การเขียนโปรแกรม ด้วยภาษา C#

Delegate

Delegate

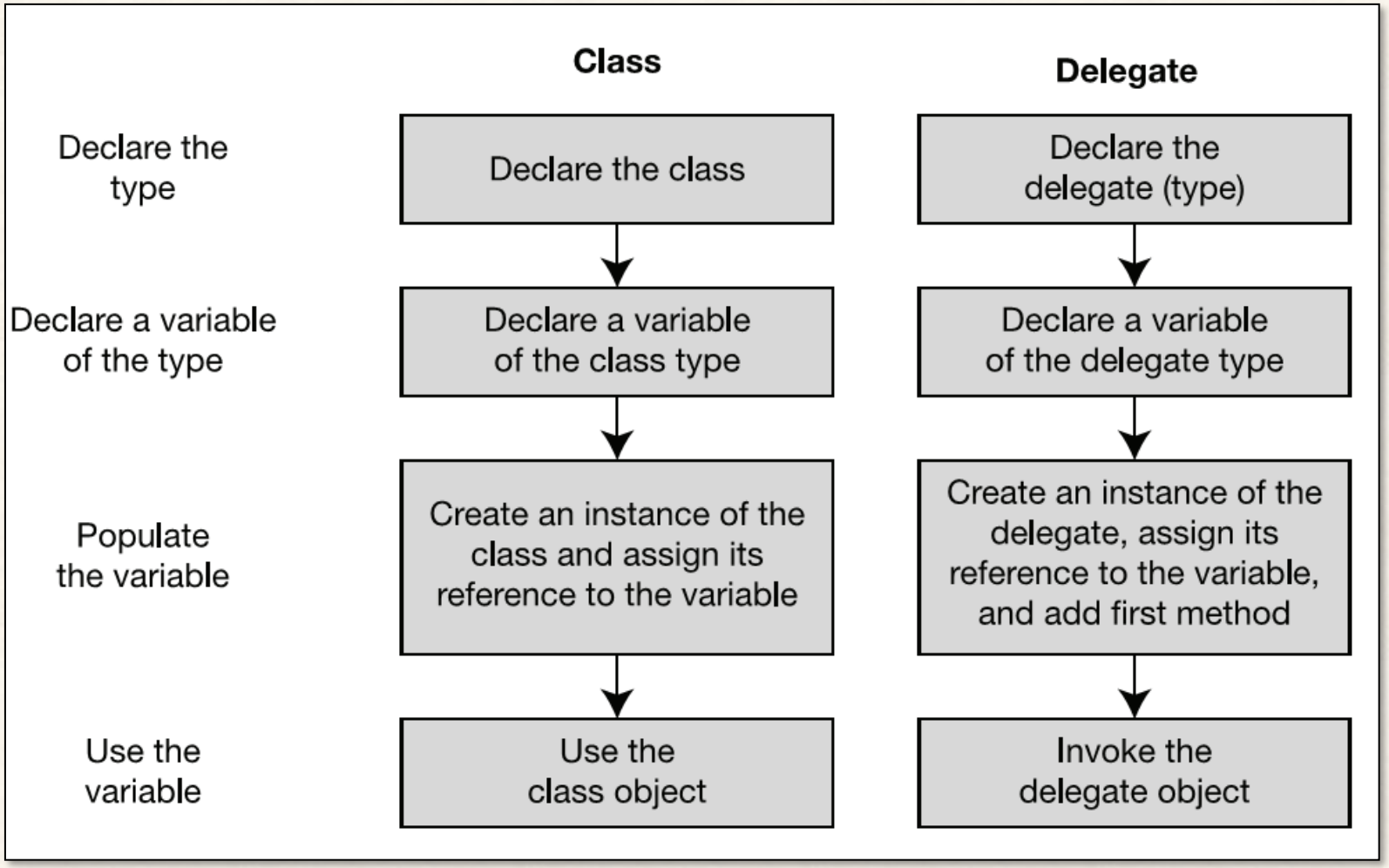
Delegate

- Delegate เป็น object ที่ใช้เก็บ reference ไปยัง method หรือหลายๆ method
- เวลาเรียกใช้ delegate จะทำให้ method เหล่านั้นถูกเรียกใช้ในคราวเดียวกันทั้งหมด (ตามลำดับ)
- เราสามารถเพิ่ม method จากคลาสใดๆ ได้ แต่ต้องมี signature ตรงกับ delegate
  - method ของ instance = ต้องสร้าง object ก่อน
  - static method = เรียกใช้ได้ทันที

