

В.В. Подбельский

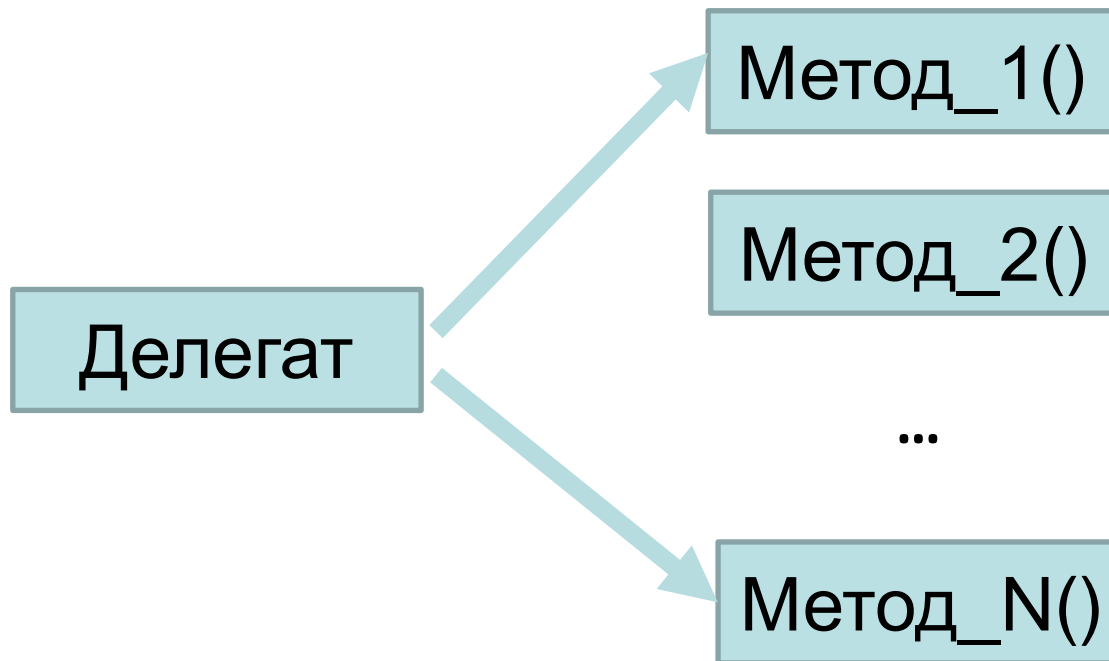
Иллюстрации к курсу лекций
по дисциплине
«Программирование»
C#
Делегаты (Delegates)

*Использованы материалы
Daniel Solis, Cal Schrottenboer Illustrated C# 7*

Что такое делегат?

Делегат – тип, объект которого может ссылаться на метод.

Делегат – это объект, которому известно, как вызывать метод.



Еще раз о сигнатурах

СИГНАТУРА МЕТОДА



Перегрузка методов

Включает:

- имя метода;
- типы параметров (с модификаторами).

Не включает:

- имена параметров;
- возвращаемый тип.

Делегаты

Включает:

- возвращаемый тип;
- типы параметров (с модификаторами).

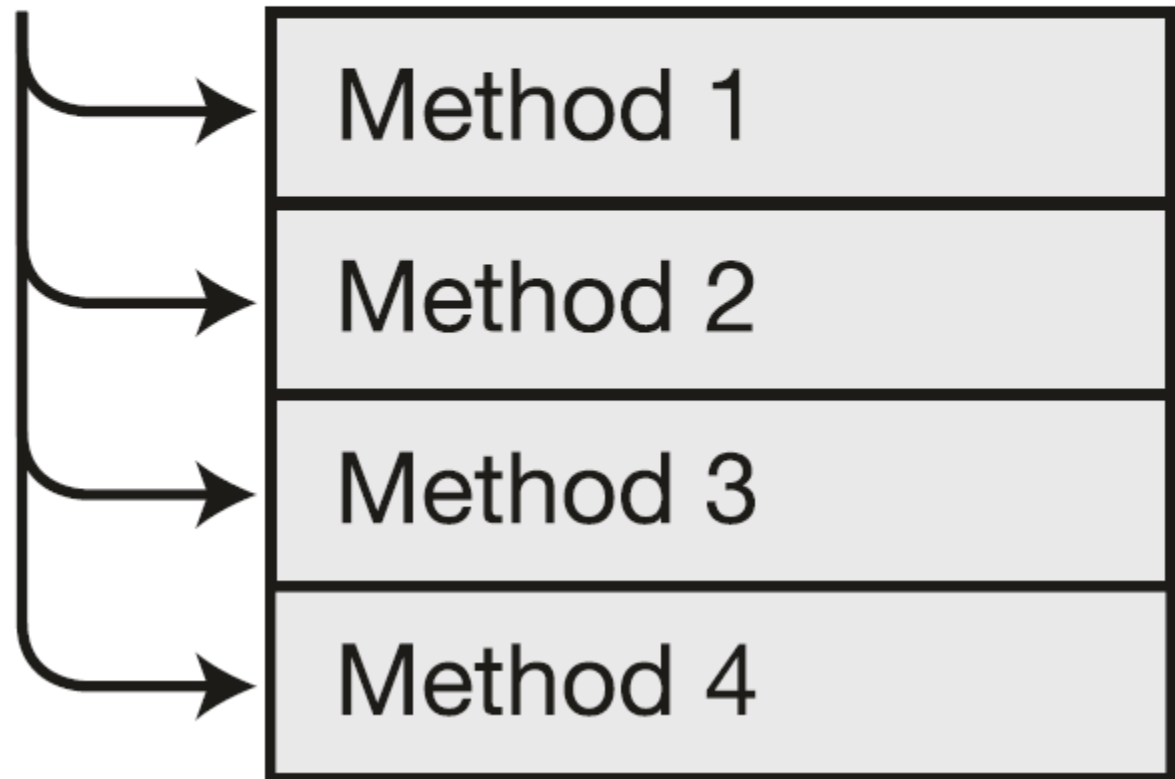
Не включает:

- имена параметров;
- имя метода.

Что такое делегат? Список вызова

Delegate

Invocation List

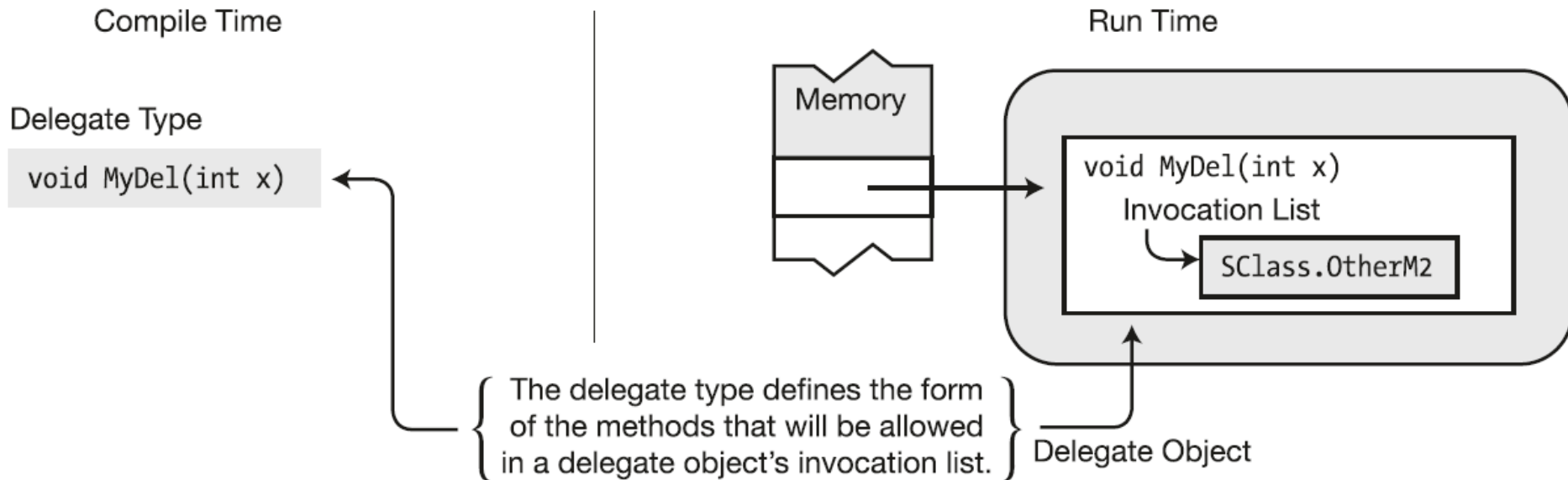


Объявление делегата-типа

Кл. слово **Имя типа делегата**

↓ ↓
delegate void MyDel (int x);

Делегат-тип и экземпляр делегата



Создание экземпляра делегата

Объявление ссылки:

Тип делегата **переменная**

↓
MyDel

↓
delVar;

Создание экземпляров делегата:

Метод объекта



```
delVar = new MyDel( myInstObj.MyM1 );  
dVar = new MyDel( SClass.OtherM2 );
```

