*Пространство имен* определяет область объявлений, в которой допускается хранить одно множество имен отдельно от другого. По существу, имена, объявленные в одном пространстве имен, не будут вступать в конфликт с аналогичными именами, объявленными в другой области. Так, в библиотеке классов для среды .NET Framework, которая одновременно является библиотекой классов С#, используется пространство имен System.

Пространства имен важны потому, что за последние годы в программировании

*"*расплодились*"* в огромном количестве имена переменных, методов, свойств и классов,

применяемых в библиотечных программах, стороннем и собственном коде. Поэтому

без отдельных пространств все эти имена будут соперничать за место в глобальном

пространстве имен, порождая конфликтные ситуации. Так, если в программе определен

класс Finder, то этот класс может вступить в конфликт с другим классом Finder,

доступным в сторонней библиотеке, используемой в этой программе. К счастью, подобного

конфликта можно избежать, используя отдельные пространства имен, ограничивающие

область видимости объявленных в них имен.

**Объявление пространства имен**

Пространство имен объявляется с помощью ключевого слова namespace. Ниже

приведена общая форма объявления пространства имен:

namespace имя

{

// члены

}

где *имя* обозначает конкретное имя объявляемого пространства имен. При объявлении

пространства имен определяется область его действия. Все, что объявляется непосредственно

в этом пространстве, оказывается в пределах его области действия. В пространстве

имен можно объявить классы, структуры, делегаты, перечисления, интерфейсы

или другие пространства имен.

namespace Counter

{

//simple counter

class CountDown

{

int val;

public CountDown(int n)

{

val = n;

}

public void Reset(int n)

{

val = n;

}

public int Count()

{

if (val > 0) return val--;

else return 0;

}

}

}

Ниже приведен пример программы, демонстрирующий применение пространства

имен Counter.

(***glava16\_1***)

class NSDemo

{

static void Main()

{

//take a look how class CountDown

//defines with namespace Counter

Counter.CountDown cd1 = new Counter.CountDown(10);

int i;

do

{

i = cd1.Count();

Console.Write(i + " ");

} while (i > 0);

Console.WriteLine();

Counter.CountDown cd2 = new Counter.CountDown(20);

do

{

i = cd2.Count();

Console.Write(i + " ");

} while (i > 0);

cd2.Reset(4);

do

{

i = cd2.Count();

Console.Write(i + " ");

} while (i > 0);

Console.WriteLine();

}

}

Некоторые важные аспекты данной программы заслуживают более пристального

внимания. Во-первых, при создании объекта класса CountDown необходимо дополнительно

определить его имя с помощью пространства имен Counter, как показано

ниже. Ведь класс CountDown объявлен в пространстве имен Counter.

Counter.CountDown cd1 = new Counter.CountDown(10);

Во-вторых, как только объект типа Counter будет создан, дополнительно определять

его члены с помощью пространства имен уже не придется. Следовательно, метод

cd1.Count() может быть вызван непосредственно без дополнительного указания пространства

имен, как в приведенной ниже строке кода.

i = cd1.Count();

И в-третьих, ради наглядности примера рассматриваемая здесь программа была

разделена на два отдельных файла. В одном файле содержится код объявления пространства

имен Counter, а в другом — код самой программы NSDemo. Но оба фрагмента

кода можно было бы объединить в единый файл. Более того, в одном файле исходного

кода может содержаться два или более пространства имен со своими собственными

областями объявлений. Когда оканчивается действие внутреннего пространства

имен, возобновляется действие внешнего пространства имен — в примере с Counter

это глобальное пространство имен. Ради большей ясности в последующих примерах

все пространства имен, требующиеся в программе, будут представлены в одном и том

же файле. Следует, однако, иметь в виду, что их допускается распределять по отдельным

файлам, что практикуется чаще в выходном коде.

**Предотвращение конфликтов имен с помощью пространств имен**

Главное преимущество пространств имен заключается в том, что объявленные в них

имена не вступают в конфликт с именами, объявленными за их пределами. Например,

в приведенной ниже программе определяются два пространства имен. Первым из них

является представленное ранее пространство имен Counter, а вторым — Counter2.