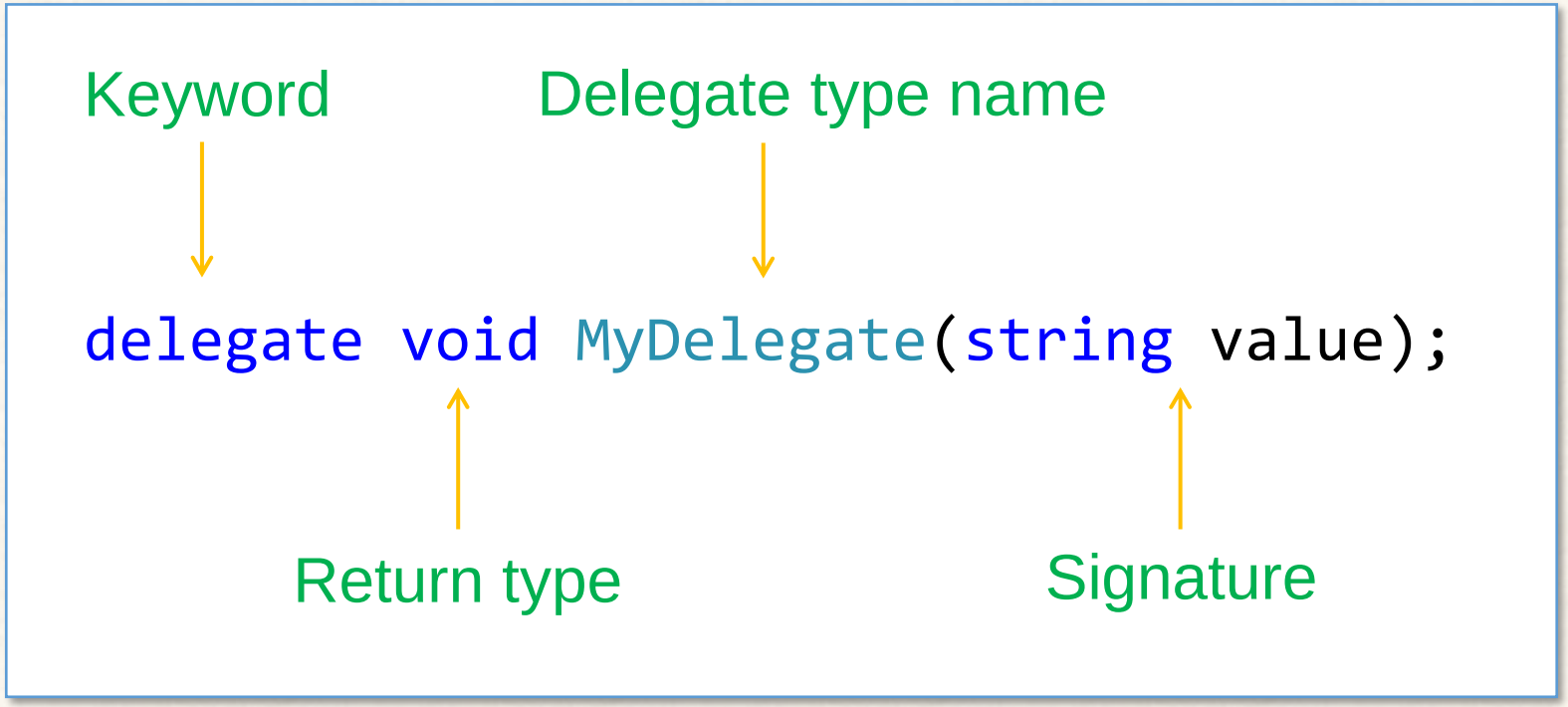
Delegate

- delegate เป็น user-defined type ชนิดหนึ่ง
- สามารถใช้งานได้แบบเดียวกับ class หรือ built-in type
- ลำดับในการสร้างและใช้งาน delegate
  - declare delegate type
  - declare delegate variable
  - create delegate object (and add some methods)
  - invoke the delegate (เราเรียก invoke เพราะ delegate ต่างจาก method และสามารถ hold หลายๆ method ไว้ในตัว)

Class vs. Delegate



Declaring the Delegate Type



ลักษณะของ Delegate

- Declare เหมือน method
  - แต่มีคำว่า delegate นำหน้า
- ต่างจาก method ตรงที่
  - ไม่มี method body
  - ไม่ต้องประกาศภายใต้ class เนื่องจากมันเป็น type เช่นเดียวกับ class

Creating the Delegate Object

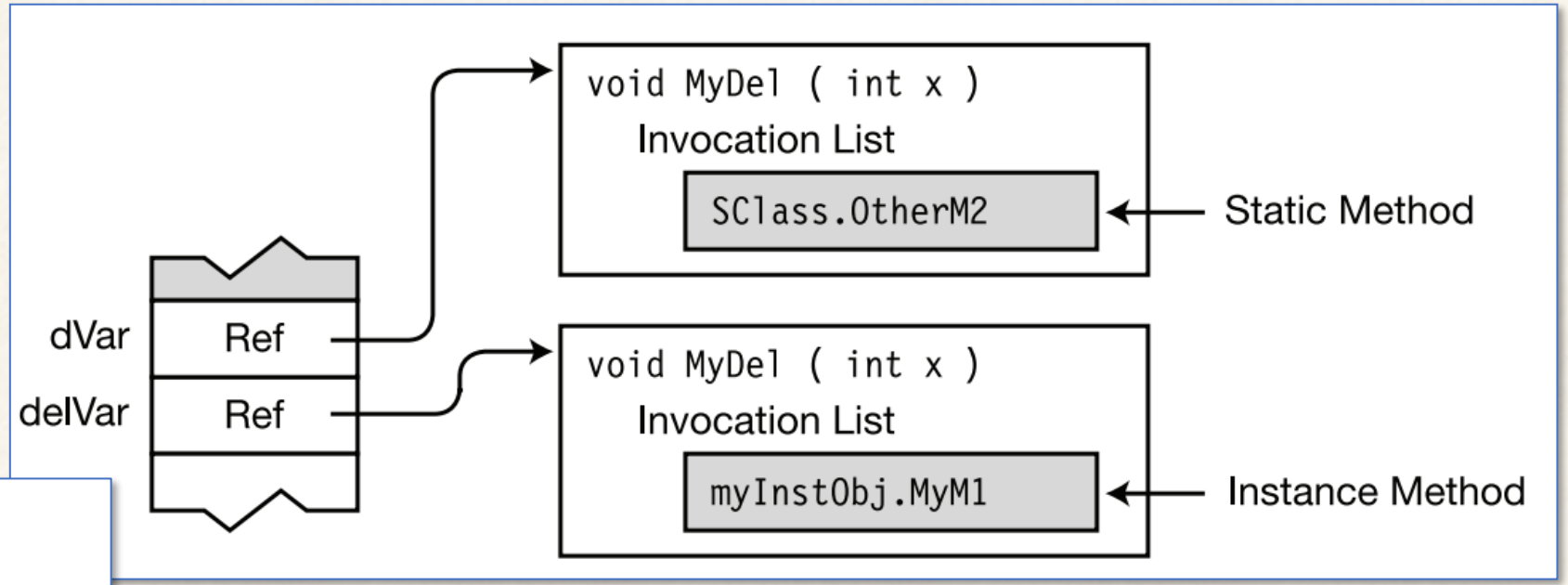
ใช้ keyword “new”

```
MyDelegate myODel = new MyDelegate { obj.method };

MyDelegate mySDel = new MyDelegate { StaticMethod };
```

ใช้ shortcut

```
MyDelegate myODels =  obj.method ;

MyDelegate mySDels =  StaticMethod;
```

Creating the Delegate Object




```
using System;

namespace ConsoleApp3
{
    delegate void MyDel(int x);
    internal class Program
    {
        static void Main(string[] args)
        {
            MyDel delVar, dVar;
            delVar = new MyDel(myInstObj.MyM1);
            dVar = new MyDel(SClass.OtherM2);
        }
    }
}
```