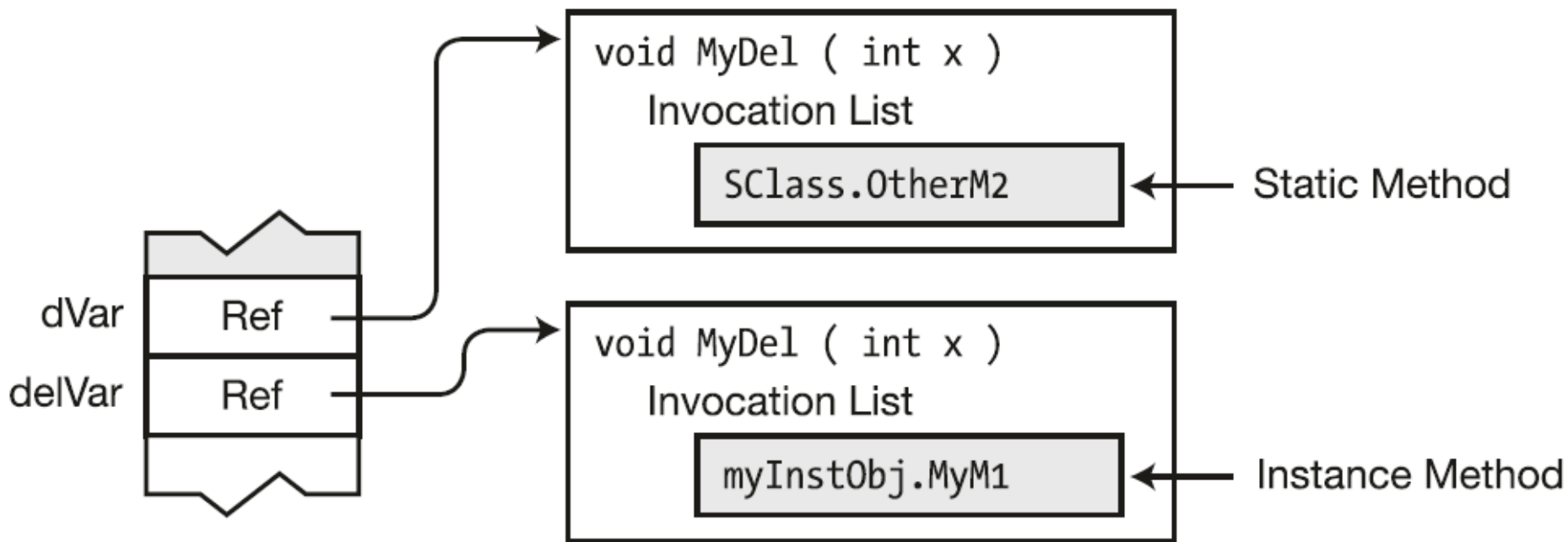


Статический метод

Упрощенный синтаксис:

```
delVar = myInstObj.MyM1; // создание делегата и сохранение ссылки  
dVar = SClass.OtherM2; // создание делегата и сохранение ссылки
```

Создание экземпляра делегата (схема)



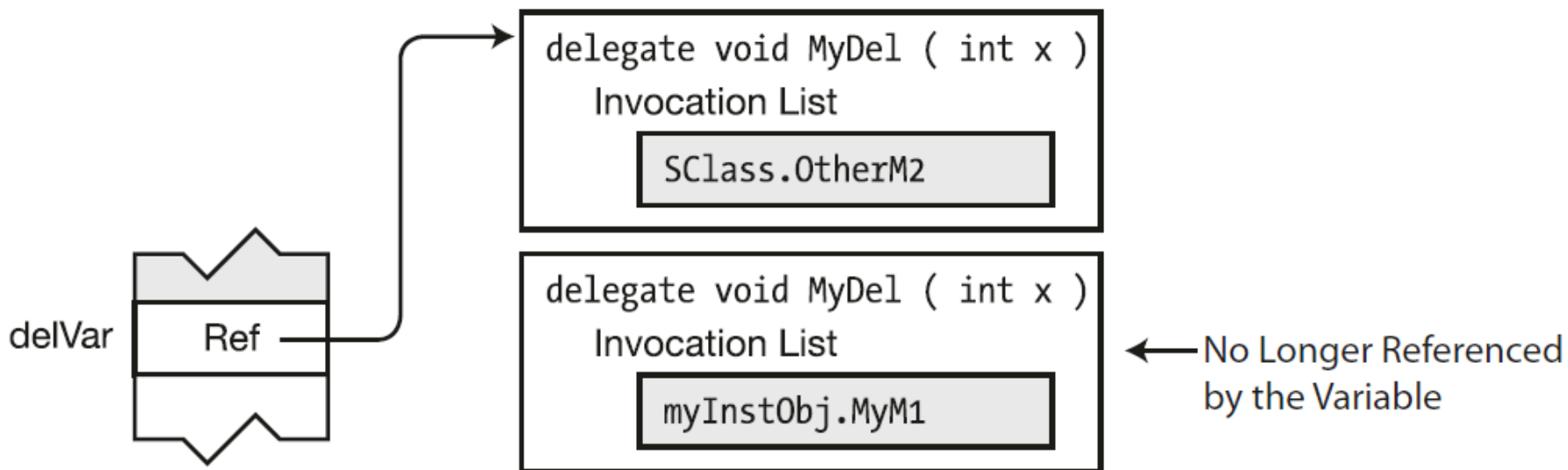
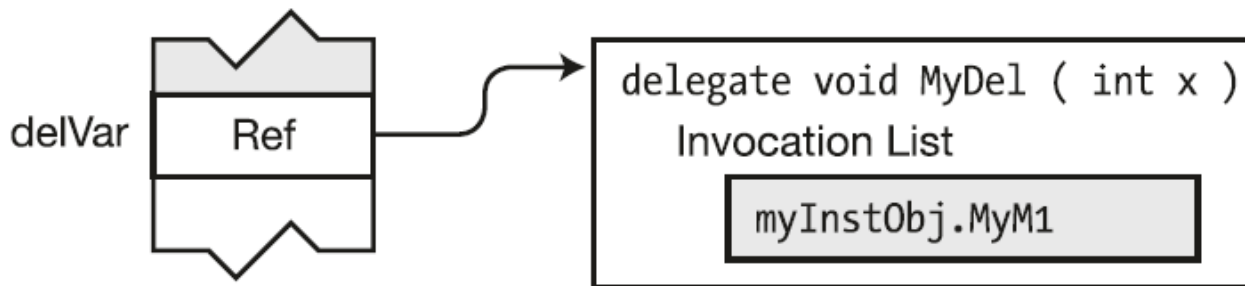
Присваивание делегатов

