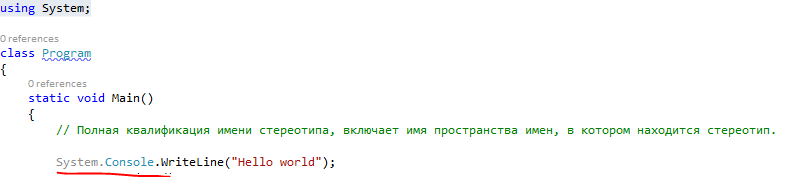


Пространство имен – некая единица изоляции программного кода (чтобы создавать одноименные классы)

Полная квалификация имя

Обращаемся к классу консоль путем полного квалифицированного имени

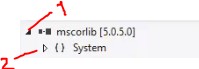
Тогда ДЕРЕКТИВУ using можно не импортировать



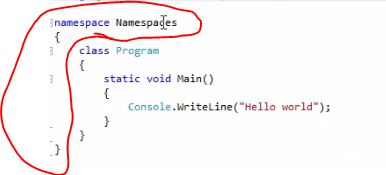
1-Сборка

2-Пространство имен

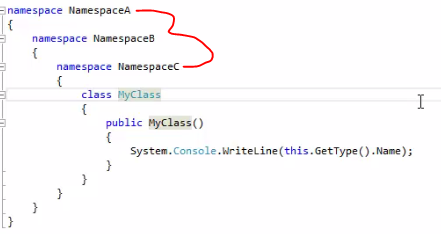
3-классы



Если не указывать пространство имен, то это будет глобальное пространство имен

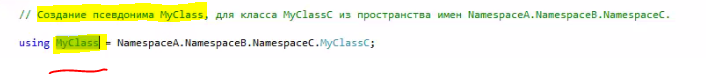


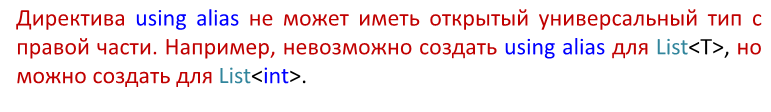
Вложенные пространство имен



Создание псевдонима класса

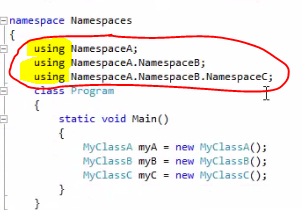
Как int – Int32 string – String (тоже псевдонимы)





Импорт как вариант можно указать внутри пространства имен

// Импорт допустим или в теле namespace, или перед всеми другими пространствами имен.

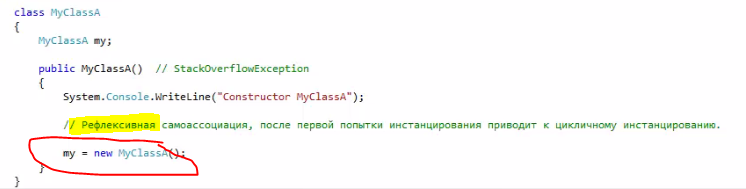


Видимость со сложенных пространствах имен

(Как при наследовании, знает базовые классы, но не знает производные классы)

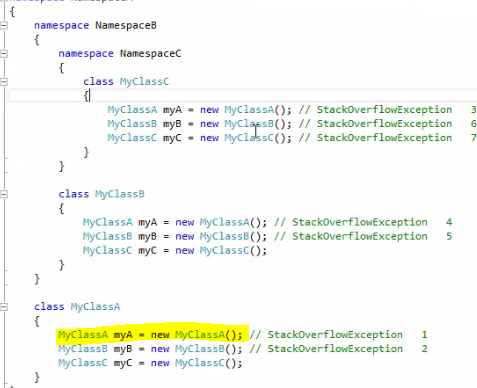


Нельзя, т.к. приведет к зацикливанию будет вызывать конструктор

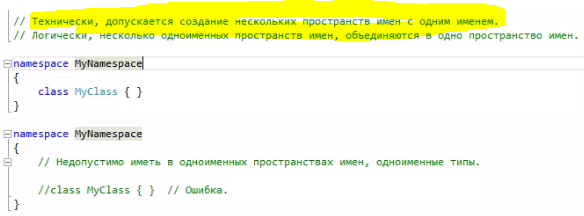


И так нельзя, создавать самого себя

Распространенные ошибки самоассоциации после первой попытки инстанцирования.