Оба пространства имен содержат классы с одинаковым именем CountDown, но поскольку

это разные пространства, то оба класса CountDown не вступают в конфликт

друг с другом. Кроме того, оба пространства имен определены в одном и том же файле.

Как пояснялось выше, это вполне допустимо. Безусловно, каждое из этих пространств

имен можно было бы выделить в отдельный файл, если бы в этом возникла

потребность.

(***glava16\_2***)

namespace Counter

{

//simple counter

class CountDown

{

int val;

public CountDown(int n)

{

val = n;

}

public void Reset(int n)

{

val = n;

}

public int Count()

{

if (val > 0) return val--;

else return 0;

}

}

}

//namespace Counter2

namespace Counter2

{

class CountDown

{

public void Count()

{

Console.WriteLine("This method Count() from namespace Count2.");

}

}

}

class NSDemo2

{

static void Main()

{

//take a look how class CountDown

//defines with namespace Counter

Counter.CountDown cd1 = new Counter.CountDown(10);

//this class CountDown from namespace Counter2

Counter2.CountDown cd2 = new Counter2.CountDown();

int i;

do

{

i = cd1.Count();

Console.Write(i + " ");

} while (i > 0);

Console.WriteLine();

cd2.Count();

}

}

Как следует из приведенного выше результата, класс CountDown из пространства

имен Counter существует отдельно от класса того же названия из пространства имен

Counter2, и поэтому конфликт имен не возникает. Несмотря на всю простоту данного

примера, он наглядно показывает, как удается избежать конфликта имен в собственном

коде и коде, написанном другими разработчиками, поместив классы с одинаковыми

именами в разные пространства имен.

**Директива using**

Если в программе присутствуют частые ссылки на члены конкретного пространства

имен, то указывать это пространство всякий раз, когда требуется ссылка на него, не

очень удобно. Преодолеть это затруднение помогает директива ***using***. Как и следовало ожидать, с помощью директивы using можно сделать видимыми вновь создаваемые пространства имен.

Существуют две формы директивы using. Ниже приведена первая из них:

using System;

Все члены, определенные в указанном пространстве имен, становятся видимыми,

и поэтому могут быть использованы без дополнительного определения их имен.

Директиву using необходимо вводить в самом начале каждого файла исходного кода

перед любыми другими объявлениями или же в начале тела пространства имен.

Приведенная ниже программа является вариантом предыдущего примера, переработанным

с целью продемонстрировать применение директивы using, делающей

видимым создаваемое пространство имен.

(***glava16\_3***)

//make visible namespace Counter !!!!!

using Counter;

namespace Counter

{

//simple counter

class CountDown

{

int val;

public CountDown(int n)

{

val = n;

}

public void Reset(int n)

{

val = n;

}

public int Count()

{

if (val > 0) return val--;

else return 0;

}

}

}

class NSDemo3

{

static void Main()

{

//take a look how class CountDown

//defines with namespace Counter

CountDown cd1 = new CountDown(10);

int i;

do

{

i = cd1.Count();

Console.Write(i + " ");

} while (i > 0);

Console.WriteLine();

}

}

Изменение состоит в том, что класс CountDown больше не нужно дополнительно определять с

помощью пространства имен Counter, как демонстрирует приведенная ниже строка

кода из метода Main().

Рассматриваемая здесь программа иллюстрирует еще одно важное обстоятельство:

применение одного пространства имен не отменяет действие другого. Когда пространство

имен делается видимым, это просто дает возможность использовать его содержимое

без дополнительного определения имен. Следовательно, в данном примере оба

пространства имен, System и Counter, становятся видимыми.

**Вторая форма директивы using**

Вторая форма директивы using позволяет определить еще одно имя (так называемый

*псевдоним*) типа данных или пространства имен. Эта форма приведена ниже:

using псевдоним = имя;

где *псевдоним* становится еще одним именем типа (например, типа класса) или пространства

имен, обозначаемого как *имя.* После того как псевдоним будет создан, он

может быть использован вместо первоначального имени.

Ниже приведен вариант программы из предыдущего примера, измененный с целью

показать создание и применение псевдонима MyCounter вместо составного имени

Counter.CountDown.

