В.В. Подбельский

Использованы иллюстрации пособия Daniel Solis, Illustrated C#

Иллюстрации к курсу лекций по дисциплине «Программирование на C#»

Модуль 3. Лекция 1

Делегаты

Благодарность Д.Д. Сагалову за помощь в переработке презентаций

Использование Методов в Качестве Параметров

При решении задач возникают случаи, когда удобным решением является **возможность передать метод в качестве входного параметра** другого метода.

Подумайте, как бы Вы могли реализовать подобное поведение средствами ООП? Какие недостатки такого подхода можно выделить?

Например, можно представить метод **TransformElements** для массива:

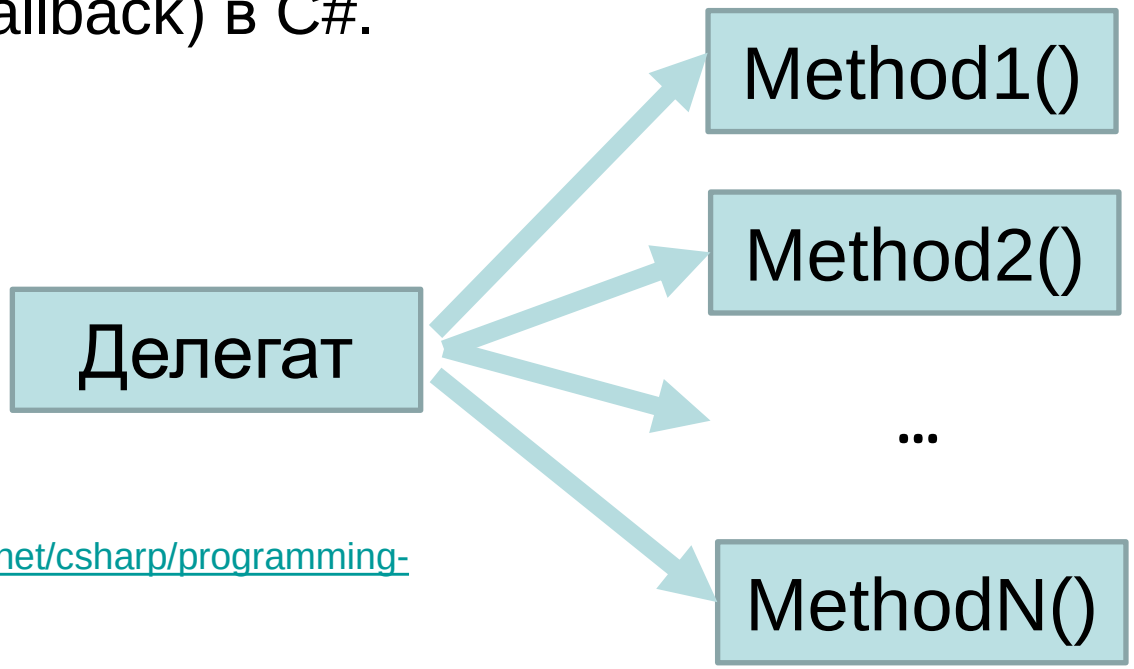
- Принимает на вход некоторое действие;
- Выполняет это действие над каждым из элементов массива.

Кроме того, возможность передавать методы в качестве параметров может быть полезной в сценариях, когда некоторое действие нужно выполнить при достижении некоторого момента времени/условия. Подобные действия в программировании называют методами обратного вызова (**callback**).

Что Такое Делегаты

Делегат-тип – специальный вид класса, экземпляр которого ссылается на методы с определённым типом возвращаемого значения и списком параметров.
Делегат – экземпляр делегата-типа, хранящий список ссылок на методы, соответствующие сигнатуре делегата-типа.
Экземпляр делегата является **неизменяемым**.

Делегаты являются одним из основных способов реализации механизма обратных вызовов (callback) в C#.



https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/delegates/using-delegates

Сигнатура Делегата-Типа и Сигнатура Метода

Для делегата-типа вводится собственное понятие сигнатуры, которое отличается от определения сигнатуры методов:

Сигнатура Метода

Включает:

- Имя метода;
- Типы параметров (с модификаторами);
- Порядок типов параметров.

Не включает:

- Имена параметров;
- Тип возвращаемого значения.

Сигнатура Делегата-Типа

Включает:

- Тип возвращаемого значения;
- Типы параметров (с модификаторами);
- Порядок типов параметров.

Не включает:

- Имена параметров;
- Имя метода.

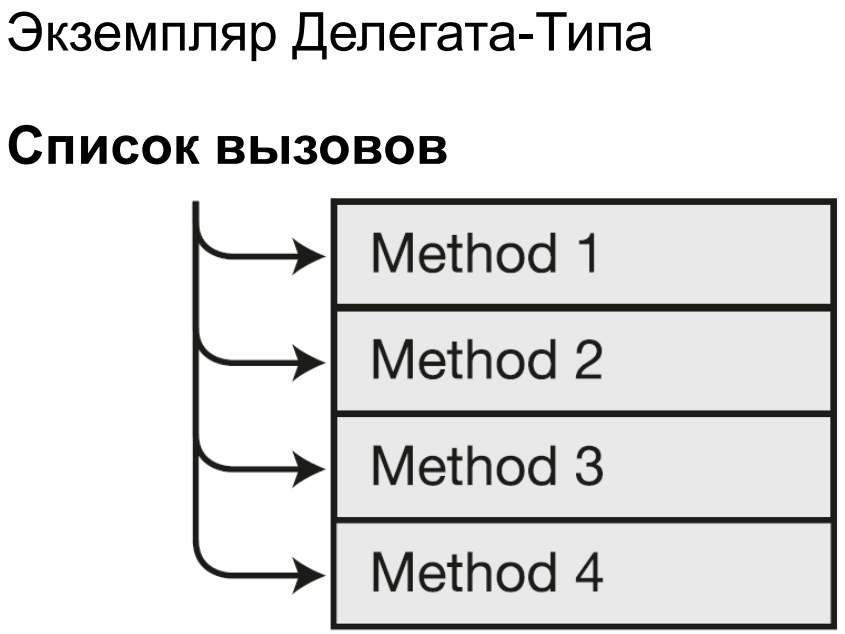
Помните: в обоих случаях модификаторы параметров **ref**, **out** и **in** считаются одним и тем же; не учитывается модификатор **params**.

