C# Essential

Пространства имен. Директивы препроцессора.

После урока обязательно

Повторите этот урок в видео формате на [ITVDN.com]

Проверьте как Вы усвоили данный материал
на [TestProvider.com]

Тема

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.

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

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

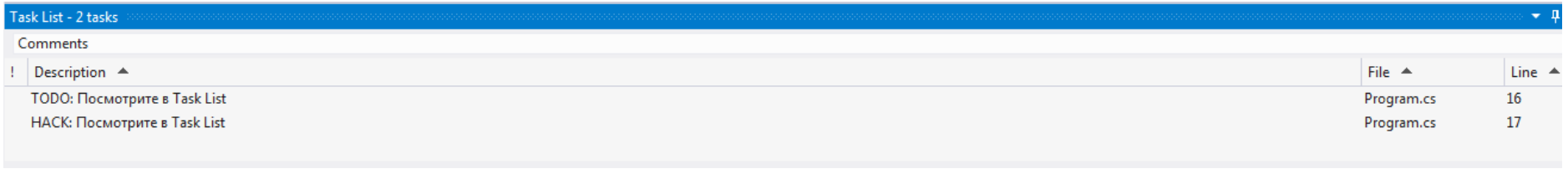
#if #endif

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");
}
```



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

Q&A

Информационный видеосервис для разработчиков программного обеспечения

ITVDN

IT VIDEO DEVELOPERS NETWORK

TestProvider PROMETRIC TEST CENTER CyberBionic systematics Microsoft Partner Silver Learning Windows Azure Cloud Partner EBA European Business Association