(***glava16\_4***)

//make nickname for name Counter.Countdown

using MyCounter = Counter.CountDown;

namespace Counter

{

//simple counter

class CountDown

{

int val;

public CountDown(int n)

{

val = n;

}

public void Reset(int n)

{

val = n;

}

public int Count()

{

if (val > 0) return val--;

else return 0;

}

}

}

class NSDemo4

{

static void Main()

{

//MyCounter instead of Counter.CountDown

MyCounter cd1 = new MyCounter(10);

int i;

do

{

i = cd1.Count();

Console.Write(i + " ");

} while (i > 0);

Console.WriteLine();

}

}

Псевдоним MyCounter создается с помощью следующего оператора.

//make nickname for name Counter.Countdown

using MyCounter = Counter.CountDown;

После того как псевдоним будет определен в качестве другого имени класса

Counter.CountDown, его можно использовать для объявления объектов без дополнительного

определения имени данного класса.

**Аддитивный характер пространств имен**

Под одним именем можно объявить несколько пространств имен. Это дает возможность

распределить пространство имен по нескольким файлам или даже разделить

его в пределах одного и того же файла исходного кода. Например, в приведенной

ниже программе два пространства имен определяются под одним и тем же именем

Counter. Одно из них содержит класс CountDown, а другое — класс CountUp. Во время

компиляции содержимое обоих пространств имен Counter складывается.

(***glava16\_5***)

using Counter;

namespace Counter

{

//simple counter

class CountDown

{

int val;

public CountDown(int n)

{

val = n;

}

public void Reset(int n)

{

val = n;

}

public int Count()

{

if (val > 0) return val--;

else return 0;

}

}

}

//another one Counter

namespace Counter

{

//just sum counter

class CountUp

{

int val;

int target;

public int Target

{

get

{

return target;

}

}

public CountUp(int n)

{

target = n;

val = 0;

}

public void Reset(int n)

{

target = n;

val = 0;

}

public int Count()

{

if (val < target) return val++;

else return target;

}

}

}

class NSDemo5

{

static void Main()

{

CountDown cd = new CountDown(10);

CountUp cu = new CountUp(8);

int i;

do

{

i = cd.Count();

Console.Write(i + " ");

} while (i > 0);

Console.WriteLine();

do

{

i = cu.Count();

Console.Write(i + " ");

} while (i < cu.Target);

Console.WriteLine();

}

}

Обратите также внимание на то, что директива

using Counter;

делает видимым все содержимое пространства имен Counter. Это дает возможность

обращаться к классам CountDown и CountUp непосредственно, т.е. без дополнительного

указания пространства имен. При этом разделение пространства имен Counter на

две части не имеет никакого значения.

**Вложенные пространства имен**

Одно пространство имен может быть вложено в другое. В качестве примера рассмотрим

следующую программу.

(***glava16\_6***)

namespace NS1

{

class ClassA

{

public ClassA()

{

Console.WriteLine("Contrcut class ClassA.");

}

}

namespace NS2 //nested namespace

{

class ClassB

{

public ClassB()

{

Console.WriteLine("Contrcut class ClassB.");

}

}

}

}

class NestedNSDemo

{

static void Main()

{

NS1.ClassA a = new NS1.ClassA();

//NS2.ClassB b = new NS2.ClassB(); Error! NS2 invisible

NS1.NS2.ClassB b = new NS1.NS2.ClassB();

}

}

В этой программе пространство имен NS2 вложено в пространство имен NS1. Поэтому

для обращения к классу ClassB необходимо дополнительно указать пространства

имен NS1 и NS2.

Вложенные пространства имен можно указать в одном операторе namespace, разделив

их точкой. Например, вложенные пространства имен

namespace OuterNS

{

namespace InnerNS

{

// ...

}

}

могут быть указаны следующим образом.

namespace OuterNS.InnerNS

{

// ...

}

**Глобальное пространство имен**

Если в программе не объявлено пространство имен, то по умолчанию используется

глобальное пространство имен. Именно поэтому в примерах программ, представленных

в предыдущих главах книги, не нужно было обращаться для этой цели к ключевому

слову namespace. Глобальное пространство удобно для коротких программ,

как в примерах из этой книги, но в большинстве случаев реальный код содержится в

объявляемом пространстве имен. Главная причина инкапсуляции кода в объявляемом

пространстве имен — предотвращение конфликтов имен. Пространства имен служат

дополнительным средством, помогающим улучшить организацию программ и приспособить

их к работе в сложной среде с современной сетевой структурой.

