5. Пространства имен

Все определяемые классы не существуют сами по себе, а, как правило, заключаются в специальные контейнеры - пространства имен. Создаваемый по умолчанию класс Program уже находится в пространстве имен, которое обычно совпадает с названием проекта:

```
namespace HelloApp
{
   class Program
   {
      static void Main(string[] args)
      {
      }
   }
}
```

Что же представляет собой пространство имён в C#?

Это некая область объявления данных, обеспечивающую логическую взаимосвязь типов.

Другими словами, пространство имён - это способ группирования ассоциированных типов, включая классы, структуры, делегаты, интерфейсы, перечисления, а также другие пространства имён.

Это логический способ объединения типов.

Пространство имен определяется с помощью ключевого слова namespace, после которого идет название. Так в данном случае полное название класса Program будет ConsoleApplication1.Program.

Класс Program видит все классы, которые объявлены в том же пространстве имен. Например класс Student нам будет виден в Main если он будет размещен в пространстве имен ConsoleApplication1:

```
namespace ConsoleApplication1//первый листинг
{
   class Program
   {
      static void Main(string[] args)
      {
         Student std = new Student(«Vasya»,17);
      }
   }
}
namespace ConsoleApplication1//второй листинг
{
   class Student
   {
      string name;
      int age;

      public Student(string name, int age)
      {
      this.name = name;
      this.age = age;
      }
   }
}
```

Но чтобы задействовать классы из других пространств имен, эти пространства надо подключить с помощью директивы using. Например класс Student имел свое собственное пространство имен –

MyStudent, то в верху нам бы пришлось добавить: using MyStudent, так как будет нарушена область видимости:

```
using MyStudent; //первый листинг добавление пространства имен MyStudent.
namespace ConsoleApplication1
{
   class Program
   {
      static void Main(string[] args)
      {
          Student std = new Student(«Vasya»,17);
      }
   }
}
namespace MyStudent//второй листинг
{
   class Student
   {
      string name;
      int age;

      public Student(string name, int age)
      {
       this.name = name;
       this.age = age;
      }
   }
}
```

Или же если мы не хотим подключать целое пространство имен (там могут быть другие классы, которые нам не нужны, и соответственно это увеличит нагрузку на проект) при объявлении экземпляра класса использовать полное имя класса с пространством имен. Например в Main:

```
static void Main(string[] args)
    {
      MyStudent.Student std = new MyStudent.Student("Vasya", 17);
    }
```

Предотвращение конфликтов совпадения имен обьектов

Если какой-нибудь тип (пусть это будет класс Poetry) бы объявлен в своём пространстве имён (пусть он называется Inspiration) то это обозначает, что этот тип будет мирно существовать в своё пространстве имён, не конфликту одноимёнными типами, объявленными в пространствах имён, отличных от данного. Например мы можем использовать два класса которые имеют одинаковые имена:

```
namespace StudentStepAcademy//первый листинг
{
   class Student
   {
      string name;
      int age;

      public Student(string name, int age)
      {
       this.name = name;
       this.age = age;
```

```
        }
    }
}
 namespace StudentStepAcademy//третий листинг
 {
    class Student
    {
        string name;
        int age;

        public Student(string name, int age)
        {
        this.name = name;
        this.age = age;
        }
    }
}
```

Соответственно в Main:

```
static void Main(string[] args)
    {
      StudentStepAcademy.Student std = new StudentStepAcademy.Student("Vasya", 17);
      StudentBerklyAcademy.Student std1 = new StudentBerklyAcademy.Student("John", 17);
    }
```

Таким образом мы можем разместить один или оба класса с одинаковым именем в разные пространства имен. Самое главное чтобы имена пространств не совпадали, или были вложены в пространства имен с разными названиями.

Вложенные пространства имен

Пространства имен могут быть определены внутри других пространств:

```
namespace ConsoleApplication1
{
  class Program
  {
    static void Main(string[] args)
    {

    }
  }

  namespace Area
  {
    class Area1
    {
      public double area;

      public Area1(double area)
      {
        this.area = area * area;
      }
    }
  }
}
```

В данном случае в пространство имен ConsoleApplication1 вложено пространство имен Area1 с обьектом Area. Чтобы им воспользоваться нужно добавить using ConsoleApplication1.Area; (перейти на внутренне пространство с внешнего) и в Main написать следующий код:

```
Area1 a = new Area1(12.3);
Console.WriteLine(a.area);
```

Или же не подключая пространства имен писать:

```
Area.Area1 a = new Area.Area1(12.3);
Console.WriteLine(a.area);
```