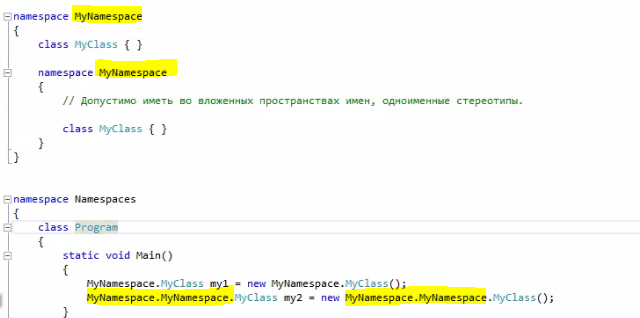


Можно создавать одноименные пространства имен, но в них тогда нельзя создавать одноименные классы



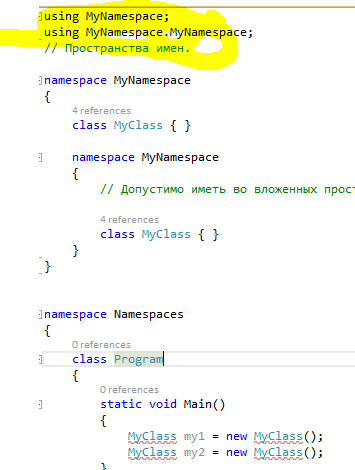
Допустимо вкладывать одноименные пространства имен,

Но тогда нужно использовать полное квалифицированное имя класса с учетом всей вложенности пути

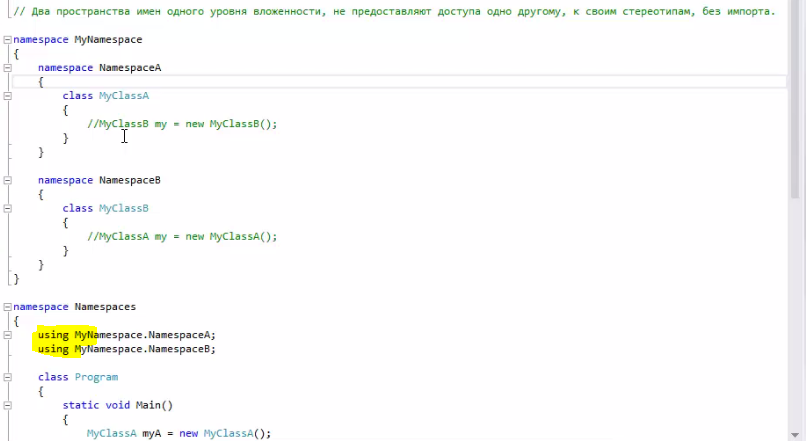


Вот так ошибка

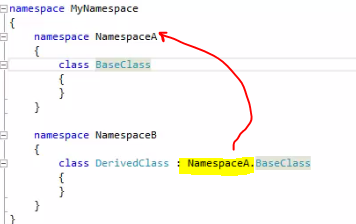
ambiguous reference - неоднозначным ссылки



При обращении к одному из пространств имен на одном уровне вложенности следует импортировать пространства имени



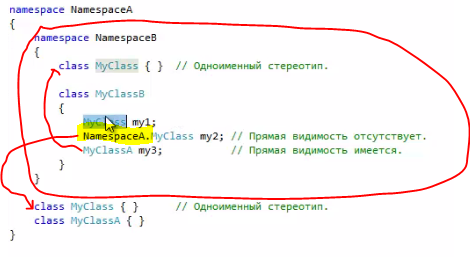
При наследовании можно использовать полное квалифицированное имя типа



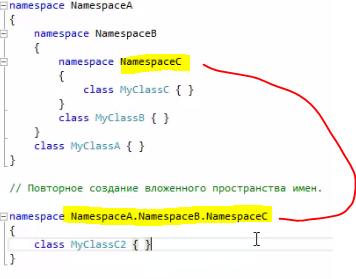
// В случае наличия одноименных стереотипов во вложенном и во внешнем пространствах имен,

// происходит сокрытие имени стереотипа внешнего пространства имен.

// Обращение к одноименному стереотипу внешнего пространства имен, потребует полной квалификации имени стереотипа.