Список Вызовов Экземпляров Делегат-Типов

Особенностью делегатов в C# является **наличие списка вызовов** (*Invocation List*), который может хранить ссылки **на несколько методов**, соответствующих сигнатуре делегата.

При этом есть ряд особенностей:

- Допустимы ссылки и на статические, и на экземплярные методы;
- Порядок вызова методов в списке вызовов соответствует их порядку добавления;
- Допустимо добавить ссылку на метод одного класса/объекта в список вызовов более одного раза;
- Попытка вызова экземпляра с пустым списком вызовов приводит к **NullReferenceException**.



Объявление Делегат-Типа и Создание Экземпляра

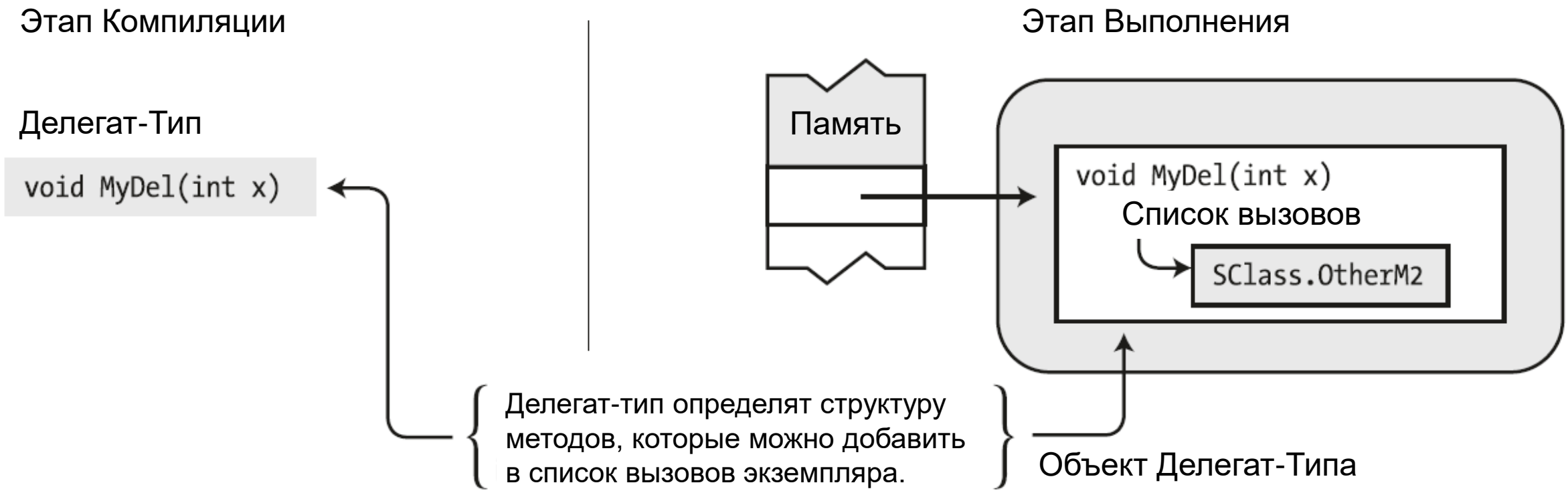
```
ключевое слово      Имя делегат-типа
      ↓                   ↓
delegate void MyDel(int x);                    // Объявление делегата-типа.

class DemoObject
{
    public void Print(int value) => System.Console.WriteLine(value);
    public static void StaticPrint(int value) => System.Console.WriteLine(value);
}
```

Для делегатов существует 2 варианта синтаксиса создания объектов:

- Стандартный с помощью new;
- Укороченный – через присваивание.

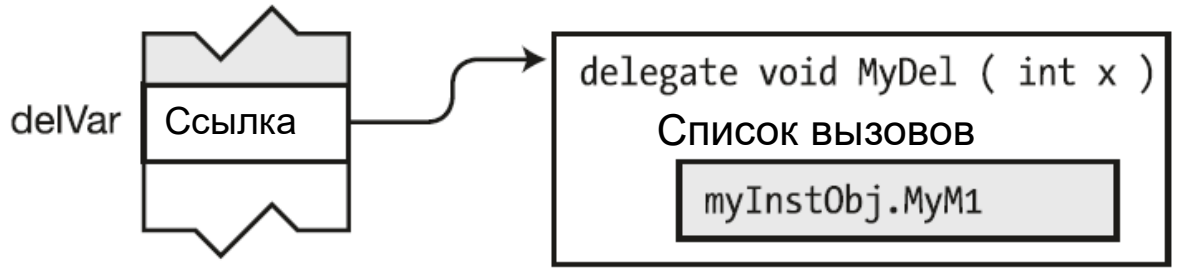
```
DemoObject myInstObj = new DemoObject();
MyDel delEx1 = new MyDel(myInstObj.Print);    // Ссылка на метод объекта myInstObj.
MyDel delEx2 = DemoObject.StaticPrint;        // Укороченный синтаксис создания.
```