MyDel delVar;

delVar = myInstObj.MyM1; // создание делегата и присваивание ссылки

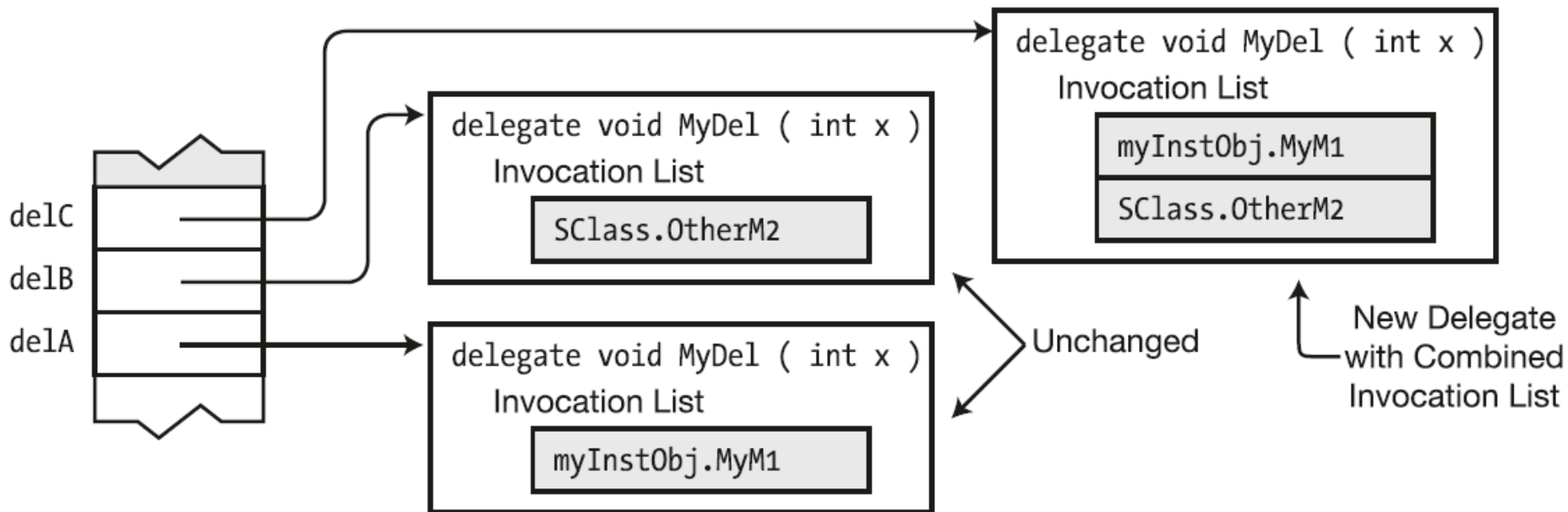
...

delVar = SClass.OtherM2; // создание нового объекта делегата и присв.



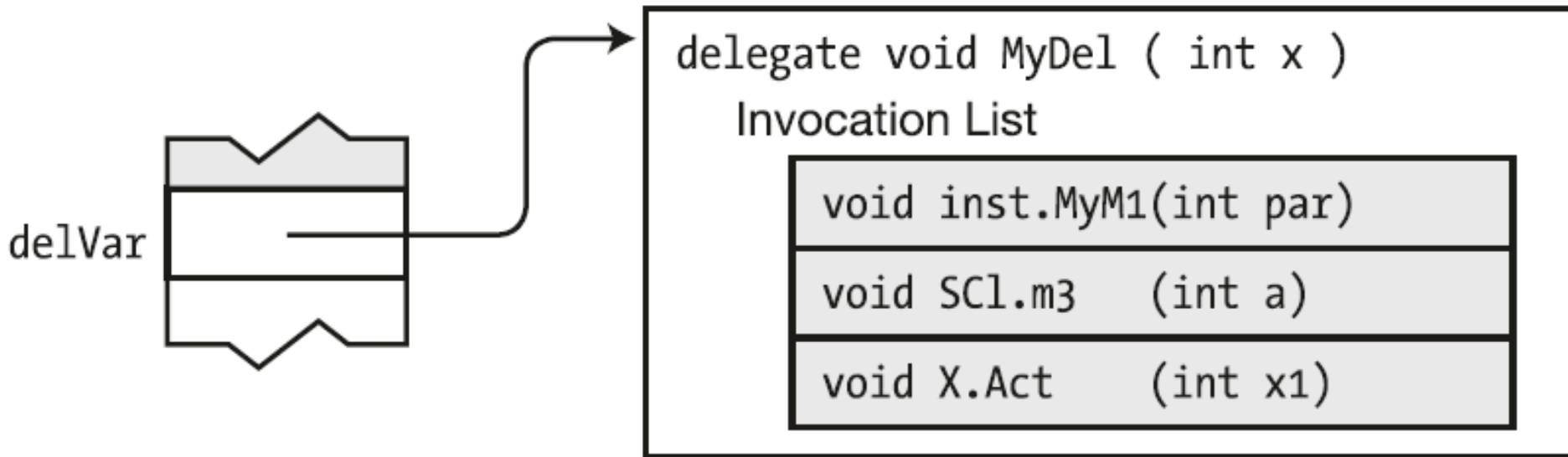
Объединение делегатов

```
MyDel delA = myInstObj.MyM1;  
MyDel delB = SClass.OtherM2;  
MyDel delC = delA + delB;    // объединение списка вызовов
```



