В.В. Подбельский

Иллюстрации к курсу лекций по дисциплине «Программирование»

Использованы материалы пособия Daniel Solis, Illustrated C#

Классы и наследование

Объявление производного класса

Указание базового класса (базы)

↓

```
class OtherClass  : SomeClass
{
```

↑ ↑

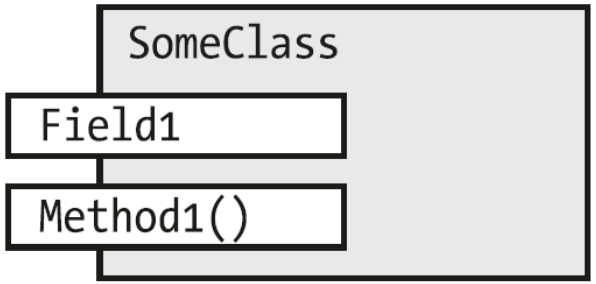
двоеточие базовый класс

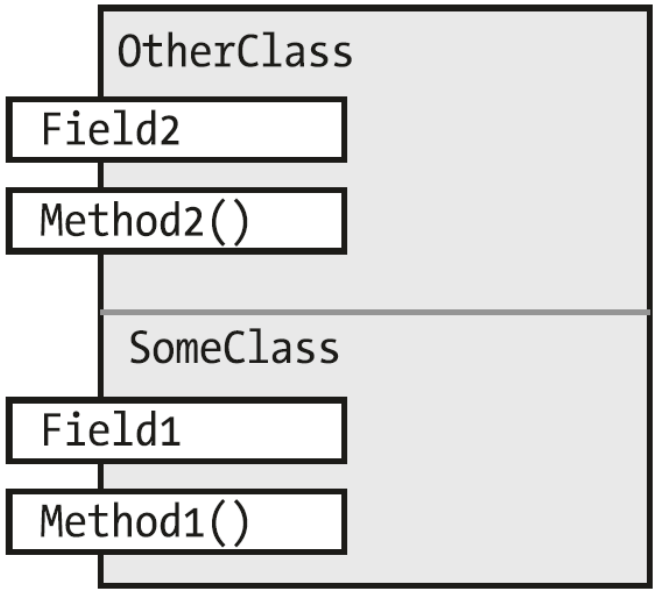
```
}
```

Коды классов при наследовании

```
// базовый класс
class SomeClass
{
    public string Field1 = "base class field";
    public void Method1(string value)
    {
        Console.WriteLine("Base -- Method1: {0}", value);
    }
}

// производный класс
class OtherClass : SomeClass
{
    public string Field2 = "derived class field";
    public void Method2(string value)
    {
        Console.WriteLine("Derived -- Method2: {0}", value);
    }
}
```

Объекты базового и производного классов





Взаимодействие членов классов

```
class Program
{
    static void Main()
    {
        OtherClass oc = new OtherClass();
        oc.Method1(oc.Field1); // Метод базового класса с полем базового класса
        oc.Method1(oc.Field2); // Метод базового класса с полем производного класса
        oc.Method2(oc.Field1); // Метод производного класса с полем базового класса
        oc.Method2(oc.Field2); // Метод производного класса с полем производ. класса
    }
}
```

Результаты работы программы:

Base -- Method1: base class field
Base -- Method1: derived class field
Derived -- Method2: base class field
Derived -- Method2: derived class field

Наследование от object

неявно

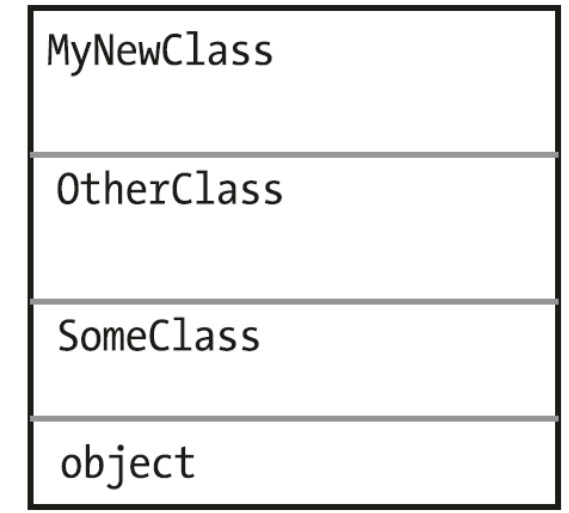
```
class SomeClass
{
    ...
}
```

явно

```
class SomeClass : object
{
    ...
}
```

Иерархия классов

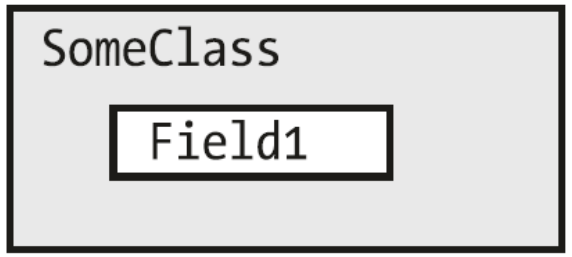
```
class SomeClass
{ ... }

class OtherClass: SomeClass
{ ... }

class MyNewClass: OtherClass
{
      ...
}
```

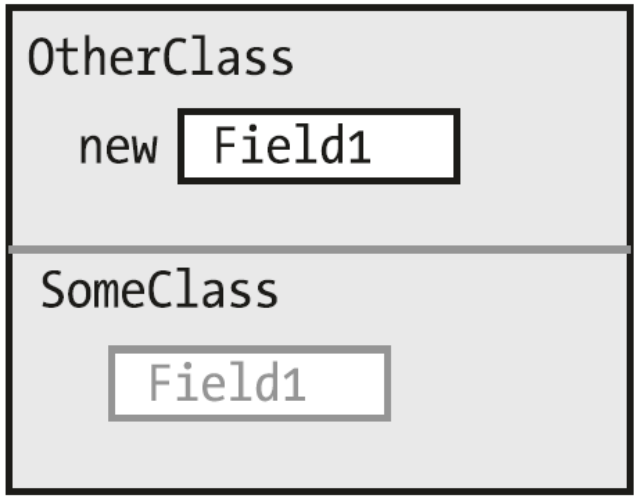


Экранирование (hiding) членов базового класса

```
class SomeClass { // базовый класс
    string Field1;
    //...
}

class OtherClass : SomeClass { // производный класс
          new string Field1;  // маскируем член базового класса
}
```

