DemoB` und `DemoC` des folgenden Codefragments sind demselben Namespace zugeordnet und benötigen deshalb keine vollqualifizierte Namensangabe.

```
namespace MyApplication
{
  class DemoA {[...]}
  class DemoB {{...}}
  class DemoC {[...]}
}
```

Eingebettete Namespaces

Eingebettete Namespaces

Ein Namespace kann mit einem Ordner des Dateisystems verglichen werden. So wie ein Ordner mehrere Unterordner enthalten kann, können auch Namespaces eine hierarchische Struktur bilden. Der oberste Namespace, der entweder dem Projektnamen entspricht oder manuell verändert worden ist, bildet die Wurzel der Hierarchie, ähnlich einer Laufwerksangabe.

Soll dieser Stamm-Namespace eine feinere Strukturierung aufweisen und eingebettete Namespaces verwalten, wird innerhalb eines Namespaces ein weiterer, untergeordneter Namespace definiert:

```
namespace Outer {
  class DemoA {
    static void Main(string[] args) {
      DemoB obj = new DemoB();
    }
  }

  namespace Inner {
    class DemoB {
      public void TestProc() {/*...*/}
    }
  }
}
```

Listing 3.51 Verschachtelte Namespaces

Ein Typ in einem übergeordneten Namespace hat nicht automatisch Zugriff auf einen Typ in einem untergeordneten Namespace. Damit das Codefragment auch tatsächlich fehlerfrei kompiliert werden kann, ist es erforderlich, mit

```
using Outer.Inner;
```

den inneren Gültigkeitsbereich den Typen in der übergeordneten Ebene bekannt zu geben.

3.10.6 Der „::“ – Operator

Auch für Namespaces lässt sich ein Alias festlegen, beispielsweise:

```
using EA = System.IO;
```

Sie können nun wie gewohnt den Punktoperator auf den Alias anwenden, also:

```
EA.StreamReader reader = new EA.StreamReader("...");
```

Seit dem .NET Framework 2.0 bietet sich aber auch die Möglichkeit, mit dem ::-Operator auf Typen aus Namespace-Aliasen zu verweisen.

```
EA::StreamReader reader = new EA::StreamReader("...");
```

- Der ::-Operator ist notwendig, um mit Hilfe von `global` auf den globalen Namespace zuzugreifen.
- Der ::-Operator sollte benutzt werden, um bei Verwendung eines Namespace-Alias kenntlich Eindeutigkeitskonflikte zu vermeiden.

3.10.7 Unterstützung von Visual Studio 2012 bei den Namespaces

3.10.7 Unterstützung von Visual Studio 2012 bei den Namespaces▲

Mit dem Anlegen eines neuen Projekts gibt Visual Studio eine Reihe von Namespaces mit `using` an. Welche das sind, hängt von der Projektvorlage ab. Dabei sind Namespaces, die durchaus benötigt werden, aber auch solche, von denen angenommen wird, dass ein Entwickler sie vielleicht gebrauchen könnte.

Sie können die in einer Quellcodedatei nicht benötigten Namespaces mit Hilfe der Entwicklungsumgebung sehr einfach loswerden. Öffnen Sie dazu das Kontextmenü des Code-Editors, und wählen Sie hier USING-DIREKTIVEN ORGANISIEREN. Anschließend können Sie entweder die nicht benötigten Direktiven löschen oder alle sortieren lassen (siehe Abbildung 3.9).

Ein ebenfalls sehr sinnvolles Feature ist das automatische Hinzufügen von benötigten `using`-Direktiven. Das setzt allerdings voraus, dass Sie die Klasse und natürlich auch deren Schreibweise hinsichtlich der Groß- und Kleinschreibung kennen. Geben Sie einfach den Klassenbezeichner im Editor ein, z. B. `StreamReader`. Anschließend setzen Sie den Mauscursor auf die Typangabe, öffnen das Kontextmenü und wählen hier AUFLÖSEN. Sie können sich dann entscheiden, ob die entsprechende `using`-Direktive in den Kopf der Quellcodedatei geschrieben werden soll oder der Typ vollqualifizierend im Code angegeben wird (siehe Abbildung 3.10).

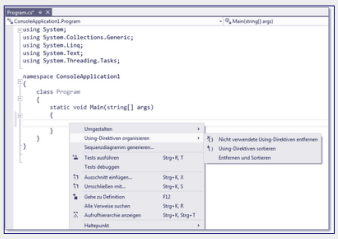

Abbildung 3.9 Verwalten der »using«-Direktiven in Visual Studio 2012