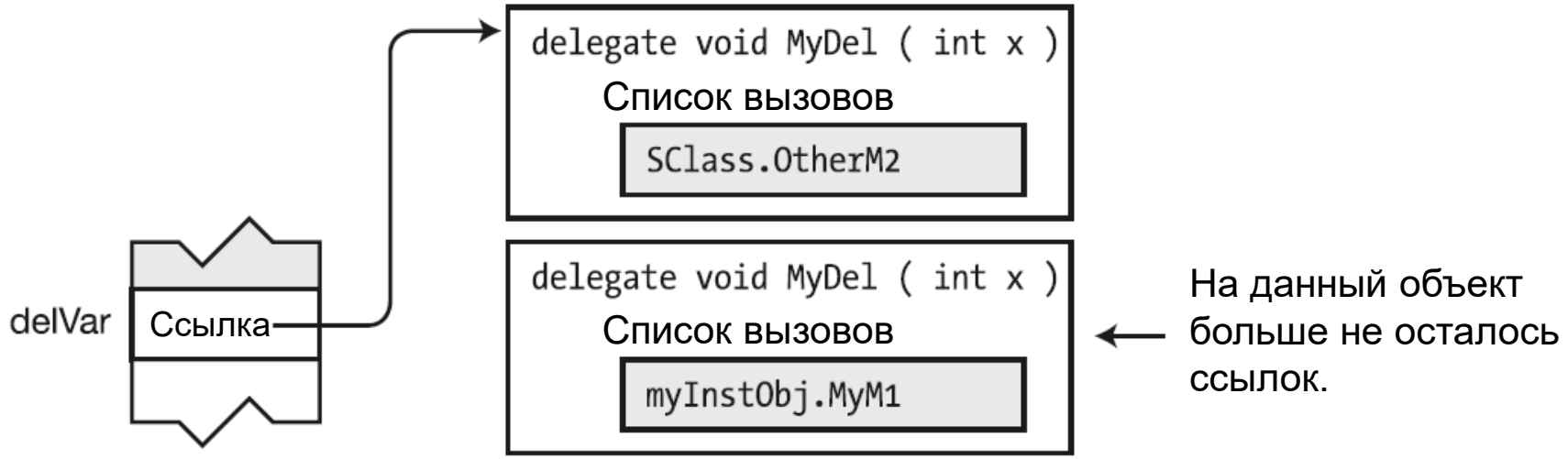
Делегат-Тип и Экземпляр Делегата



Создание Экземпляров Делегат-Типов

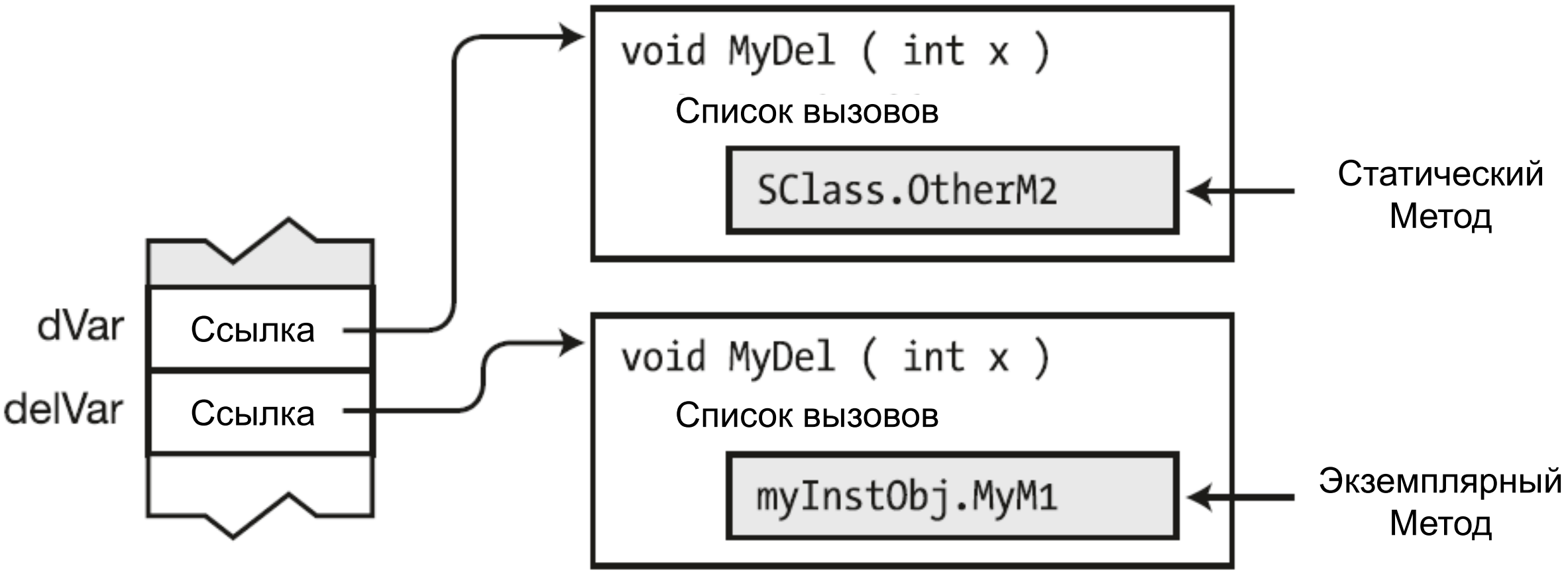
```
MyDel delVar;
delVar = myInstObj.MyM1; // Создание делегата и присваивание ссылки.
delVar = SClass.OtherM2; // Создание нового экземпляра по той же ссылке.
```





Несколько Экземпляров Делегат-Типов в Памяти

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



Операции над Делегатами

Для делегатов перегружены операции **+**, **-**, **+=** и **-=**.

