

# C# Essential

Пространства имен. Директивы препроцессора.

# C# Essential

Автор курса



Александр Шевчук  
MCT



MCID: 9230440

# C# Essential

После урока обязательно



Повторите этот урок в видео формате на [ITVDN.com](http://itvdn.com)

Доступ можно получить через руководство вашего учебного центра



Проверьте как Вы усвоили данный материал на [TestProvider.com](http://testprovider.com)

# Пространства имен. Директивы препроцессора.

# Пространства имен

## Понятие пространства имен

**Пространства имен (namespace)** – это способ, благодаря которому .NET избегает конфликтов имен между классами.

**Пространство имен**



```
System.Console.WriteLine("Hello world");
```



Полная квалификация имени стереотипа, включает имя пространства имен, в котором находится стереотип.

# Пространства имен

## Свойства пространства имен

**Пространства имен имеют следующие свойства:**

- Организация крупных проектов по созданию кода.
- Для их разделения используют оператор . (точка).
- Директива `using` исключает требование на указание имени пространства имен для каждого класса, избавляя от необходимости полной квалификации имен стереотипов
- Пространство имен `global` является корневым пространством имен: `global::System` всегда будет ссылаться на пространство имен платформы **.NET Framework System**

# Директива

## using

Директива **using** – импортирует пространство имен, избавляя от необходимости полной квалификации имен стереотипов

```
using System;

namespace Namespaces
{
    class Program
    {
        static void Main()
        {
            Console.WriteLine("Hello world");
        }
    }
}
```



Ключевое слово **using** также используется для создания операторов **using**, которые обеспечивают правильную обработку объектов **IDisposable**, например файлов и шрифтов

# Пространства имен

## using

Директива `using` позволяет создавать псевдонимы пространства имен или типа. Это называется директива `using alias`

```
using MyClass = NamespaceA.NamespaceB.NamespaceC.MyClassC;
```

Создание псевдонима `MyClass`, для класса `MyClassC` из пространства имен `NamespaceA.NamespaceB.NamespaceC`.



Директива `using alias` не может иметь открытый универсальный тип с правой части. Например, невозможно создать `using alias` для `List<T>`, но можно создать для `List<int>`.



# Пространства имен

## using

В случае отсутствия импорта пространства имен **System**, полные имена базовых типов будут недоступными.

```
//using System;  
  
class Program  
{  
    static void Main()  
    {  
        // Int32 a = 1;  
  
        int b = 2;  
    }  
}
```



Доступными будут только псевдонимы типов.

# Пространства имен

## internal

Ключевое слово **internal** является модификатором доступа для типов и членов типов.

```
extern alias L1;

namespace Namespaces
{
    class Program
    {
        static void Main()
        {
            L1.Library.MyClass my1 = new L1.Library.MyClass();

            my1.Method();
        }
    }
}
```

Доступ к типам или членам с модификатором доступа **protected internal** может осуществляться из текущей сборки или из типов, которые являются производными от содержащего их класса

# Пространства имен

## Подключение алиаса

**Для подключения алиаса выполните следующие действия:**

- Добавьте в [References](#) необходимые сборки.
- Откройте папку [References](#).
- Правой кнопкой мыши кликните по сборке, откроется контекстное меню, в котором выберите пункт [Properties](#).
- В открывшемся окне свойств, в свойстве [Aliases](#), замените значение [global](#) на свое название

# Модификаторы доступа

## Обзор и применение модификаторов доступа

**public** – доступ к типу или члену возможен из любого другого кода в той же сборке или другой сборке, ссылающейся на него.

**protected** – доступ к типу или элементу можно получить только из кода в том же классе или структуре, либо в производном классе.

**internal** – доступ к типу или члену возможен из любого кода в той же сборке, но не из другой сборки.

**protected internal** – доступ ограничен текущей сборкой или типами, которые являются производными от содержащего класса.