Если бы в нас обьект в пространстве Area имел бы имя как и пространство – тоже Area. То соответсвенно подключение ConsoleApplication1.Area; нам бы не помогло и нам бы пришлось писать с приведением пространства имен и имени обьекта, поскольку оба имени совпадают:

```
namespace ConsoleApplication1
{
  class Program
  {
    static void Main(string[] args)
    {
      Area.Area a = new Area.Area(12.3);
      Console.WriteLine(a.area);
    }
  }

  namespace Area
  {
    class Area
    {
      public double area;

      public Area(double area)
      {
        this.area = area * area;
      }
    }
  }
}
```

Аналогично если в нас в пространстве имен ConsoleApplication1 уже существовал класс с названием Area как и во вложенном пространстве имен ConsoleApplication1.Area.Area для избегания совпадения имен объектов в внешнем пространстве имен и во внутреннем:

```
namespace ConsoleApplication1
{
  class Program
  {
    static void Main(string[] args)
    {
      Area b= new Area(14.3);//обращение к классу Area во внешнем пространстве имен
      Console.WriteLine(b.area);
      Area.Area a = new Area.Area(12.3); /обращение к классу Area во внутреннем пространстве имен
      Console.WriteLine(a.area);
    }
  }
    class Area//класс Area во внешнем пространстве имен ConsoleApplication1
    {
      public double area;
```

```
        public Area(double area)
        {
           this.area = area * area;
        }
      }
   namespace Area
   {
      class Area//класс Area во внутреннем пространстве имен ConsoleApplication1.Area
      {
        public double area;
        public Area(double area)
        {
           this.area = area * area;
        }
      }
}
```

По поводу вложенности пространств имён нужно понимать одно простое правило: вы создаёте области видимости, вложенные друг в друга, а это значит, что все типы, объявленные в пространстве имён на ступеньку ниже, не доступны из объемлющих пространств имён.

Разбиение пространства имён на части

Мы можем в пределах одного или несколько файлов разделяете пространство имён на части**.**
Например:

```
//Class1.cs
using System;
namespace A
{
   public class ClassA
   {
     public void Print()
     {
      Console.WriteLine("Printing from A.ClassA");
     }
   }
}
//Class2.cs
namespace A
{
  class ClassB
  {
    public void Print()
    {
       Console.WriteLine("Printing from A.ClassB");
    }
}
```

И в Main подключив using A мы увидим оба класса:

```
using A;
namespace ConsoleApplication1
{
   class Program
   {
      static void Main(string[] args)
      {
         ClassA a = new ClassA();
```

```
            a.Print();
            ClassB b = new ClassB();
            b.Print();
        }
    }
```

Вторая форма using. Псевдонимы классов и пространства имен

Для различных классов мы можем использовать псевдонимы. Затем в программе вместо названия класса используется его псевдоним. Например, для вывода строки на экран применяется метод Console.WriteLine(). Но теперь зададим для класса Console псевдоним:

```
using printer = System.Console;
class Program
{
    static void Main(string[] args)
    {
        printer.WriteLine("Hello from C#");
        printer.Read();
    }
}
```

Особенно это эффективно когда мы используем длинные имена пространств имен, с вложенными пространствами имен или классами. Например:

```
using NS = Wrox.ProCSharp.WindowsApplication.SimpleForm;
```

Но бывают ситуации когда псевдонимы пространства имен совпадают с именами класса:

```
using Area = RectArea;//пространству имен ReactArea задаем псевдоним Area
namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
            Area a = new Area(14.4);
            Console.WriteLine(a.area);
            Area::AreaRectangle ra = new RectArea.AreaRectangle(23.7);
            Console.WriteLine(ra.area);
        }
    }
    class Area//имя класс Area  совпадает с псевдонимом пространства имен RectArea - Area
    {
        public double area;

        public Area(double radius)
        {
            this.area = Math.PI*Math.Pow(radius,2);
        }
    }
}

namespace RectArea
{
    class AreaRectangle
    {
        public double area;

        public AreaRectangle(double area)
```

```
    {
      this.area = area * area;
    }
  }
}
```

Как видим псевдоним пространства имен Area совпадает с именем класса Area. Компилятор дает в данной ситуации преимущество классу Area:

```
Area a = new Area(14.4);
Console.WriteLine(a.area);
```

А для вызова метода AreaRectangle через псевдоним пространства имен Area имени пространства RectArea нам нужно использовать квалификатор - ::

```
Area::AreaRectangle ra = new Area::AreaRectangle(23.7);
Console.WriteLine(ra.area);
```

Импорт функциональности классов

Начиная с версии C# 6.0 (Visual Studio 2015) в язык была добавлена возможность импорта функциональности классов. Например, опять же возьмем класс Console и применим новую функциональность:

```
using static System.Console;
namespace HelloApp
{
  class Program
  {
    static void Main(string[] args)
    {
      WriteLine("Hello from C# 6.0");
      Read();
    }
  }
}
```