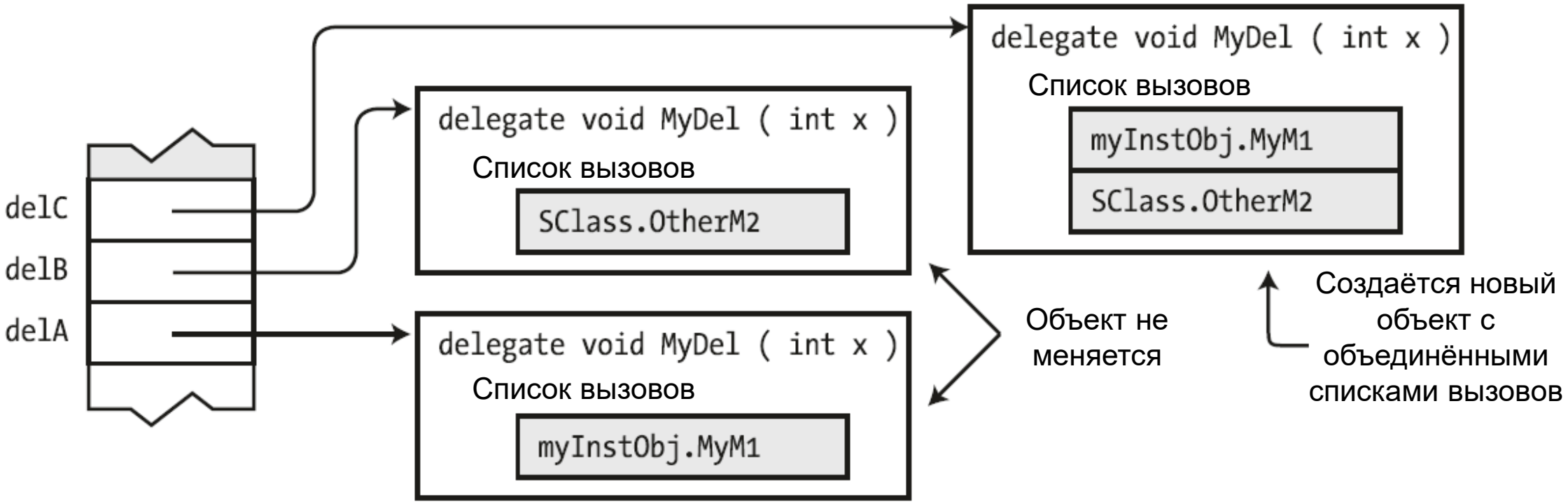
+ позволяет получить новый делегат, состоящий из списков вызовов методов двух других. Вы можете также объединять делегаты с методами.
- позволяет убрать методы/списки методов из делегата. Если указанного метода в списке вызовов нет, ничего не произойдёт, если их несколько – будет удалено последнее вхождение метода в список вызовов.
+= и **-=** работают интуитивным образом.

Помните: операндами в данном случае могу быть как делегаты, так и отдельные методы.

```
MyDel myDel = DemoObject.StaticPrint;
DemoObject myInstObj = new DemoObject();
myDel += myInstObj.Print;                    // Добавление в список вызовов.
```

Объединение Списков Вызовов: Схема

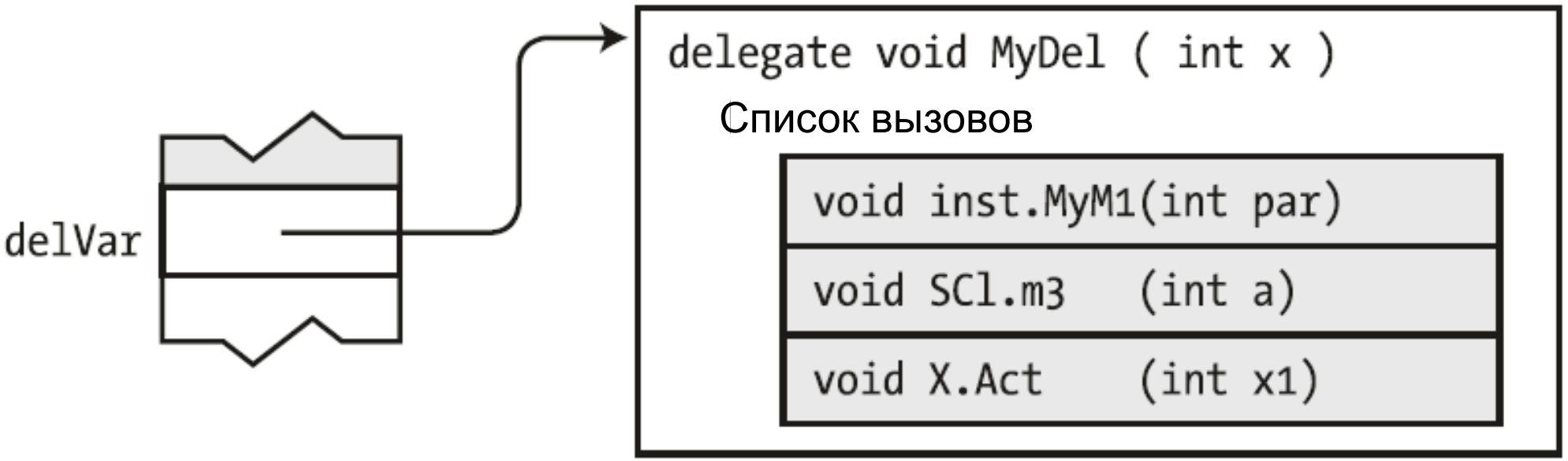
```
MyDel delA = myInstObj.MyM1;
MyDel delB = SClass.OtherM2;
MyDel delC = delA + delB;      // Объединение списков вызовов.
```



Добавление Методов в Список Вызовов: Схема

```
MyDel delVar = inst.MyM1;   // Создание и инициализация.
delVar += SCl.m3;           // Добавление метода в список вызовов.
delVar += X.Act;            // Добавление ещё одного метода.
```