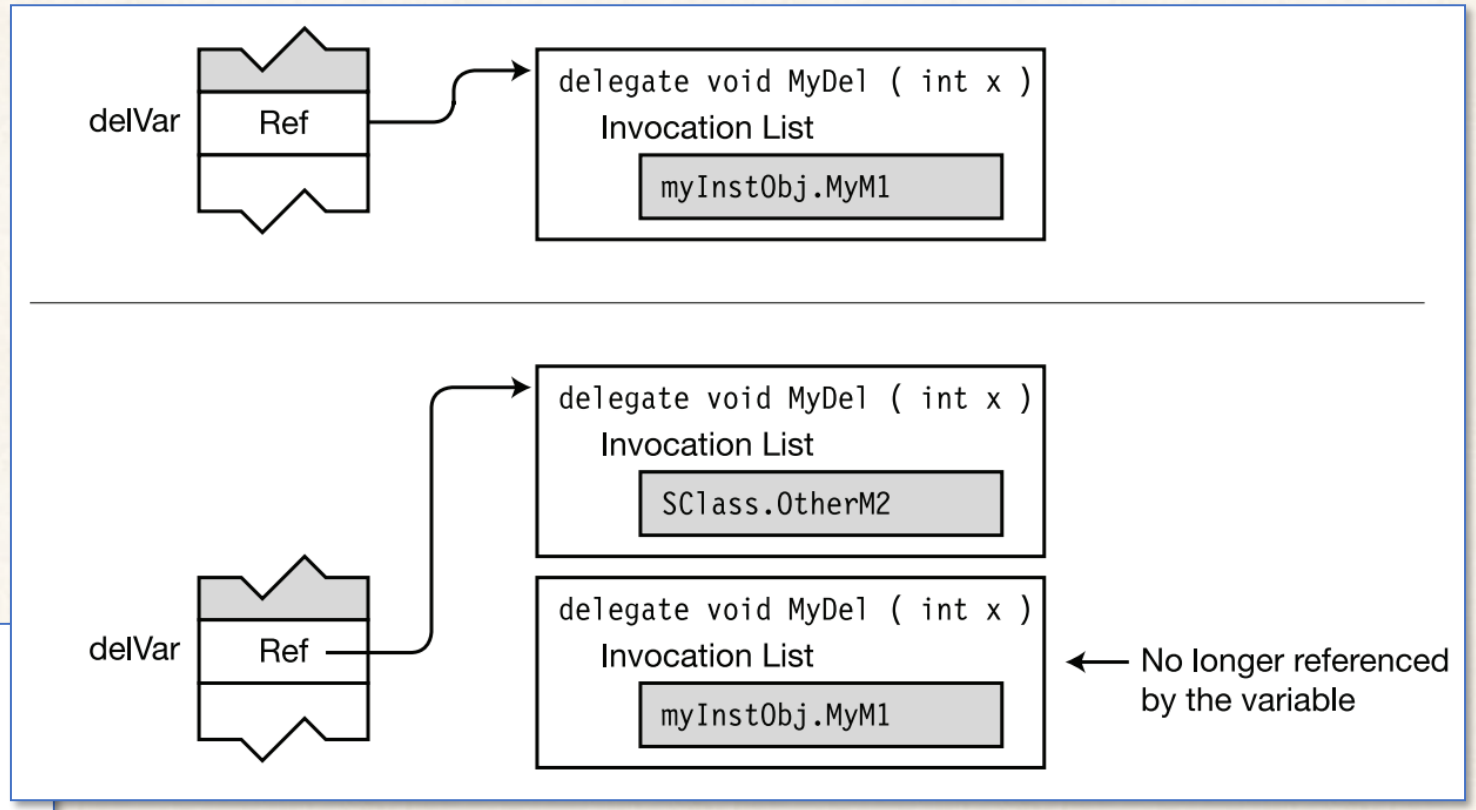
```
MyDel delVar = myInstObj.MyM1;
MyDel dVar = SClass.OtherM2;
```

Assigning Delegates

- Delegates จัดเป็น reference types อย่างหนึ่ง
- เราสามารถกำหนด reference ไปยัง delegate ได้ในภายหลัง

```
MyDel delVar;

// Create and assign the delegate object.
delVar = myInstObj.MyM1;
...


// Create and assign the new delegate object.
delVar = SClass.OtherM2;
```

Assigning Delegates



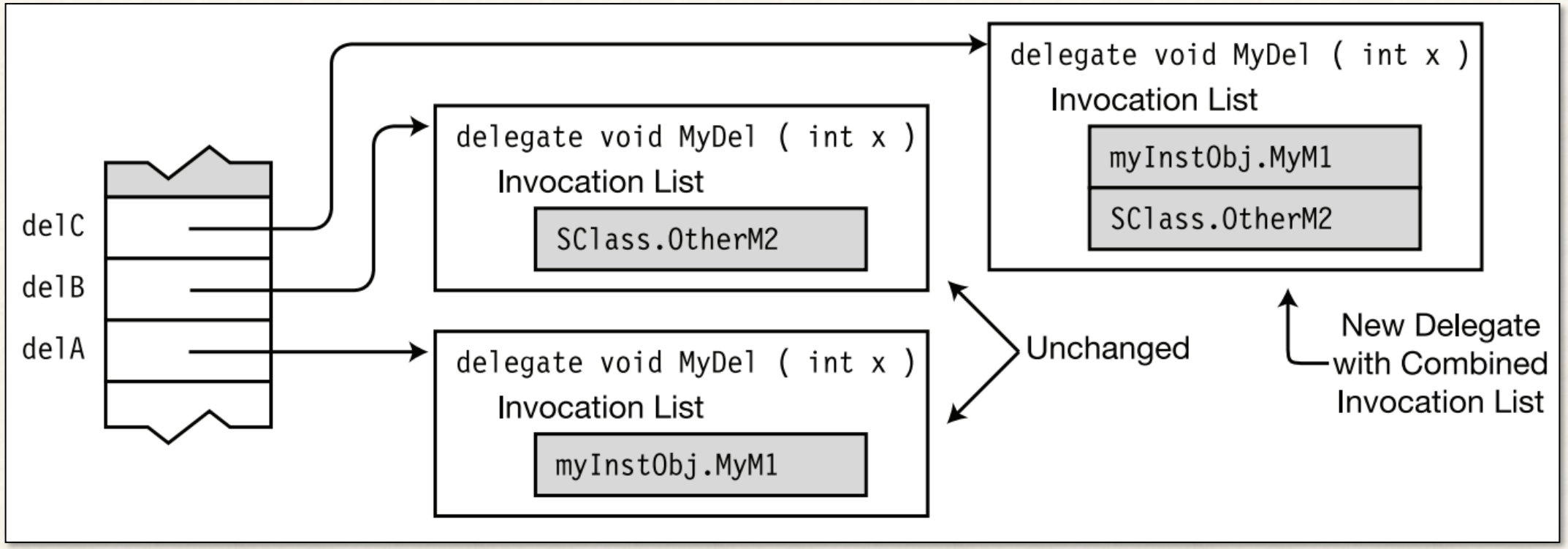

```
MyDel delVar;

// Create and assign the delegate object.
delVar = myInstObj.MyM1;
...


// Create and assign the new delegate object.
delVar = SClass.OtherM2;
```

Combining Delegates

- เราสามารถทำการ combine ให้ delegate หลายๆ ตัว เข้าด้วยกันได้
- ผลจากการ combine จะได้ object ของ delegate ตัวใหม่ โดยไม่ส่งผลกระทบใดๆ กับ delegate เดิมที่เป็นต้นฉบับ

```
MyDel delA = myInstObj.MyM1;

MyDel delB = SClass.OtherM2;

MyDel delC = delA + delB;

// Has combined invocation list
```