↑ ключевое слово



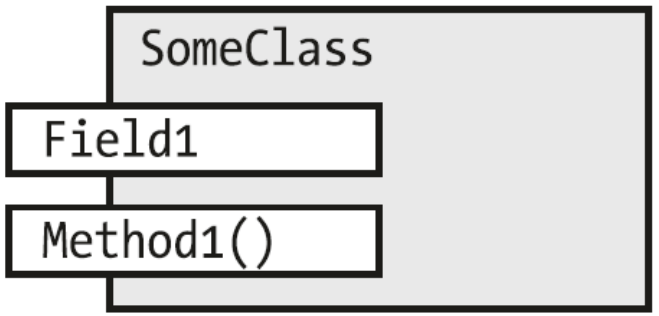


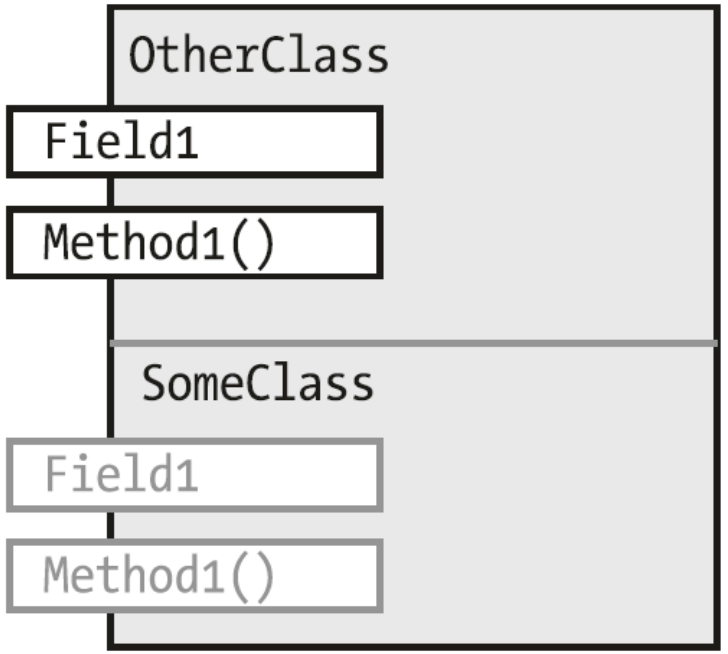
Экранирование двух членов

```
class SomeClass {     // базовый класс
    public string Field1 = "SomeClass Field1";
    public void Method1(string value)
    {
        Console.WriteLine("SomeClass.Method1: {0}", value);
    }
}


class OtherClass : SomeClass {     // производный класс
    new public string Field1 = "OtherClass Field1"; // Mask ...
    new public void Method1(string value) // Mask ...
    {
        Console.WriteLine("OtherClass.Method1: {0}", value);
    }
}
```

Экранирование поля и метода базового класса

```
class Program
{
    static void Main()
    {
        OtherClass oc = new OtherClass();
        oc.Method1(oc.Field1); // Используем маскирующие члены
    }
}
// Результат:
// OtherClass.Method1: OtherClass Field1
```





Доступ к базовому классу

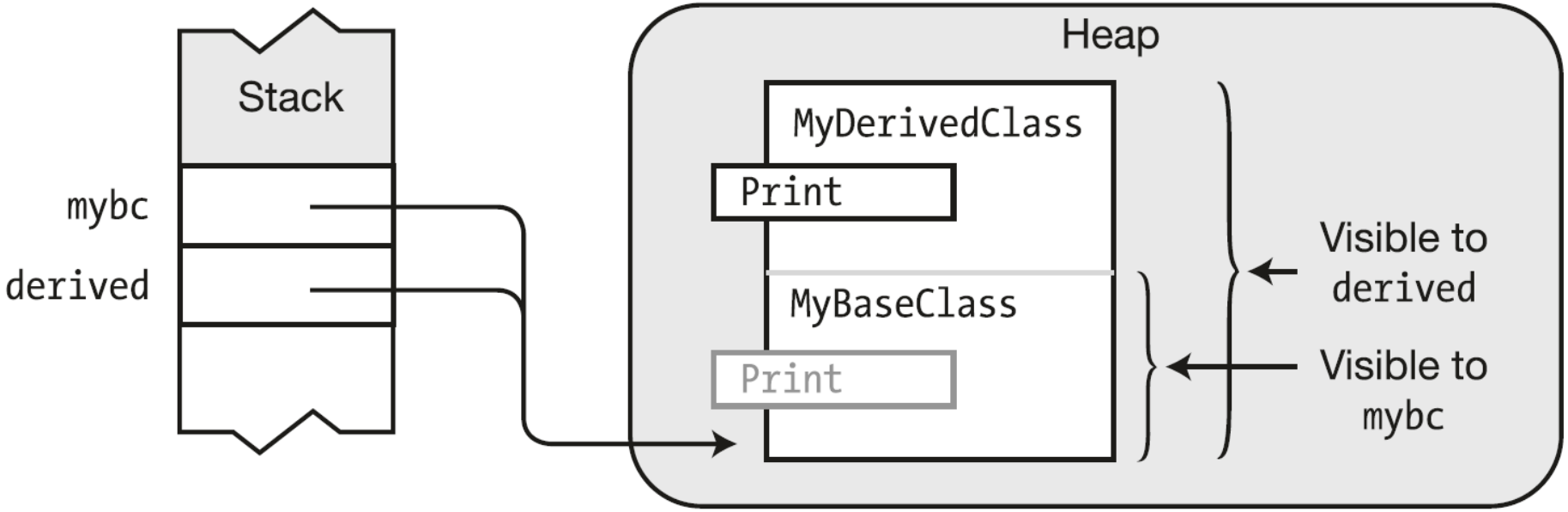
```
class SomeClass { // базовый класс
    public string Field1 = "Field1 -- In the base class";
}

class OtherClass : SomeClass { // производный класс
    new public string Field1 = "Field1 -- In the derived class";
    public void PrintField1()
    {
        Console.WriteLine(Field1); // доступ к производному классу
        Console.WriteLine(base.Field1); // доступ к базовому классу
    }
}
class Program {
    static void Main()
    {
        OtherClass oc = new OtherClass();
        oc.PrintField1();
    }
}
```

Результаты работы программы:

Field1 -- In the derived class
Field1 -- In the base class

Ссылки на базовые классы

```
MyDerivedClass derived = new MyDerivedClass();
MyBaseClass mybc = (MyBaseClass) derived;
```



Ссылки на базовые классы (пример)

```
class MyBaseClass {
    public void Print() {
        Console.WriteLine("This is the base class.");
    }
}
class MyDerivedClass : MyBaseClass {
    new public void Print() {
        Console.WriteLine("This is the derived class.");
    }
}
class Program {
    static void Main() {
        MyDerivedClass derived = new MyDerivedClass();
        MyBaseClass mybc = (MyBaseClass) derived; // необязательное приведение типа
        derived.Print(); // Print из производного класса
        mybc.Print();   // Print из базового класса
    }
}
```

Результаты работы программы:
This is the derived class.
This is the base class.