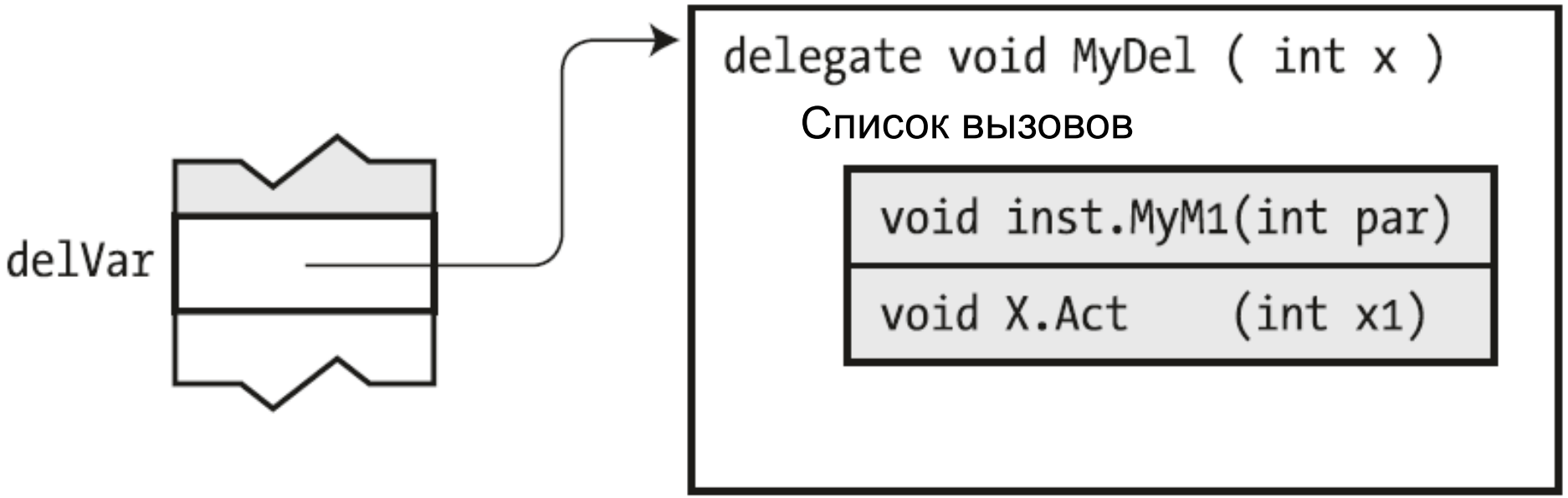
Результат:



Удаление Методов из Списка Вызовов: Схема

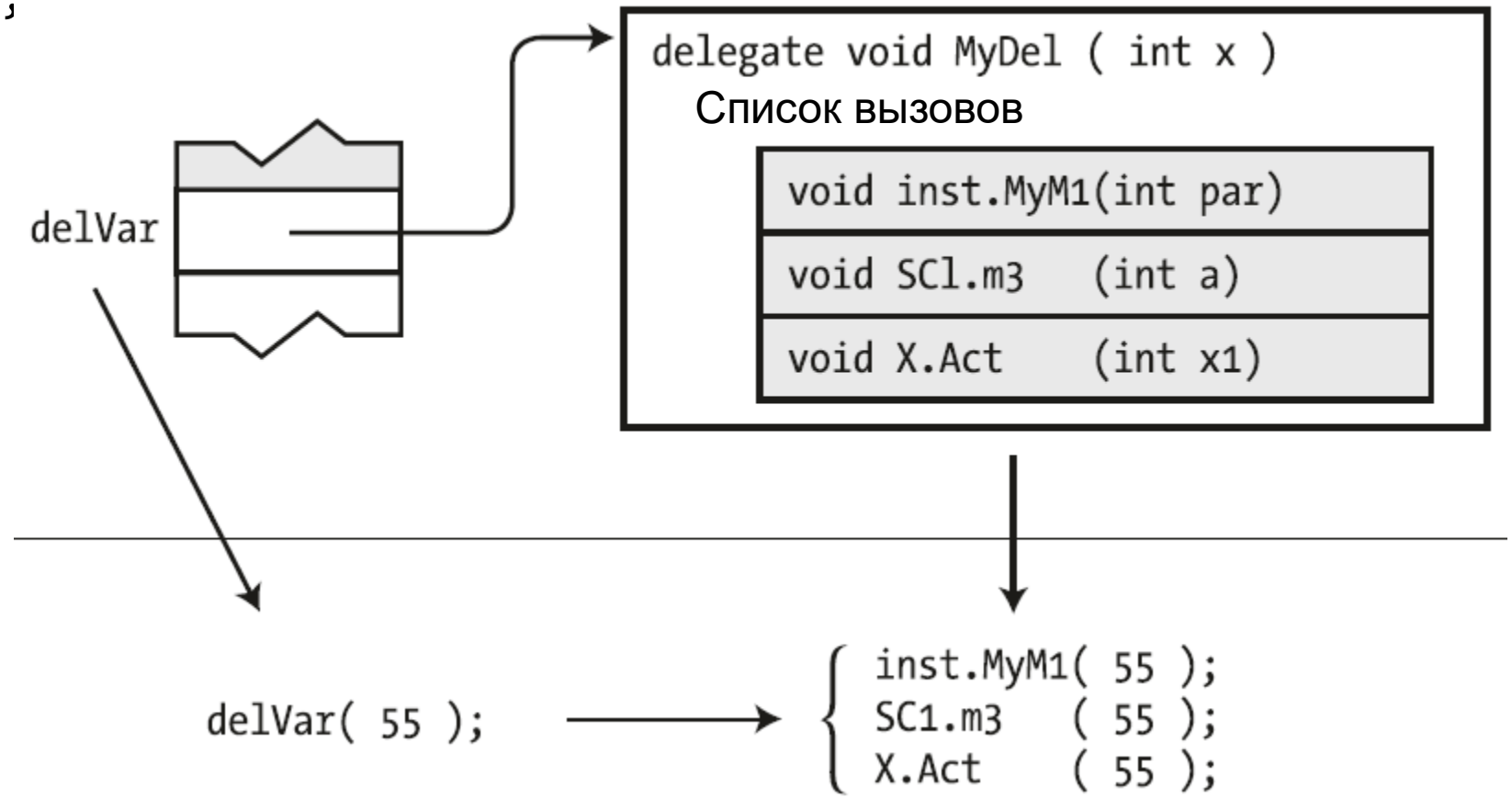
```
delVar -= SCl.m3; // Удаление метода из списка вызовов.
```