Combining Delegates

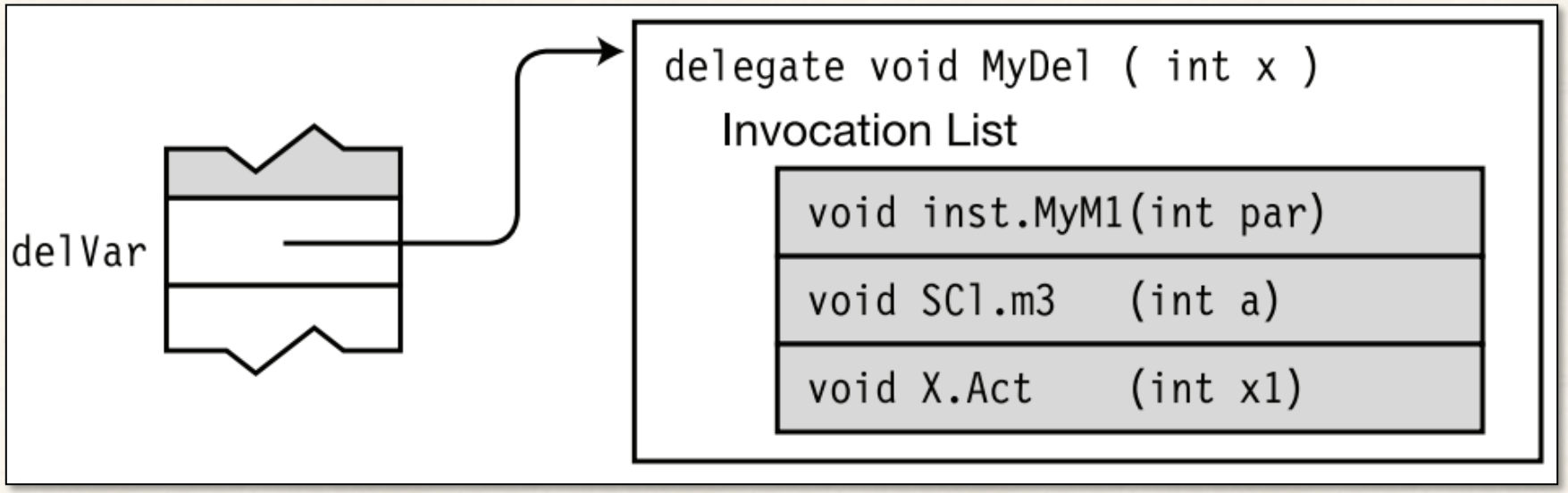


การเพิ่ม Methods ไปยัง Delegates

- เราสามารถเพิ่มความสามารถให้กับ delegate โดยการเพิ่ม method ทำได้โดยการใช้ operator +=
  - เช่นการประมวลผลข้อมูลรูปแบบต่างๆ ที่ต้องใช้หลายกระบวนการย่อย ๆ
  - การเดินทางจากจุดหนึ่งไปอีกจุดหนึ่ง ที่ต้องใช้พาหนะหลายอย่าง
- การเพิ่ม method เข้าไปใน invocation list จะทำให้เกิดความยืดหยุ่นในการปรับเปลี่ยน algorithm

การเพิ่ม Methods ไปยัง Delegates

```
MyDel delVar = inst.MyM1;    // Create and initialize.
delVar += SCl.m3;            // Add a method.
delVar += X.Act;             // Add a method.
```



การเพิ่ม Methods ไปยัง Delegates

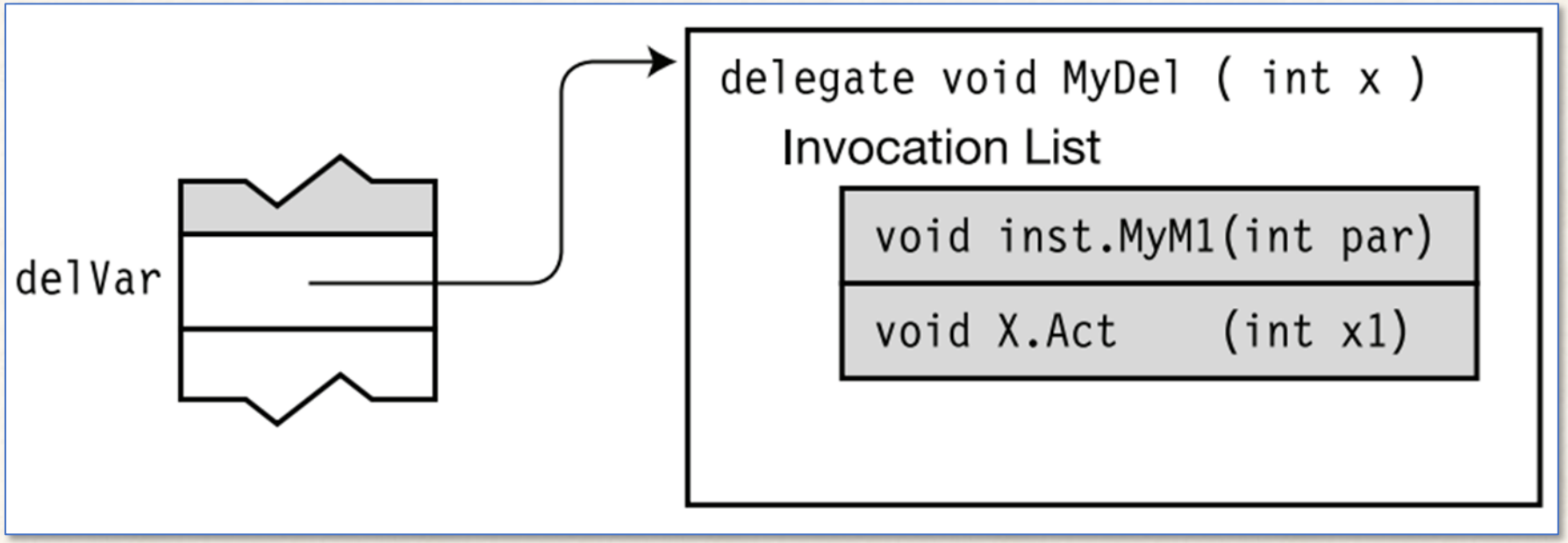
- ในความจริง delegate จะมีคุณสมบัติ immutable คือไม่สามารถเปลี่ยนรูปได้
- การเพิ่ม method ลงไปใน delegate จะเป็นการสร้าง delegate ขึ้นมาใหม่ แล้วย้าย reference ไปจาก delegate object เดิม
- เราสามารถเพิ่ม method ได้เรื่อย ๆ ทุกครั้งที่มีการเพิ่ม ระบบจะสร้าง delegate object ขึ้นมาใหม่ แล้วย้าย reference ไปชี้ยัง delegate ใหม่นั้นเสมอ