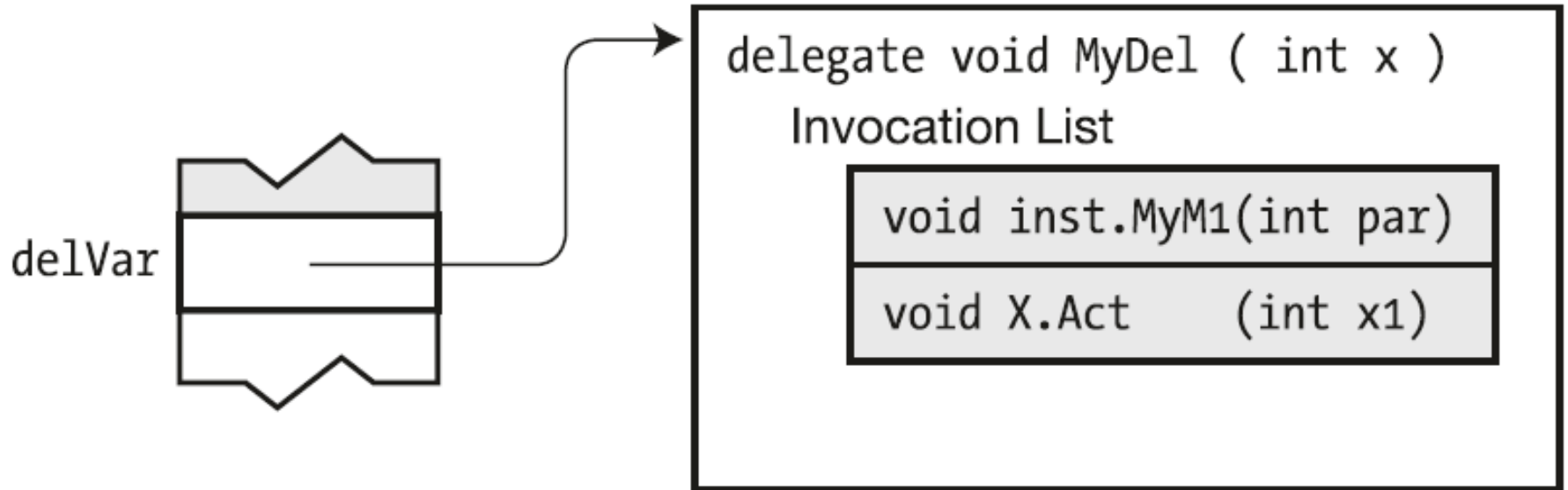
Добавляем методы в делегаты

```
MyDel delVar = inst.MyM1; // создание и инициализация  
delVar += SCl.m3; // добавление метода  
delVar += X.Act; // добавление метода
```