Результат:



Вызов Делегата: Схема

```
MyDel delVar = inst.MyM1;
delVar += SCl.m3;
delVar += X.Act;
// Вызов делегата синтаксически не отличается от вызова метода:
delVar(55);
```



Пример: Класс и Делегат. Часть 1

```
using System;

// Файл: Student.cs. Определяет 2 метода с одинаковым набором параметров и типов
// возвращаемого значения для соответствия сигнатуре делегата.
public class Student
{
    private static uint _studentCount = 0;
    private int _group;

    public Student(int group)
    {
        ++_studentCount;
        _group = group;
    }
    public void PrintGroup() => Console.WriteLine(_group);
    public static void PrintStudentsCount()
            => Console.WriteLine($"There are {_studentCount} student(s).");
}
```

Пример: Класс и Делегат. Часть 2

```
// Файл: Program.cs. Объявляется делегат-тип, используется его
// экземпляр с методами класса/объекта Student.
class Program
{
    static void Main()
    {
        Student student = new(216);
        MyDel studentCaller = student.PrintGroup;
        studentCaller += Student.PrintStudentsCount;
        studentCaller += student.PrintGroup;

        studentCaller();
    }
}
```

Вывод:
216
There are 1 student(s).
216

Метод одного и того же объекта может быть добавлен более одного раза.

Обращение к Делегатам в Методах

При передаче экземпляров делегат-типов в методы необходимо учитывать, что вызов делегата с пустым списком вызовов приведёт к **NullReferenceException**. Для обработки этого случая можно выполнить явную проверку:

- `if (myDel != null) myDel();  // Явная проверка.`
- `myDel?.Invoke();  // Операция ?. + явный вызов.`

Пример:

```
public delegate int ArrayTransformer(int value);

static void ArrayForEach(int[] array, ArrayTransformer transform)
{
    if (transform == null) { return; }
    for (int i = 0; i < array.Length; ++i)
    {
        array[i] = transform(array[i]);
    }
}
```

Делегаты с Непустым Возвращаемым Значением

Допускается объявление делегат-типов с непустым возвращаемым значением (т.е. любым, кроме void).

Хотя отработают все методы, будет возвращено **только значение, полученное в результате работы последнего метода, добавленного в список вызовов**.

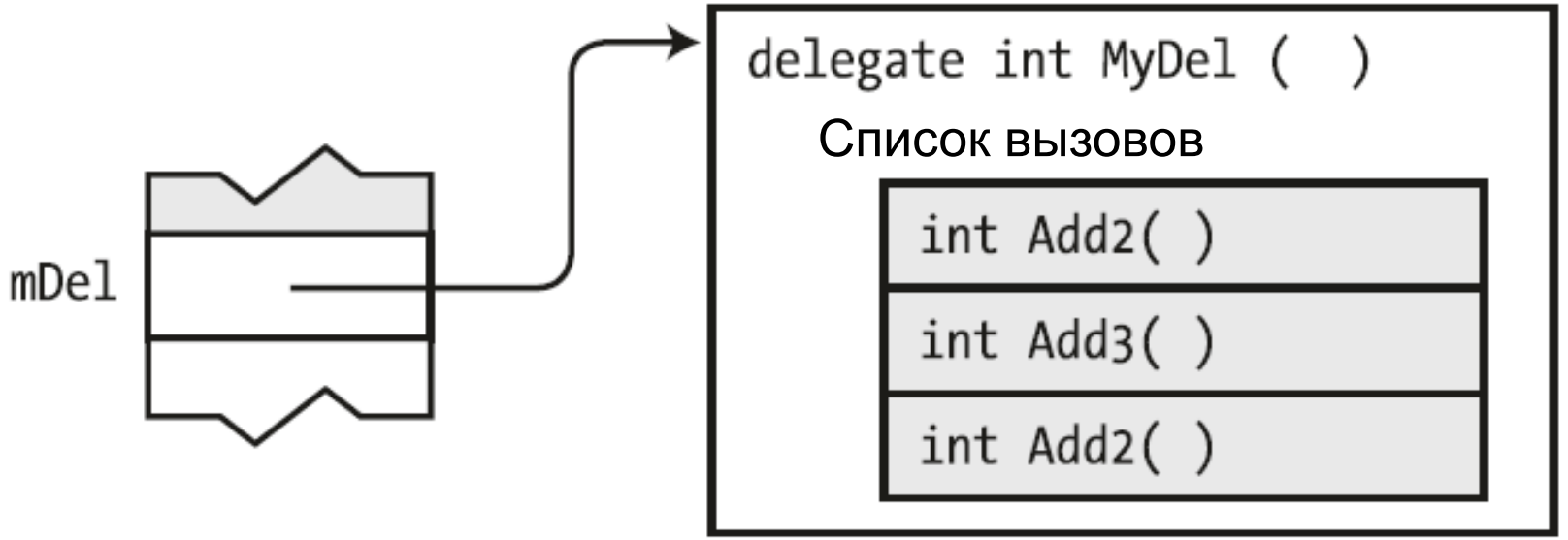
```
class Program
{
    static int number = 0;
    static int GetVal(int value) => number += value + 1;
    delegate int Returner(int val);
    static void Main()
    {
        Returner valReturn = GetVal;
        valReturn += GetVal;
        System.Console.WriteLine(valReturn(2)); // Вернёт и напечатает только 6.
    }
}
```