Removing Methods from a Delegate

- เราสามารถลดความสามารถของ delegate โดยการลบ method ทำได้โดยการใช้ operator -=
  - เช่นเดียวกับการเพิ่ม method, การลบ method จะทำได้โดยการสร้าง delegate object ใหม่โดยไม่นำ method ที่ลบออกมารวมด้วย
  - จากนั้นจะย้าย reference มายัง delegate ใหม่ที่สร้างขึ้น
- ผู้ใช้ไม่ต้องสนใจว่าทำได้อย่างไร เป็นหน้าที่ของ framework

การลบ Methods ออกจาก Delegate

```
delVar -= SCl.m3;
// Remove the method from the delegate.
```



การลบ Methods ออกจาก Delegate

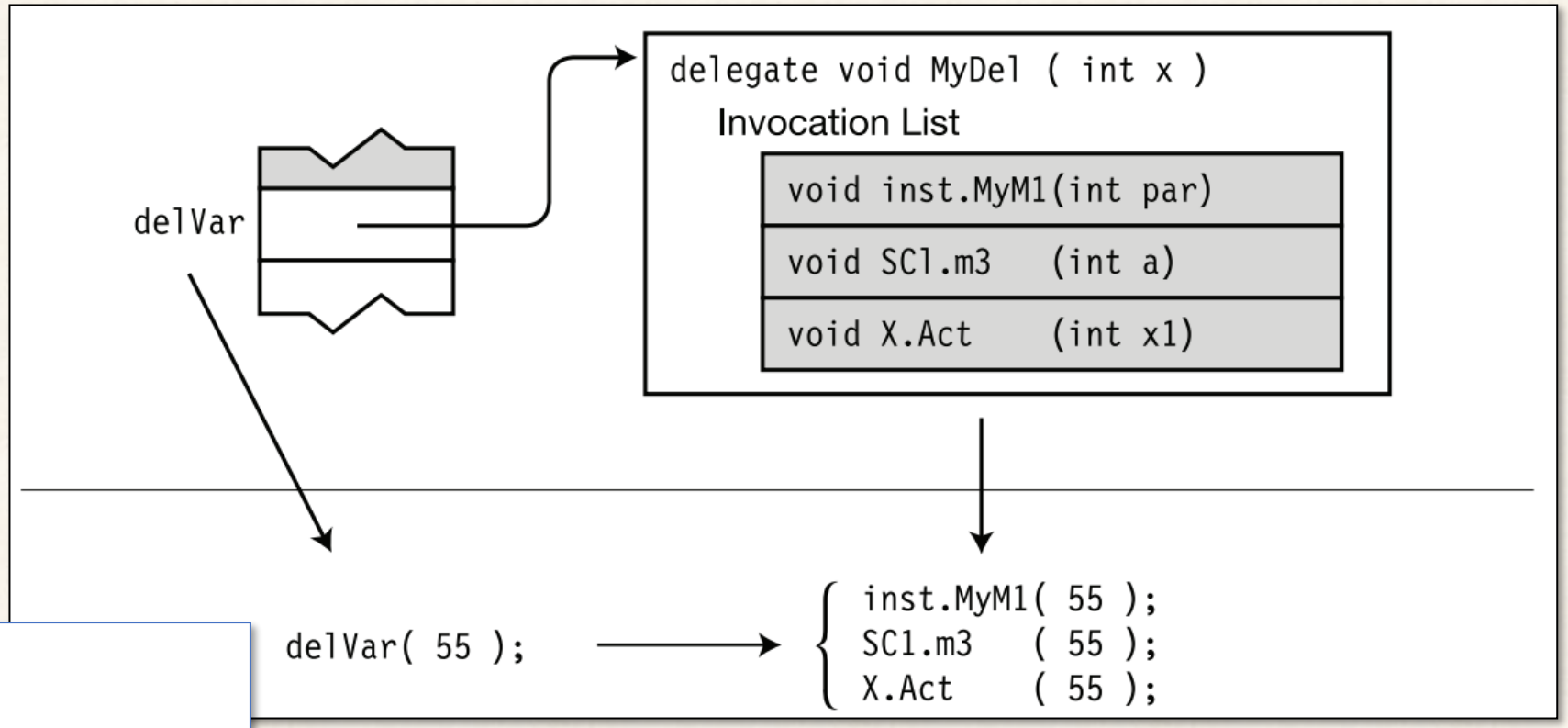
- ถ้าใน invocation list มี method เดียวกันซ้ำๆ กันหลายที่
  - operator ‘-=’ จะเริ่มค้นหาจากด้านล่างของ invocation list และลบ method แรกที่ตรงกัน
- การพยายามลบ method ที่ไม่อยู่ใน invocation list จะไม่ส่งผลกระทบใดๆ
- การพยายามเรียกใช้ (invoke) delegate ที่ว่างเปล่า (ถูกลบออกจนหมด) จะเกิด exception ดังนั้นควรตรวจสอบว่า delegate เป็น null หรือไม่ก่อนที่จะ invoke

การเรียก Delegate (Invoking a delegate)

- การ invoke delegate ก็เหมือนการเรียกใช้ method ทั่ว ๆ ไป
  - ต้องเรียกตามรูปแบบที่ประกาศไว้เท่านั้น
- เมื่อมีการ invoke ด้วย parameter ก็จะส่ง parameter นั้นไปยัง method ที่อยู่ใน invocation list ด้วย

```
MyDel delVar = inst.MyM1;
delVar += SCl.m3;
delVar += X.Act;
...
delVar( 55 ); // Invoke the delegate.
...
```

การเรียก Delegate (Invoking a delegate)




```
MyDel delVar = inst.MyM1;
delVar += SCl.m3;
delVar += X.Act;
...
delVar( 55 ); // Invoke the delegate.
...
```

Delegate Example 1:

Invoking Delegate

ตัวอย่างการใช้งาน Delegate

```
delegate void PrintFunction();
class Test
{
    public void Print1()
    {
        Console.WriteLine("Print1  -- instance method");
    }
    public static void Print2()
    {
        Console.WriteLine("Print2  -- static method");
    }
}
```

ตัวอย่างการใช้งาน Delegate

```
class Program
{
    static void Main(string[] args)
    {
        Test t = new Test(); // create instance of Test class.
        PrintFunction pf;    // create null delegate.
        pf = t.Print1;       //Instance and initialixe the delegate.

        // add three more methods to the delegate.
        pf += Test.Print2;
        pf += t.Print1;
        pf += Test.Print2;

        if (null != pf)
        {
            pf();           // Invoke the delegate
        }
        else
            Console.WriteLine("Delegate is empty.");
    }
}
```