**private** – доступ к типу или члену можно получить только из кода в том же классе или структуре.

# Директивы компилятора

## #region

Директива **#region** позволяет указать блок кода, который можно разворачивать и сворачивать с помощью функции структурирования в редакторе кода **Visual Studio**.

```
static void Main()
{
    #region MyRegion

    Console.WriteLine("Hello...");

    #endregion
}
```

В больших файлах кода очень удобно сворачивать или скрывать одну или несколько областей, чтобы не отвлекать внимание от той части файла, над которой в настоящее время идет работа

# Директивы компилятора

## #if #endif

При обнаружении компилятором C# директивы `#if`, за которой далее следует директива `#endif`, компиляция кода между двумя директивами выполняется только в том случае, если определен указанный символ

```
static void Main()
{
    #if DEBUG
        Console.WriteLine("Debug version");
    #endif

    // TODO: Посмотрите в Task List
    // HACK: Посмотрите в Task List

    Console.WriteLine("Release version");
}
```

Task List - 2 tasks		
Comments		
!	Description ^	
	TODO: Посмотрите в Task List	File ^ Line ^
	HACK: Посмотрите в Task List	Program.cs 16
		Program.cs 17

# Директивы компилятора

## #if #endif

Оператор `#if`, вместе с операторами `#else`, `#elif`, `#endif`, `#define` и `#undef`, позволяет включать или исключать код на основе существования одного или нескольких символов.

```
static void Main()
{
    #if (DEBUG && !VC_V7)
        Console.WriteLine("DEBUG is defined");
    #elif (!DEBUG && VC_V7)
        Console.WriteLine("VC_V7 is defined");
    #elif (DEBUG && VC_V7)
        Console.WriteLine("DEBUG and VC_V7 are defined");
    #else
        Console.WriteLine("DEBUG and VC_V7 are not defined");
    #endif
}
```

Это особенно полезно при компиляции кода для построения отладки или при компиляции для определенной конфигурации

# Смотрите наши уроки в видео формате

ITVDN.com



Посмотрите этот урок в видео формате на образовательном портале [ITVDN.com](http://itvdn.com) для закрепления пройденного материала.

Все курсы записаны сертифицированными тренерами, которые работают в учебном центре CyberBionic Systematics





# Проверка знаний

TestProvider.com

TestProvider

Мы помогаем людям оценить себя

Главная Услуги и цены Центр Тестирования Поддержка О нас

Регистрация Войти

Поиск сертификата

Мы в социальных сетях

## Тестирование

Языки программирования и информационные технологии

**Microsoft**

C# ASP.NET MVC JavaScript Patterns OF Design SQL Architecture Guide WCF HTML&CSS XML SEO WPF HTML5&CSS3 JQuery XNA SharePoint GUI for Android Windows Azure Platform Microsoft Patterns&Practices TFS SCRUM ReSharper TDD WWF LINQ Entity Framework Windows Forms Refactoring Microsoft Expression Blend 4 Windows Phone 8 Windows 8 AppStore Visual Studio Tips&Tricks MSF MEF SilverLight AJAX MEF Service Oriented Architecture

Добро пожаловать на TestProvider.com!

Сайт перенесен на новую облачную платформу с использованием системы единой авторизации Single Sign On. Если вы хотите восстановить статистику по предыдущим экзаменам обратитесь в [службу поддержки](#). Для восстановления информации с предыдущей версии сайта, просба написать в службу поддержки Ваш старый и новый логины.

ITVDN PROMETRIC TEST CENTER CyberBionic Microsoft Partner Windows Azure Cloud Partner EBA

TestProvider – это online сервис проверки знаний по информационным технологиям. С его помощью Вы можете оценить Ваш уровень и выявить слабые места. Он будет полезен как в процессе изучения технологии, так и общей оценки знаний IT специалиста.

После каждого урока проходите тестирование для проверки знаний на [TestProvider.com](http://TestProvider.com)

Успешное прохождение финального тестирования позволит Вам получить соответствующий Сертификат.



# C# Essential

Q&A

# Информационный видеосервис для разработчиков программного обеспечения