Обращения к членам базового класса

```
using System;
class C1 {
    protected void Print()
    {  Console.WriteLine("C1"); }
}
class C2 : C1  {
    new protected void Print()
    {  base.Print();
       Console.WriteLine("C2");
    }
}
class C3 : C2  {
    new public void Print()
    {   Console.WriteLine("C3");
        base.Print();
    }
}
```

```
class Program   {
  static void Main( ) {
     C3 ob3 = new C3();
     ob3.Print();
  }
}
```

Результаты:

C3
C1
C2

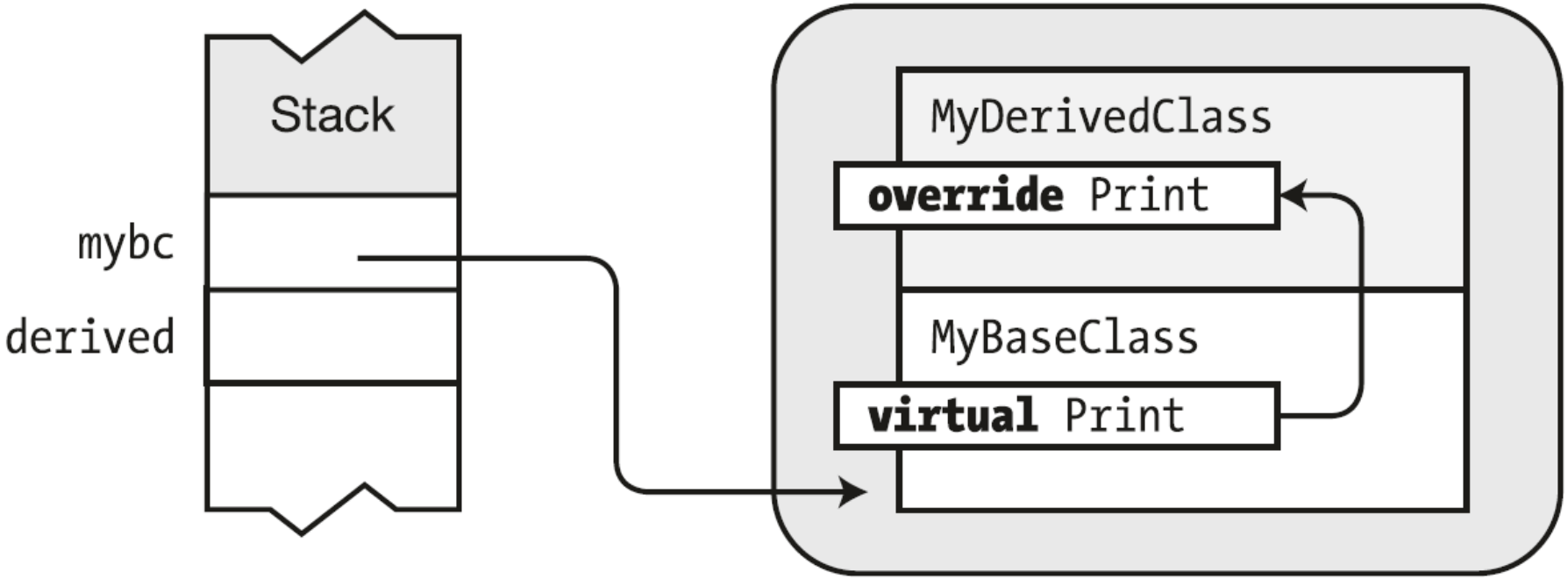
Использование virtual и override в методах

```
class MyBaseClass   // базовый класс
{
    virtual public void Print()
    { ... }
}


class MyDerivedClass : MyBaseClass // производный класс
{
    override public void Print()
    { ... }
}
```

Виртуальность допустима

- Метод
- Свойство
- Индексатор
- Событие

Виртуальный и переопределенный метод



Пример с виртуальным методом

```
class MyBaseClass {
    virtual public void Print() {
        Console.WriteLine("This is the base class.");
    }
}

class MyDerivedClass : MyBaseClass {
    override public void Print() {
        Console.WriteLine("This is the derived class.");
    }
}


MyDerivedClass derived = new MyDerivedClass();
MyBaseClass mybc = (MyBaseClass) derived;
derived.Print();  // вызов Print из производного класса
mybc.Print();     // вызов Print из производного класса
```

Переопределение метода, помеченного override

```
class MyBaseClass {                                  // базовый класс
    virtual public void Print()
    { Console.WriteLine("This is the base class."); }
}

class MyDerivedClass : MyBaseClass {                 // производный класс
    override public void Print()
    { Console.WriteLine("This is the derived class."); }
}

class SecondDerived : MyDerivedClass {               // 2-производный класс
    //  Print() с модификатором override или new
}
```

Обращения к Print()

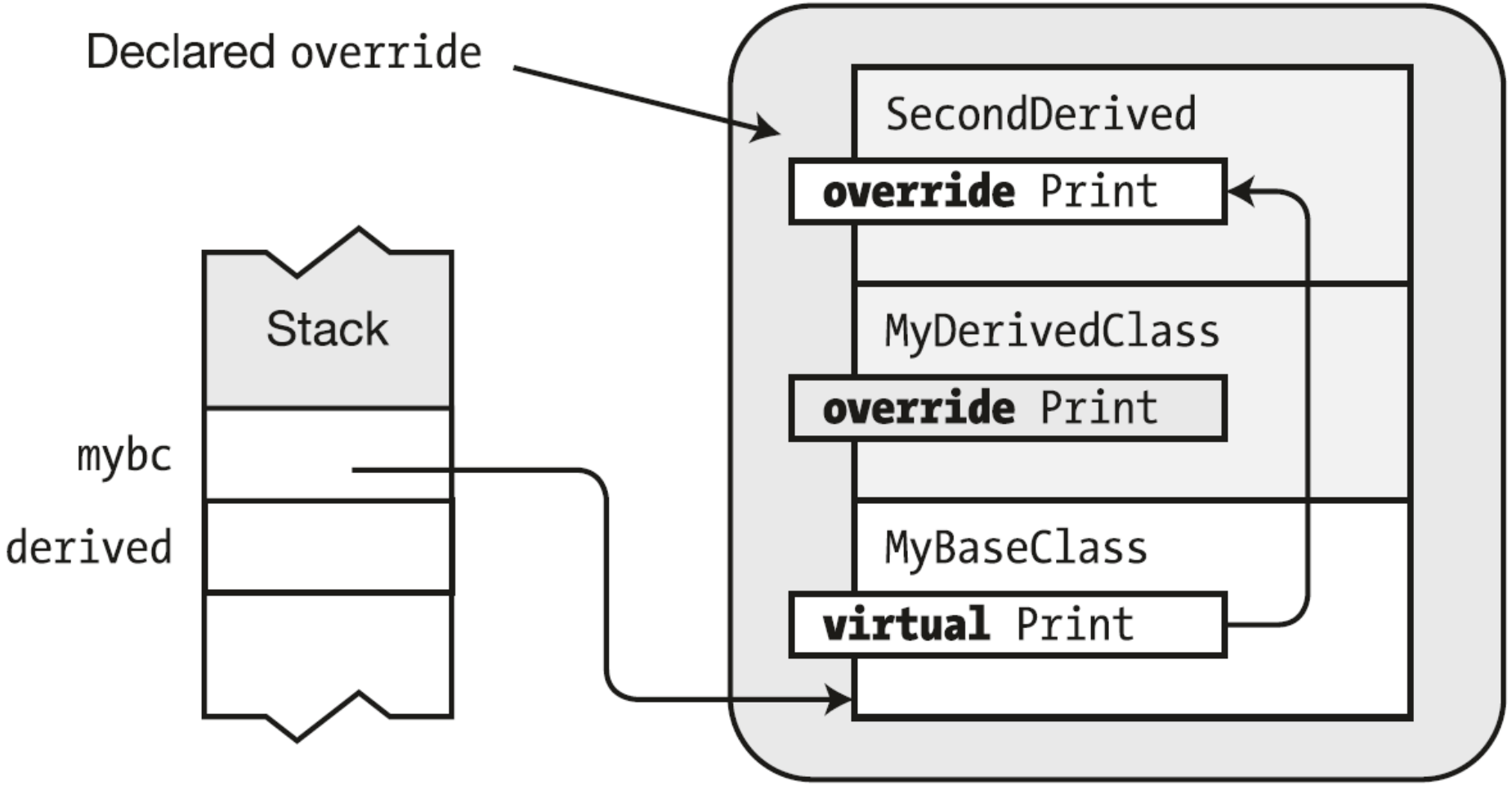
```
// создаем объект типа SecondDerived:
SecondDerived derived = new SecondDerived();

// Создаем ссылку типа MyBaseClass:
MyBaseClass mybc = (MyBaseClass) derived;
mybc.Print();
```