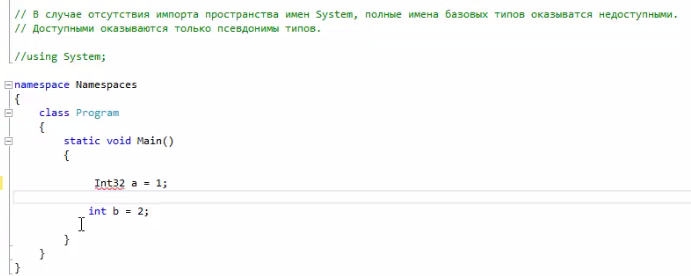


Расширение пространства имен (одна область)



Полные имена базовых типов не доступны, только их псевдонимы

Без импорта System можем видеть алиасы (псевдонимы), но типы видеть не можем

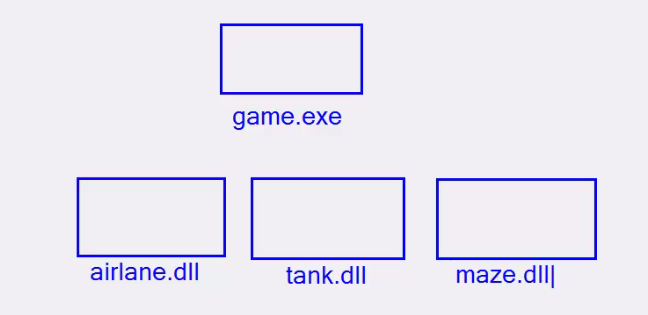


DLL библиотека

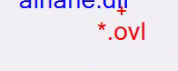
Dynemic linq lyblary

Динамически подключаемые библиотеки

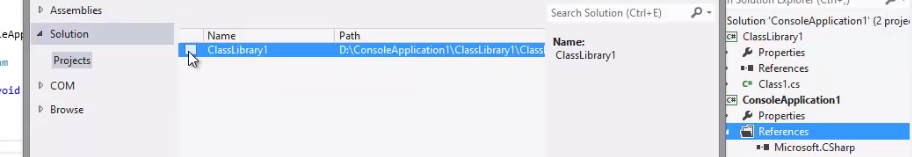
Нужны для того чтобы не подгружать в память все возможности игры за один раз



Раньше под DOS назывались overlay



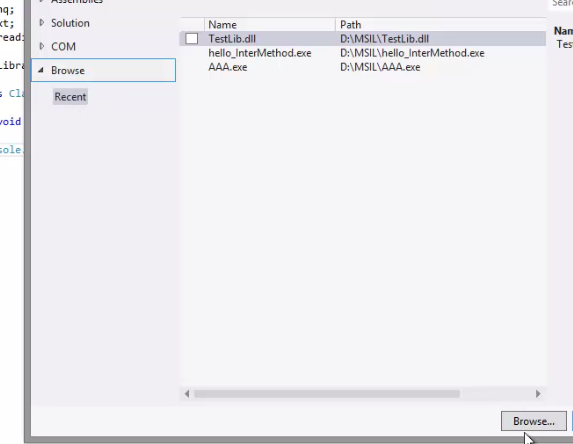
Подключение библиотеки (подключаем как проект)



После делаем импорт



Второй способ подключения dll сборки (подключаем как файл (чужой файл)) и после делаем импорт



Разница между первым и вторым способом подключения состоит в том, что мы не можем добавить комментарий либо изменить тело метода когда библиотека подключается через Browse

В случае когда при подключении библиотек совпадают их пространства имен и имена классов, необходимо

// Совпадение имен двух пространств имен в разных сборках и совпадение имен стереотипов в этих пространствах имен.

// Для подключения алиаса выполните следующие действия:

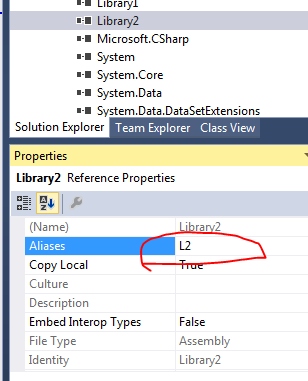
// 1. Добавьте в References необходимые сборки Library1 и Library2.

// 2. Откройте папку References.

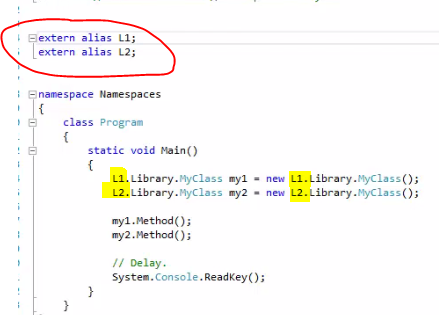
// 3. Правой кнопкой мыши кликните по сборке Library1, откроется контекстное меню, в котором выберите пункт Properties.

// 4. В открывшемся окне свойств, в свойстве Aliases, замените значение global на L1.