การเรียก Delegates ที่มี Return Values

- ถ้า delegate มีการส่งค่ากลับ และใน invocation list มี method มากกว่า 1 แล้ว การ invoke delegate จะเป็นไปตามกฎต่อไปนี้
  - เฉพาะค่าส่งกลับจาก method สุดท้ายใน invocation list จะถูกนำไปใช้
  - ค่าส่งกลับจาก methods อื่นๆ ใน invocation list จะถูกเพิกเฉย

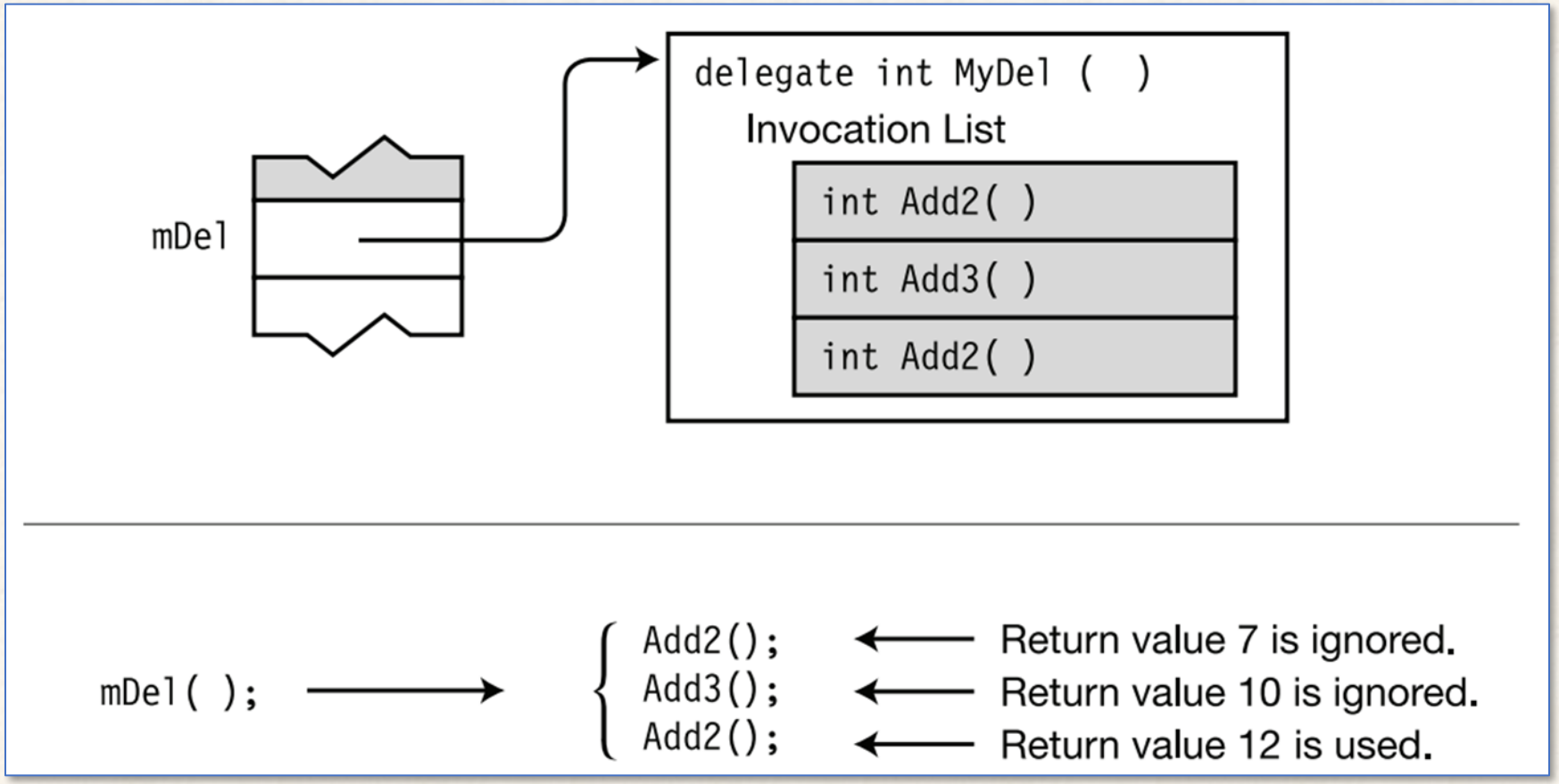
Delegate Example 2:

Invoking Delegates with Return Values

การเรียก Delegates ที่มี Return Values

```
delegate int MyDel();
class MyClass
{
    int IntValue = 5;
    public int Add_2() { return IntValue += 2; }
    public int Add_3() { return IntValue += 3; }
}

class Program
{
    static void Main(string[] args)
    {
        MyClass mc = new MyClass();
        MyDel md = mc.Add_2;
        md += mc.Add_3;
        md += mc.Add_2;
        int sum = md();
        Console.WriteLine($"Value = {sum}");
    }
}
```

การเรียก Delegates ที่มี Return Values



การเรียก Delegates ที่มี Reference Parameters

- ถ้า delegate มี parameter เป็นแบบ reference ค่าของ parameter จะถูก update ให้เป็นปัจจุบันก่อนเรียกใช้เสมอ