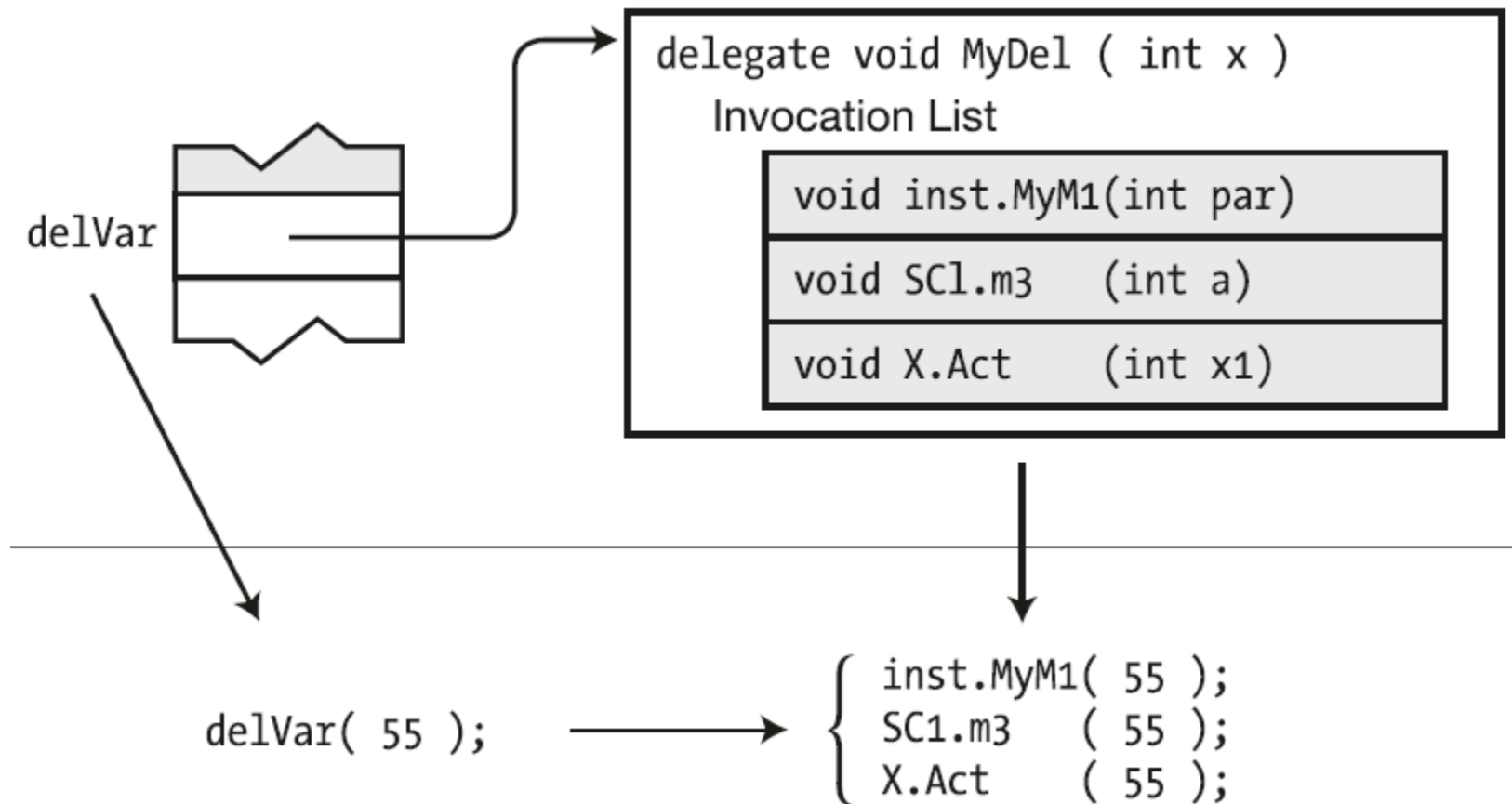
Удаляем методы из делегатов

`delVar -= SCl.m3; // удаление метода из списка вызовов`



Вызов делегата (Invocation)

```
MyDel delVar = inst.MyM1;  
delVar += SC1.m3;  
delVar += X.Act;  
delVar( 55 ); // вызов делегата
```



Пример – делегат и класс

// Определить тип делегата без возвращаемого значения и
// без параметров

```
delegate void PrintFunction();
```

```
class Test
```

```
{
```

```
    public void Print1()
```

```
    {
```

```
        Console.WriteLine("Print1 -- instance");
```

```
    }
```

```
    public static void Print2()
```

```
    {
```

```
        Console.WriteLine("Print2 -- static");
```

```
    }
```

```
}
```

Применение делегата

```
class Program    {
    static void Main()    {
        Test t = new Test(); // создаем объект
        PrintFunction pf;   // создаем переменную-делегат
        pf = t.Print1;      // инициализируем переменную
        // добавляем три метода в список вызова делегата
        pf += Test.Print2;
        pf += t.Print1;
        pf += Test.Print2;
        // список вызова делегата содержит четыре метода
        if ( null != pf )    // убеждаемся в наличии методов
            pf();            // вызов делегата
        else
            Console.WriteLine("Delegate is empty");
    }
}
```

Обращение к делегатам, возвращающим значение

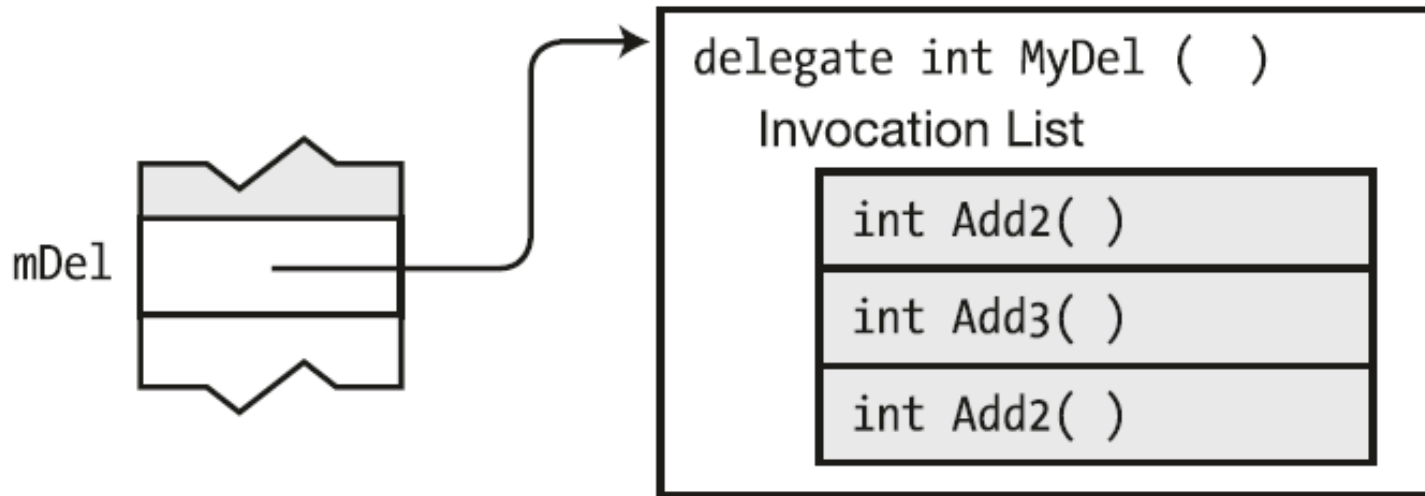
```
delegate int MyDel( ); // метод с типом возврата  
class MyClass    {  
    int IntValue = 5;  
    public int Add2() { IntValue += 2; return IntValue; }  
    public int Add3() { IntValue += 3; return IntValue; }  
}
```