**Применение описателя псевдонима пространства имен ::, global**

Допустим, что два пространства имен содержат

класс MyClass. Если попытаться сделать видимыми оба пространства имен с помощью

директив using, то имя MyClass из первого пространства вступит в конфликт с

именем MyClass из второго пространства, обусловив появление ошибки неоднозначности.

В таком случае для указания предполагаемого пространства имен явным образом

можно воспользоваться *описателем псевдонима пространства имен* ::*.*

***псевдоним\_пространства\_имен :: идентификатор***

Здесь *псевдоним\_пространства\_имен* обозначает конкретное имя псевдонима пространства имен, а *идентификатор —* имя члена этого пространства.

Для того чтобы стало понятнее назначение описателя псевдонима пространства

имен, рассмотрим следующий пример программы, в которой создаются два пространства

имен, Counter и AnotherCounter, и в обоих пространствах объявляется класс

CountDown. Затем оба пространства имен становятся видимыми с помощью директив

using. И наконец, в методе Main() предпринимается попытка получить экземпляр

объекта типа CountDown.

Для того чтобы воспользоваться описателем ::, ***необходимо сначала определить***

***псевдоним для пространства имен***, которое требуется описать, а затем дополнить описание

неоднозначного элемента этим псевдонимом. Ниже приведен вариант предыдущего

примера программы, в котором устраняется упомянутая выше неоднозначность.

(***glava16\_7***)

using System;

//use both namespaces

using Counter;

using AnotherCounter;

//define :: for NS !!!!!!!!!!!!!!!!!!

using Ctr = Counter;

//define namespace for Counters

namespace Counter

{

class CountDown

{

int val;

public CountDown(int n)

{

val = n;

}

}

}

//one more namespace

namespace AnotherCounter

{

//another class CountDown

class CountDown

{

int val;

public CountDown(int n)

{

val = n;

}

}

}

class WhyAliasQualifier

{

static void Main()

{

Ctr::CountDown cd1 = new Ctr.CountDown(10);

}

}

В этом варианте программы для класса Counter сначала указывается псевдоним

Ctr в следующей строке кода.

//define :: for NS

using Ctr = Counter;

А затем этот псевдоним используется в методе Main() для дополнительного описания

класса CountDown, как показано ниже.

Ctr::CountDown cd1 = new Ctr.CountDown(10);

Описатель :: устраняет неоднозначность, поскольку он явно указывает на то, что

следует обратиться к классу CountDown из пространства Ctr, а фактически — Counter.

Именно это и делает теперь программу пригодной для компиляции.

Описатель :: можно также использовать вместе с предопределенным идентификатором

global для ссылки на глобальное пространство имен. Например, в приведенной

ниже программе класс CountDown объявляется как в пространстве имен Counter,

так и в глобальном пространстве имен. А для доступа к варианту класса CountDown в

глобальном пространстве имен служит предопределенный псевдоним global.

(***glava16\_8***)

using System;

using Counter;

//define :: for NS

using Ctr = Counter;

//define namespace for Counters

namespace Counter

{

class CountDown

{

int val;

public CountDown(int n)

{

val = n;

}

}

}

//another class CountDown, its in global namespace

class CountDown

{

int val;

public CountDown(int n)

{

val = n;

}

}

class GlobalAliasQualifier

{

static void Main()

{

//here is :: using CountDown from NS

Ctr::CountDown cd1 = new Ctr.CountDown(10);

//now create CountDown from Global namespace

global::CountDown cd2 = new CountDown(10);

}

}

Обратите внимание на то, что идентификатор global служит для доступа к классу

CountDown из используемого по умолчанию пространства имен.

//now create CountDown from Global namespace

global::CountDown cd2 = new CountDown(10);

Этот подход можно распространить на любую ситуацию, в которой требуется указывать

используемое по умолчанию пространство имен.

И последнее: описатель псевдонима пространства имен можно применять вместе с

псевдонимами типа extern, как будет показано в главе 20.