В зависимости от **new** / **override** в классе SecondDerived:

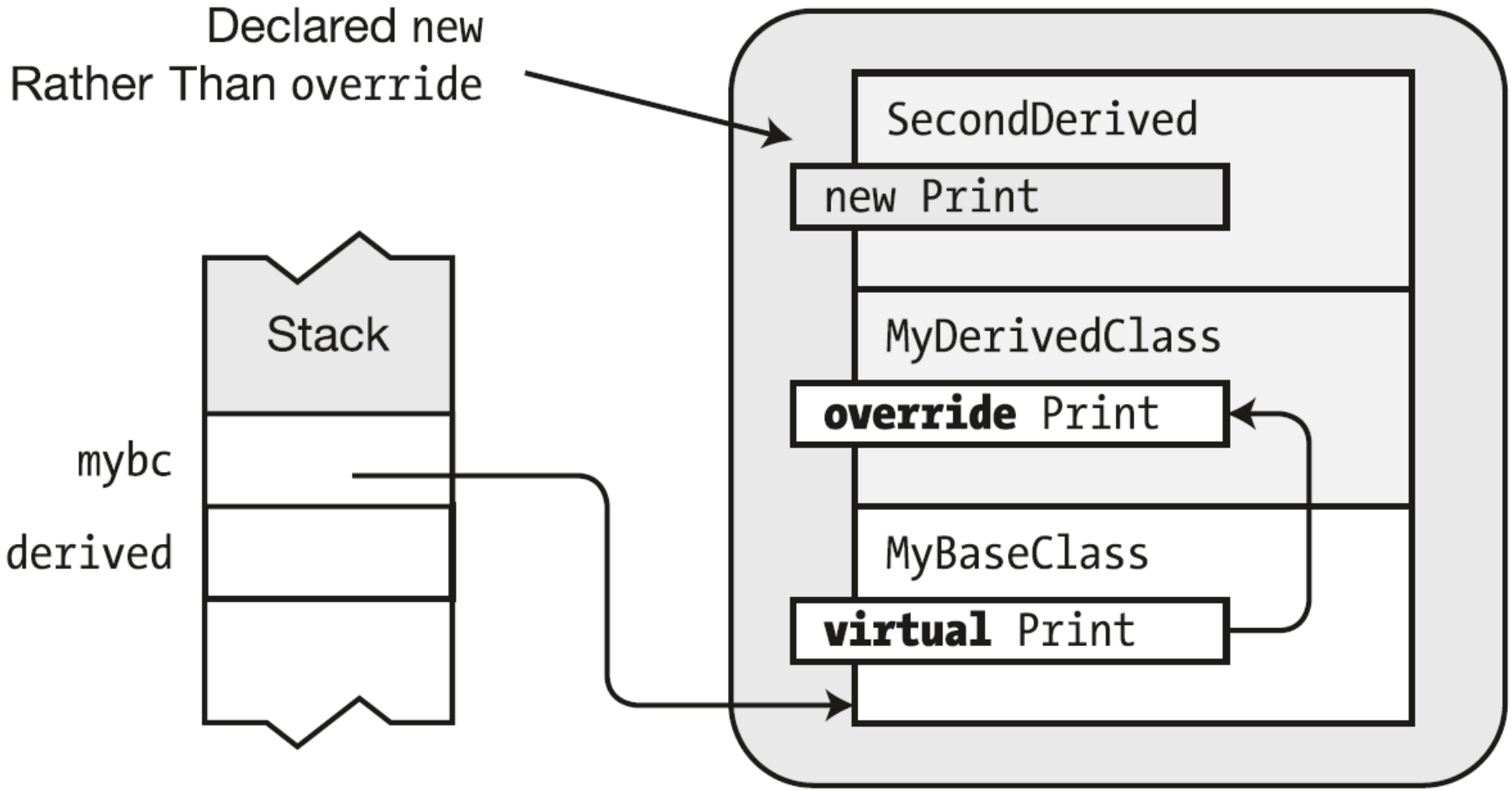
При **override** ➔ Print из **SecondDerived**

При **new** ➔ Print из **MyDerivedClass**

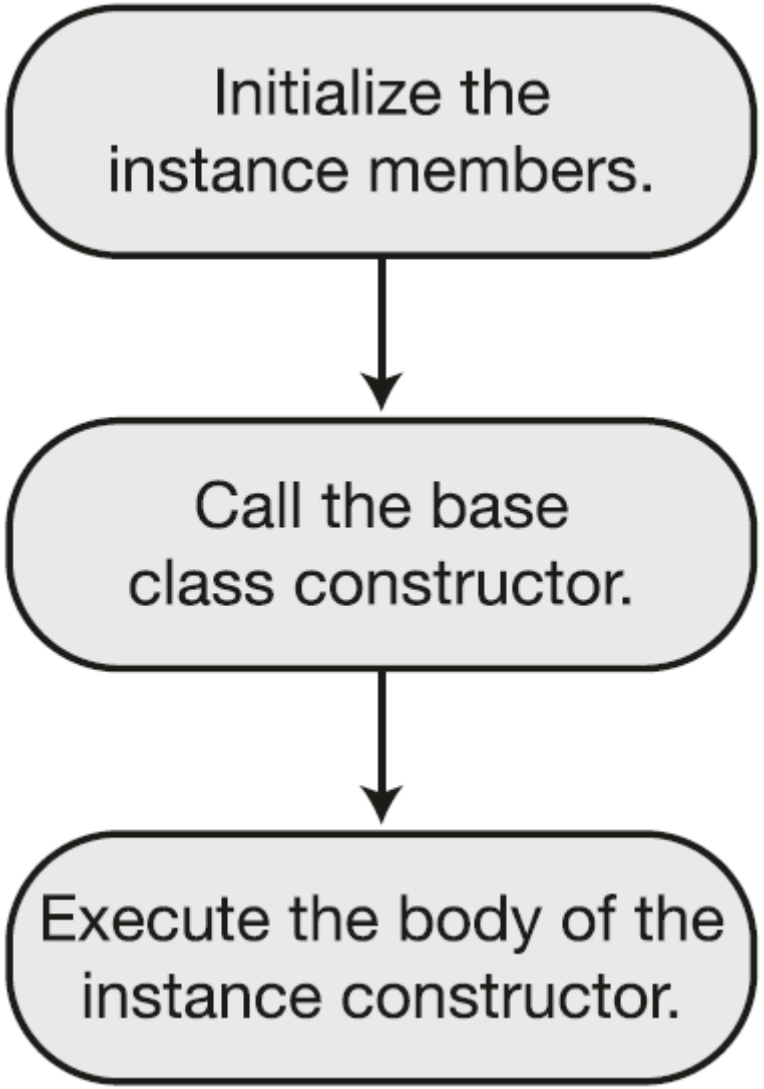
override public void Print()



new public void Print()