Обращение к делегатам, возвращающим значение

```
class Program    {  
    static void Main( )    {  
        MyClass mc = new MyClass();  
        MyDel mDel = mc.Add2; // создаем и инициализируем  
        mDel += mc.Add3;      // добавляем метод  
        mDel += mc.Add2;      // добавляем метод  
        // вызов делегата и использование результата:  
        Console.WriteLine("Value: {0}", mDel ( ) );  
    }  
}
```

Результат работы программы:
Value: 12

Схема к примеру

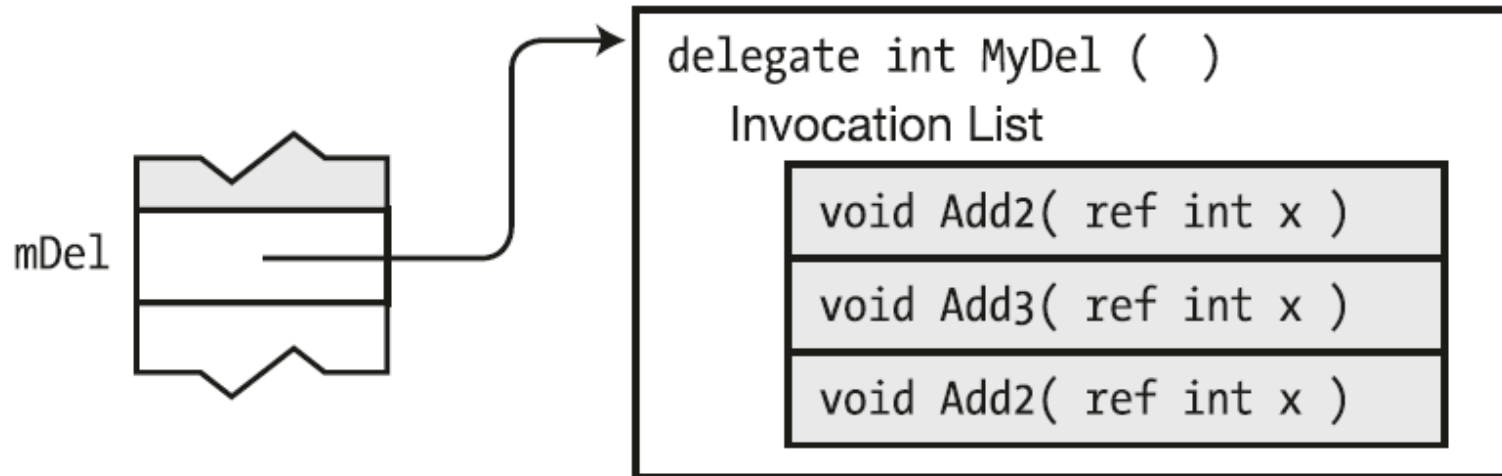


mDel(); \longrightarrow $\left\{ \begin{array}{ll} \text{Add2();} & \longleftarrow \text{Return value 7 is ignored.} \\ \text{Add3();} & \longleftarrow \text{Return value 10 is ignored.} \\ \text{Add2();} & \longleftarrow \text{Return value 12 is used.} \end{array} \right.$

Обращение к делегату с параметрами, передаваемыми по ссылке

```
delegate void MyDel ( ref int X );  
class MyClass      {  
    public void Add2(ref int x) { x += 2; }  
    public void Add3(ref int x) { x += 3; }  
    static void Main()    {  
        MyClass mc = new MyClass();  
        MyDel mDel = mc.Add2;  
        mDel += mc.Add3;  
        mDel += mc.Add2;  
        int x = 5;  
        mDel(ref x);  
        Console.WriteLine("Value: {0}", x);  
    }  
}
```

Схема к примеру



mDel(); → { Add2(x = 5); ← Initial Value of ref x
Add3(x = 7); ← New Input Value of ref x
Add2(x = 10); ← New Input Value of ref x

Два свойства делегатов: Method и Target

```
public delegate int[ ] Row(int num); // делегат-тип
public class Example {
    // Метод возвращает массив цифр целого числа-параметра:
    static public int[ ] Series(int num) { /* ... */ }
}

class Program {
    static void Main() {
        Row delRow; // Ссылка на делегат
        delRow = new Row(Example.Series); // Экземпляр делегата
        .....
        Console.WriteLine("delRow is equals {0}", delRow.Method);
        Console.WriteLine("delRow.Target is equals {0}", delRow.Target);
    }
}
```

Результат выполнения операторов:

```
delRow is equals Int32[ ] Series(Int32)
delRow.Target is equals
```

Массивы делегатов

```
delegate void Steps(); // делегат-тип

class Robot // класс для представления робота
{
    int x, y; // положение робота на плоскости

    public void Right() { x++; } // направо
    public void Left() { x--; } // налево
    public void Forward() { y++; } // вперед
    public void Backward() { y--; } // назад

    public static void Stub() { } // заглушка

    public void Position() // сообщить координаты
    {
        Console.WriteLine("The Robot position: x={0}, y={1}", x, y);
    }
}
```