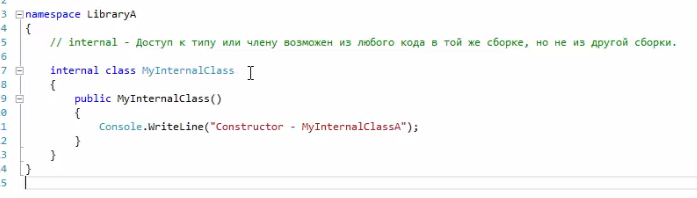
Расширили ПРОСТРАНСТВО ИМЕН



После

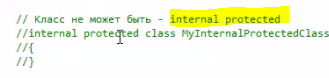


Модификатор доступа INTERNAL



Могут быть методы

Класс не может быть помечен как Internal protected



Internal – для другой сборке он является private

/ internal - Доступ к типу или члену возможен из любого кода в этой сборке, но не из другой сборки.

internal void InternalMethod()

{

Console.WriteLine("InternalMethod");

}

Internal protected – для другой сборке он является protected

// internal protected - Доступ к типу или члену возможен из любого кода в этой сборке,

//или из производного класса в другой сборке.

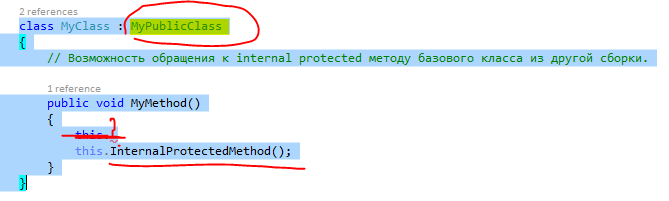
internal protected void InternalProtectedMethod()

{

Console.WriteLine("InternalProtectedMethod");

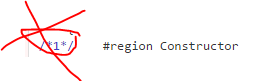
}

Т.е. если мы в другой сборке наследуемся от класса в котором имеются данные методы с указанным модификатором доступа то internal protected void InternalProtectedMethod() можем использовать в теле данного производного класса

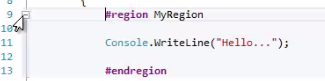


Дерективы препроцессора

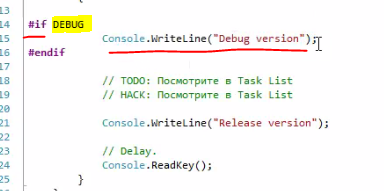




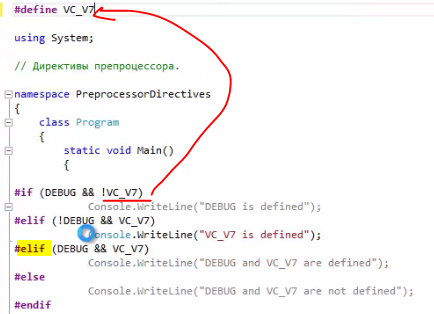
Деректива #region – используется для удобства (сворачивания кода)



#if #def – выполнить какое либо условие, в данном примере если в режиме Debug, то строка выполнится



Если Debug и 7 версия продукта (образно)



#line

В данном примере показывает строку 123, а не 13

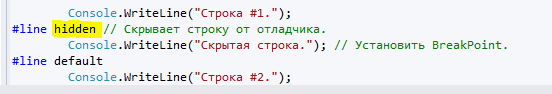
И при отладке F11 переносит курсор на указанную строку





# line hidden

Отладчик пропускает указанный участок кода

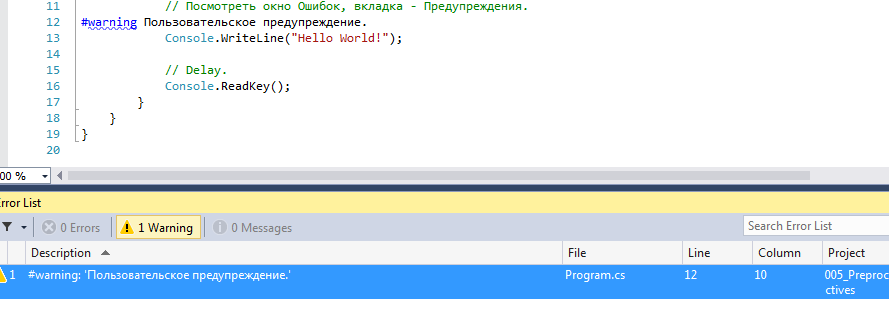


#line default

Убирает hidden

#warning

Пользовательское предупреждение



#error

Ошибка определенная пользователем, программа перестанет билдиться

Если над проектом работают много людей, чтобы ни кто не смог запустить