Выполнение конструктора



Инициализация членов объекта

Вызов конструктора базового класса

Выполнение тела конструктора

Этапы создания объекта

```
class MyDerivedClass : MyBaseClass    {
   int MyField1 = 5;              // 1. инициализация поля
   int MyField2;                  //    инициализация поля
   public MyDerivedClass() // 3. выполнение тела конструктора
   {      ...      }
}

class MyBaseClass          {
   public MyBaseClass()    // 2. выполнение конструктора базового класса
   {      ...      }
}
```

ВНИМАНИЕ: вызов виртуального метода в конструкторе **настоятельно** не рекомендуется!

Инициализатор в конструкторе

инициализатор конструктора

↓

```
public MyClass(int x) : this(x, "Using Default String")
{
```

↑

кл. слово

==========

инициализатор конструктора

↓

```
public MyDerivedClass( int x, string s ) : base( s, x )
{
```

↑

кл. слово

Эквивалентные объявления

```
class MyDerived: MyBase
{
   MyDerived()
   {
      ...
   }
   ...
```

Constructor Implicitly Using Base Constructor `MyBase()`

```
class MyDerived: MyBase
{
   MyDerived() : base()
   {
      ...
   }
   ...
```

Constructor Explicitly Using Base Constructor `MyBase()`

Модификаторы доступа к классам

кл. слово

↓

public class MyBaseClass

{ ...

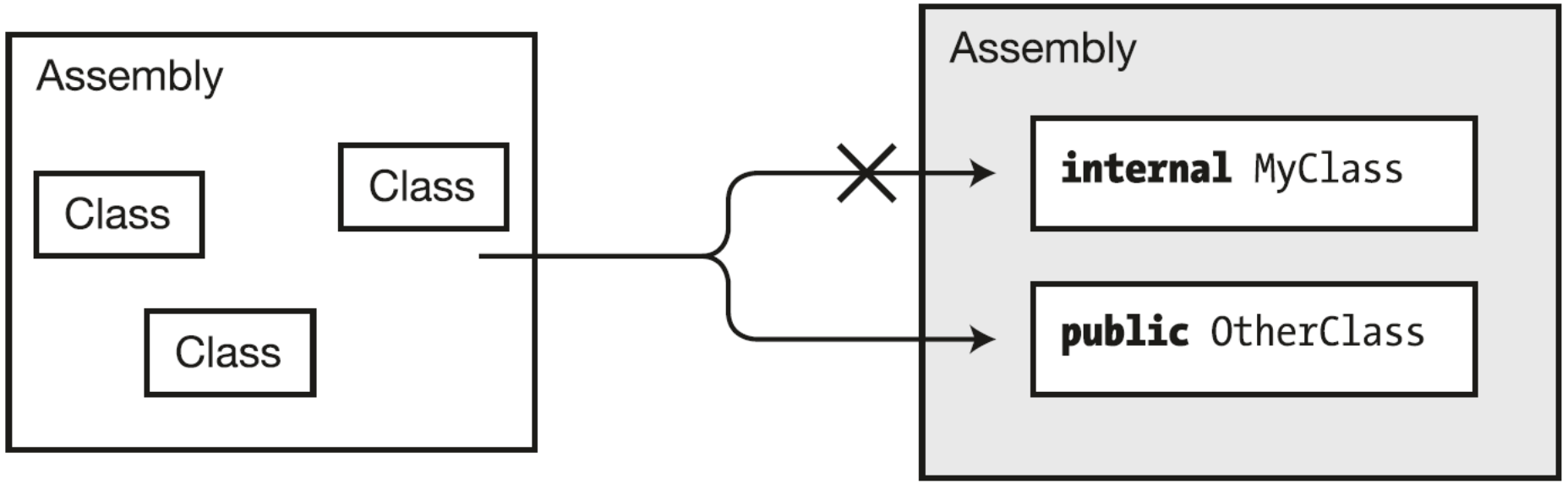
кл. слово

↓

internal class MyBaseClass

{ ...

Модификаторы доступа к классам (2)



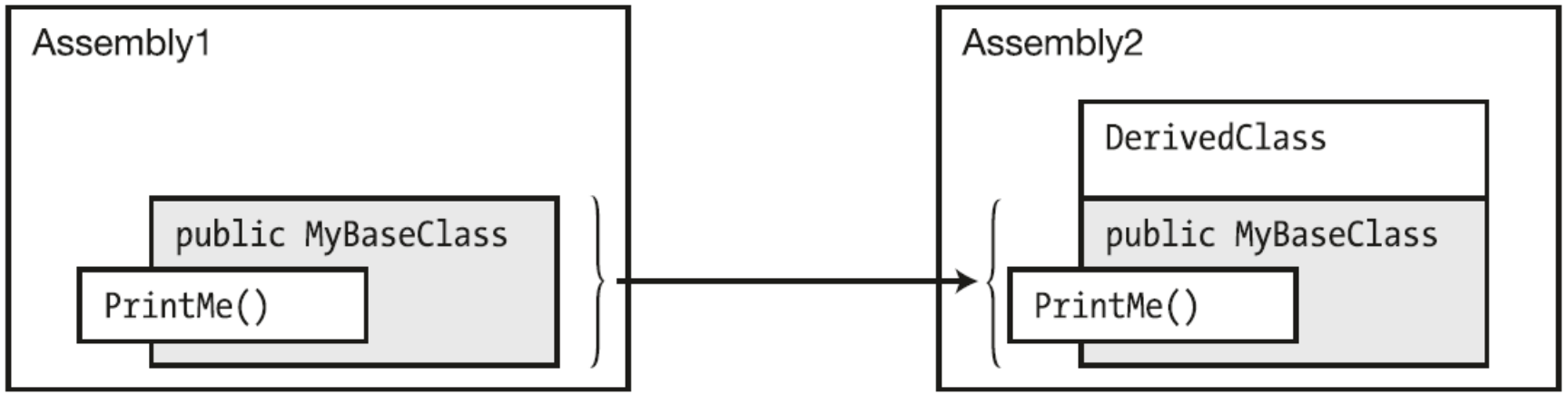
Наследование между сборками (Inheritance Between Assemblies)

```
// Assembly1.cs
namespace BaseClassNS {

    public class MyBaseClass {
        public void PrintMe() {
            Console.WriteLine("I am MyBaseClass");
        }
    }

}
```

Наследование между сборками 2

```
// Assembly2.cs
using BaseClassNS;
namespace UsesBaseClass       {
    class DerivedClass: MyBaseClass {
        // пустое тело
    }
    class Program {
        static void Main( )    {
                DerivedClass mdc = new DerivedClass();
                mdc.PrintMe();
        }
    }

}
```

Наследование между сборками - 3



MyBaseClass is declared in Assembly1.