Пример: Вывод Возвращаемого Делегатом Значения

```
class Program
{
    delegate int MyDel();
    static int value = 1;
    static int Add2() => value += 2;
    static int Add3() => value += 3;
    static void Main()
    {
        MyDel mDel = Add2;          // Объявление и инициализация.
        mDel += Add3;               // Добавление метода Add3.
        mDel += Add2;               // Добавление метода Add2.
        // Вызов делегата и вывод результата:
        System.Console.WriteLine($"Value: {mDel()}");
    }
}
```

Вывод:
Value: 8

Схема к Примеру на Прошлом Слайде



```
mDel( );  ⟶  { Add2();   ⟵ Значение 3 игнорируется.
               Add3();   ⟵ Значение 6 игнорируется.
               Add2();   ⟵ Значение 8 выводится.
```

Делегаты и Передача Параметров по Ссылке

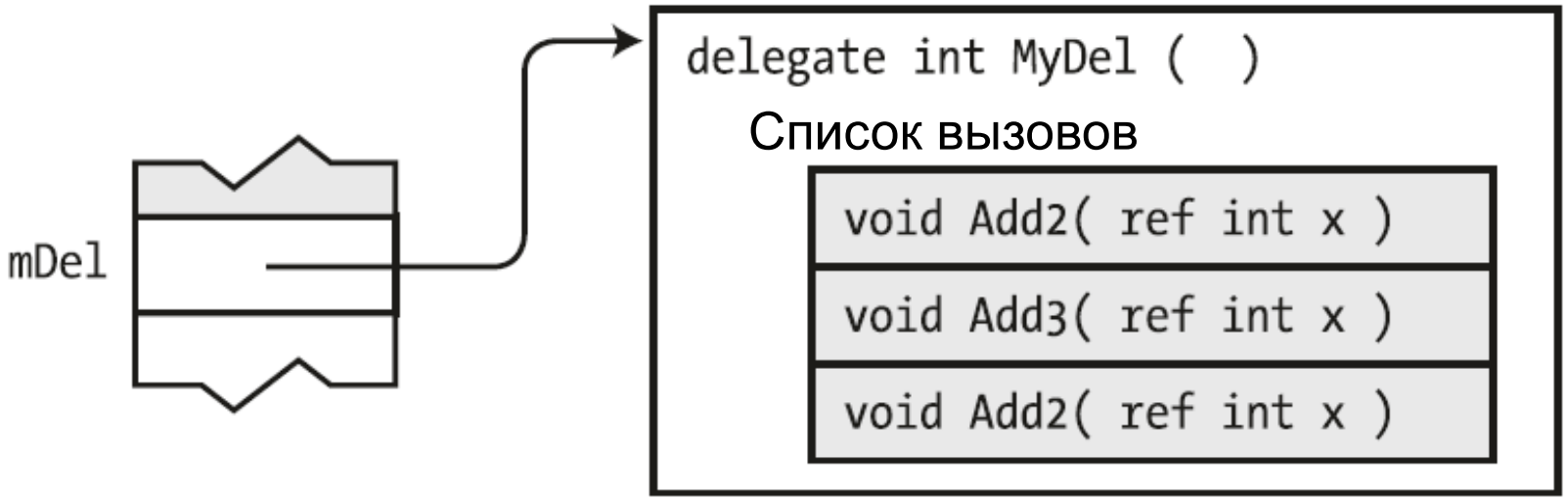
При работе с делегатами может возникнуть необходимость изменять переданное значение в процессе вызовов методов по цепочке.

Для реализации такого сценария используется **модификатор ref**, который позволяют передавать ссылку последовательно по цепочке.

```
MyDel mDel = Add2;
mDel += Add3;
mDel += Add2;
int x = 5;
mDel(ref x);
System.Console.WriteLine($"Value: {x}");

void Add2(ref int x) { x += 2; }
void Add3(ref int x) { x += 3; }

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

Значение x передаётся в каждый из методов по ссылке в порядке их добавления.

Делегаты и Передача Параметров по Ссылке: Схема



```
mDel( );  ⟶  { Add2( x = 5  );   ⟵ Изначальное значение x.
               Add3( x = 7  );   ⟵ Значение x после Add2.
               Add2( x = 10 );   ⟵ Значение x после Add3.
```

Как Устроены Делегаты

Все делегаты неявно наследуются от класса MulticastDelegate:

Object → Delegate → MulticastDelegate

При этом все делегаты пользовательских типов фактически генерируются средой выполнения (CLR).

Ниже представлено несколько ссылок, позволяющих подробнее ознакомиться со внутренним устройством делегатов:

Исходный код MulticastDelegate:
https://github.com/dotnet/coreclr/blob/01a9eaaa14fc3de8f11eafa6155af8ce4e44e9e9/src/mscorlib/src/System/MulticastDelegate.cs#L622-L627

Описание, как внутри устроены делегаты и события:
https://www.codeproject.com/Articles/26936/Understanding-NET-Delegates-and-Events-By-Practice#Internal

Статья про внутреннее устройство делегатов с большим количеством ссылок:
https://mattwarren.org/2017/01/25/How-do-.NET-delegates-work/

Информация об Экземплярах Делегат-Типов

Экземпляры делегат-типов **хранят в себе информацию о вызываемых методах и объектах**, для которых их нужно вызывать.

Все делегаты-типы имеют следующие функциональные члены:

- `public Delegate[] GetInvocationList()` – метод, возвращающий массив объектов Delegate, содержащих информацию о каждом из вызываемых методов;
- `public MethodInfo Method { get; }` – информация о последнем в списке вызовов методе;
- `public object? Target { get; }` – объект, связанный с последним методом в списке вызовов или null, если метод статический.

Обратите внимание: т.к. делегаты хранят в себе ссылку на объект, относительно которого вызывается метод, может происходить продление жизни объектов (иногда нежелательное).
Попробуйте самостоятельно смоделировать данный сценарий.

Пример 1: Method и Target Объекта. Часть 1