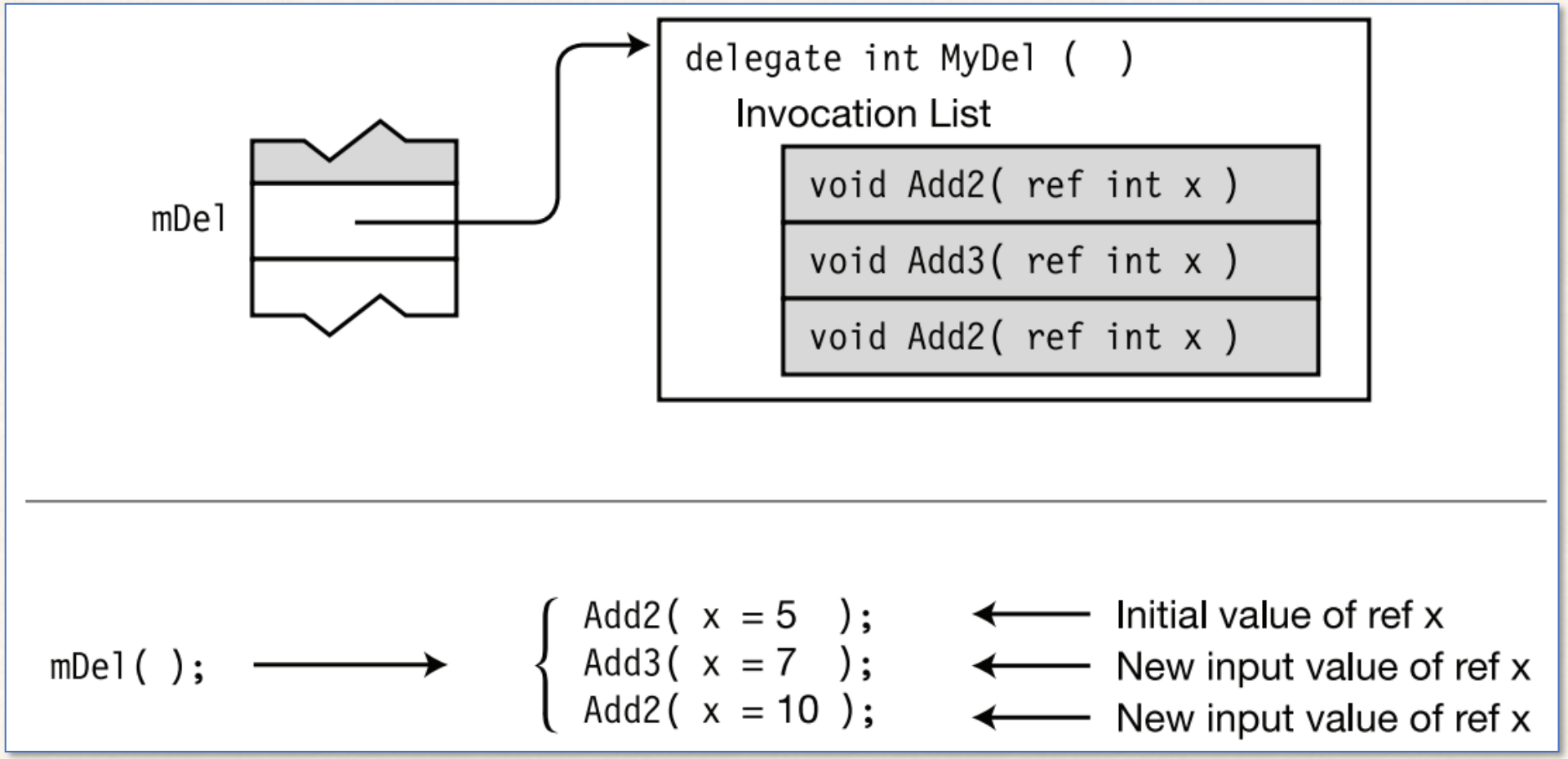
Delegate Example 3:

Invoking Delegates with Return Values

การเรียก Delegates ที่มี Reference Parameters

```
delegate void MyDel(ref int x);
class Program
{
    public void Add_2(ref int x) { x += 2; }
    public void Add_3(ref int x) { x += 3; }
    static void Main(string[] args)
    {
        Program pc = new Program();
        MyDel md = pc.Add_2;
        md += pc.Add_3;
        md += pc.Add_2;
        int x = 5;
        md(ref x);
        Console.WriteLine($"Value = {x}");
    }
}
```

การเรียก Delegates ที่มี Reference Parameters



Anonymous methods

- ใน delegate เราสามารถเพิ่ม method ได้ทั้งแบบ static และ instance
  - method เหล่านั้น ต้องถูกสร้างเป็น member ของ struct หรือ class ก่อนเสมอ
- แต่ถ้าเราต้องการใส่ code ที่ทำงานเพียงครั้งเดียว ไม่จำเป็นต้องสร้างเป็น method ไว้ใน delegate
  - ไม่จำเป็นต้องนำ code เหล่านั้นไปใส่ใน method และหาคลาสให้อยู่
  - สามารถใช้ anonymous method มาทำงานแทนได้

Named methods

```
class Program
{
    public static int Add_20(int x)
    {
        return x + 20;
    }
    delegate int SomeDel(int param);
    static void Main(string[] args)
    {
        SomeDel del = Add_20;

        Console.WriteLine(del(5));
        Console.WriteLine(del(6));
    }
}
```

Anonymous methods

```
class Program
{
    //public static int Add_20(int x)
    //{
    //    return x + 20;
    //}
    delegate int SomeDel(int param);
    static void Main(string[] args)
    {
        SomeDel del = delegate(int x)
                      {
                          return x + 20;
                      };

        Console.WriteLine(del(5));
        Console.WriteLine(del(6));
    }
}
```

Named Method vs. Anonymous methods

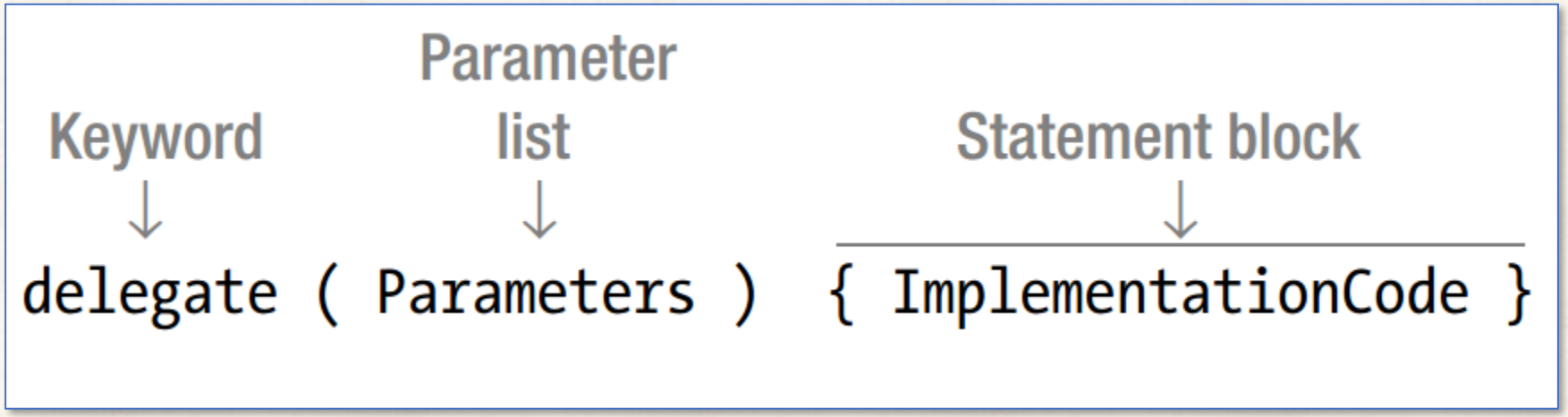
```
class Program
{
    public static int Add_20(int x)
    {
        return x + 20;
    }
    delegate int SomeDel(int param);
    static void Main(string[] args)
    {
        SomeDel del = Add_20;

        Console.WriteLine(del(5));
        Console.WriteLine(del(6));
    }
}
```

```
class Program
{
    //public static int Add_20(int x)
    //{
    //    return x + 20;
    //}
    delegate int SomeDel(int param);
    static void Main(string[] args)
    {
        SomeDel del = delegate(int x)
                      {
                          return x + 20;
                      };

        Console.WriteLine(del(5));
        Console.WriteLine(del(6));
    }
}
```