DerivedClass derives from MyBaseClass in Assembly1.

Модификаторы доступа к членам типа

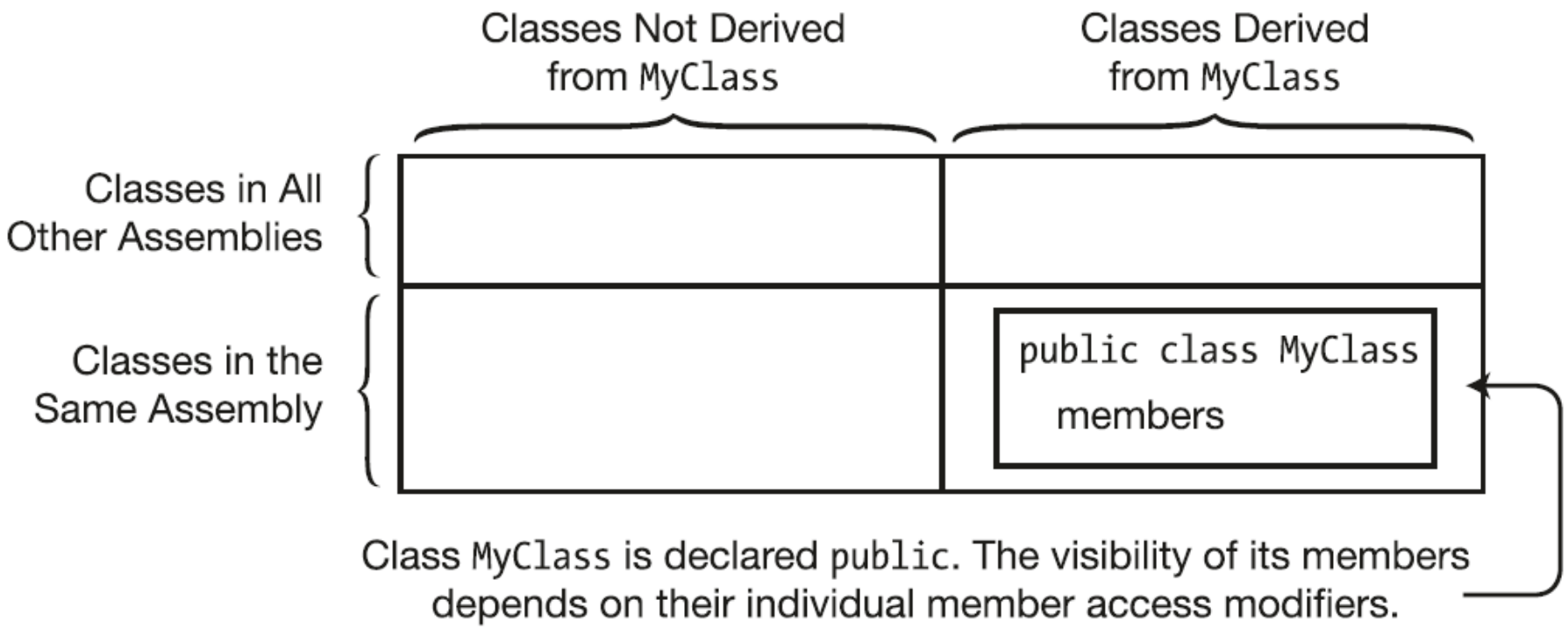
Существует 6 уровней доступа:

- public
- private
- protected
- internal
- protected internal
- private protected (C# 7.2)

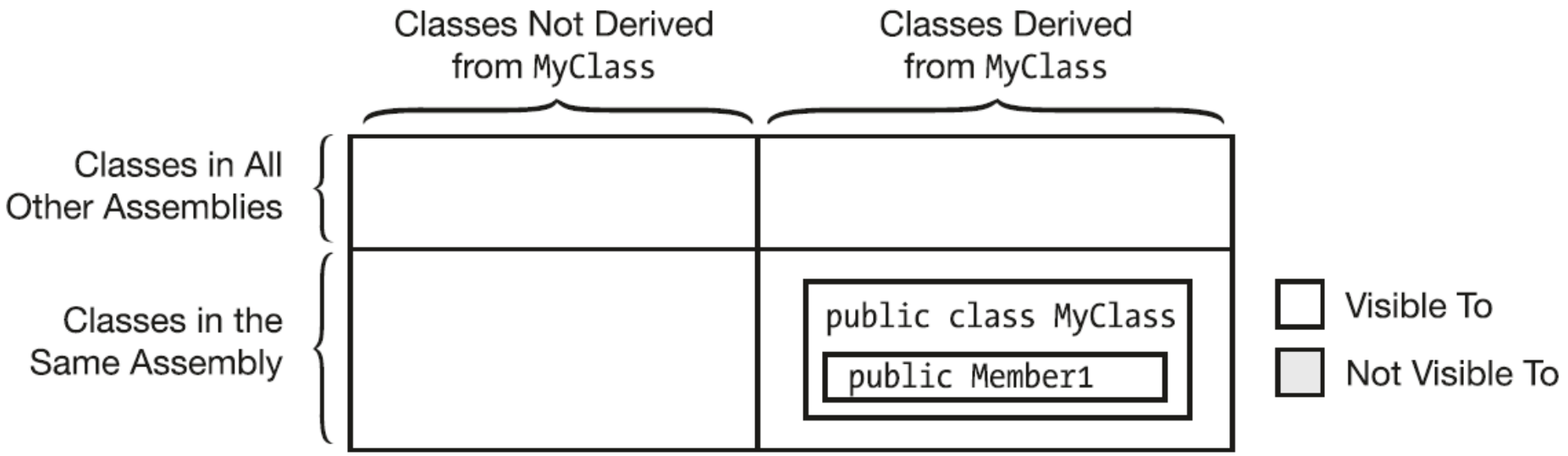
Уровни доступности членов класса

```
public class MyClass        {
    public                  int Member1;
    private                 int Member2;
    protected               int Member3;
    internal                int Member4;
    protected internal      int Member5;
    private protected       int Member6;
    ...

}
```