Массивы делегатов

```
static void Main02()
{
    Robot rob = new Robot(); // конкретный робот
    Steps[] trace = { new Steps(rob.Backward), new
        Steps(rob.Backward), new Steps(rob.Left) };

    for (int i = 0; i < trace.Length; i++)
    {
        Console.WriteLine("Method={0},Target={1}", trace[i].Method,
trace[i].Target);
        trace[i]();
    }
    rob.Position(); // сообщить координаты
}
```

Результат выполнения программы:

Method=Void Backward(), Target=Robot

Method=Void Backward(), Target=Robot

Method=Void Left(), Target=Robot

The Robot position: x=-1, y=-2

Анонимные методы (C# 2.0)

```
class Program
{
    public static int Add20(int x)
    {
        return x + 20;
    }

    delegate int OtherDel(int InParam);
    static void Main()
    {
        OtherDel del = Add20;

        Console.WriteLine("{0}", del(5));
        Console.WriteLine("{0}", del(6));
    }
}
```

Named Method

```
class Program
{

    delegate int OtherDel(int InParam);
    static void Main()
    {
        OtherDel del = delegate(int x)
        {
            return x + 20;
        };

        Console.WriteLine("{0}", del(5));
        Console.WriteLine("{0}", del(6));
    }
}
```

Anonymous Method

Синтаксические правила анонимных методов

- служебное слово delegate;
- список параметров;
- блок операторов.

тип возврата для делегата



```
delegate int OtherDel(int InParam); // тип-делегат
```

```
static void Main()    {  
    OtherDel del = delegate(int x) // экземпляр делегата  
    {  
        return x + 20 ;           // возврат int  
    };  
    ...  
}
```

Пропуск списка параметров

```
delegate void SomeDel ( int X ); // объявление типа-делегата
```

```
SomeDel SDel = delegate // отсутствие ()  
{  
    PrintMessage();  
    Cleanup();  
};
```

Массивы-параметры params

params keyword used in delegate type declaration



```
delegate void SomeDel( int X, params int[] Y);
```

params keyword omitted in matching anonymous method

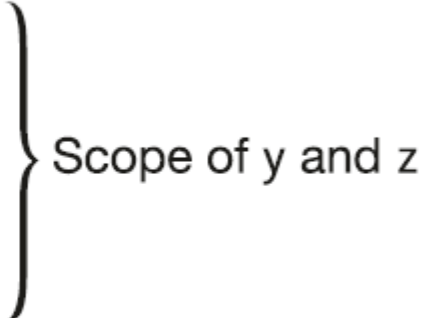


```
SomeDel mDel = delegate (int X, int[] Y)  
{  
...  
};
```

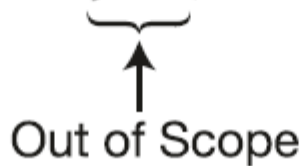
Область видимости переменных и параметров

```
delegate void MyDel( int x );  
...
```

```
MyDel mDel = delegate ( int y )  
{  
    int z = 10;  
    Console.WriteLine("{0}, {1}", y, z);  
};
```



```
Console.WriteLine("{0}, {1}", y, z); // Compile error.
```



Out of Scope

Внешние переменные

```
int x = 5;  
...
```

← Variable x is defined before the scope of the anonymous method.

```
MyDel mDel = delegate  
{  
    Console.WriteLine("{0}", x);  
};
```

↑
Using outer variable x.

} Variable x can be used inside the scope of the anonymous method.

Расширение времени жизни захваченных переменных

```
delegate void MyDel( );
static void Main()
{
    MyDel mDel;
    {
        int x = 5;
        mDel = delegate
        {
            Console.WriteLine("Value of x: {0}", x);
        };
    }
    // Console.WriteLine("Value of x: {0}", x);
    if (null != mDel)
        mDel( );
}
```

Variable x is defined inside the outer block and outside the anonymous method.

Scope of x

Variable x is captured by the anonymous method.

Variable x is out of scope here and would cause a compile error.

But x is used here, inside the anonymous method.

Лямбда выражения (C# 3.0)

// определим делегат-тип:

```
delegate int MyDel( int X );
```

// Анонимный метод в объявлении экземпляра делегата:

```
MyDel del = delegate(int x)    { return x + 1; } ;
```

// Лямбда-выражение:

```
MyDel le1 = (int x) => { return x + 1; } ;
```

Синтаксис лямбда-выражений

$$\left\{ \begin{array}{l} (\text{Parameter}, \text{Parameter}, \dots) \\ (\text{Parameter}) \\ \text{Parameter} \\ () \end{array} \right\} \Rightarrow \left\{ \begin{array}{l} \{ \text{Statements} \} \\ \text{Expression} \end{array} \right.$$

Лямбда-выражения в примерах

```
delegate int MyDel( int X );
```

// делегат-тип

```
MyDel del = delegate(int x) { return x + 1; } ;
```

// анонимный метод

```
MyDel le1 = (int x) => { return x + 1; } ;
```

// лямбда-выражение

```
MyDel le2 = (x) => { return x + 1; } ;
```

// лямбда-выражение

```
MyDel le3 = x => { return x + 1; } ;
```

// лямбда-выражение

```
MyDel le4 = x => x + 1 ;
```

// лямбда-выражение