```
using System.Collections.Generic;

public delegate int[] Row();     // Делегат-тип и класс в файле DigitSplitter.cs.
public class DigitSplitter {
    private int _value;
    public DigitSplitter(int value) => _value = value;
    public int[] GetDigitsArray() {
        int copy = _value;
        List<int> digits = new List<int>();
        while (copy != 0) {
            digits.Add(copy % 10);
            copy /= 10;
        }
        return digits.ToArray();
    }
}
```

Пример 1: Method и Target Объекта. Часть 2

```
using System;

// Основная программа – класс Program.cs.
class Program
{
    static void Main()
    {
        DigitSplitter splitter = new(12345);
        Row delRow = splitter.GetDigitsArray;
        Console.WriteLine($"The method called is {delRow.Method}");
        Console.WriteLine($"The target object is {delRow.Target}");
    }
}
```

Вывод:
The method called is Int32[] GetDigitsArray()
The target object is DigitSplitter

Пример 2: Вызов Экземплярных Методов. Часть 1

```
using System;

delegate void PrintDel(int val);

// Файл TargetDelegateDemo.cs – определяет делегат и класс
// с двумя методами, один из которых статический.
public class TargetDelegateDemo
{
    public void Print(int value) => Console.WriteLine(value);
    public static void StaticPrint(int value)
            => Console.WriteLine($"static: {value}");
}
```

Пример 2: Вызов Экземплярных Методов. Часть 2

```
class Program {
    static void Main() {
        TargetDelegateDemo p = new TargetDelegateDemo();
        PrintDel printDel = p.Print;
        printDel += TargetDelegateDemo.StaticPrint;
        printDel += TargetDelegateDemo.StaticPrint;
        printDel += p.Print;
        Delegate[] delegates = printDel.GetInvocationList();
        for (int i = 0; i < delegates.Length; ++i) {
            if (delegates[i].Target != null) {
                ((PrintDel)delegates[i]).Invoke(i);
            }
        }
    }
}
```

Вывод:
0
3

Проверка, что метод принадлежит объекту.

Приведение типов требуется: неизвестно, как делать вызов для базового типа Delegate.

Пример 3: Массивы Делегатов. Часть 1

```
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; }              // Вперёд.
    public void Backward() { --y; }             // Назад.

    public void PrintPosition()                     // Печать текущих координат.
            => Console.WriteLine($"The Robot is located at: x={x}, y={y}");
}
```

Пример 3: Массивы Делегатов. Часть 2

```
class Program
{
    static void Main()
    {
        Robot rob = new Robot();          // Создание робота.
        Steps[] trace = { rob.Backward, rob.Backward, rob.Left };

        for (int i = 0; i < trace.Length; i++)
        {
            Console.WriteLine($"Method={trace[i].Method}, Target={trace[i].Target}");
            trace[i]();
        }
        rob.PrintPosition();                    // Вывод итоговых координат.
    }
}
```

Вывод:

Method=Void Backward(), Target=Robot
Method=Void Backward(), Target=Robot
Method=Void Left(), Target=Robot
The Robot is located at: x=-1, y=-2

Библиотечные Делегаты в BCL

В пространстве имён System определён набор стандартных делегат-типов, который позволяет не создавать собственные делегат-типы без необходимости.

Наличие таких делегат-типов в первую очередь нужно для унификации кода и фактически покрывает большинство сценариев за исключением случаев, когда нужны модификаторы ref/in/out или params.

System.Action – делегат-типы, для которых указывается 0-16 типов входных параметров, а тип возвращаемого значения всегда void.

System.Func – делегат-типы, для которых указывается 0-16 типов входных параметров, а тип возвращаемого значения задаётся последним параметром.

Action: https://docs.microsoft.com/en-us/dotnet/api/system.action?view=net-6.0
Func: https://docs.microsoft.com/en-us/dotnet/api/system.func-1?view=net-6.0

Ссылки с Источниками по Делегатам

Краткий обзор возможностей: https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/delegates/

Инструкция по использованию делегатов:
https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/delegates/using-delegates

Основы работы с делегатами: https://docs.microsoft.com/en-us/dotnet/csharp/delegate-class

Класс Delegate: https://docs.microsoft.com/en-us/dotnet/api/system.delegate?view=net-6.0

Класс MulticastDelegate: https://docs.microsoft.com/en-us/dotnet/api/system.multicastdelegate?view=net-6.0

Исходный код System.Delegate (partial class):
https://github.com/dotnet/runtime/blob/main/src/coreclr/nativeaot/System.Private.CoreLib/src/System/Delegate.cs
https://github.com/dotnet/runtime/blob/main/src/libraries/System.Private.CoreLib/src/System/Delegate.cs

Исходный код System.MulticastDelegate:
https://github.com/dotnet/runtime/blob/main/src/coreclr/nativeaot/System.Private.CoreLib/src/System/MulticastDelegate.cs

Использование рефлексии для работы с делегатами:
https://docs.microsoft.com/en-us/dotnet/framework/reflection-and-codedom/how-to-hook-up-a-delegate-using-reflection

Статья с информацией и разными источниками про внутреннее устройство делегатов:
https://mattwarren.org/2017/01/25/How-do-.NET-delegates-work/

Мини-викторина

Отметьте все верные утверждения:

1. Объект делегат может ссылаться только на один метод;
2. Тип возврата метода учитывается в сигнатуре делегата;
3. Имя метода входит в сигнатуру делегата;
4. Сигнатуры делегатов, отличающиеся только модификаторами параметров ref и out, эквивалентны;