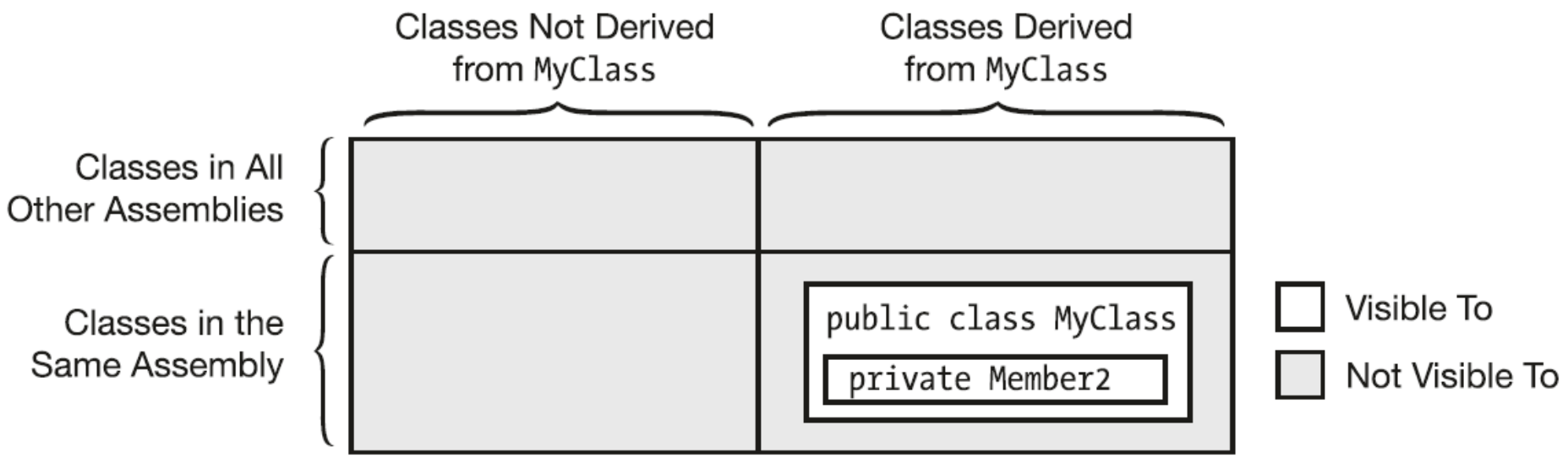
Области доступности



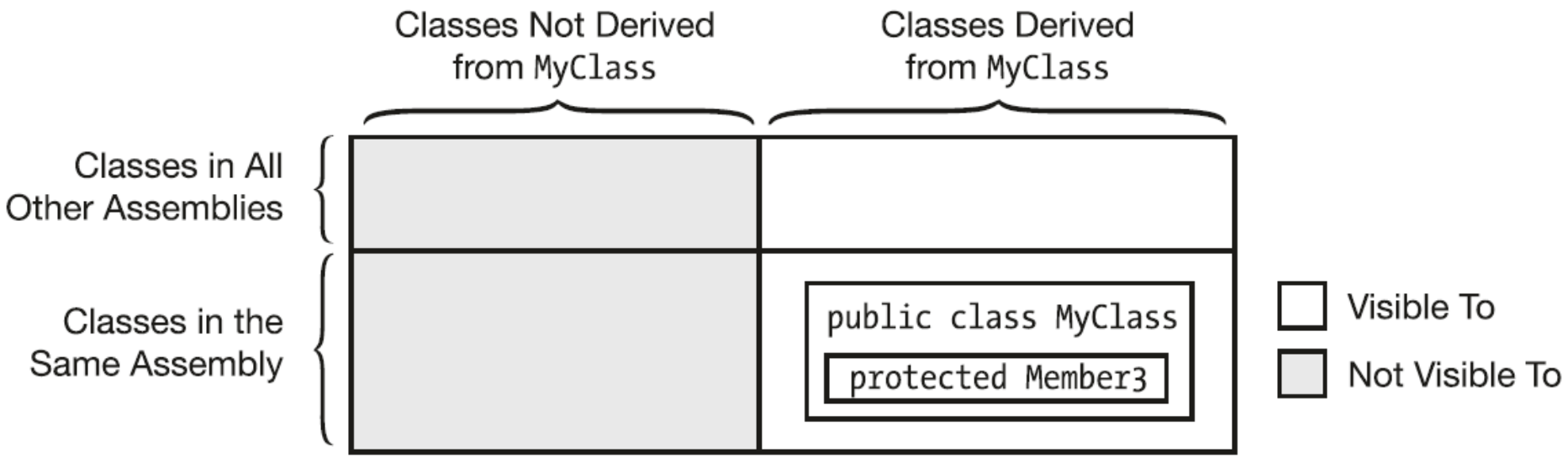
Доступность членов класса с public



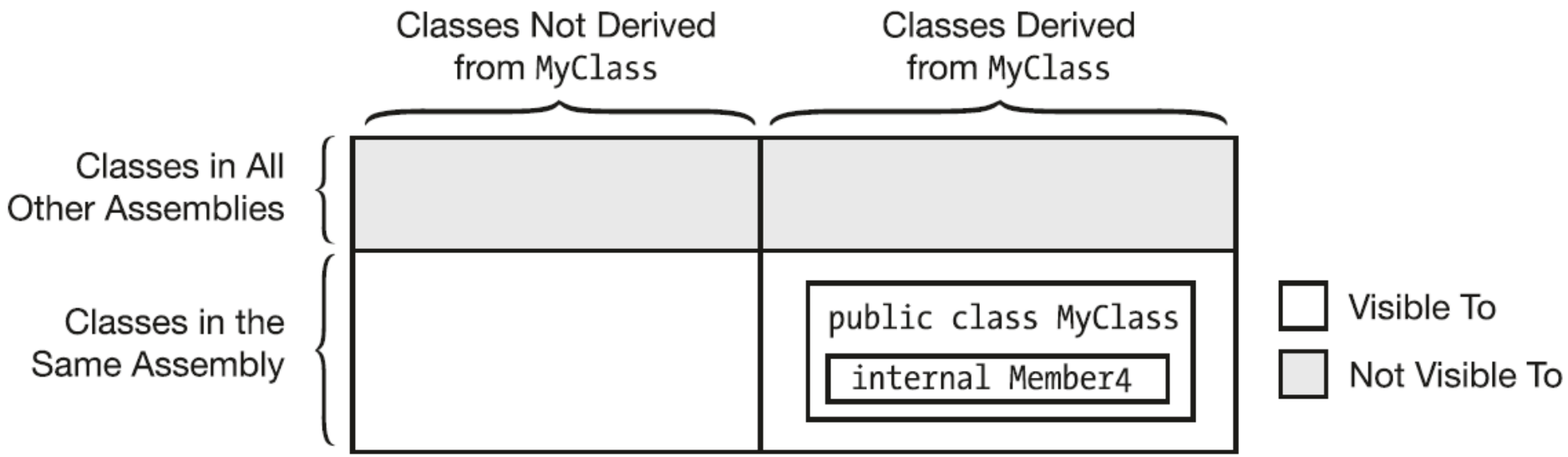
Доступность членов класса с private



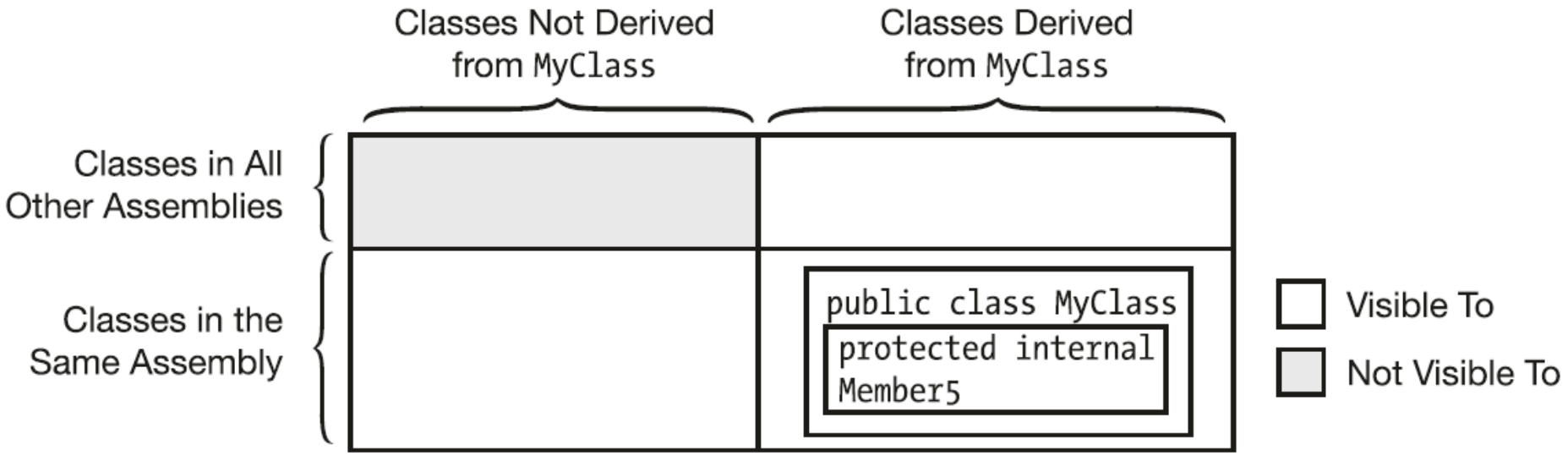
Доступность членов класса с protected



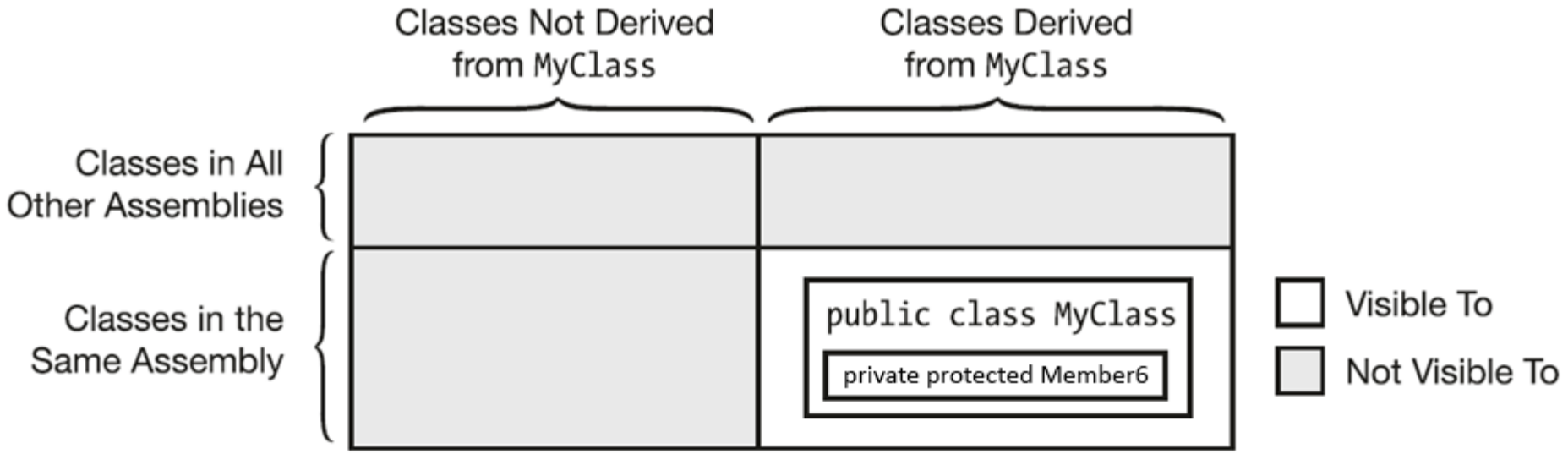
Доступность членов класса с internal



Доступность членов класса с protected internal



Доступность членов класса с private protected



Модификаторы доступа к членам типа

Modifier	Meaning
private	Доступность только внутри типа.
internal	Доступность для всех типов в рамках текущей сборки.
protected	Доступность для всех производных (от данного) классов.
protected internal	Доступность для всех производных классов **или** типов из текущей сборки.
private protected	Доступность для всех классов, объявленных производными **и** в текущей сборке.
public	Доступность для всех типов.

Модификаторы доступа к членам типа

	Классы в одной сборке		Классы в другой сборке	
	Non-Derived	Derived	Non-Derived	Derived
private				
internal	✓	✓		
protected		✓		✓
protected internal	✓	✓		✓
private protected		✓		
public	✓	✓	✓	✓

Абстрактные члены

кл. слово точка с запятой вместо реализации

↓ ↓

```
abstract public void PrintStuff(string s);
abstract public int MyProperty  {
    get; ← точка с запятой вместо реализации
    set; ← точка с запятой вместо реализации
}
```

Виртуальные и абстрактные члены

	Виртуальный член	**Абстрактный член**
кл. слово	**virtual**	**abstract**
Тело с реализацией	есть тело с реализацией	нет тела с реализацией – точка с запятой
Переопр. в произв. классе	***Может*** быть переоп-реределен с override	***Обязан*** быть переопределен с использованием override
Типы членов	Методы Свойства События Индексаторы	Методы Свойства События Индексаторы

Абстрактные классы

кл. слово

↓

```