การใช้งาน anonymous method

- ใช้เป็น initializer expression ในขณะประกาศตัวแปร delegate
- ใช้ที่ด้านขวาของสมการตอนทำ combining delegates.
- ใช้ที่ด้านขวาของการกำหนด delegate ให้กับ event
  - เนื่องจากการเขียนโปรแกรม multithread จะเป็นแบบ asynchronous จึงต้องใช้ delegate ในการเรียกใช้ method ที่อยู่คนละ thread

รูปแบบของ anonymous method



ใน anonymous method ไม่จำเป็นต้องระบุ return type

anonymous method : Return type

```
class Program
{
    delegate int SomeDel(int param);
    static void Main(string[] args)
    {
        SomeDel del = delegate(int x)
                      {
                          return x + 20;
                      };
    }
}
```

anonymous method : Parameters

- Parameter list ของ anonymous method จะต้องตรงตามของ delegate ตามส่วนประกอบดังนี้
  - จำนวน
  - ชนิดและตำแหน่ง
  - modifiers
- แต่เราสามารถละ parameters ได้ถ้า
  - delegate parameter ไม่มีตัวใดที่เป็น out parameter
  - anonymous method ไม่จำเป็นต้องใช้ parameter

anonymous method : params Parameters

- params parameter ช่วยให้เราสามารถป้อน parameter ให้กับ method เป็นจำนวนเท่าใดก็ได้
- ถ้ามีการประกาศ params parameter ในรายการ parameter ของ delegate list แล้ว ให้ตัดคำว่า params ออกจาก anonymous method

```
        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)
         {
            ...
         };
```

Lambda Expressions

- ใน anonymous method คำว่า delegate ถือเป็นส่วนเกิน
  - เนื่องจาก compiler รู้แล้วว่าเราต้องการกำหนด method ให้ใช้งานกับ delegate (ใช้กับอย่างอื่นไม่ได้)
- เราสามารถเปลี่ยนรูปแบบการประกาศ delegate ของ anonymous method ให้ง่ายขึ้นโดยการ
  - ตัดคำว่า delegate
  - เพิ่ม => ระหว่าง parameter list และ body ของ anonymous method (ก็คือส่วนที่ล้อมรอบด้วย {})

Lambda Expressions

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

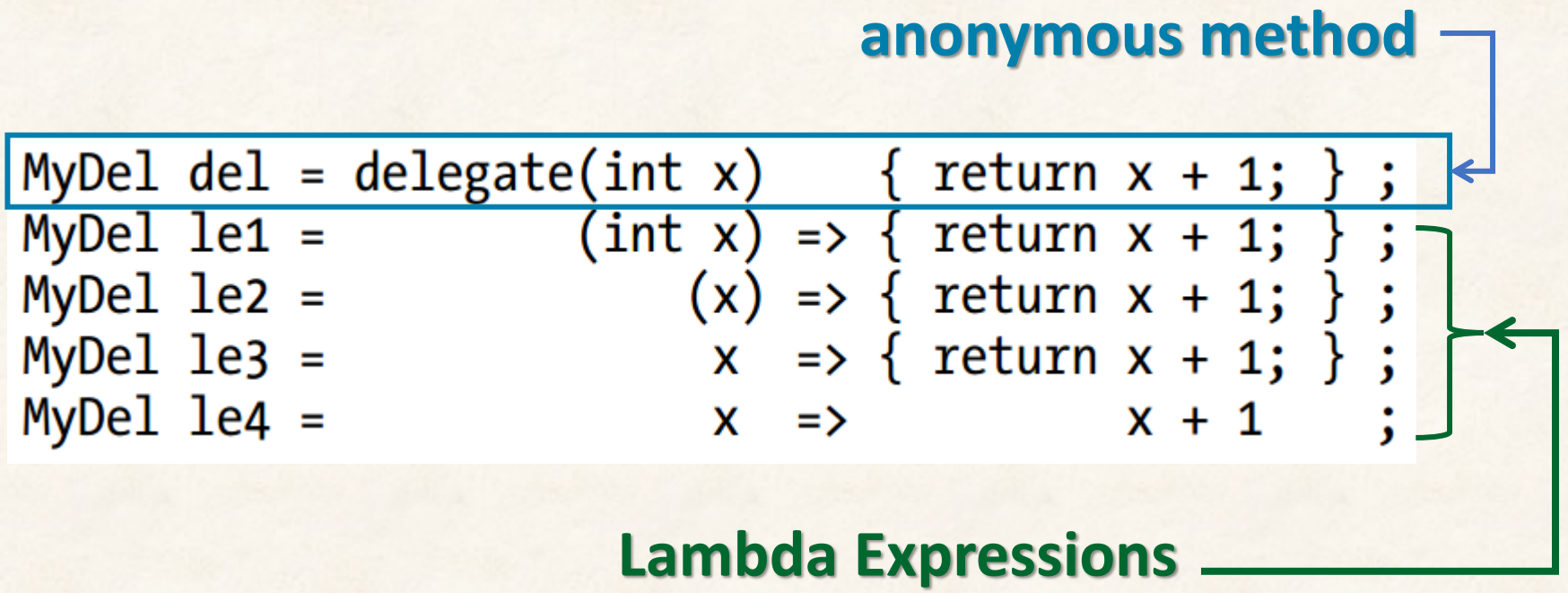
ตัดออก

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

เพิ่ม

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

Lambda Expressions



methods ทั้งหมดข้างบน ให้ผลลัพธ์เหมือนกัน

เลือกเองว่าจะเขียนแบบไหน

ตอนเริ่มต้นอาจทำความเข้าใจยากหน่อย

Anonymous vs. Lambda Expressions

```
class Program
{
    delegate int SomeDel(int param);
    static void Main(string[] args)
    {
        SomeDel del = delegate(int x)
                      {
                          return x + 20;
                      };

        Console.WriteLine(del(5));
        Console.WriteLine(del(6));
    }
}
```

```
class Program
{
    delegate int SomeDel(int param);
    static void Main(string[] args)
    {
        SomeDel del = x => x + 20;

        Console.WriteLine(del(5));
        Console.WriteLine(del(6));
    }
}
```