abstract class MyClass  {
   ...
}
```

Наследование с участием абстрактного класса

```
abstract class AbClass                          // абстрактный класс
{
    ...
}
abstract class MyAbClass : AbClass      // абстрактный класс
                                                // унаследованный от
{                                               // абстрактного класса
    ...
}
```

Опечатанные (sealed) классы

кл. слово

↓

```
sealed class MyClass
{
    ...
}
```

Методы расширения (extension methods)

```
// Дано: внешний класс, который мы не можем изменить
sealed class MyData        {
    private double D1;                    // поля
    private double D2;
    private double D3;
    // конструктор
    public MyData(double d1, double d2, double d3)
    {     D1 = d1; D2 = d2; D3 = d3; }
    // метод
    public double Sum()
    {     return D1 + D2 + D3; }
}
```

Методы расширения (extension methods)

Расширение функциональности (вариант 1)

```
static class ExtendMyData
{
    public static double Average( MyData md )
    { return md.Sum() / 3; }
}
class Program   {
    static void Main()  {
        MyData md = new MyData(3, 4, 5);
        Console.WriteLine("Average: {0}", ExtendMyData.Average(md));
    }
}
```

ссылка на объект типа MyData

используем ссылку на объект MyData

объект типа MyData

вызов статического метода

Желательно иметь такое обращение: **md.Average();**

Метод расширения (рецепт приготовления)

Обязательно static!

↓

```
static class ExtendMyData
{ Обязательно public и static!     Ключевое слово
          ↓                              ↓
    public static double Average( this MyData md )
    {     ...       }
}
```

Использование расширяющего метода:

```
MyData md = new MyData(3, 4, 5);
Console.WriteLine("Average: {0}", md.Average());
Console.Write("Average: {0}", ExtendMyData.Average(md));
```

Схема методов расширения